POLITECNICO DI TORINO

Master's Degree in Electronics Engineering

Department of Electronics and Telecommunications

Master's Degree Thesis

Design and simulations of a class D audio amplifier featuring novel digital modulations

Supervisors:

Prof. Francesco Musolino

Prof. Paolo Stefano Crovetti

Candidate:

Samuele Gisonno

Academic Year 2020-2021

July 2021

Tutto ciò che è comodo,
è stupido

Ringraziamenti

Senza le persone, un lavoro tecnico perde di senso. La vicinanza, la pazienza e il supporto morale di tante persone hanno permesso la realizzazione di questo lavoro di tesi. In particolare, i miei pensieri di gratitudine vanno

- a mio Padre e a mia Madre che hanno sostenuto economicamente il mio percorso, hanno avuto pazienza e hanno creduto in me più di quanto non lo facessi io;
- a mia Sorella che è stata una figura di d'esempio per fare sempre meglio secondo la mia coscienza e mi ha ispirato a vedere il mondo con occhi diversi;
- a Michele Ferrero, Riccardo Placenti e Marta Prelli per aver condiviso con me le mie preoccupazioni, sopportato i miei limiti e capito l'importanza che questo percorso ha avuto per me;
- a Davide Gualtieri e Davide Belgradi, da sempre compagni di viaggio, senza i quali conoscerei meno il mondo e me stesso e mi insegnano che "nella vita si vive una volta sola...";
- ad Elena Perrone, che ha sempre saputo ascoltare, suggerirmi e ha creduto in me incondizionatamente;
- a Nonno Nicola, che mi ha sempre mostrato che attraverso la Volontà, l'Intraprendenza, la Determinazione e l'Adattamento si raggiungono gli obiettivi;
- ad Arianna Di Guida, che mi ha permesso di crescere come uomo e fare esperienza di un aspetto della vita non comprensibile con la ragione e la matematica, ma ben più folle, misterioso e affascinante.

Un ringraziamento particolare al professor Francesco Musolino e al professor Paolo Stefano Crovetti per l'umanità e la professionalità mostrate e per avermi sostenuto, stimolato e consigliato sempre in maniera costruttiva, aiutandomi a crescere a livello professionale.

Contents

Abstract

In recent years there has been an increasing demand of portable electronic devices, from mobile phones to personal computers, from MP3 players to hearing-aid systems which have to be smart, light and low-power consuming. As far as the power dissipation is concerned, the audio power results one of the most critical. For this reason, a lot of research has been carried out to find out solutions that allow to reach very high power efficiency, without reducing the audio quality. In fact, even if traditional linear amplifier classes (e.g. A, B, AB) can reach high performance in terms of audio quality, with low distortion, they have severe limitations in terms of efficiency, dissipated heat and weight (in particular for high power levels). Class D audio switching amplifiers, proposed in 1958, can reach 100% power efficiency (in real cases about 80%-90%) against 75% of class AB (in real cases about 30%-40%). The increase in power efficiency has an impact both in terms of less heat generation and lower weight, increasing the portability of the system in which they operate.
Moreover, advances in research and development of semiconductor technology have allowed the integrability of those amplifier circuits and higher *switching frequency.*
One of the main drawback of class D amplifiers is the generation of higher distortion level of amplified signal with respect to that of traditional classes. To reduce this distortion, solutions have been proposed most of which exploit the features of *closed loop negative feedback.* In this context, control laws implemented through digital programmable circuits it is preferred to analog implementations both in terms of noise immunity and in terms of higher flexibility since the control can be specified in terms of software.
In this thesis, a class D audio amplifier has been designed taking advantage of the most recent state of the art architectural and technological solutions. In particular, the modulation section is analyzed in detail, highlighting the limitation of this section implemented in digital way by means of DPWM and DDPWM and overcome them by means of a new proposed modulator that is a combination of DPWM and DDPM modulation techniques. The models of the open- and closed-loop amplifier, implemented in MATLAB, are designed under some specifications and a rigorous methodology design for closed-loop system is shown. Then, a comparison among

the proposed solutions is done, highlighting the differences among them in terms of distortion, noise and speed, which influence the efficiency of the overall system. Finally, a possible physical implementation is suggested using GaN and MOS transistors, to compare in future the effect of different technology on the experimental performances of a class D amplifier.

Chapter 1

Introduction

The applications of audio systems are very numerous and cover the major part of our daily life aspects: entertainment field, like games, videos, films (where the role of the audio in the entire production cover the 50% of the work), music, communication applications like video-calls, video-conferences and calls; even in medical applications is extremely important, from hearing-aid devices to therapies to remove the perception of tinnitus or to treat the autism in children; in mobile phone, where the audio is important not only for call, but even to give commands and instructions to apps and to interface the phone by use of own voice. All these applications are requiring an increasing demand for better audio quality and more portability. Moreover, in a world where renewable energy is becoming always more important, to reduce energetic environmental impact and wastes, the efficiency of electronic systems is fundamental. In this context, main characteristics of audio system have to be

- portability;
- high audio quality;
- energy efficiency.

1.1 What is an audio system

"The audio is an electronic information that represents the sound"[1]. This definition put in evidence the difference between *sound* and *audio*

- with *audio* is intended an electrical information related to sound;
- with *sound* is referred to a wave propagation in an elastic mean (like air) due to a oscillating object (like a string of a guitar).

Both entities bring information because the *sound* is the variation of a bias pressure condition and the *audio* is related to the sound, so it brings the same information of the sound. The difference between the two stays in how this information is propagated: *sound* moves the information through the elastic mean (e.g. air, water) while *audio* moves the information through electrical physical quantity (e.g. voltage, current). In other words, the audio information is a *translation* of sound information in electrical world. This translation is called *transduction* and it is made by a sensor (e.g. microphone); the opposite *transduction*, from electrical to mechanical world, is done by the *actuator* (e.g. speaker). A generic audio system to play music or to amplify voice consists in these sections (fig. 1.1)

- *sensor*: generally is the microphone (indicated as MIC in figure 1.1) and allows to *transduce* the sound in electrical physical quantity (generally voltage);
- *ADC* (*analog-to-digital converter*): convert analog information from analog to digital domain; in this way it is possible to process the converted signal by mean of algorithms; generally, analog-to-digital converter includes amplification section to adapt the dynamic of the signal to the dynamic of the converter;
- *DSP*: it is a digital block that processes, elaborates, edits and modifies the digital information provided by the ADC in order to get specific goals; this could be a typical configuration of studio recording; the advantage of signal elaboration by means of digital strategies has strong advantage in terms of configurability, flexibility and cost;
- *DAC* (*Digital-to-Analog converter*): opposite to ADCs, it allows to convert the digital audio signal, modified by the DSP, back to analog domain;
- *output stage*: is the amplifier driven by the output signal of DA and provides the nominal power to the load (the speaker);
- *speaker*: is the transducer that has the opposite function of microphone; *trasduces* an electric signal in sound.

This general scheme in figure 1.1 could include, depending on the applications, crossover filters placed after or before the output stage to avoid the damage of speaker; in fact depending on the driven speaker (twitters, mid-range, woofers, subwoofers), if the bandwidth of the signal is not adequate, the speaker could be damaged, so it is necessary limit the band of the signal.
The quality of the final product depends on different aspects starting from the position of the microphone to record the instruments, the quality of microphone, amplifier, ADC converter and the algorithm to manipulate the recorded audio (DSP,

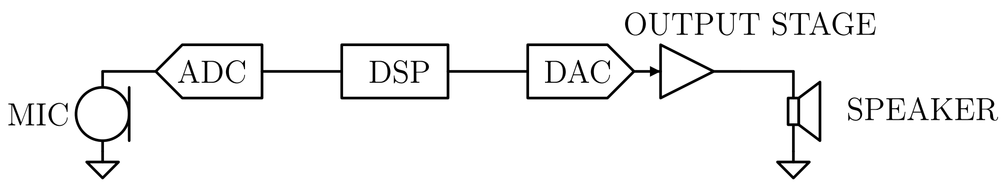


Figure 1.1: General scheme of an audio system

plugin, *DAW, Digital Audio Workstation*): so, even if the presence of qualified audio technicians is a must, in these productions, to obtain a good result, it's important that all the *electroacustic chain* has high quality, in particular the hardware part. In fact, the quality of the *trasducers* (speaker), *sensors* (microphone), ADC and amplifier influence the quality of the transduced sound information. The electroacustic chain inevitably introduces a modification of the original signal, even if no DSP algorithm is applied to original signal. In fact non-linear effects, the resolution of ADC, the thermal noise, the frequency response of sensors, have an impact on the *transduction* of sound information.

1.2 Output stage limitations

In a system similar to that of figure 1.1, one of the most problematic section is the output stage. In fact, it is one of the main responsible of output signal distortion and, against the goals of low-power system consuming, one of the most power consuming section. This drawback is particularly important for portable applications because reduces the battery duration and increases the heat generation. These issues are particularly evident when the output stage is implemented by using linear amplifier classes which characteristics are in contrast with the requirements of energy efficiency and portability.
For these reasons, *switching amplifiers* are replacing the traditional linear classes thanks to their power efficiency, high portability and integration. Particularly important are becoming class D audio amplifiers, more and more used in hearing-aid devices, phones, MP3, TV, multimedia applications. The working principle of these amplifiers is the same of buck converters: the audio input signal is sampled by means of a comparation with high frequency (switching frequency) triangular waveform, which result is a PWM signal that drives the power transistors of the output stage, amplifying the PWM signal. Finally this signal is filtered by an output filter (typically a second order one) and the amplified audio information is recovered. The use of this architecture allows to reach very high power efficiency and high density. The PWM modulator can be easily implemented in digital way: the input

signal is sampled and hold at switching frequency into a N bits register and the resulting code is compared with the one generated by a N bits free-running counter that works at switching frequency, giving as output an high value if input code is lower than counter one or low value if it's the contrary. This is the so-called *uniform digital PWM modulation* (DPWM). Moreover, recent works have introduced digital-assisted structure for closed-loop class D amplifiers that have increased the performances in terms of distortion and SNR (Signal-to-Noise Ratio) thanks to the possibility to design the complex compensator transfer function in terms of software, by mean of FPGA or microcontroller.

1.3 Thesis goals and outline

In this overview, the goals of the thesis is to implement a model of a switching class D amplifier, both in open- and closed-loop configurations. The main goals is to evaluate the impact of the modulator on the amplifier performances. In particular, two types of modulation are considered: DPWM and DDPWM, one of the most recent kind of modulation introduced in literature in power converter field. The first chapter, **Class D amplifiers**, covers a detail review of characteristics, properties and literature works about class D amplifiers; than, in **Design of a class D amplifier** chapter, are shown the results of MATLAB design and Simulink simulations starting from design specs and varying some system parameters and configurations; in particular, configurations with DPWM, DDPWM and a new combination between DPWM and DDPM modulation are implemented and tested in closed- and open-loop comparing their performances and drawbacks; in chapter **Conclusions** are summarized the results found in the thesis, suggesting future works and possible studies.

Chapter 2

Class D amplifiers

Class D amplifier topology was proposed the first time in 1958. The main difference among this class and the traditional ones (A, B, AB) is the operating mode of active device: considering MOS transistor as active devices, in linear amplifiers they operate in saturation region, while in class D amplifiers they operate in linear or cutoff regions [2]. *"This condition of operation allows class D amplifiers to achieve higher efficiency than linear amplifiers. When the output transistor is completely on, it shows almost no voltage drop, and if it is completely off, it presents an almost null current, and thus the power dissipation is negligible. Therefore, ideally the efficiency can reach 100%"* [2]. Due to its high efficiency and thanks to development of technologies, this kind of amplifiers have become widely use in portable systems like hearing-aids devices [2].
In audio system is not only important the efficiency: even audio quality is a fundamental requirement. For this reason, because of the open-loop configuration of class D amplifiers shows very high distortion obtaining a relatively low audio quality, closed-loop topologies are developed. These topologies are much complex to design, but thanks to the feedback and improvement of integration technologies, it is obtained high efficiency and low-distortion class D amplifiers [3][2][4].
Among the different types of analog modulations [5][6], in recent years digital architectures are developed adding lot of advantages in terms of reliability, cost and design difficulties. In particular digital approach has allowed a better control on EMI emission: in fact, due to high switching frequency, in particular of the output stage, this kind of circuits induce lot of high frequency noise that can create *electromagnetic compatibility* problems. Digital techniques allows to reduce those problems in different ways: by changing modulation type (*ternary scheme* [7]), using *digital spread spectrum modulation* [8] or even reducing crosstalk effect between different channels [9].
In this chapter the working principles, the topologies and the studies done about

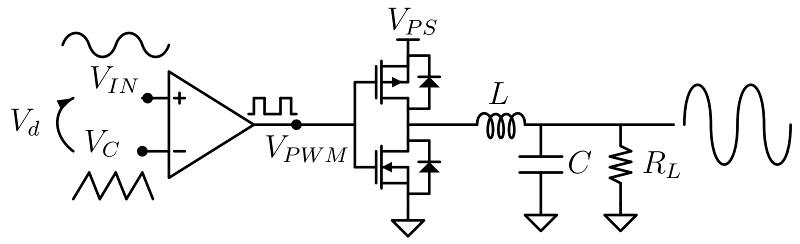


Figure 2.1: Analog CDA in open-loop configurations

class D amplifiers (CDA) in literature are analyzed, enlightening the advantages and disadvantages of each of them. Moreover, the main parameters traditionally used to analyze the performances of amplifiers are described. Finally, particular attettion is paid on similarities and differences among analog and digital topologies and to PWM and DPWM modulations due to their intensive used in power applications and the advantage of the implementation of the innovative DDPWM one on it.

2.1 Structure of class D audio amplifiers

The class D audio amplifier which scheme is shown in figure 2.1 is even known as *switching power amplifier*, due to the presence of switching devices as output stage. The working principle is the following: the analog input signal V_{IN} (see figure 2.1) is modulated by an high frequency signal V_C generating a *pulse-width modulated* signal V_{PWM}. This signal is basically *a square wave with variable duty cycle, where the average value corresponds to the input signal*, as shown in figure 2.2. This signal drives two power switches that amplify the driving signal. Finally, appropriately filtering the amplified signal, it is possible to recover the amplified input signal.

2.1.1 Pulse-width modulator

Pulse-width modulated (PWM) signal is generated by a comparison between the input signal V_{IN} (which has a band f_i) and a carrier signal at frequency $f_{sw} \gg f_i$. Due to the Shannon theorem, in principle f_{sw} should be $2f_i$. In practical cases, due to non-idealities of the circuit components, $f_{sw} > 10f_i$ [3]. Considering that the input signal is in the audio band ($20 - 20\,000\,\text{Hz}$), $f_{sw} > 200\,\text{kHz}$. Generally, switching frequency of amplifiers designed and described in literature are between $300\,\text{kHz} - 1\,\text{MHz}$ [3], but using digital architecture can reach $2\,\text{MHz}$ [10].
Because of the switching frequency is much higher than the input signal maximum one, a graphical analysis of the PWM modulator can be done considering the input signal constant (fig. 2.2). V_{PWM} is the output voltage of differential amplifier. The

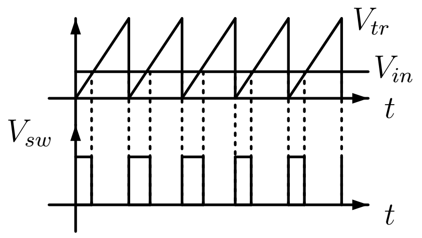


Figure 2.2: Two-level pulse width modulated signal

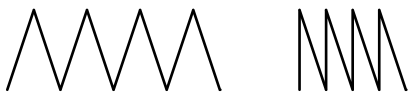


Figure 2.3: dual edge (on the left) and leading edge (on the right) triangular waveform

comparison is done taking the difference between carrier signal and input signal

$$V_d = V_{in} - V_{tr} \Rightarrow V_{PWM} = \begin{cases} V_H & \text{for } V_d > 0 \\ V_L & \text{for } V_d < 0 \end{cases} \tag{2.1}$$

where V_H is the maximum voltage that amplifier can give (basically its power supply voltage), V_L is the lowest one (in single power supply, ground), V_d the differential input of differential amplifier (see fig. 2.1). In figure 2.2 it is shown a *trailing edge triangular waveform* for the carrier signal, but other kind of waveforms can be used

- *leading edge triangular waveform*;
- *dual edge triangular waveform*;

Different studies have shown that the waveform used as carrier signal modifies the amount of distortion of the output signal: in particular, for the same switching frequency, DC voltage offset and modulation index, the number of harmonics is much higher for trailing edge carrier than dual edge [5]. Considering a sinewave input signal, the **modulation index**, $\mathbf{m_a}$ is the ratio between the peak-to-peak voltage of input signal ($V_{peak,ref}$) and the peak-to-peak voltage of the triangular waveform ($V_{peak,carrier}$)

$$m_a = \frac{V_{peak,ref}}{V_{peak,carrier}} \in (0, 1) \tag{2.2}$$

Comparators for pulse width modulator, in analog design should have the following characteristics

- high slew rate (to minimize propagation delay);

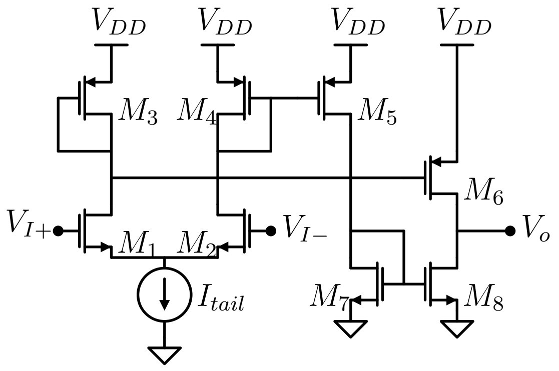


Figure 2.4: Three current mirror transconductance amplifier with push-pull output stage [5, p.97]

- high *Power Supply Rejection Ratio* (*PSRR*) to suppress the noise contribution from power supply.

In [5], a design for *integrated circuit* (IC) architecture is shown, using a *three current mirror transconductance amplifier with push-pull output stage* (fig. 2.4). Resuming, this stage, the **pulse-width modulator**, *gives information about the input signal through the variation of duty cycle of generated PWM signal*: so the transfer function is the ratio between the duty cycle d and the input signal V_{in}. In particular, depending on the power supply used for the differential stage, there are two possible transfer functions:

- **Single power supply**
  The stage is supplied between 0 V and a non-zero voltage power supply V_{DD}, generally positive: this means that the input signal V_{in} and V_C have to stay between 0 V and V_{DD} (neglecting the limitation due to *common mode input range* of the differential input pair). In this condition

  - if $V_{in} = V_C = V_{DD} \Rightarrow V_d = 0 \Rightarrow d = 1$;
  - if $V_{in} = 0 \Rightarrow V_d = V_C \Rightarrow d = 0$.

  The transfer function can be expressed by a proportion, where $V_C = V_{tr}$ is the peak-to-peak amplitude of carrier signal

  $$1 : V_{tr} = d : V_{in} \Rightarrow \boxed{G_M = \frac{d}{V_{in}} = \frac{1}{V_{tr}}} \tag{2.3}$$

  where G_M is the *modulator gain*, related to the duty cycle.

- **Dual power supply**
  The stage in supplied by a negative and a positive power voltage, $-V_{DD}$ and

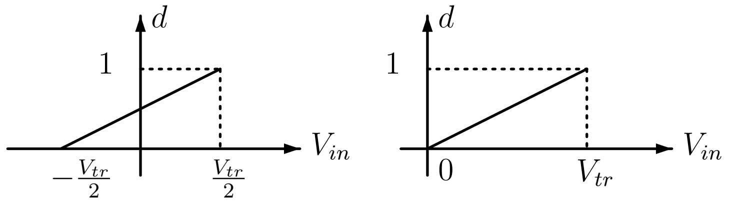


Figure 2.5: Transfer function of pulse-width modulator with single (on the right) and double power supply (on the left)

V_{DD} respectively. In a similar way to the previous case, the two input signal can have a voltage range $[-V_{DD}, V_{DD}]$. In this condition the output voltage of the comparator could be

- if $V_{in} = V_C = -V_{DD} \Rightarrow V_{PWM} = -V_{DD} \Rightarrow d = 0$;
- if $V_{in} = V_C = V_{DD} \Rightarrow V_{PWM} = V_{DD} \Rightarrow d = 1$;

As before, the transfer function is linear again, but with an offset of 1/2 (see fig. 2.5). It can be written as

$$\boxed{d = \frac{1}{V_{tr}} V_{in} + \frac{1}{2}} \tag{2.4}$$

2.1.2 Output stage

The **Output stage** shown in figure 2.1 is an **half bridge** stage: basically when $V_{PWM} = V_H$, the low-side transistor (an NMOS transistor) conducts, while, if $V_{PWM} = V_L$ is the high-side one (a PMOS transistor) that conducts. So the two transistors works as *complementary switches*: they do not work in saturation region, differently from traditional classes. This working mode reduces power losses because when the transistors are conducting, they show a small on resistance while when they are not conducting show a very high resistance ($> 1\,\mathrm{M\Omega}$), obtaining (ideally) a zero current flow: basically this stage is an *inverter that amplifies and inverts the driving signal*

$$V_{sw} = \begin{cases} V_{PS} & \text{if } V_{PWM} = V_L \\ 0 & \text{if } V_{PWM} = V_H \end{cases} \tag{2.5}$$

where V_{sw} is the voltage at the drains of the switches. The main disadvantages due to the presence of the switches are two:

- because of the output signal of the output stage is basically a square wave, an output filter is necessary to recover the amplified input signal and reducing the effect of higher harmonics due to PWM modulator; this increases the cost of the system and the occupied area on Printed-Circuit Board (PCB);

- the switches generate high $\frac{dv}{dt}$ and $\frac{di}{dt}$ and this induces electromagnetic interference that can affect devices like ADC, radio frequency devices, voltage and/or current supply; so techniques like *spread-spectrum modulation* are often used to limit this drawback; even changing the type of modulation can reduce the *electromagnetic interference* (EMI).

As said before, ideally the CDA can reach 100% efficiency. Anyway due to non-ideality of the switches and of the other components of the amplifier, power losses are present. In particular [5]

$$P_{loss} \simeq P_Q + P_{CL} + P_{SW} + P_{BD} \tag{2.6}$$

where

- P_Q: *quiescent power*;
- P_{CL}: *conduction losses*;
- P_{SW}: *switching losses*;
- P_{BD}: *body-diode losses.*

Conduction losses are related to the conduction time of the transistors, when they behave as a resistance of value R_{dsON}: so they are basically ohmic losses related to the drain-source resistance. This dissipated power is dominant when the amount of output power (and of the output current) is large respect to the maximum achievable value. Switching losses are related to the transition between *ON* and *OFF* states of the switch, in particular to the charging and discharging of the parasitic MOSFET capacitances. Body-diode losses occur due to body-diode conduction and reverse recovery charge. Power losses can be written as [5]

$$P_{CL} = I_{o,RMS}^2 R_{dsON} \tag{2.7}$$

$$P_{SW} = \sum_i (f_{sw} \cdot V_{CP}^2 C_{P_i}) + V_{DD} \cdot I_{o,pk} \cdot \frac{2t_{trans}}{t_{sw}} \tag{2.8}$$

$$P_{BD} \simeq V_{SD} \cdot f_{sw} \cdot (I_{o,pk} t_{dead} + I_{rrm} t_{rr}) \tag{2.9}$$

where

- $I_{o,RMS}$: *mean root square of output current*;
- f_{sw}: *switching frequency*;
- C_{P_i}: *i-th parasitic capacitance*;
- V_{CP}: *parasitic capacitance C_{P_i} voltage drop*;
- $I_{o,pk}$: *peak output current*;
- V_{SD}: *body-diode source-drain volt-*

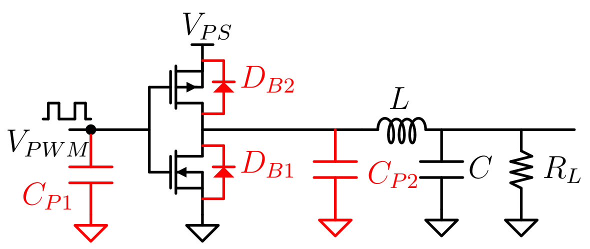


Figure 2.6: Class-D output stage with parasitic elements [5, p.33]

age;

- t_{trans}: *ON/OFF state transistion time*;
- t_{dead}: *dead time* (to avoid shortcircuit between V_{PS} and ground);
- I_{rrm}: *maximum reverse recovery current*;
- t_{rr}: *reverse recovery time.*

All the parasitic elements are shown in figure 2.6 in red

The efficiency can be characterized by three different regions (fig. 2.7), depending on the amount of output power respect to the maximum achievable one:

- ***for low power levels***: *switching losses* and *quiescent losses* dominate and the efficiency is the minimum;
- ***for medium power levels***: all power losses contributions have more or less the same impact on the efficiency;
- ***for high power levels***: *conduction* and *body-diode power losses* dominate the reduction of efficiency, due to high amount of output current.

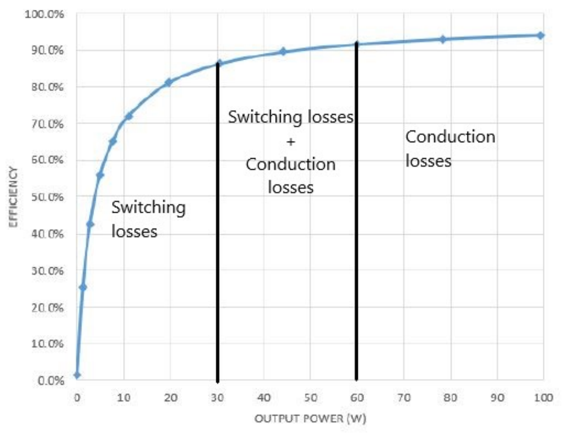
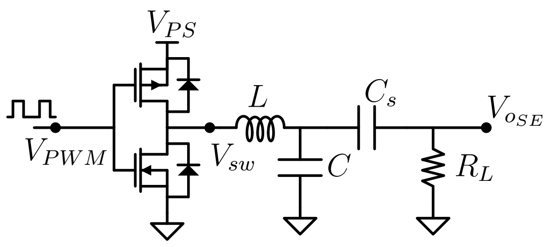
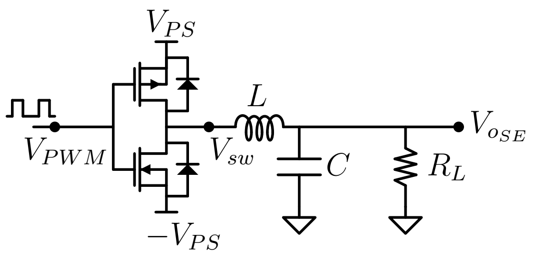


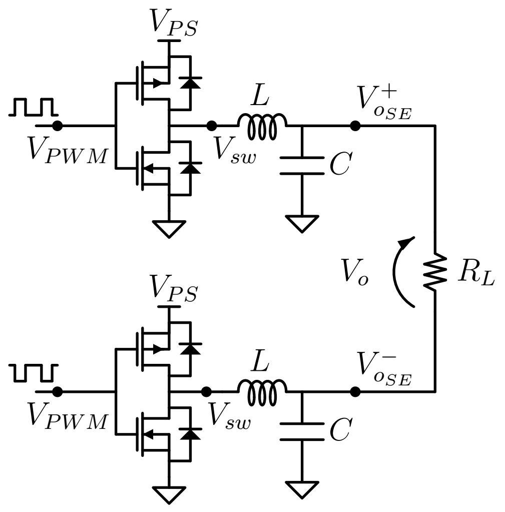
Figure 2.7: Power vs efficiency with power losses regions



(a) *Single-ended configuration with single power supply*



(b) *Single-ended configuration with dual power supply*



(c) *Bridge-Tied Load configuration*

Figure 2.8: Output stage configurations

2.1.3 Other output stage configuration

The half-bridge configuration described in figure 2.1 is a *single-ended (SE)* topology: basically one terminal of the load is connected to the output of the amplifier, while the second one is connected to ground. In this case the power supply of the output stage should be *single* or *dual.* In the first case, if power supply is V_{PS}, the output voltage of output stage shows a DC component that should be block because it must not reach the speaker to avoid damages. This means that is necessary to add an additional capacitor in series to the load: as a consequence, cost and occupied area increases. Moreover, even if with dual power supply this problem should be theoretically removed ($V_{sw_{DC}} = 0$ if the system is perfectly symmetric), two power supplies are not so simple to obtain. To overcome these limitations, a *differential configuration*, ***Bridge-Tied Load (BTL) configuration*** is used, in particular for general speaker applications. The topology is based on ***Full-bridge*** output stage and have three main advantages with respect to the single ended one

1. it need a single power supply;
2. for same power supply, can reach output power four times higher than the single ended configuration;
3. does not require a series capacitor to block the DC component; in fact, being a differential configuration, the load has a terminal connected to a branch and one terminal connected to the other branch (fig. 2.8c) so common mode contribution are basically null (in perfect symmetric condition).

2.1.4 Amplification factor of class D amplifier

Ideally, the amplified input signal can be measured after the output filter. The *amplification factor* is related to the PWM gain and to the power supply of output stage. Let's consider the amplifier as two gain stages in cascade:

- **PWM stage**: the gain of this stage, as shown in formula 2.3 and 2.4, is related to duty cycle d of the generated PWM signal and it depends on power supply (single or dual)
- **output stage and filter**: the output of this stage $V_{o_{SE}}$ is *the average value of* V_{sw} *voltage* (fig. 2.8);

So, for single-ended stage, depending on the power supply of output stage, there are two possible cases.

- **Single ended power supply** (fig. 2.8a)
  In this case the output voltage is given by

$$V_{o_{SE}} = \overline{V_{sw}} = 0d + V_{PS}(1-d) = V_{PS}(1-d) = \underbrace{V_{PS}}_{bias} - \underbrace{V_{PS}d}_{signal} \tag{2.10}$$

  It is possible to see that a DC offset appears (V_{PS}); the other term is the one related to the amplification of the useful signal. Because of the input signal of output stage comes from the output of PWM stage, this signal gives information through duty cycle. This means that the gain of output stage is the ratio between the output voltage and the duty cycle of the PWM signal. In formula

$$\boxed{A = \frac{V_{o_{SE}}}{d} = -V_{PS}} \tag{2.11}$$

  From a system point of view, a single ended stage with single power supply can be consider as a stage of gain $A = -V_{PS}$. To obtain the overall gain of the amplifier, it has to be considered the PWM stage gain. Because all the system can be considered as supplied by a single power supply, the gain of PWM is the one in equation 2.3. Replacing the gain of the PWM in 2.11 can be obtained

$$V_{o_{SE}} = V_{in}(-V_{PS}d) = V_{in}(-V_{PS} \cdot \frac{1}{V_{tr}}) \Rightarrow \boxed{\frac{V_{o_{SE}}}{V_{in}} = A \cdot G_M = -\frac{V_{PS}}{V_{tr}}} \tag{2.12}$$

It can be observed that using an *inverter* gate with the same PWM signal that drives high-side and low-side switches, the output signal results with opposite phase respect to the input signal. If the switches are driven so that

$$V_{sw} = \begin{cases} V_{PS} & \text{if } V_{PWM} = H \\ 0 & \text{if } \ V_{PWM} = L \end{cases} \tag{2.13}$$

the minus sign in 2.11 disappears and there's no phase inversion.

- **Dual power supply**
  From figure 2.8b the output voltage can be calculated as

$$V_{o_{SE}} = \overline{V_{sw}} = -V_{PS}d + V_{PS}(1-d) = V_{PS}(1-2d) = \tag{2.14}$$

$$\underbrace{=}_{2.4} V_{PS}\Big[1 - 2\Big(\frac{1}{V_{tr}}V_{in} + \frac{1}{2}\Big)\Big] = -\frac{2V_{PS}}{V_{tr}}V_{in} \tag{2.15}$$

$$\Rightarrow \frac{V_{o_{SE}}}{V_{in}} = -\frac{2V_{PS}}{V_{tr}} \Rightarrow \boxed{A = -2V_{PS}, \quad G_M = \frac{1}{V_{tr}}} \tag{2.16}$$

  where A is the gain of output stage block and G_M is the PWM block gain. The overall gain of a dual stage single ended amplifier is double that the single-ended one.

For BTL configuration the voltage on the load is given by the difference of output voltages of two single-ended output stage. This means that to avoid zero voltage at the output, the PWM signal of low-side single-ended stage has to be phase inverted (see fig. 2.8c, V_{PWM} signals). In this way the voltage on the load is

$$V_o = V_{o_{SE}}^+ - V_{o_{SE}}^- = V_{o_{SE}}^+ + V_{o_{SE}}^+ \underbrace{=}_{V_{o_{SE}}^+ = V_{o_{SE}}} 2V_{o_{SE}} \tag{2.17}$$

From this equation it can be seen that for the same single power supply the BTL configuration gives an amplification two times higher than single-ended configuration. As a consequence the output power is four times higher than single-ended configuration. In formula

$$P_{o_{BTL}} = \frac{(2V_{o_{SE}})^2}{R_L} = \frac{4V_{o_{SE}}^2}{R_L} \tag{2.18}$$

$$P_{o_{SE}} = \frac{(V_{o_{SE}})^2}{R_L} \tag{2.19}$$

$$\Rightarrow P_{o_{BTL}} = 4P_{o_{SE}} \tag{2.20}$$

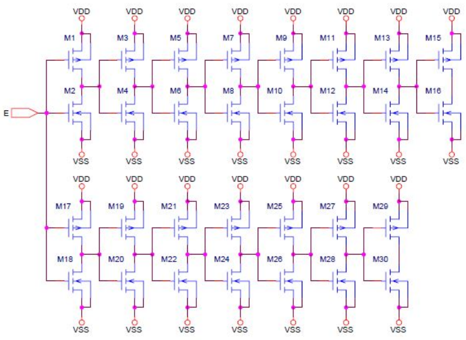
where $P_{o_{BTL}}$ and $P_{o_{SE}}$ are respectively the power on the load for single-ended (fig.2.8a) and bridge-tied load (fig. 2.8c) configurations.

2.1.5 Gate drivers

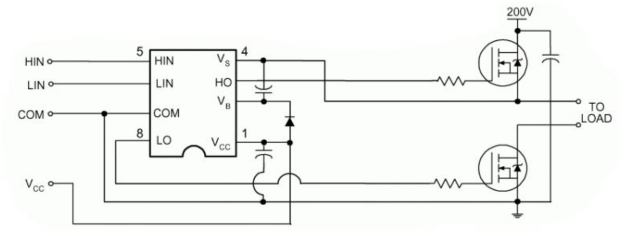
In the basic schemes shown until now one important element is missing: the ***gate drivers***. They are circuits that drive the transistors of output stage gates correctly. The PWM comparator generally cannot reach the correct value of voltage and current to charge the capacitances of output stage switches. In particular gate drivers should have the following characteristics:

- minimize the transition time between ON and OFF state (or viceversa) of switches;
- allowing a *dead time* to avoid the simultaneous conduction of the output switches;
- amplify control signal from PWM modulator to correctly drive the switch gates.

In integrated topologies, most of the time the gate drivers are designed together with the output stage like in [5, p. 101] and in [2] where the output stage is designed with a *Fixed Taper Buffer* structure (fig. 2.9a). It consists in making the *n-th* couple of transistors $M^{1/N}$ times larger than the couple $n-1$, where N is the number of couples between the first couple and the output filter and M is the size ratio of the last transistors couple in the buffer. In this way it can be obtained the best trade-off between power and speed because the first (small) transistors have lower parasitic capacitances and can drive less current for the next stage, while the last stage (the biggest one) is the slower, but can provide the correct power to the load. In this case, even if it is not specified in the article [2], the dead time is obtained due to the different number of stages for the two branches (8 for the high-side, 7 for the low-side): this modifies the propagation delay of the two branches, introducing, as a consequence, a dead time. On the other hand an example of gate driver for discrete design is shown in figure 2.9b. In conclusion, a possible complete schematic of an open loop bridge-tied load class D amplifier can be seen in figure below (2.10).



(a) *Fixed Tapered Buffer* [2]



(b) *Discrete driver* [11]

Figure 2.9: Gate drivers examples

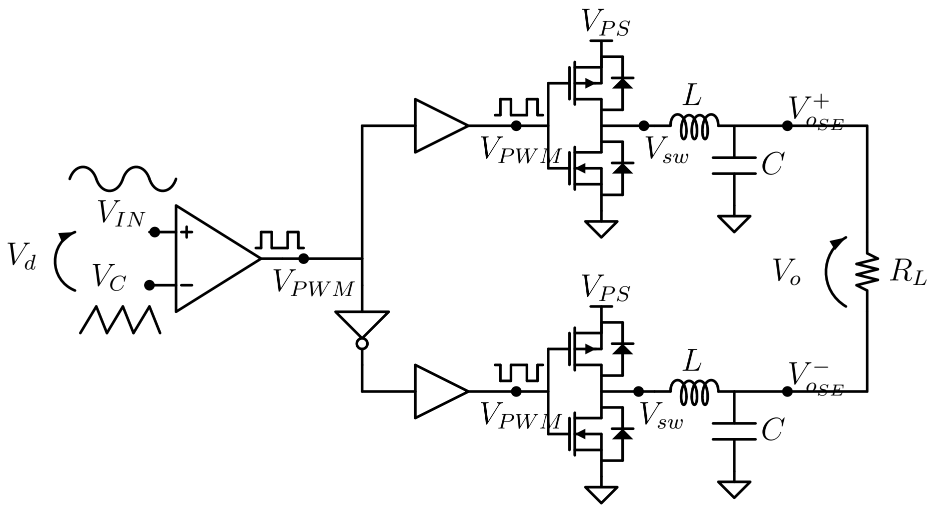


Figure 2.10: Bridge-tied load class D amplifier

2.2 Audio amplifier parameters

Performances of amplifier are mainly characterized by two parameters

1. ***distortion***;
2. ***efficiency***.

2.2.1 Efficiency

Efficiency is defined as

$$\eta = \frac{P_o}{P_o + P_{loss}} \tag{2.21}$$

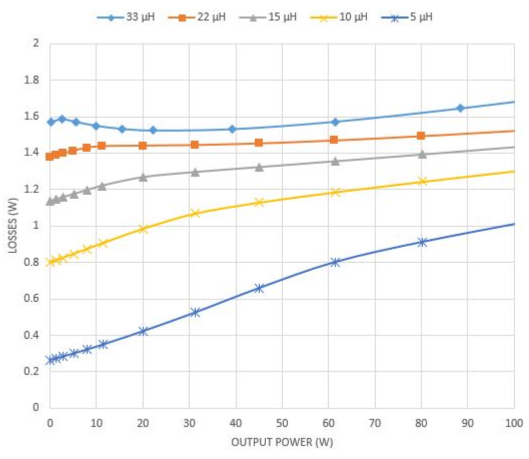
Genre	**Max power levels**		
	3 W	10 W	100 W
Rock	9.73%	26.3%	75.2%
House	14.0%	34.9%	81.3%
Hip Hop	14.7%	36.3%	82.1%
Jazz	1.79%	5.73%	37.5%

Figure 2.11: Efficiency for different music genres and output power levels

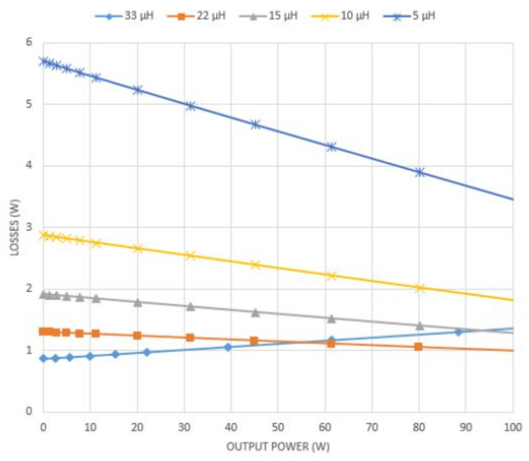
where P_o is the output power and P_{loss} is the dissipated power due to non-idealities and quiescent power. η is always less than 1 and depends on the power losses: higher the power losses, lower the efficiency. In class D amplifier the efficiency can reach 90% (and higher in recent designs like [10][2][4]). Generally, efficiency reaches higher values for high output power near to the maximum achievable by the amplifier. Moreover, in [12] it is shown that the efficiency of amplifier depends not only on the output power, but even on the musical genre (so more in general on the kind of input signal). This behaviour is related to the amount of *compression* of the input signal: music with higher dynamic, so without a strong compression (like jazz), have a medium volume lower than musics with higher amount of compression (like pop or house). High compressed input signal has a larger RMS value, which means that input power is higher than signal with lower one; as a consequence, to reach the same output power, high RMS input signal requires less average current to the output to be amplified, increasing the efficiency.
Other elements that reduce the efficiency, in addition to the parasitic of the output stage transistors described in subsection 2.1.2, are

- ***Filter losses***, in particular
  - *Equivalent series resistance* (*ESR*) of output capacitor: these losses can be minimized choosing output capacitors with low ESR;
  - *magnetic core losses*: due to ripple current through inductor; inductor losses depend on material of magnetic core and the value of inductor. In particular in [12] is shown that
    * increasing inductance value, switching losses are increased;
    * reducing inductance value, inductor losses are increased.

    Considering both effects, the overall efficiency is reduced due to the larger value of inductor losses. To reduce these losses, magnetic core should be removed. Using an *air core inductor*, the core losses become basically null. Moreover, because switching losses are lower reducing inductance value, it can be possible take a low inductance value. The problem is that air core inductors occupy large area.



(a) *Switching losses vs inductance value for GaN FET switching device* [12]



(b) *Inductance losses vs inductance value for GaN FET switching device* [12]

Figure 2.12: Switching and inductance losses at varying of inductance value

- ***Switching device***: material of the switching devices (e.g GaN, Si, SiC) has strong impact on the performances of amplifiers in terms of efficiency and distortion. The main parameters to consider are
  - R_{dsON}: the source-drain resistance influences the efficiency for high power values, where the main losses are the conduction ones; lower R_{dsON}, higher the efficiency;
  - Q_g: is the *total gate charge* related to parasitic capacitances of the switching device; in datasheet is generally given as a function of command voltage (V_{gs}); total gate charge has two main contributes:
    * Q_{gs}: is the charge accumulated in C_{gs} when $V_{GS} < V_{TH}$, where V_{TH} is the threshold voltage of the device;
    * Q_{gd}: the charge accumulated in C_{gd}, usually even called *Miller charge* [11].

  In [11] a total charge less than 20 nC is recommended for mid- and high power full-bandwidth amplifiers. Moreover, Q_g is related to speed of the amplifier and so to its distortion: with a lower Q_g, the amplifier can operate at larger frequency and, as a consequence, higher switching frequency can be used, reducing distortion, but increasing the switching losses [11].

- ***Reverse recovery charge***: as mentioned before, is the charge related to the body-diode conduction. In particular, the body-diode conducts during the

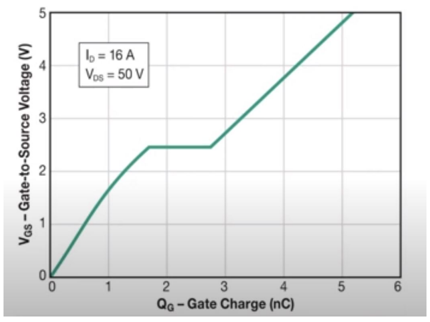


Figure 2.13: Q_g vs V_{GS} of a GaN FET device [13]

dead time to allow the continuity of the current in the inductor (fig. 2.14); after dead time, the high-side switch is turned on and V_D voltage, the voltage drop on low-side body-diode, becomes negative, so a *reverse current* start to flow in the diode discharging the junction capacitance; the same phenomenon happens to the body-diode of the high-side switch during the dead time between the turn off of the high-side switch and the turn on of the low-side switch. The main parameters that describe this phenomenon are

- Q_{rr}: total charge accumulated in the junction capacitance during t_{rr};
- t_{rr}: time during diode conducts reversely; it is considered the time necessary to reach the 10% of I_{rr};
- I_{rr}: peak value of the reverse diode current.

A descriptive graph of this phenomena is reported in figure 2.14. As far as this parameter is concerned, a strong advantage of GaN FET is the absence of Q_{rr} because no body-diode exists between source/drain and substrate [13].

- ***Stray PCB inductance and package of switching devices***: parasitic inductance due to PCB and the package inductance have a strong impact on the efficiency. A graph of a manufacturer of GaN transistors (figure 2.15) shows how the simulated efficiency of a power converter with GaN switching device is reduced and converges to experimental values if parasitic elements of PCB and drivers non-idealities are taken into account.

- ***modulation index***: efficiency increases exponentially with modulation index [3].

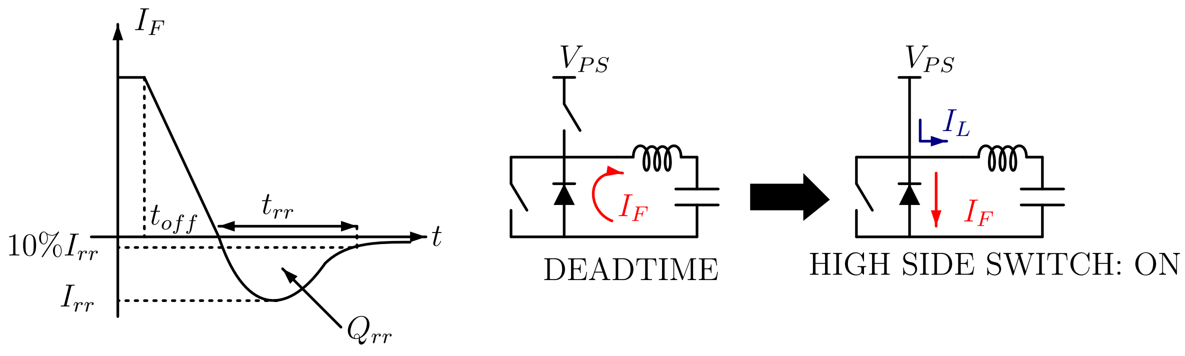


Figure 2.14: Descriptive graph of *reverse recovery charge* phenomena

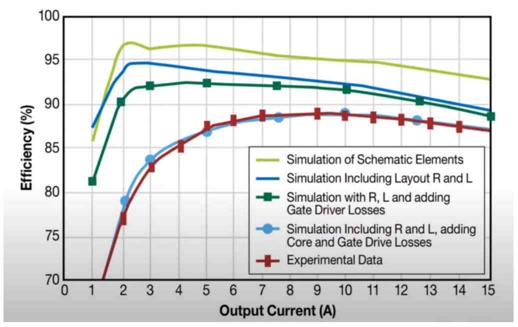


Figure 2.15: EPC simulation of power vs efficiency with different parasitic elements and experimental results [13]

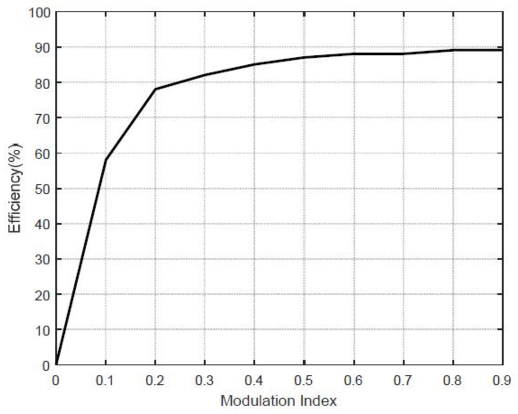


Figure 2.16: Output power vs modulation index [3]

2.2.2 Distortion

The distortion is usually measured by the ***total harmonic distortion (THD)*** defined as

$$THD = \frac{\sqrt{V_2^2 + V_3^3 + \cdots + V_n^2}}{V_1} \tag{2.22}$$

where V_i with $i = 1, 2, \ldots n$ is the amplitude voltage of the i-th harmonics. Ideally, amplifier should increase the voltage only of the fundamental harmonic V_1, but due to non-linear effects of the system, higher harmonics are generated. Lower THD, lower distortion and higher accurate reproduction of audio signal at the output. [6]. Most of the time the parameter considered is ***total harmonic distortion + noise (THD+N)*** that takes into account the contribution of amplifier noise floor; it is defined as

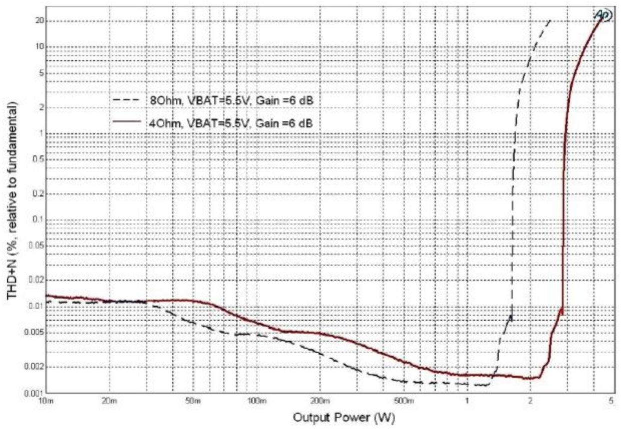
$$THD + N = \frac{\sqrt{\sum_{i=2}^{N} V_i^2 + V_n^2}}{V_1} \tag{2.23}$$

where V_n is *the integrated noise RMS voltage* [5]. Since THD and THD+N vary of several order of magnitude, they are expressed in logarithmic scale or percentage.

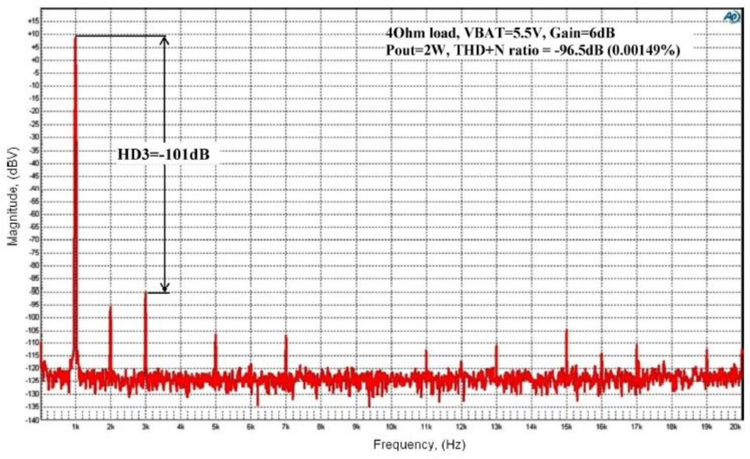
In particular [5]

$$THD + N(\text{dB}) = 20 \cdot \log \frac{THD + N(\%)}{100} \tag{2.24}$$

$$THD + N(\%) = 100 \cdot 10^{\left(\frac{THD+N(\text{dB})}{20}\right)} \tag{2.25}$$



(a) *Frequency vs THD* [4]



(b) *Output spectra voltage output to measure the THD from harmonics amplitude* [4]

Figure 2.17

Elements and parameters that influence THD are

- ***parasitic elements***: presence of stray inductance and parasitic capacitance introduce non-ideality that increase the non-linearities (and so the distortion) of the whole system;
- ***switching device characteristics***: parameters described in previous subsection (2.2.1) like Q_g, *dead time*, Q_{rr} introduce differences from ideal behaviour that increase distortion;

- ***configuration of loop***: in open-loop class D amplifier the effect of non-linearities are totally shown at the output; in *closed-loop* configuration, thanks to *feedback* connection, the non-linearities are much reduced: this is the main reason why the most part of amplifiers are designed in closed-loop configuration.

THD can be measured at a fixed audio input tone (generally 1 kHz), at varying the output power or applying an input sweep at the input from 20 Hz to 20 kHz and fixing the output power: with the first method, THD vs output power is measured; with the second one, THD vs frequency is obtained. In [14][4][15] these kind of measurements are taken.

2.2.3 Signal To Noise Ratio (SNR)

SNR is a parameter that defines the ratio between the signal power (the useful power) to the noise power (the useless one). The noise power is the sum of all noises introduced in the system like power supply noise, switching noise, thermal noise, radio frequency interference [3]. It is evaluated integrating the noise floor of the output signal in the audio frequency range, with no audio signal at the input [5]. In formula [5]

$$SNR = 10 \log \frac{P_o}{P_n} = 20 \log \frac{V_o}{V_n} = 20 \log V_o - 20 \log V_n \tag{2.26}$$

where P_o is the output power, P_n the noise power and V_o and V_n respectively the output voltage and the noise voltage.

2.2.4 PSRR (Power Supply Rejection Ratio) and PS-IMD

PSRR describes the capability of an amplifier to be immune to power supply noise. It is defined as [3]

$$PSRR = 20 \log \frac{V_n}{V_o} \tag{2.27}$$

Generally it is measured at 217 Hz because it's the frequency related to *GSM burst used for device communication* [5]. In practical terms, often is applied a small voltage amplitude square wave at frequency 217 Hz, as it is done in [4][16].
Another parameter related to the power supply noise is the ***power supply intermodulation (PS-IMD)***. Applying a noise tone to the power supply (e.g. $f_n = 217\,\text{Hz}$) the output shows *intermodulation products* related to the input signal tone. In particular, considering an input tone of $f_{in} = 1\,\text{kHz}$ at output can be measured

$$f_{in} - f_n = 783\,\text{Hz} \tag{2.28}$$

$$f_{in} + f_n = 1217\,\text{Hz} \tag{2.29}$$

as it happens in [4], in figure 2.18.

2.2.5 Intermodulation products (IMD)

When more than one audio tone at are applied to the input signal, other unwanted tones related to the ones applied at the input are generated. This happens due to non-linearities of the system, *"but sometimes"* they "may be low enough to be imperceptible" [6]. In formula it is defined as [6]

$$IMD = \frac{\sqrt{\sum_{n=1}^{\infty}\sum_{m=1}^{\infty}[V_o(mf_1+nf_2)+V_o(mf_1-nf_2)]^2}}{V_o(f_2)} \tag{2.30}$$

where f_1 and f_2 are the two input tone. An example of IMD measurement is in figure 2.19.

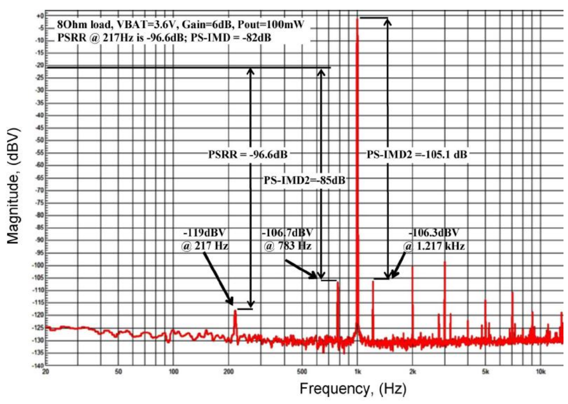


Figure 2.18: PS-IMD example [4]

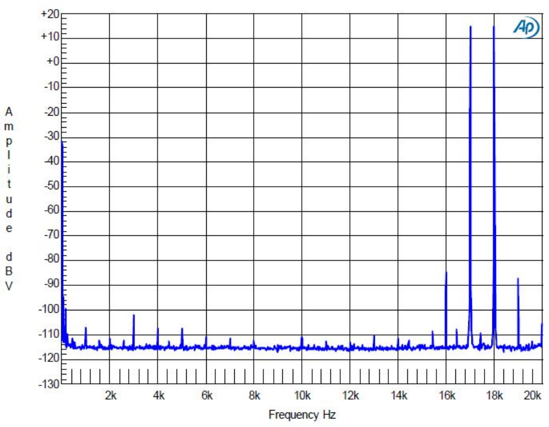
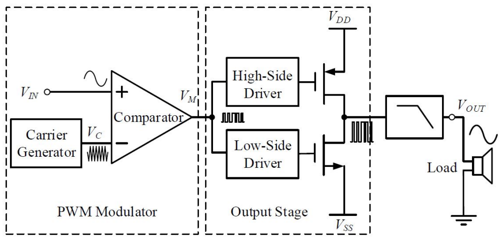


Figure 2.19: IMD measurement [14]

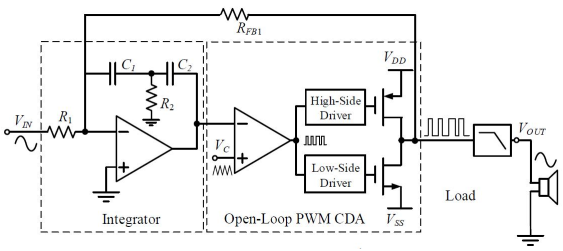
2.3 Closed-loop configurations and other modulation techniques

Due to non-idealities and switching noises, the open-loop configuration described so far, often gives high THD and low SNR that are often unacceptable. To increase the performances most of the *closed-loop configuration* are typically employed which presence reduces the effect of non-idealities and parasitic noises. The use of feedback needs the design of a *compensator* to ensure strong stability. Typically, in CDA the compensator consists of an *integrator* because it allows to reach a null *feedback* error and better performances in terms of distortion and SNR due to its high loop gain [6]. Therefore, higher loop gain, lower distortion is generated, so less power at unwanted harmonics. Increasing the order of integrator, the performances of the system improve. In practical analogue design, the order of compensator is typically not larger than 2 [4][5][9, fig. 3], leading to the use of different kinds of modulations and topologies. In particular, *"CDA can be classified into four general types based on different modulations schemes"* [6]:

- **Pulse Width Modulation (PWM)**, seen in previous section: is the most employed due to its simple hardware and ease of design; moreover, it allows to obtain low power dissipation. Two blocks schemes, one in open-loop and the other in closed-loop configurations are shown in figures 2.20.



(a) *Open-loop configuration CDA with PWM modulation* [6]



(b) *Closed-loop configuration CDA with PWM modulation* [6]

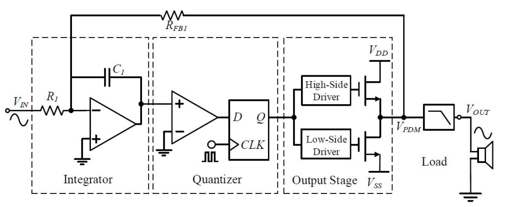
Figure 2.20: PWM configurations for CDA

- **Sigma-Delta Modulation (SDM)**: this kind of modulation is well known due to its low non-linearities and noise shaping that allows to obtain higher SNR respect to previous modulation [5]; two topologies are possible:

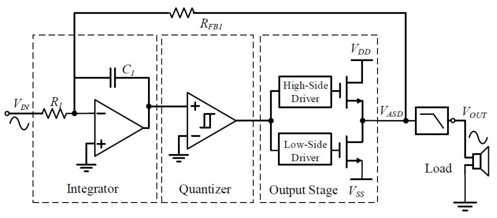
  – *synchronous*: after the quantizer, there is a flip-flop with fixed clock that determines the sampling frequency; this technique allows to reduce non-linearities and quantization noise;

  – *asynchronous*: an hysteresis comparator is used as quantizer: in this way the quantization error becomes, theoretically, zero; the problem is that the switching frequency is varying and there could be *"undesired coupling for multiple channel designs on a single IC"* [6].

  The main problems of this kind of modulation scheme is the high power dissipation and the lower sample frequency respect to the PWM modulation. Block schemes diagrams are shown in figures 2.21.



(a) *Closed-loop configuration CDA with Synchronous SDM modulation* [6]



(b) *Closed-loop configuration CDA with asynchronous SDM modulation* [6]

Figure 2.21: SDM configurations for CDA

- **Bang-bang control**: it is the most efficient power structure; it consists in an *hysteresis comparator* and a feedback network to achieve the stability; the main advantage is that is the most low-power consumption configuration, because only the hysteresis comparator is the quiescent power dissipating block (see fig. 2.22).

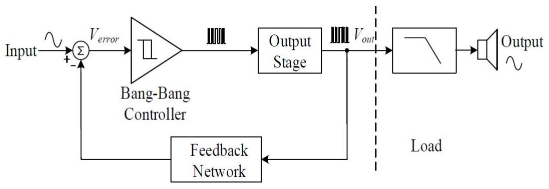


Figure 2.22: Bang-bang control CDA [6]

- **Self-oscillating**: this modulation is possible only with a feedback path because it is based on oscillator operation mode. The feedback network has to be designed to oscillate at the switching frequency. The drawback is that the self-oscillating CDA design is much more complex than the previous modulations.

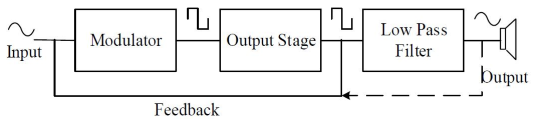


Figure 2.23: Self-oscillating block scheme CDA [6]

There are different ways to start the oscillation, but all of them do not require any external waveform generator. The general idea of all these architectures is to have an open loop transfer function that is a first order integrator (which have a 90° phase shift over all the band) and adding a delay time in closed-loop fashion to obtain a 180° phase and unitary loop gain at the oscillation frequency. The oscillation techniques are the following [17]:

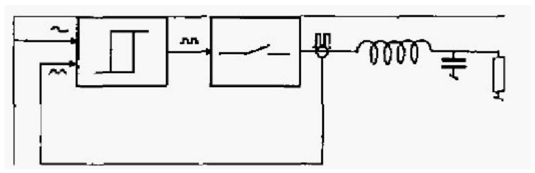


Figure 2.24: Current mode hysteresis modulator [17]

– ***Hysteresis modulator (current mode)***: because of the inductor current is the integral of its drop voltage, the feedback signal is the measured value of inductor current: the error signal it is obtained by the subtraction between the voltage reference and the inductor current. This signal feds an hysteresis comparator that generates the PWM signal to drive the switch (fig. 2.24); the oscillation frequency depends on the delay time due to the hysteresis comparator

$$t_d = \frac{V_{hyst}}{\alpha_{carrier}} \text{ [17]} \tag{2.31}$$

where V_{hyst} is the height of hysteresis window and $\alpha_{carrier}$ the slope of the carrier. The switching frequency depends on modulation index M *the ratio between output voltage and power supply voltage* [17], the inductance value L, the delay time t_d, the power supply V_s and the height hysteresis window I_{hyst} [17]

$$f_s(M) = \frac{V_s}{4} \frac{1 - M^2}{L \cdot I_{hyst} + \frac{1}{2} t_d V_s (1 + M^2)} \text{ [17]} \tag{2.32}$$

– ***Hysteresis modulator (voltage mode)***: the working principle is the same of the current mode, but the feedback signal is the input voltage of inductor (the PWM signal generated by the switch) that has to be integrated. So, a first order integrator is added and is fed by the error signal, the difference between the reference signal and the inductor input voltage (fig. 2.26). In this case the switching frequency is

$$f_s(M) = \frac{V_s}{4} \frac{1 - M^2}{\tau_{int} \cdot V_{hyst} + \frac{1}{2} \cdot t_d V_s (1 + M^2)} \text{ [17]} \tag{2.33}$$

where τ_{int} is the integrator time constant.

– ***1st order fixed delay self oscillating modulator***: the additional phase shift is reached adding in the feedback a delay line to obtain the Barkhausen condition at the required frequency. In this case the switching frequency becomes

$$f_s(M) = \frac{1}{2} \frac{1 - M^2}{t_d (1 + M^2)} \text{ [17]} \tag{2.34}$$

- ***Controlled oscillation modulator (COM)***: the hysteresis comparator is removed and the closed-loop transfer function is shaped adding high frequency poles to obtain the Barkhausen condition at the switching frequency (fig. 2.25).

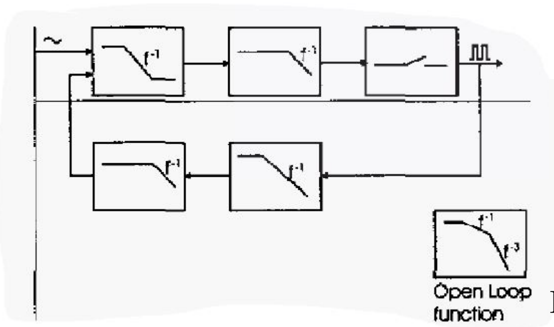


Figure 2.25: Block diagram of COM modulator [17]

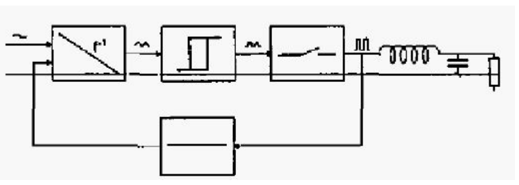


Figure 2.26: Voltage mode hysteresis modulator [17]

Another important feature to be considered to decide the modulation technique to implement in a class D amplifier is the *electromagnetic interference*: in fact at the changing of modulation techniques and the noise shape and the high frequency harmonics that can be generated in RF band.

2.4 Electromagnetic Interference

Electromagnetic noise in electronic systems is not a marginal problem: in fact, the Federal Communications Commission (FCC) and European Union (EU) rules about the *electromagnetic compatibility* (respectively in USA and EU) must be respected to sell the system [18]. This is one of main problems and drawback in switch-mode amplifiers.
ElectroMagnetic Compatibility (*EMC*) problems are concerned it is common to define two types of disturbances:

$$DM = OutA - outB \qquad \textbf{DIFFERENTIAL MODE NOISE} \tag{2.35}$$

$$CM = \frac{OutA + OutB}{2} \qquad \textbf{COMMON MODE NOISE} \tag{2.36}$$

as shown in [7], where $OutA$ and $OutB$ are the two output of a Bridge-Tied Load class D amplifier. While the differential mode noises can be attenuated by a symmetric structure, the common mode noise can be reduced only by adding extra filtering [7]. Depending on modulation type, different classes of amplifiers are defined:

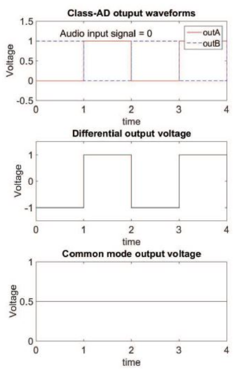
- ***Class AD***: *"A speaker that is connected to these two output lines will always see an excitation voltage of $\pm V_{DD}$ causing maximum current flow"* [7]. This voltage stress so much the speaker because the output current is very high. To improve the efficiency and reduce the stresses, an output filter is added to both branches of amplifiers, reducing the power dissipation. With this modulation scheme (that is basically the *natural PWM)* both in presence and absence of a signal, the common mode noise is theoretically *half of the power stage power supply*, so basically no CM harmonics. In fig. 2.28a it can be seen that CM components are not null: this is due to the asymmetry of the two stage and the effect of switching delay of the two branches;

- ***Class BD***: this class exploits the fact that the speaker already works as a low pass filter. Changing the modulation scheme (fig. 2.27b) it is possible to avoid the use of the output filter, increasing the efficiency and recovering area on the board: for those motivations this amplifier class is even called *filterless Class D* and it is widely used in mobile applications. This modulation scheme allows to nullify differential output voltage when no signal is applied, increasing the efficiency of the system; on the other hand, the common mode noise is increased, too(fig. 2.27b). Observing the spectrum of CM *"at duty cycle 50%, the odd harmonics have a maximum amplitude, while the even harmonics disappear"* [7].
- ***Ternary scheme***: it is a development of the class BD scheme and is composed by a digital logic; it allows to reduce the CM noise respect to BD scheme (fig. 2.27c), doubling the fundamental frequency of the CM signal when the audio signal is applied at the input; the other advantage is the possibility to remove the output filter because with no audio signal only short pulses are shown in DM waveform (fig. 2.27c). Observing the CM waveform, decomposing the signal with Fourier series, it can be seen that odd and even harmonics of twice the PWM frequency are generated.

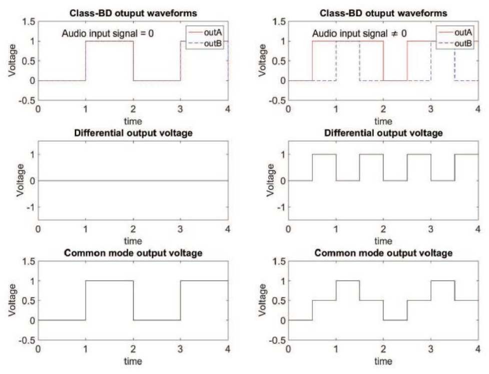
2.4.1 EMI-Reduction techniques

EMI has a strong impact in mobile and integrated applications like phones because it can influence the functionality of other circuits like converters, source currents, voltage references. EMI problems could not be neglected in *filterless* amplifiers, so techniques for reducing EMI are developed, both in analog and digital ways. The digital solutions are used in integrated class D amplifiers because the analog ones are difficult to be implemented in integrated products [8]. Three EMI reduction techniques implemented by digital mean are shown in [8]:

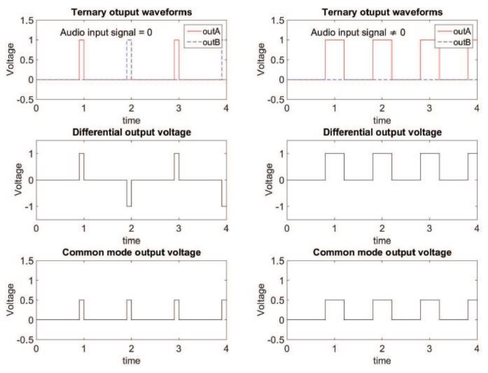
- ***Common-mode (CM) modulation***: the idea is to modulate the common mode component of a tri-level PWM (class BD modulation scheme): inverting and swapping pseudo-randomly the two output of a bridge tied-load amplifier, the common mode can be nullified without affecting the differential mode (fig. 2.29).



(a) *Output signal of class AD modulation scheme* [7]

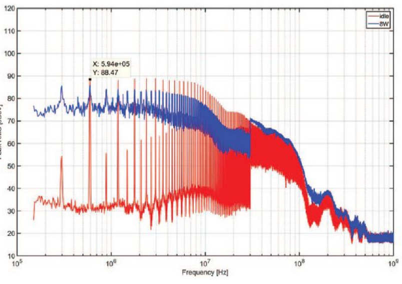


(b) *Output signal of class BD modulation scheme* [7]

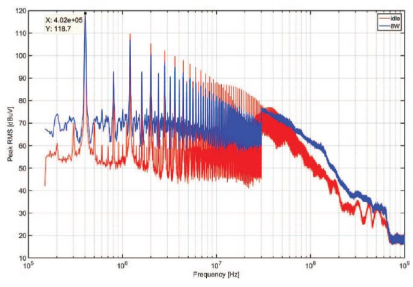


(c) *Output signal of Ternary modulation scheme* [7]

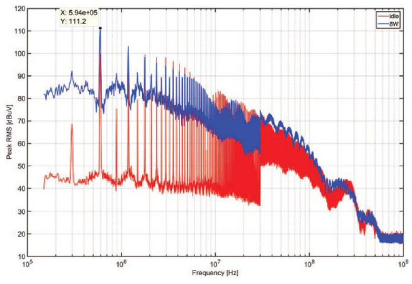
Figure 2.27: Output signal of different modulation schemes



(a) *Class AD modulation scheme CM spectrum* [7]



(b) *Class BD modulation scheme CM spectrum* [7]



(c) *Ternary modulation scheme CM spectrum* [7]

Figure 2.28: CM spectra of different modulation schemes

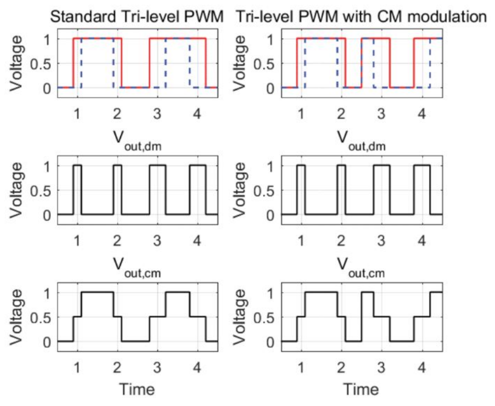


Figure 2.29: Waveform comparison between tri-level PWM and tri-level PWM with CM modulation [8]

- ***Spread-spectrum modulation***: it is a very simple technique that is used in many implementation like USB3.0 [8]; the idea is to distribute over a wider frequency range the energy of the disturbances; the drawbacks is that the clock jitter is increased; in digital implementation is difficult to obtain good performances since *"due to the digital clock division the frequency deviation for the PWM signal is relatively small"* [8].

- ***Digital spread-spectrum modulation***: digital implementation of the previous modulation.

- ***Digital spread-spectrum modulation with additional common mode modulation***: this technique implements together, in a digital fashion, the common mode and the spread spectrum modulation, avoiding the impact on audio quality and the reducing EMI noise, as shown in [8].

The procedure to measure electromagnetic disturbances is defined in the standard IEC61967-4 and it is used in the works [7][8].

2.5 Digital modulations (DPWM)

The PWM modulation is the simplest modulation technique. For this reason in several works [19][20][16][21][22][14] it is implemented and studied to improve the performances of the class D amplifier. In fact, the possibility to digitalize this techniques brings lot of advantages:

1. allows to reach higher switching frequency improving the performances of the whole system;

2. in closed-loop fashion, the integrator and filter can be implemented with much more complex transfer functions without adding any extra external hardware;

3. the integration of the whole system is much easier;

4. the area occupied is much lower than in analog implementations;

5. the system, and in particular the modulation section, is more protected by external noise, being digital.

On the other hand, the digital implementations show several drawbacks and bottlenecks, in particular resolution of a digital PWM is a trade-off with the switching frequency, clock frequency and area and could generate **LCO** (**Limit Cycle Oscillation**)[19][20][21][23][24].
So due to all these reasons, different implementations of **DPWM** (**Digital Pulse-Width Modulation**) are studied.

2.5.1 DPWM counter and comparator-based implementation

The most simple implementation of a DPWM converter in made by

- ***Free-running counter***: it's a n bit digital counter that works at a clock frequency f_{clk}. From those two parameters the switching frequency f_{sw} can be derived as

  $$f_{sw} = f_{clk}/2^n \tag{2.37}$$

  This circuit substitutes a triangular trailing edge oscillator in analog implementation of PWM ;

- ***input register***: is the register where the input digital code is stored and sampled at f_{sw};

- ***comparator***: it compares the register code with the number reached by the counter; than it output a high value if the register code is higher than the counter code, or a low value otherwise.

An example of DPWM converter is shown in figure 2.30. The output signal is composed by

- n "ones";

- $2^N - n$ "zeros

where N is counter and input register number of bits and n the converted number. So the resulting waveform is a square wave with a frequency $f_{sw} = f_{clk}/2^N$ and a duty cycle $D = n/2^N$.

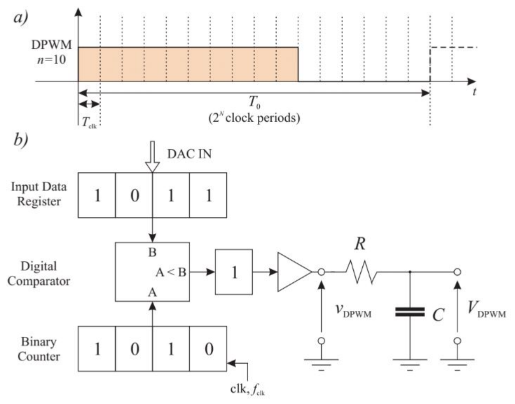


Figure 2.30: Example of architecture and output signal of a DPWM DAC converter [20]

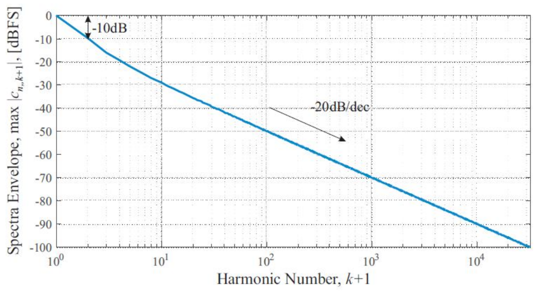


Figure 2.31: DPWM spectrum

So the output waveform can be expressed as [20]

$$v_{DPWM,n}(t) = V_{DD} \sum_{k=-\infty}^{k=+\infty} \Pi\left(\frac{t}{nT_{clk}} - \frac{1}{2} - \frac{2^N}{n}k\right) \tag{2.38}$$

where [20]

$$\Pi(x) = \begin{cases} 1 & |x| < \frac{1}{2} \\ \frac{1}{2} & |x| = \frac{1}{2} \\ 0 & |x| > \frac{1}{2} \end{cases} \tag{2.39}$$

and its spectrum results [20]

$$V_{dpwm}(f) = V_{DD} \sum_{k=-\infty}^{k=+\infty} c_{k,n}\delta(f - kf_0) \tag{2.40}$$

$$c_{k,n} = Dsinc(kD)e^{j\pi kD} = \frac{n}{2^N} sinc\left(\frac{kn}{2^N}\right)e^{-\frac{j\pi kn}{2^N}} \tag{2.41}$$

with $sinc(x) = \sin(\pi x)/x$, $f_0 = 2^N \frac{1}{T_{clk}}$ and T_{clk} is the clock period of the counter. This architecture is simple to be implemented and requires a minimum hardware area. The main problem, as it is shown in [20], is the filter requirement to recover the input information. As a matter of fact, the highest harmonic component of the output spectrum $V_{dpwm}(f)$ is $f_0(k = 1)$ with a factor $\alpha_0 = 1/\pi$ below V_{DD}. The attenuation filter at this frequency should be that the f_0 amplitude has to be below the quantization error $\epsilon = \frac{V_{DD}}{2^{N+1}}$. So the output voltage of output filter OF should be

$$OF < \epsilon \Rightarrow \alpha_F \alpha_0 V_{DD} < \frac{V_{DD}}{2^{N+1}} \Rightarrow \alpha_F \frac{1}{\pi} V_{DD} < \frac{V_{DD}}{2^{N+1}} \Rightarrow \alpha_F < \frac{\pi}{2^{N+1}} \tag{2.42}$$

where α_F is the attenuation filter, that is a function of frequency. Considering the order of filter P, can be found the *corner frequency* f_c of the filter by

$$\begin{aligned} \alpha(f) = (f_c/f)^P \Rightarrow \left(\frac{f_c}{f}\right)^P < \frac{\pi}{2^{N+1}} \\ \Rightarrow f_c < \sqrt[P]{\pi} 2^{-\frac{N+1}{P}} f_0 = \sqrt[P]{\pi} 2^{-\frac{N+1}{P}} f_{clk} 2^{-N} = \\ = \sqrt[P]{\pi} 2^{-\frac{N+1}{P}} f_{clk} 2^{\frac{-N-1-PN}{P}} = \sqrt[P]{\pi} f_{clk} 2^{-\frac{N(P+1)+1}{P}} \end{aligned} \tag{2.43}$$

Expression 2.43 shows that ***for a given clock frequency*** f_{clk} ***the resolution*** N ***is a trade-off with corner frequency*** f_c. Moreover the implementation of this filter is difficult because it requires large resistors and capacitors (fig. 2.30). Let's consider the following examples

- $f_{clk} = 100\,\text{MHz}$, $P = 1$, $\mathbf{N = 8 \Rightarrow f_c} = 2.4\,\text{kHz}$
- $f_{clk} = 100\,\text{MHz}$, $P = 1$, $\mathbf{N = 12 \Rightarrow f_c} = 37.45\,\text{Hz}$

It can be observed that increasing the resolution N, f_c has to decrease, reducing the reliability of the filter in integrated applications and requiring large area even in discrete application because filter component should be large (e.g. 2nd order filter with inductor and capacitor).
This important drawback has brought different implementations of DPWM, increasing the resolution and overcoming the f_{clk} and f_{sw} trade-off.

2.5.2 Alternative implementations of DPWM

The increase of resolution in the DPWM implementation shown before, needs an increase in clock frequency, especially in high switching frequency applications. For this reason, different implementations of DPWMs are developed

- ***delay lines-based DPWM***: it consists in a set of delay elements to *"achieve sub-clock cycle resolution"*[21], but requires large area on silicon and it is very dependent on temperature, parameter variation and power supply[21][25];
- ***thermometric dithering***: it consists in a *variation of duty cycle of one LSB over pre-defined patterns*[21] controlling the average duty cycle with a sub-LSB resolution. The idea is shown in figure 2.32. *"[...]the digitally quantized*

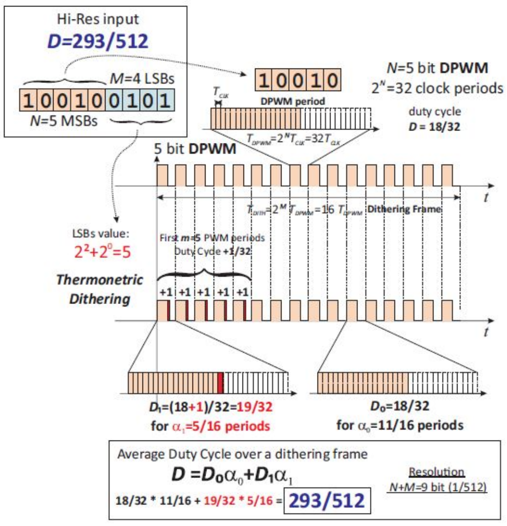


Figure 2.32: *High resolution DPWM by thermometric duty cycle dithering over switching periods* [21]

duty cycle is $(n+1)/2^N$ in the first M *switching periods of a 2^M dithering pattern and $n/2^N$ in the remaining periods, so that an average duty cycle $(n \cdot 2^M + m)/2^{N+M}$ quantized over $N+M$ bits can be achieved, thus increasing the effective DPWM resolution of M bits"*[21]. This solution is inexpensive and needs a simple architecture, but introduces switching frequency sub-harmonics that cannot be properly filtered out.

2.5.3 Digital Sigma-Delta modulation

Another solution adopted to solve the resolution limitations is to *shape* the quantization noise: the idea is to move the low frequency quantization noise to higher frequencies, reducing its effect in the band of interest. This technique is called *noise shaping* and ***Sigma Delta modulation ($\Sigma\Delta$)*** is one of the most used implementation of this technique. Usually, this kind of modulation is applied in analogue-to-digital converter, *"in digital-to-analog converter and in digital signal processing (DSP) to reduce the number of bit of binary representation, without reducing the information of the signal"*[25]. The parameters that characterize $\Sigma\Delta$ converter are:

- *oversampling ratio (OSR)*: is the ratio between the sample frequency of the modulator f_{clk} and the Nyquist frequency $2f_s$ related to the band of the signal f_s

$$OSR = \frac{f_{clk}}{2f_s} \tag{2.44}$$

  Larger is OSR, larger is the SNR;

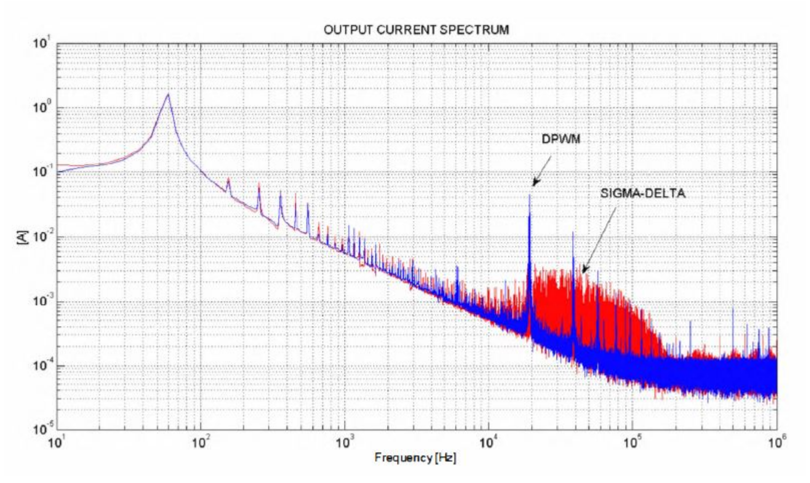


Figure 2.33: DPWM vs ΣΔ modulation spectra

- ***modulator order***: ΣΔ modulator is based on the presence of an integrator that has the function to reduce quantization noise in signal bandwidth; increasing the order of the integrator is more effective on quantization noise reduction;

- ***resolution of quantizer***: increasing the number of bit of ΣΔ converter, SNR increases.

Quantitatively, SNR of ΣΔ converter can be expressed as a function of $OSR = 2^{r}$[1], amplitude of input sinusoidal signal A, power supply of ADC V_{DD} and number of bits b [5]

$$\text{SNR [dB]} = 20 \log \frac{A}{V_{DD}} + 6.02b + 3.01r + 1.76 \tag{2.45}$$

This solution is applied in [25], where a ΣΔ modulator is used to control a MOSFET switch of a step-down converter in a DC-AC converter. The solution is compared with another one that use a DPWM converter. The spectral results, in figure 2.33, show that solution with ΣΔ modulation has a spread noise at high frequency, differently from DPWM modulator, where the fundamental, harmonics of clock frequency and intermodulation products are visible. Moreover, at low frequency, spurious tones are generated in particular with DPWM modulation. The problems of ΣΔ modulation are that the implementation is more complex and required area and power dissipation are larger. Moreover, the clock sample frequency is generally lower than DPWM one.

[1]generally the oversampling ratio is set as a power of 2

2.5.4 Limit Cycle Oscillation (LCO)

In a closed-loop systems with quantizers like ADCs and DPWMs in the loop, undesirable oscillations may occur on output voltage due to the resolution limitation. This behaviour is called **Limit Cycle Oscillation (LCO)**. An example of this kind of system is shown in figure 2.34. Qualitatively, with figure 2.35 as reference, the oscillation on the output is due to the fact that DPWM resolution is less than the ADC one and *"there is no DPWM level that can be mapped into the ADC bin corresponding to the reference voltage V_{ref}"*[24]. In this situation, what happen is that the digital controller tries to drive V_{out} to the zero-error bin, but the lower resolution of DPWM does not permit this correction, triggering an oscillation on the output: this means that *the first step toward eliminating limit cycles is to ensure that under all circumstances there is a DPWM level that maps into the zero-error bin*[24].

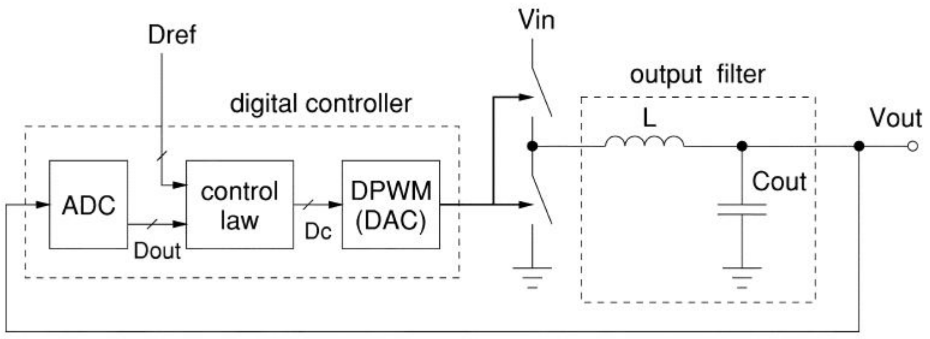
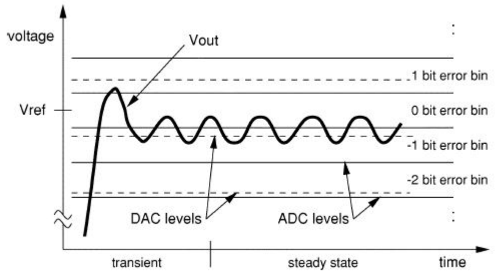
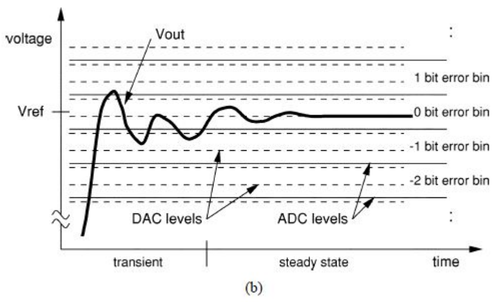


Figure 2.34: Block diagram of a digitally controlled buck converter [24]



(a)



(b)

Figure 2.35: a) LCO phenomena in case of DPWM resolution lower than ADC; b) LCO phenomena in case of DPWM resolution higher than ADC

As shown in [24] the conditions to remove the LCO are

1. $\boldsymbol{DPWM_{res} > ADC_{res}}$;

2. ***Integrator gain 0<K_i<1***;

3. ***Nyquist criterion*** must be ensured.

If the last two conditions can be easily respected by proper compensator design, the first depends on quantization of ADC and DPWM, influencing the clock frequency of DPWM. Given a switching frequency f_{sw}, considering a DPWM solution based on comparator and free-running counter of N bits, the necessary clock frequency f_{clk} is given by

$$f_{clk} = 2^N f_{sw} \tag{2.46}$$

The equation 2.46 highlights that increasing the resolution N, clock frequency rises up. For example, given $f_{sw} = 1\,\text{MHz}$ and $N = 10$, clock frequency overcomes 1 GHz that could be not so easy to reach in microcontrollers[26]. Moreover, increasing the clock frequency means higher power dissipation.

2.6 Dyadic Digital Pulse Modulation (DDPM)

As shown in subsection 2.5.1 the output filter requirements for digital-to-analog conversion with DPWM are very stringent and create problems in integrated circuits implementations because it needs large capacitors and resistors. Moreover, as shown in eq. 2.46, increasing the switching frequency to attenuate the requirement of output filter, is a trade-off with the resolution and the clock frequency. Even if the resolution can be increased with thermometric dithering technique, *"it may introduce noise at switching frequency sub-harmonics, which cannot be properly rejected by the converter output filter"*[23]. To overcome these problems, it can be observed that the mean value of DPWM stream is $n/2^N V_{DD}$, where n is the number of "ones" of the output digital stream, N the DPWM number of bits and V_{DD} the power supply of digital architecture (fig.2.30): the idea developed in [20] is to change the output waveform of a non-DPWM modulator without changing the mean value.

2.6.1 Dyadic sequence

"The strict requirement for the output filter of DDPWM DAC are due to the fact that most of the harmonic content of DPWM signal is concentrated at the fundamental frequency $f_0 = f_{clk}/2^N$"[20]. The worst case for the output of a DPWM modulator is when the input code is such that the output digital stream is a square wave with 50% duty cycle: in this case the first harmonic (the one with highest power, $f_0 = f_{clk}/2^N$) has an amplitude of $\frac{1}{\pi}V_{DD}$[20]. The input code that generates this waveform is 2^{N-1} and in particular the output digital stream is made by 2^{N-1} "ones"

and 2^{N-1} "zeros" ($S_{N-1} = 1111...10000$). If this input code (2^{N-1}) is associated to the output sequence $S_{N-1} = ...10101010...$, the fundamental of this signal is increase at $f_0 = f_{clk}/2$ (instead of $f_{clk}/2^N$ of DPWM sequence) reducing the requirement of the output filter. Moreover, the resulting DC components does not change because the number of "ones" and "zeros" is the same.
In a similar way, the DPWM pattern associated to the input code 2^{N-2} is composed by $\frac{1}{4}N$ of "ones" and $\frac{3}{4}N$ of "zeros". This pattern can be rearranged in the output sequence $S_{N-2} = ...10001000...$, maintaining the same DC component of DPWM sequence and increasing the fundamental frequency to $f_0 = f_{clk}/4$.
This pattern can be recursively applied down to LSB. In general, the pattern associated to the code 2^{N-x} ($N \leq x < 0 \in \mathbb{N}$) is a sequence with the following characteristics

- 2^{N-x} periods;
- each period is composed by a "one" (T_{clk} duration) and $2^x - 1$ zeros (each zero of duration T_{clk}).

Example: DPWM vs DDPM sequences

- $N = 4 \Rightarrow 2^N = 2^4 = 16$ sequences

Sequence	DPWM	DDPM
$S_0 = 2^0(0001)$	1000000000000000	0000000100000000
$S_1 = 2^1(0010)$	1100000000000000	0001000000010000
$S_1 = 2^2(0100)$	1111000000000000	0100010001000100
$S_2 = 2^3(1000)$	1111111100000000	1010101010101010

Sequence	$f_{0_{DPWM}}$	$f_{0_{DDPM}}$
$S_0 = 2^0(0001)$	$f_{clk}/16$	$f_{clk}/16$
$S_1 = 2^1(0010)$	$f_{clk}/16$	$f_{clk}/8$
$S_2 = 2^2(0100)$	$f_{clk}/16$	$f_{clk}/4$
$S_3 = 2^3(1000)$	$f_{clk}/16$	$f_{clk}/2$
$S_i = 2^i$	$f_{clk}/2^N$	$f_{clk}/2^{N-i}$

This kind of sequence is called *dyadic* and an example with $N = 4$ is shown is figure 2.36.

2.6.2 Dyadic sequence of an integer number

Observing the figure 2.36 it can be seen that the sequences that represents the 2^{N-x} numbers are orthogonal (there's not superposition among all the ones). Starting from this, any integer number $n \in [0, 2^N)$ can be represented by a proper sum of the dyadic sequences. In terms of binary representation can be written as [20]

$$n = \sum_{i=0}^{N-1} b_{i,n} 2^i \tag{2.47}$$

where $b_{i,n}$ is the *i-th* bit of the binary code. The related dyadic sequence of n is [20]

$$\Sigma_n = \sum_{i=0}^{N-1} b_{i,n} S_i \tag{2.48}$$

where S_i is the dyadic sequence associated to 2^i.

Example: DDPM sequence

- $n = 10 \Rightarrow 1010$

Applying the 2.48

$$\Sigma_{10} = \sum_{i=0}^{4-1} b_{i,10} 2^i = \sum_{i=0}^{3} b_{i,10} 2^i = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 = 2 + 8 = 10$$

Sequence	DDPM
$S_1 = 2^1(0010)$	0001000000010000
$S_2 = 2^3(1000)$	1010101010101010
$\Sigma_{10} = S_1 + S_2$	1011101010111010

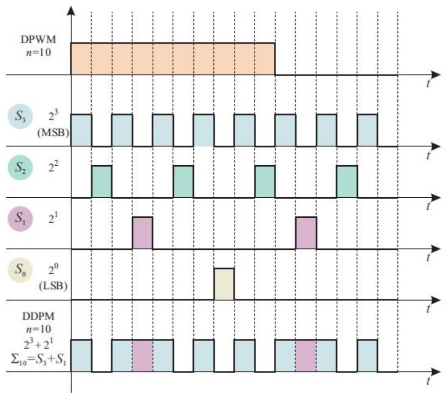


Figure 2.36: Dyadic sequences with N=4 bits [20]

2.6.3 Spectral analysis of DDPM

The spectral analysis reported in [20] shows interesting advantages of the DDPM over standard DPWM. *“The DDPM sequence associated to a constant input code n*

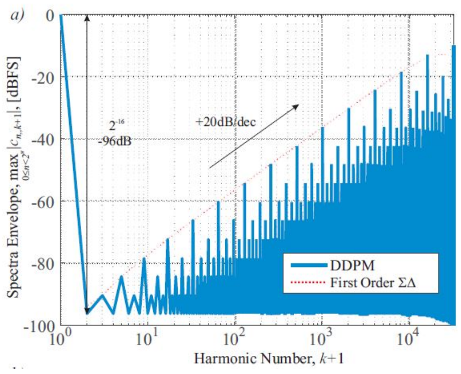


Figure 2.37: Dyadic Digital Pulse Modulation spectrum, N=16 bit resolution [20]

corresponds to a digital waveform [20]

$$v_{DDPM,n}(t) = V_{DD} \sum_{k=-\infty}^{+\infty} x_n(t - 2^N kT_{clk}) \text{ [20]} \tag{2.49}$$

where [20]

$$x_n(t) = \sum_{i=0}^{N-1} \sum_{h=0}^{2^i-1} b_{i,n} \prod \Big(\frac{t}{T_{clk}} - 2^{N-i}h - 2^{N-i-1} - \frac{1}{2}\Big) \tag{2.50}$$

with a spectrum[20]

$$V_{ddpm,n(f)} = V_{DD} \sum_{k=-\infty}^{+\infty} c_{k,n} \operatorname{sinc}\Big(\frac{k}{2^N}\Big)\delta(f - kf_0) \tag{2.51}$$

where $f_0 = 1/T_0 = f_{clk}/2^N$ *and* [20]

$$c_{k,n} = \sum_{i=0}^{N-1} b_{i,n} 2^{i-N} \sum_{m=0}^{2^{N-i}-1} \delta[k - 2^i m] e^{-j\pi m(1+2^{i-N})} \tag{2.52}$$

with $\delta[\cdot]$ *is the Kroenecker function defined as* [20]

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \tag{2.53}$$

A graphical representation of DDPM spectrum is shown in figure 2.37. As expected the DC component is not changed ($n/2^N V_{DD}$) with respect to DPWM. Moreover, harmonics amplitude increases of 20 dB/dec (linearly scaled by 2^i) and frequencies are spaced by $2^i f_0$.

2.6.4 Output filter requirement

The spectral distribution of the DDPM pattern modifies the requirements for the output filter. In particular, looking at fig. 2.40, it can be seen that to attenuate the

$f_0 = f_{clk}/2^N$ (the lowest harmonic) under quantization error ϵ, it is necessary only an attenuation of 6 dB at f_0, with a slope of $-20\,\text{dB/dec}$ at least. In fact

$$\begin{gathered}\alpha|_{f=f_0} = \alpha_0 = 2^{-N} V_{DD} \\ \Rightarrow \alpha_F \alpha_0 < \epsilon \Rightarrow \alpha_F \alpha_0 < \frac{V_{DD}}{2^{N+1}} \\ \alpha_F \frac{1}{2^N} < \frac{1}{2^{N+1}} \Rightarrow \alpha_F < \frac{1}{2} \Rightarrow \alpha_F|_{dB} = -6\,\text{dB}\end{gathered}$$

where α_0 is the amplitude of the lowest harmonics, α_F the attenuation of output filter and ϵ the quantization error. Considering an RC filter

$$|H(jf)|_{f=f_0} = \left.\sqrt{\frac{1}{1+\left(\frac{f}{f_c}\right)^2}}\right|_{f=f_0} < \frac{1}{2} \Rightarrow f_c < \frac{f_0}{\sqrt{3}} \tag{2.54}$$

Example: DPWM vs DDPM

- $N = 16$, $P = 1$, $f_{clk} = 100\,\text{MHz}$

Applying 2.43

$$f_{c_{DPWM}} < 37\,\text{mHz}$$

Applying 2.54

$$f_{c_{DDPWM}} < \frac{f_{clk}}{2^N}/\sqrt{3} = \frac{100\,\text{MHz}}{2^{16}}/\sqrt{3} = 2.65\,\text{kHz}$$

The requirements for DDPM output filter are much less stringent.

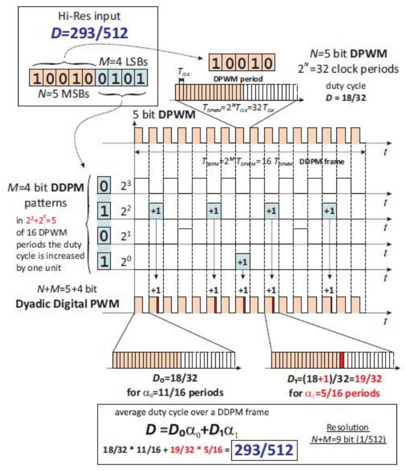
2.7 Dyadic Digital Pulse-Width Modulation (DDPWM)

In [23] the DDPM is applied to DPWM to increase the effective resolution of DPWM and reducing the spurious effect of sub-harmonics in baseband due to the thermometric dithering technique, without affecting DC components. This technique is called *Dyadic Digital Pulse-Width Modulation.* It consists in modulating the DPWM signal between two adjacent quantization levels [23]

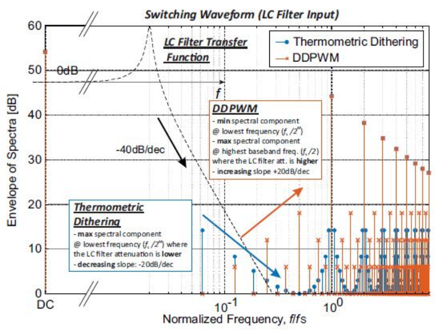
$$D_0 = \frac{n}{2^N} \text{ and } D_1 = D_0 + 1LSB_{DPWM} = \frac{n+1}{2^N} \tag{2.55}$$

where n is the value represented by the first N MSBs of input code. The duty cycle D_0 is applied $2^M - m$ times while D_1 is applied the remaining m times, where M are the LSBs of the digital input. Using this technique, the average duty cycle is given by [23]

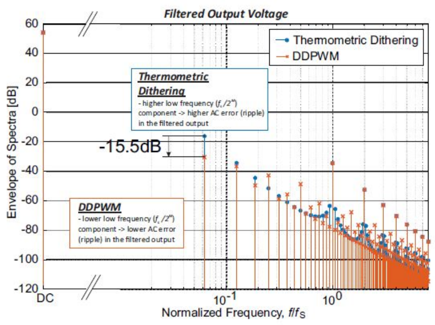
$$D = \frac{n}{2^N}\frac{2^M - m}{2^M} + \frac{n+1}{2^N}\frac{m}{2^M} = \frac{n \cdot 2^M + m}{2^{N+M}} \tag{2.56}$$



(a) *$N + M$ bit Dyadic Digital Pulse Width Modulation (DDPWM)* [23]



(b) *Comparison between DDPWM spectra and thermometric dithering DPWM (switching voltage)* [23]



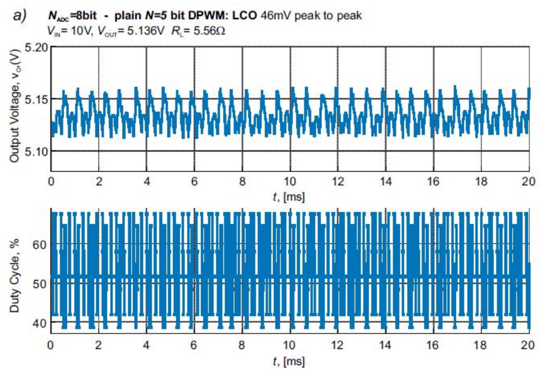
(c) *Comparison between DDPWM spectra and thermometric dithering DPWM (output voltage)* [23]

Figure 2.38: DDPWM pattern and comparisons between DDPWM and thermometric dithering DPWM spectra

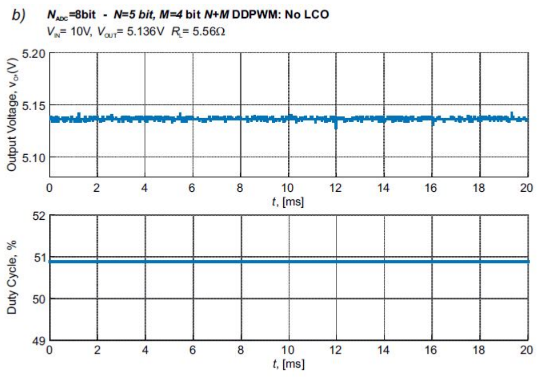
Comparing the obtained duty cycle with the one shown in subsection 2.5.2 with thermometric dithering, the results are the same. The difference is that in thermometric dithering D_1 is applied m *consequtive* times, while in DDPWM D_1 is generated by means of a DDPM architecture that receives the M LSBs (fig. 2.38a). This characteristics has a direct effect on output signal spectrum. The advantage of the DDPWM can be observed in spectra comparison (fig. 2.38)

- the lowest sub-harmonic component results the minimum one for DDPWM modulation, contrary to thermometric dithering where is the maximum one and higher sub-harmonics decrease with 20 dB/dec;
- the highest sub-harmonic of DDPWM is at $f_{sw}/2$ where the attenuation of the filter is higher; as a consequence, the output voltage ripple is reduced.

Another advantage of DDPWM is *LCO suppression* due to the fact that an higher



(a) *DPWM output with LCO phenomena* [23]



(b) *DDPWM output with suppresion of LCO phenomena* [23]

Figure 2.39: Comparison between DPWM and DDPWM output voltage where the LCO is put in evidence

resolution can be easily achieved with a suitable clock frequency. This effect is shown in [23], where *"the output voltage of the buck converter with* $N_{ADC} = 8$ *bit ADC operated* $f_s = 100\,\text{kHz}$ *switching frequency by a plain DPWM modulator clocked at* $f_{clk} = 3.2\,\text{MHz}$*, resulting in a* $N = 5$ *bit resolution"* is measured and compared with a DDPWM modulation with $M = 4$ and $N = 5$, using the same switching frequency and the same clock frequency for DPWM. The results show that the LCO-free operation is not reached by DPWM modulator because its resolution ($N = 5$) is lower than the ADC one ($N_{ADC} = 8$). On the other hand, the DDPWM suppresses the LCO phenomena because its effective resolution is $N + M = 9$, one bit higher than the ADC one (fig. 2.39).
In conclusion, DDPWM modulation allows to reach better performances than other kind of modulations like DPWM. In particular, in closed-loop system LCO-free operation can be reached and a better accuracy on DC output voltage can be obtained [23]. Moreover, the requirements on the output filter and the bottleneck between resolutions and clock frequency are reduced.

2.8 Devices technology

One of the main goals of power systems is to increase the efficiency. This aim can be achieved by the use of proper architecture (e.g. switching mode driving), but even through an appropriate technology that limits some non-idealities of devices. At the state of the art of class D amplifiers, the most employed power devices for power stage are MOSFETs. In this active devices, the limitations are related to parasitic capacitances and body diode, which reduce the speed of the device and increase

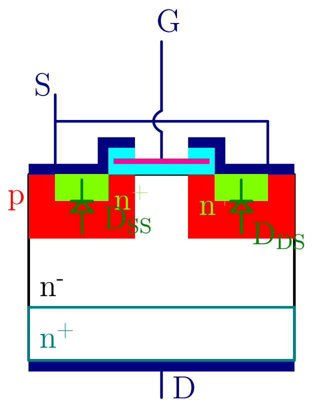


Figure 2.40: MOSFET structure

the power consumptions. Those two effects have direct impact both on distortion and power efficiency. In fact, as mentioned on page 27, parasitic capacitances (e.g. C_{gs}) need to be charged with a charge (e.g. Q_g) depending on value capacitance and voltage drop on it. Moreover, the presence of body diode influences the efficiency and the speed, which is related to the *reverse recovery charge* that has to be considered when the device change state during dead time (from ON to OFF or from OFF to ON) (see page 27). On the other hand, the body diode is not a perfect short circuit when it conducts, but there's a voltage drop on it and, as a consequence, a power consumption during conduction time[27][28].

2.8.1 Body diode effect

Considering an n-channel MOSFET, the cross section of physical structure is the one shown in figure 2.40. As it can be observed, the p-substrate and the n-wells (source and drain) form two pn-junctions and, as consequence, two parasitic diodes. If substrate and source are short circuited, the only diode that could conduct is the one formed by the substrate (p-well) and the drain (n-well), $\mathrm{D_{DS}}$. To point out when $\mathrm{D_{SD}}$ conducts, the circuit in figure 2.41 is considered. Basically, it is an output stage with output filter of a class D amplifier. During dead time, both transistor are turned-off. Because of the current that flows through the inductor must continue to flow (due to the continuity condition of the current), the body diode of the low-side transistor is forced to conduct and a negative voltage drop falls on it. A qualitative graphical representation of V_{DS} behaviour is shown in figure 2.41. Because of the voltage drop on the diode and the inductor current flows through it, during dead time there's a power dissipation which depends on dead time duration. In a similar way, this happens to the body diode of high-side transistor when the current flowing through the inductor is negative. Consequently, the presence of body diodes are necessary to the correct behaviour of class D output stage, but they increase power

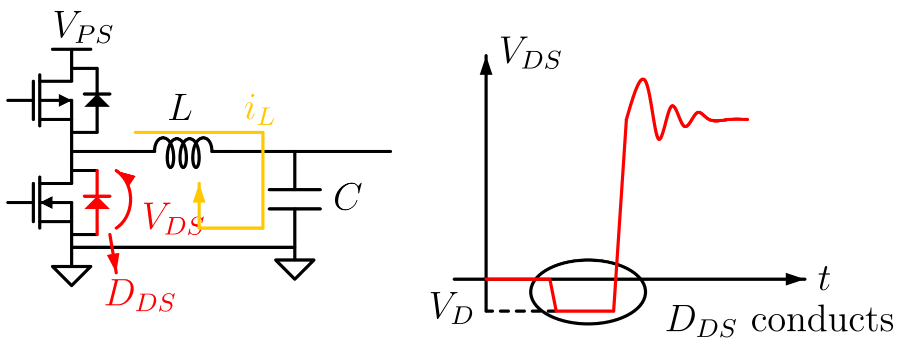


Figure 2.41: Dead time V_{DS} behaviour and D_{DS} conduction

Parameter	**Silicon**	**GaN**	**SiC**
Band Gap E_g (eV)	1.12	3.39	3.26
Critical Field E_{crit} (MV/cm)	0.23	3.3	2.2
Electron Mobility (cm^2/Vs)	1400	1500	950

Figure 2.42: Comparison among Si, GaN and SiC technologies characteristics

dissipation.

2.8.2 GaN devices

In recent years, in power applications have been introduced active devices based on different materials and technologies like Silicon Carbide (SiC) and Gallium Nitride (GaN) FETs. In particular, the last ones are becoming a valid alternative to MOS technology because their physical characteristics allow to reach

- power losses reduction;
- system size reduction;
- higher efficiency;
- system cost reduction;
- higher speed.

A comparison among Silicon, GaN and SiC technologies is shown in table 2.42. GaN devices are characterized by a higher *bandgap* value with respect to the two competitors, obtaining the capability to work at higher temperature before performance degradation. Moreover, GaN devices have higher *critical field* and so, as a consequence, a larger *breakdown voltage*: this characteristic allows the reduction of size device. Finally, larger mobility means faster device.

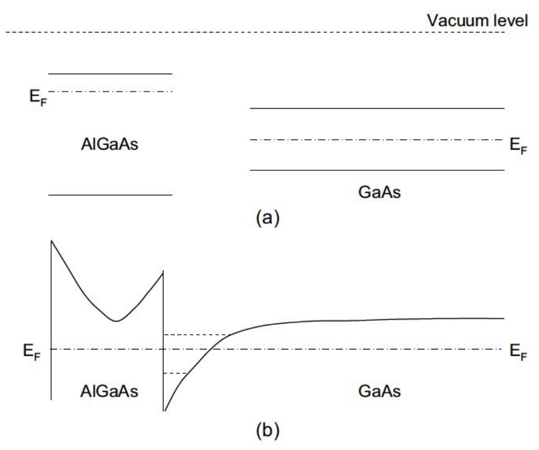


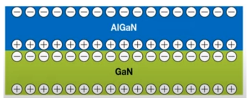
Figure 2.43: Band diagram of an heterostructure formed by AlGaAs and GaAs alloys [29]

2.8.3 GaN physical structure

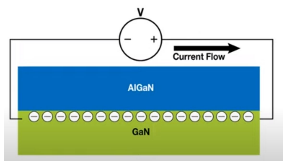
Differently from MOS, GaN devices are characterized by *heterostructure* of $Al_xGaN_{1-x}N$ alloy growth on GaN substrate. The parameter x influences the bandgap and lattice constant structure. In this kind of technology, one of the key characteristics of the matarials is the large bandgap difference of the alloys used: in fact, considering the graph in figure 2.43, putting in contact two alloys of AlGaAs and GaAs, at contact interface between the two materials a quantum well is formed. *The electrons of the n-type doped barrier layer collected in the quantum well are commonly referred as two dimensional electron gas (2DEG), since they behave as free particles in two directions (in the plane of the interface) while their motion is quantized only in orthogonal direction. The separation of electrons from their donor atoms N_D reduces the Coulomb scattering and leads to a high mobility of the electrons in the 2DEG*[29]. This structure increases significantly the mobility of electrons due to the absence of ionized impurities that in MOS structure cause scattering events. As a results, the speed of the device is larger than the MOSFET one.

2.8.4 GaN FET polarization

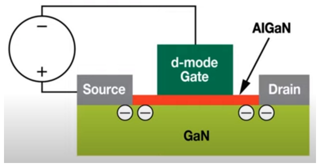
When there's no voltage applied to the structure, the system is at equilibrium. This equilibrium derives from the fact that when the AlGaN and GaN alloys are put in contact, electrons flow from AlGaN to GaN. This mechanism leaves ionized positive charge in the AlGaN layer at the interface between the two materials (see fig. 2.44a). Applying a voltage to the structure, the 2DEG becomes the source of electrons to make current flow (fig. 2.44b). The conduction of a FET can be controlled in two ways, depending on the physical design:



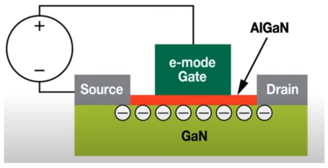
(a) HEMT at equilibrium



(b) HEMT polarized



(c) Depletion mode polarization GaN FET device



(d) Enhancement mode polarization GaN FET device

Figure 2.44

- **depletion mode** (**D-mode**): the channel of electrons is already formed at equilibrium condition, so that without any V_{GS} applied the current can flow; this kind of conduction is called *normally-on* because the device conducts current without any *command voltage* (V_{GS}) application; to stop the conduction, a negative V_{GS} has to be applied (fig. 2.44c). Due to the need of negative voltage, these kinds of devices are not so frequently used in power applications;

- **Enhancement mode** (**E-mode**): contrary to depletion mode devices, the channel has to be formed by a positive V_{GS} voltage application (*normally-off*) (fig. 2.44d); for this reason, these kind of devices are mostly used in power applications.

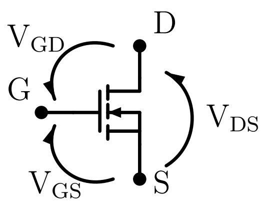
2.8.5 Electrical characteristics

Similarly to Si MOS devices, the basic transistor GaN parameters are:

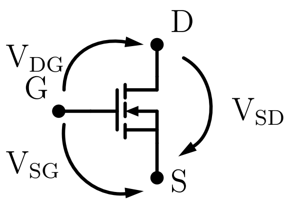
- **On-resistance** R_{on}: it is related to the equivalent 2DEG resistance and the source and drain contact resistances; it's variation in temperature is similar to the silicon MOSFETs [13] (fig. 2.46a); in general, the R_{on} of GaN FET is lower than that of silicon MOSFET [12][15];

- **Threshold voltage $\mathbf{V_{th}}$**: it is the voltage necessary to turn on the device; as happens for MOSFET, it has a negative sensitivity to temperature, but the slope is not so steep than MOSFET one[13] (fig. 2.46b);

- **capacitance**: C_{GS}, C_{DS} and C_{GD} are the three main capacitances that characterize GaN transistors; observing figure 2.46c, it can be seen a drop in the



(a) *Forward* conducted GaN



(b) *Reverse* conducted GaN

Figure 2.45: Forward and reverse conducted voltages on GaN FET

capacitance value at increasing of V_{DS} value: this is due to the depletion charges in the 2DEG near the surface at increasing of V_{DS}[13];

- **reverse conduction**: as shown in subsection 2.8.1, MOS transistors have intrinsic parasitic diodes; on the contrary, for GaNs, the absence of drain and source wells makes impossible the formation of parasitic pn-junctions: this allows to drive the GaN FETs in reverse direction by only changing the driving voltage from V_{GS} to V_{GD}. To point out this behaviour, figure 2.45 can be considered.

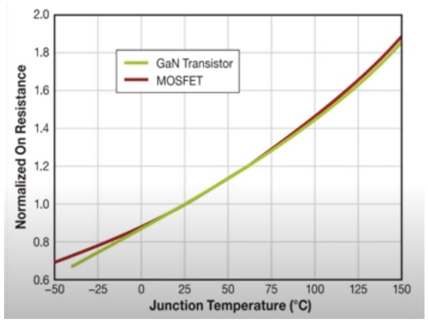
  In *forward* conduction, the device can conduct when $V_{GS} > V_{TH}$. In this condition all the voltages are positive (fig. 2.45a). The voltages equation are

$$\begin{aligned} V_{GD} - V_{GS} + V_{DS} = 0 \\ V_{GD} = V_{GS} - V_{DS} \\ V_{DS} = V_{GS} - V_{GD} \end{aligned}$$

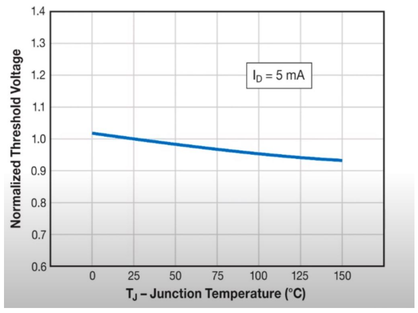
  Suppose $V_{GS} < 0$ and $V_{DS} < 0$ (fig. 2.45b). If V_S is made more and more positive, V_{DG} increases: when $V_{DG} = -V_{GD}$ reaches the *threshold voltage* V_{TH_2} the device starts to conduct in *reverse direction.* The subscript 2, in V_{TH_2} notation, indicates the fact that in reverse conduction the threshold voltage in general is not equal to the threshold voltage V_{TH_1} of *forward conduction* ($V_{TH_1} \neq V_{TH_2}$).
  As a consequence, the advantage of GaNs over MOSFETs in reverse conduction is the *absence of power dissipation due to the presence of parasitic diodes.* This means an increase in power efficiency. Moreover, the *reverse recovery charge* is zero ($Q_{rr} = 0$): this increases the speed of device because the parasitic capacitance related to the pn-junction does not exist.

2.8.6 GaN FET in Class D amplifier applications

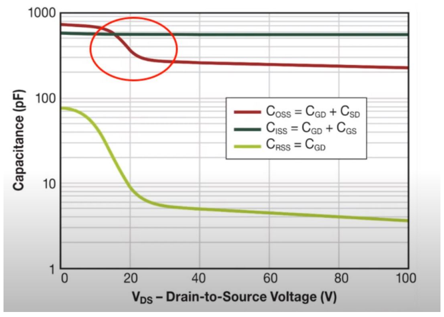
Due to their advantageous physical characteristics, GaN FETs are becoming a valid alternative to the traditional technologies like Si or SiC. In particular, for class D amplifier some studies are done in literature to evaluate the performance in terms of efficiency, distortion and sensitivity to temperature [12][15].
In [12], a class D amplifier in half-bridge open-loop configuration is taken into consideration to investigate what are the elements that causes an efficiency reduction. The specification of the designed system is in table 2.47.



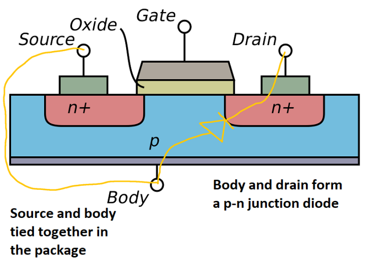
(a) *R_{on} Si and GaN FETs comparison* [13]



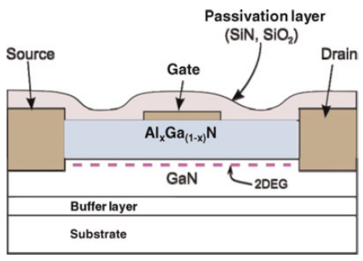
(b) *V_{th} vs I_D Si and GaN FETs comparison* [13]



(c) *Capacitances vs V_{DS} for GaN devices* [13]

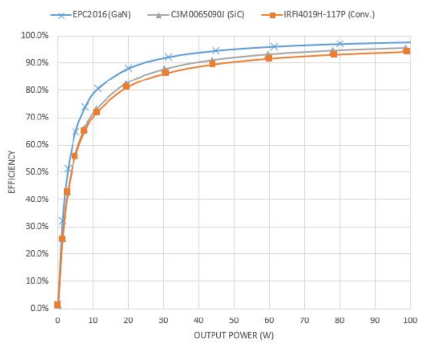

Gate
Oxide
Source
Drain
n+
n+
p
Body
Source and body tied together in the package
Body and drain form a p-n junction diode

(d) *MOSFET physical structure*



(e) *GaN FET physical structure* [30]

Figure 2.46



(a) Efficiency comparison among different devices [12]

Device	V_{DS}	I_{DS}	R_{dsON}	C_{oss}
IRF4019H-117P (Conventional)	150 V	8.7 A	80 mΩ	100 pF
EPC2016 (GaN)	100 V	11 A	16 mΩ	225 pF
C3M0065090J (SiC)	900 V	35 A	65 mΩ	60 pF

(b) Electrical characteristics of devices used in [12]

Figure 2.48

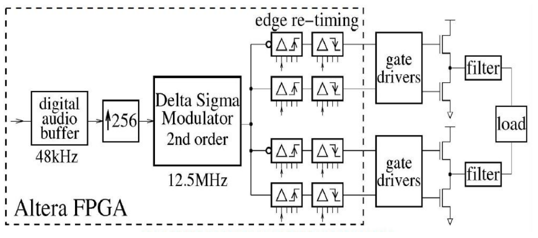
Rated power	100 W RMS
Switching frequency	400 kHz
Supply voltage	±45 V
Freq. response	20 Hz – 20 kHz
Load impedance	8 Ω
Modulation index	≤ 0.9

Figure 2.47: Specifications for simulated amplifiers designed in [12]

The paper shows that one of the main section that influences the system power dissipation is the output stage. In particular, comparing three different devices made in three different technologies (GaN, Si, SiC), GaN is the most efficient one (fig. 2.48a). Comparing the electrical characteristics in table 2.48b, it can be seen that GaN R_{dsON} is the lowest one and, as a consequence, the power dissipation is lower, in particular for higher output power.
In [15], two class D amplifiers in open-loop full-bridge configuration are designed and performances in terms of efficiency, distortion and sensitivity to temperature are evaluated. The output stage of the first amplifier is designed using Si MOSFET, while the other one with GaN FET devices. Audio data are streamed into FPGA at 48 kHz, up-sampled at 12.5 MHz for the second order ΣΔ modulator (fig. 2.49a). The employed modulation, differently from most part of literature systems, is *Pulse Densisty Modulation* (*PDM*). Two switching frequencies are tested:

- $f_{sw} = 1.1\,\text{MHz}$;
- $f_{sw} = 360\,\text{kHz}$.

The TI drivers used are LM5113 for GaN FET and UCC27211 for Si FET configuration. THD+N and efficiency versus output power and efficiency versu output power are measured:



(a) CDA block diagram [15]

	GaN	Silicon
Manufacturer	GaNSystems	Texas Instrument
Part Number	GS61004B	CSD19537Q3
Voltage rating	100 V	100 V
R_{on}	15 mΩ	12.1 mΩ
Q_g	6.6 nC	16 nC
V_{th}	1.3 V	3 V
FOM	$9.9 \times 10^{-11}\Omega{\cdot}C$	$1.94 \times 10^{-10}\Omega{\cdot}C$

(b) Devices characteristics in [15]

Figure 2.49

- at 20 W output power, third harmonic has less power in GaN solution (figs 2.51a and 2.51b); this means that GaN FETs give better audio performances than Si competitors;
- GaN output stage has better performances at output power level greater than 0.5 W (see fig. 2.51c); distortion on output signal is reduced using GaN devices for the most part of the output dynamic range; at low power level (less than 0.5 W) MOSFETs output stage solution show less THD+N;
- **efficiency** is better in GaN output stage for output power levels up to 20 W (fig. 2.51c); as shown in [12], [15] has confirmed the better performances in terms of efficiency of GaN FETs, thanks to lower R_{dsON} (fig. 2.51c);
- **switching performances** show that GaN devices are faster, having lower rising and falling time;

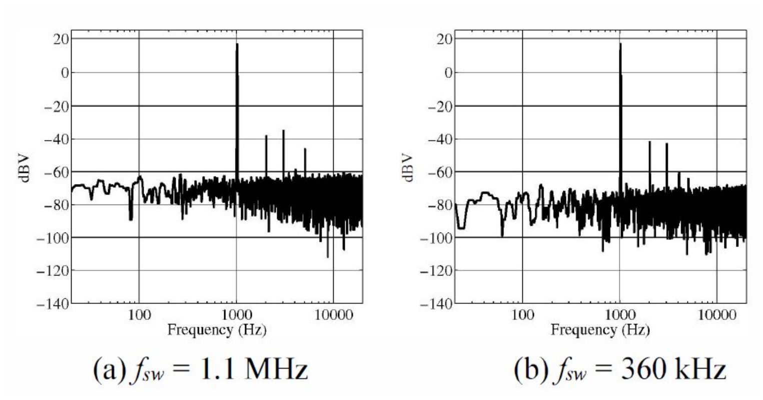
GaN		Si	
$\mathbf{t_{rise}}$	$\mathbf{t_{fall}}$	$\mathbf{t_{rise}}$	$\mathbf{t_{fall}}$
8 ns	6 ns	11 ns	21 ns

Figure 2.50: Switching performances of devices measured in [15]

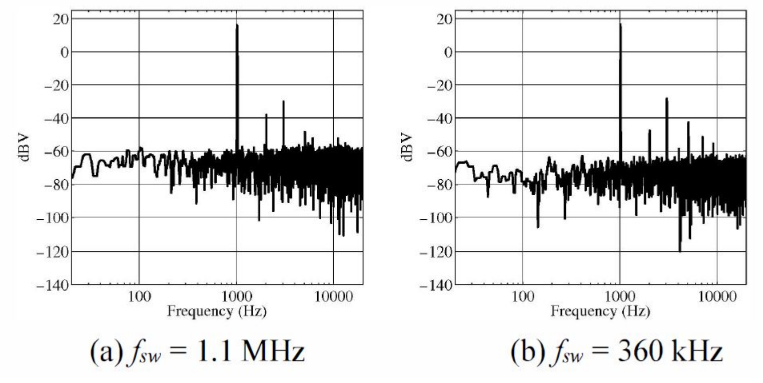
- **temperature** GaN case is $20 - 25\,°C$ lower than Si transistors in this test; this is a consequence of less power dissipation;

Performances at different temperature are evaluated, too. In particular the systems performances are measured after 1 min and 15 min. The same measurements are performed applying an air forced cooling system. Measurements are resumed in the tables 2.52. The following considerations can be done:

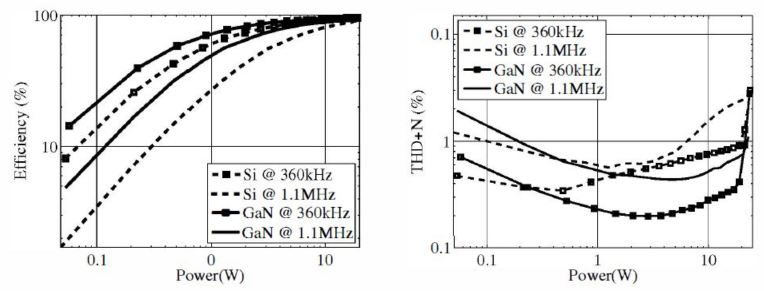
- for GaN output stage, efficiency is approximately the same at changing of temperature T: this can be explained considering that the increase of temperature also increases the V_{th} of the device; if V_{th} increases, the moment when the device is turned-on is delayed, so the dead time becomes larger than the nominal



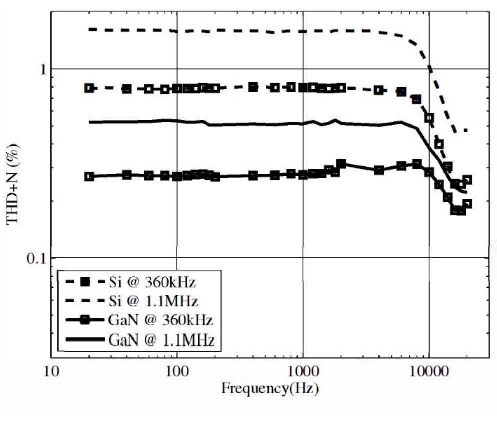
(a) *Single-tone spectrum of the GaN output stage* [15]



(b) *Single-tone spectrum of the silicon output stage* [15]



(c) *Efficiency and THD+N versus output power with* 1 kHz *tone* [15]



(d) *THD+N vs frequency at* 10 W *output power* [15]

Figure 2.51: Measured results in [15]

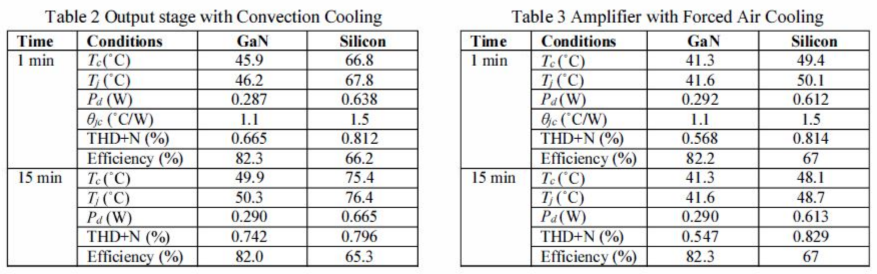


Table 2 Output stage with Convection Cooling

Time	Conditions	GaN	Silicon
1 min	T_c(˚C)	45.9	66.8
	T_j (˚C)	46.2	67.8
	P_d (W)	0.287	0.638
	θ_{jc} (˚C/W)	1.1	1.5
	THD+N (%)	0.665	0.812
	Efficiency (%)	82.3	66.2
15 min	T_c (˚C)	49.9	75.4
	T_j (˚C)	50.3	76.4
	P_d (W)	0.290	0.665
	THD+N (%)	0.742	0.796
	Efficiency (%)	82.0	65.3

Table 3 Amplifier with Forced Air Cooling

Time	Conditions	GaN	Silicon
1 min	T_c (˚C)	41.3	49.4
	T_j (˚C)	41.6	50.1
	P_d (W)	0.292	0.612
	θ_{jc} (˚C/W)	1.1	1.5
	THD+N (%)	0.568	0.814
	Efficiency (%)	82.2	67
15 min	T_c (˚C)	41.3	48.1
	T_j (˚C)	41.6	48.7
	P_d (W)	0.290	0.613
	THD+N (%)	0.547	0.829
	Efficiency (%)	82.3	67

Figure 2.52: Performances at varying of temperature with no cooling system (left table) and with cooling system (right) [15]

designed value. From this consideration, efficiency should increase. But because of R_{dsON} has a positive sensitivity to temperature, if T increases, R_{dsON} increases and, as a consequence, efficiency η is reduced[15]. The reasoning can be schematized as below

$$\text{if } T \uparrow \Rightarrow V_{th} \uparrow {}^{2} \Rightarrow t_{dead} \uparrow {}^{3} \Rightarrow \eta \uparrow \ldots$$
$$\text{but } T \uparrow \Rightarrow R_{dsON} \uparrow \Rightarrow \eta \downarrow \Rightarrow \eta \simeq$$

- in a similar way, the variation of THD and efficiency of amplifier with MOSFET can be explained considering that V_{th} goes down while R_{dsON} goes up as increases T. As a consequence, the dead time increases, reducing THD and, due to R_{dsON} variation, efficiency goes down, too [15]. Schematizing

$$\text{if } T \uparrow V_{th} \downarrow , R_{dsON} \uparrow \Rightarrow THD \downarrow , \eta \downarrow .$$

- in case of air-forced cooling system, the performances of the two amplifiers do not show particular variation in time. Again, GaN FET performances are better than Si FET ones, both in terms of efficiency and distortion.

GaN FETs: a new frontier in class D power amplifiers

As shown in previous sections, GaN devices have lot of advantages in terms of costs, occupied area, efficiency and, in audio field, distortion. Even if in literature these devices are not so much investigated for audio systems, the researches reviewed in subsection 2.8.6 have shown that their characteristics can lead an improvement in efficiency and audio quality of sound systems, promoting them as new possible devices for realization of high-fidelity low-consumption class D power amplifiers.

2.9 Class D amplifier designs

The main goals of class D amplifier design is to achieve higher power efficiency with respect to the traditional classes, without sacrificing the audio quality. These two

[3]V_{th} has positive temperature coefficient

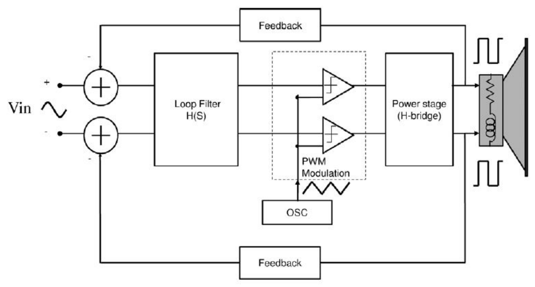
specifications are contradictory[6]. For this reason the design is complex, in particular for high fidelity applications. However, some design steps are very common among literature designs [6][14][10][4][2][5]

1. Choosing the basic scheme modulation, depending on the applications, cost, area, complexity and specifications requirements (e.g. maximum output power, signal fidelity, noise immunity[6]);

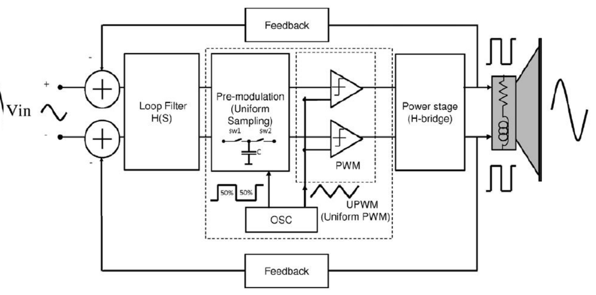
2. switching frequency (generally taken $> 500\,\text{kHz}$): increasing switching frequency means in general reducing THD+N and increasing SNR and PSRR, at the price of higher EMI[6];

3. closed- or open-loop architecture: depending on the specifications, open or closed-loop design can be done; anyway, do to high non-linearity of this amplifiers, to obtain acceptable performances, lot of literature designs are done in closed-loop [14][4][10][9].

Starting from these steps, other factors can influence the design choices, in particular if the design is fully analog like in [4][3][5] or with global feedback and digital control [10][16][14].



(a) *Conventional analog closed-loop class D amplifier with PWM* [4]



(b) *Analog closed-loop class D amplifier with UPWM* [4]

Figure 2.53: Comparison between analog closed-loop architecture with PWM and UPWM

2.9.1 Analog filterless design

An example of analog design is shown in [4]. In this design, a *filterless* closed-loop architecture is implemented with CMOS technology. One of the main problem of conventional closed-loop class D amplifier architecture (fig. 2.53a) with PWM modulation is due to the aliasing caused by the *feedback ripple when sampled by PWM*[4]: when the PWM signal is feedback to evaluate the *error* signal, it is re-sampled by

the triangular waveform. The error signal has a spectrum that contains baseband audio and PWM components and is filetered by the loop filter: as a consequence, when the compensator output signal is sampled by the PWM comparator triangular waveform, due to residual ripple at f_{sw}, the Nyquist criterion is not respected and aliasing happens, increasing THD.
To overcome this problem, a *Uniform PWM* (*UPWM*) architecture (fig. 2.53b) is implemented: the idea is to sample and hold (SAH) the output compensator signal at switching frequency f_{sw} before the PWM sampling. Sampling and hold at a determined frequency (e.g f_{sw}) means introducing infinite attenuation at sampling frequency and its harmonics (e.g. f_{sw}, $2f_{sw}$, $3f_{sw}$, ...) as shown in figure 2.54. In this way, when the compensator output signal is sampled and hold, the harmonic components related to the PWM are nullified, reducing the effect of aliasing.

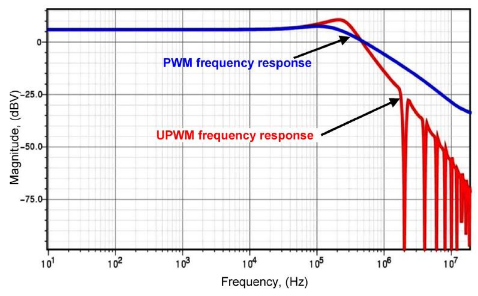


Figure 2.54: PWM and UPWM transfer function [4]

2.9.2 Loop filter design

The loop filter is designed as an active second-order integrator with two RC-pole and one zero to obtain the correct phase margin. The full circuit is shown in figure 2.55

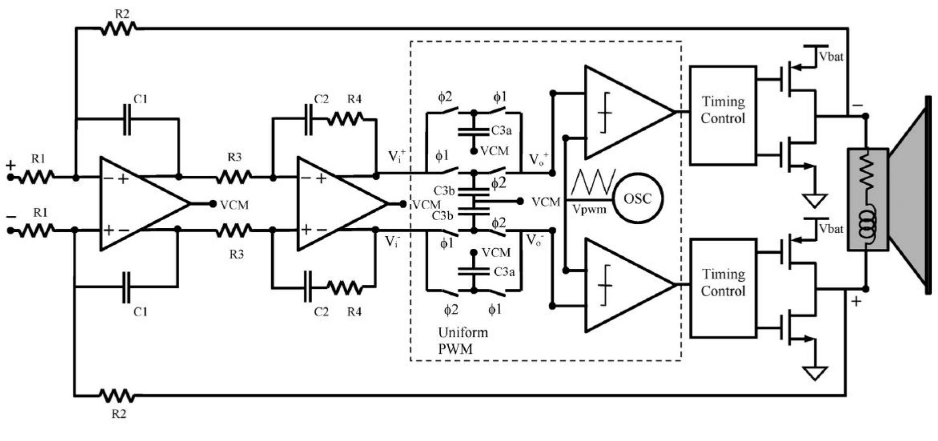


Figure 2.55: Closed-loop circuit of [4] with UPWM

The open-loop transfer function of the loop-filter is

$$H_{LF}(s) = \frac{1 + R_4 C_2 s}{R_3 C_2 R_1 C_1 s^2} \text{ [4]} \tag{2.57}$$

while the closed-loop results

$$H(s) = -\frac{R_2 G_{PWM}(1 + R_4 C_2 s)}{R_1(G_{PWM} + G_{PWM} R_4 C_2 s + R_3 C_2 R_2 C_1 s^2)} \text{ [4]} \tag{2.58}$$

where G_{PWM} is the product of modulator and output stage gain V_{bat}/V_{pwm}, with V_{bat} the power supply of power stage and V_{pwm} is the power supply of PWM modulator. In particular, $V_{pwm} = \frac{V_{bat}}{2}$ obtaining $G_{PWM} = 2$. *The unity gain of the first and second integrating stages are set at* [4]

$$f_{UG1} = \frac{1}{2\pi R_1 C_1} = 110\,\text{kHz} \tag{2.59}$$
$$f_{UG2} = \frac{1}{2\pi R_3 C_2} = 221\,\text{kHz} \tag{2.60}$$

and the zero has been located at [4]

$$f_{zero} = \frac{1}{2\pi R_4 C_2} = 147\,\text{kHz} \tag{2.61}$$

Let's observe that for $s \to 0$ the eq.2.58 becomes

$$H(s \to 0) = -\frac{R_2}{R1}\frac{G_{PWM}}{G_{PWM}} = -\frac{R_2}{R_1} \tag{2.62}$$

R_2 is set equal to $2R_1$ to fix the gain in audio band to 2 (6 dB). Qualitatively, the open-loop filter transfer function is shown in figure 2.56.

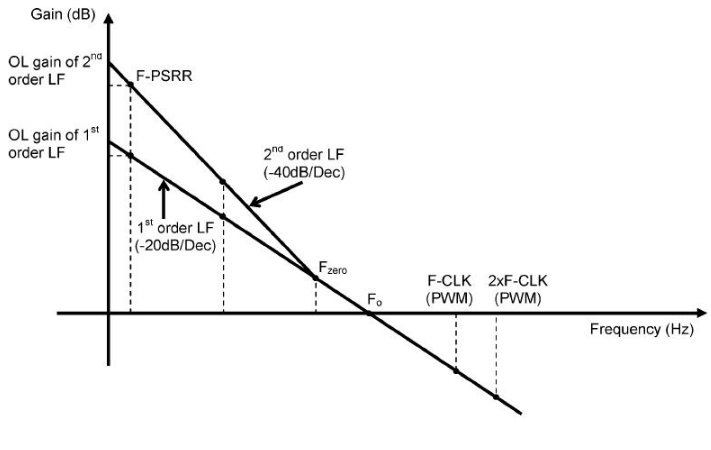


Figure 2.56: Qualitatively open-loop transfer function of a first- and second-order loop filter [4]

The resulting open-loop transfer function has 127 dB gain at 100 Hz, 114 dB at 217 Hz and 87 dB at 1 kHz. The attenuation at switching frequency ($F_{clk} = 1$ MHz) is 16 dB and 23 dB at $2F_{clk} = 2$ MHz. The amplifier performances are resumed in the following table, extracted from [4].

SNR (dB), A-weigthed	103
THD+N(%)@100 Hz	0.00093
THD+N(%) @ 1 kHz	0.00122
PSRR (dB) @ 217 Hz	96
Max. η (%), 8Ohm Load	93
F_s (kHz)	1000
Max. Pout (W), @1kHz THD+N<1%, Min. Load	3.1

Figure 2.57: Resuming table of amplifier performances designed in [4]

Block	Poles	Zeros
G_1(z)	$f_{p1} = f_{p2} = 700$ kHz	$f_{z1} = f_{z2} = 70$ kHz
G_2(z)	$f_{p1} = f_{p2} = 1$ MHz	/
G_3(z)	$f_{p1} = f_{p2} = 11.3$ kHz $f_{p3} = f_{p4} = 18.6$ kHz $z_p = 1$	$f_{z1} = f_{z2} = 31.2$ kHz $f_{z3} = f_{z4} = 41.6$ kHz
LPF	$f_{p1} = f_{p2} = 70$ kHz	/

(a) *Parameters of the filters in* [14] *design*

Block	f_{clk}	f_{sample}
DPWM (7bit)	98.304 MHz	768 kHz
Loop	19.6608 MHz	19.6608 MHz
A/D	19.6608 MHz	19.6608 MHz

(b) *Parameters of design in* [14]

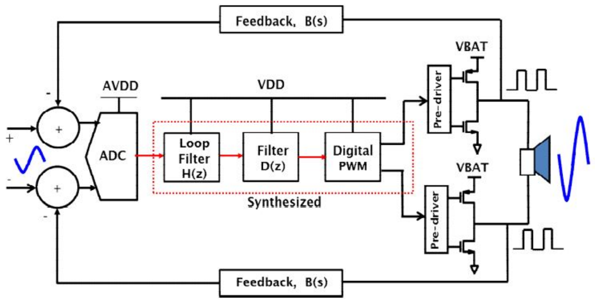
Figure 2.58: Parameters of filters and design in [14]

2.10 Digital designs

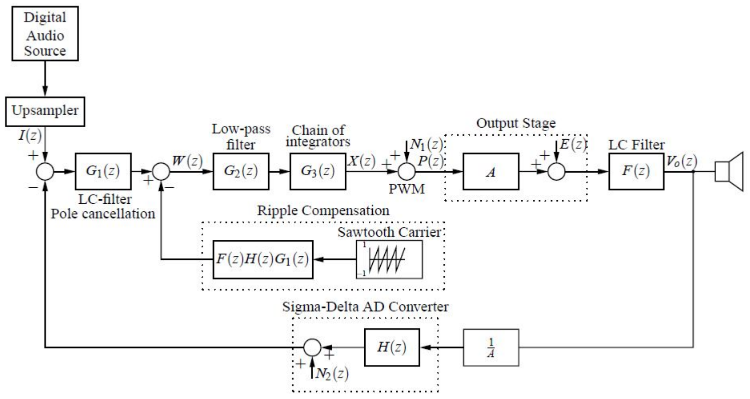
In recent years have been developed solutions of closed-loop switching amplifiers with global feedback and digital control. With respect to analog designs there are some architectural differences:

- the presence of an ADC; this circuit is used to sample the PWM signal at the output of power stage, as in [14] or to sample the error, as [16];
- PWM modulator is replaced with a DPWM modulator; this difference, with respect to analog design, introduces a problem about the SNR, due to the resolution of DPWM;
- depending on the ADC position in the loop, some additional filters have to be added to reduce the effect of high frequency noises, in particular aliasing and non-linear effects that can trigger instability [14]; this increase the design complexity of closed-loop system.

The block diagrams of digital designs in [16] and [14] are respectively shown in fig. 2.59a and 2.59b.



(a) *Block diagram of [16] design*



(b) *Block diagram of [14] design*

Figure 2.59: Digital design architectures of [16] and [14]

2.10.1 ADC in feedback path

The model of digital system in [14], fig. 2.59b, consists in different blocks:

- DPWM modulator: it is modelled as a linear component that introduces a noise $N_1(z)$ due to the finite number of bit;
- the output stage is modelled as a gain block of $A = \frac{V_d}{2}$;
- ADC Sigma Delta converter is modelled with a delay transfer function, $H(z) = z^{-1}$, with the added shaped quantization noise $N_2(z)$;
- the digital loop filter consist in:
  - $G_1(z)$: it provides the zero-pole cancellation of the output filter double pole; moreover, it has two high frequency poles to reduce the quantization noise of ADC;

- $G_2(z)$: its main goals is to attenuate the quantization noise of ADC; it is a second order Chebyshev filter;
- $G_3(z)$: is a chain of five integrators, with two double poles and a pole in the origin.

A *ripple compensation* is added after $G_1(z)$ and the stability of the system is strictly dependent on the presence of this block. The main parameters of the filters and system are resumed in the table 2.58a and 2.58b respectively. The open-loop filter compensation transfer function is shown in figure 2.60.
The digital design is implemented with FPGA programmed in VHDL. The resulting THD+N is 0.0009% at 15 W output power.

2.10.2 ADC error sampling

The main problem of previous solution is the design complexity due to the presence of zero to compensate the poles of output filter and the high number of poles to reduce the quantization noise of ADC. In [16], the idea is to implement in digital way the architecture of [4]. This means to replace the transfer function $H(s)$ with an ADC (that mimics the differential function of the amplifier in [4]) and a digital filter. Moreover, the analog switching capacitance circuit that provides uniform sampling, can be easily replaced with digital solutions.

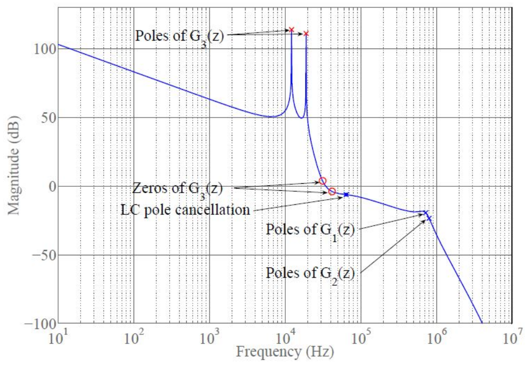


Figure 2.60: Open-loop transfer function of design in [14] ($G_1(z)G_2(z)G_3(z)F(z)$)

Placing the ADC to sample the error signal gives different advantages

1. the error signal has a lower dynamic excursion respect to the PWM signal of output stage, relaxing the dynamic input requirement of ADC;
2. *the output swing of the closed loop ADC amplifier(s) is also relatively small, which eases its design requirements and allows implementation even in low supply voltage domains*[10];

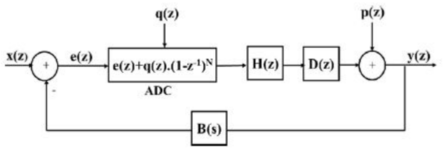


Figure 2.61: Block diagram of system in [16]

3. a front-end ADC can be implemented with a first-order one-bit resolution Sigma Delta modulator, reducing the problems with additional noises and spurious tones.

About the loop transfer function design, the first requirement is to obtain a very high loop gain in audio band. In this prototype, loop gain is designed with these specs

$$|T(z=0)|_{\mathrm{dB}} = 100\,\mathrm{dB}$$
$$|T(f < 20\,\mathrm{kHz})|_{\mathrm{dB}} > 60\,\mathrm{dB}$$

In this way can be obtain

- suppression of DPWM non-idealities;
- reduction of non linearities of output driver;
- high PSRR.

The model of the amplifier is shown in figure 2.61. The functions of each block are the following

- **ADC**: sigma-delta ADC, implemented in *switching capacitance fashion*;
- **B(s)**: RC differential filter, to reduce aliasing of output switching signal;
- **H(z)**: filter to improve PSRR and linearities;
- **D(z) & DPWM**: compensate feedback loop and generates PWM.

The design is made considering that the output signal $y(z)$ is [16]

$$y(z) = STF \cdot x(z) + NTF \cdot q(z) + PTF \cdot p(z) \tag{2.63}$$

where [16]

$$STF = \frac{H(z)D(z)}{1 + H(z)D(z)B(z)} \tag{2.64}$$
$$NTF = \frac{H(z)D(z)(1-z^{-1})^N}{1 + H(z)D(z)B(z)} \tag{2.65}$$
$$PTF = \frac{1}{1 + H(z)D(z)B(z)} \tag{2.66}$$

The loop transfer function is designed considering the effect of the previous transfer function

- STF: is designed to make pass the audio band unaltered;
- NTF: this function is designed to reduce the quantization error of the ADC; can be observed that this transfer function can be described with a parameter N, the order of the Sigma Delta converter;
- PTF: shaping accurately this function can be reduced the quantization noise of the DPWM and the THD of the amplifier.

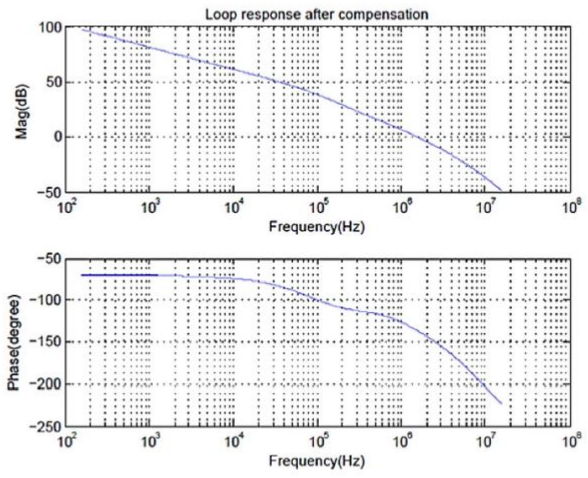


Figure 2.62: Loop transfer function of the design in [16]

The parameters of the filters are designed in the following way

- **H(Z)**: first-order integrator

$$H(z) = \frac{1}{1 - z^{-1}} \tag{2.67}$$

- **D(z)**:
  - 2 poles: $p_1 = 1\,\text{MHz}$, $p_2 = 15\,\text{MHz}$
  - 1 zero: $z_1 = 400\,\text{kHz}$ (to improve loop phase margin)

  The coefficients are determined by bilinear transformation

$$s = \frac{2F_s(z-1)}{1 - z^{-1}} \tag{2.68}$$

- **B(s)**: differential RC circuit

$$B(s) = \frac{G}{s+p} \tag{2.69}$$

where $G = 0.4$, $p = 100\,\text{kHz} \Rightarrow R_1 = 30\,\text{k}\Omega$, $R_2 = 40\,\text{k}\Omega$, $C_{FB}/2 = 25\,\text{pF}$

The overall loop gain is a 4-th order system (four poles). The resulting closed-loop transfer function is shown in figure 2.62. The measured main performances consists in SNR=105 dB and THD+N=0.0031%

	[4]	[16]	[14][4]
Integrator order	2	1	5
Poles number	2	4	11
Zeros number	2	1	6
$\mathbf{F_{sw}}$	1 MHz	2.133 MHz	768 kHz
$\mathbf{F_{clock}}$	/	51.2 MHz	19.6608 MHz
$\mathbf{F_{clock\ PWM}}$	/	400 MHz	98.304 MHz
Output stage	H-bridge (BTL)	H-Bridge (BTL)	H-bridge (SE)
$\mathbf{V_{PS}}$	$2.5 - 5$ V	$2.5 - 5.5$ V	25 V
Modulation	UPWM	three-level DPWM	DPWM
$\mathbf{P_{out\ MAX}}$**@1 kHz**	2 W	1.5 W	15 W
Technology	0.25 µm CMOS	55nm CMOS	Discrete

Figure 2.63: Design parameters

2.10.3 Comparisons among solutions

A resuming table (2.63) with the main characteristics, parameters and performances of the amplifiers designed in [4], [14] and [16]([10]) is shown. Some observations can be done about the differences among designs

- the design in [14] can provide higher power than the two competitors;
- the digital design of [16] use larger sample and switching frequency than [14], requiring faster hardware;
- the number of poles and zeros of [14] is much higher than the two competitors, increasing the design complexity and requiring the presence of the ripple compensation to be stable;
- power supplies of designs [4] and [16] make them useful for portable applications (like phones or PCs);
- the digital version ([16]) of the design in [4] has a THD+N a bit less than the analog version.

[4]not considered ripple compensation

Finally, the presence of quantization noise increases the complexity of digital loop because it is necessary a much larger low frequency gain of the loop transfer function. This increases the difficult to stabilize the system and choose the crossover frequency.

2.11 Class D amplifier: new frontiers and digital architecture approach

In this review are described the basic characteristics of class D amplifiers, showing the motivations of the advantages in terms of power efficiency. Due to the PWM driving method for the switching transistors of the output stage, open-loop class D amplifiers show much higher distortion than their competitors of traditional linear classes. For these reasons, most of the literature and industrial systems are designed in closed-loop. The difficult to stabilize and obtain good performances in closed-loop is strictly related to the aliasing of the PWM ripple in the loop and to the switching frequency: higher the switching frequency, better the performances of the amplifier (e.g. lower THD+N). A higher switching frequency is not so simple to obtain by adopting analog implementations by analog architectures and MOSFETs have speed limitations that can be overcome by GaN technology. For this reason, alternative digital architectures are implemented using DPWM, even simplifying the design of modulation section thanks to programmable devices (e.g. FPGA, microcontroller). The resolution of the DPWM and the quantization noise due to ADC conversion result the main bottleneck of the system, requiring very high low frequency loop gain that can be easily obtained with digital design. On the other hand, the DPWM resolution can be increased by using different implementation like the new DDPWM, overcoming the trade-off between resolution and clock frequency. In conclusion, the advantages of new technologies like GaN transistors which allow to reach higher switching frequency and new architectures to increase the resolution of DPWM are moving class D amplifier toward new design strategies and control architectures.

Chapter 3

Design of a class D amplifier

In this chapter, the design of a class D audio amplifier starting from a set of specifications will be described. The architecture of the amplifier will be selected taking into account the vantages and the disadvatages of the different digital solutions illustrated in the previous Chapter. Open- and closed-loop configurations are designed, verifying by mean of simulation the performances of the systems to support and validate the designs. Particular focus is given to modulator section which is investigated in order to highlight the bottleneck of different implementations (DPWM, DDPWM) and proposing an alternative one (DDPM-DPWM combination).

3.1 Proposed digital model

In this section is described the proposed model of an almost fully-digital architecture of a class D amplifier in open-and closed-loop fashion. The basic idea is to implement the PWM of analog architectures in digital way, substituting it with a DPWM one. The main point is that the analog signal has to be converted in digital one to be sampled by DPWM and so a front-end ADC has to be inserted before the DPWM section. Then, after DPWM, drivers and output stage have to be added, connected the last one to a second order output filter. This is the basic architecture shown in figure 3.1. It can be observed in figure 3.1 the presence of two identical structures to obtain a *Bridge-Tied Load* class D amplifier. The dashed line indicates how it is possible to implement a closed-loop system by taking the output stage PWM signal. To be precise this simple closed-loop architecture has some limitations:

- depending on the ADC input dynamic range, it is necessary an attenuation of the feedback PWM signal to adapt the dynamic of signal to the ADC one; moreover, this attenuation determines the overall voltage gain of the amplifier in audio band;

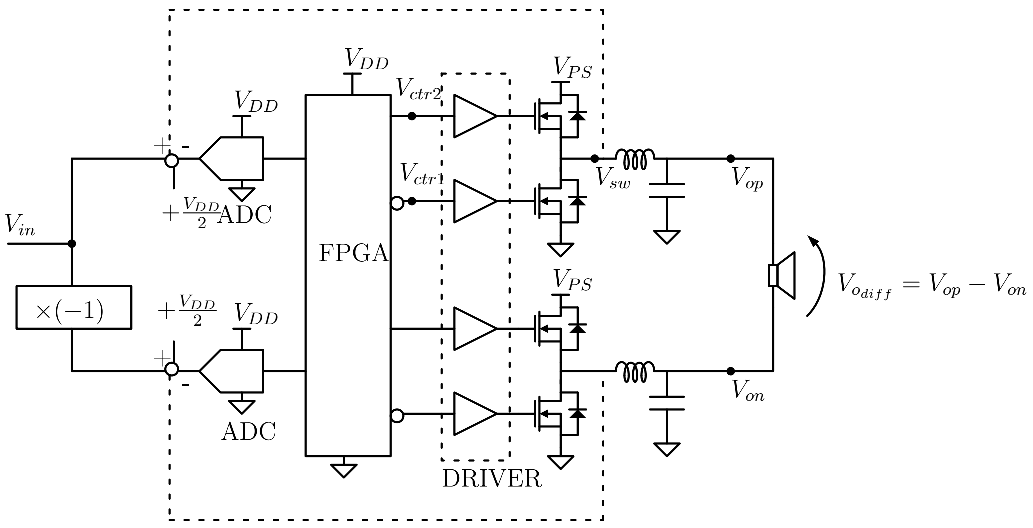


Figure 3.1: CDA schemes proposed

- to avoid the aliasing effect, as shown in [16] an anti-aliasing filter is necessary.

Moreover, the digital control and DPWM generation can be specified in terms of software, using FPGA or microcontroller. The architecture results very similar to the one in [16].

3.2 Specifications of the designed amplifier

The specifications of the designed amplifier are chosen considering low power applications, for example for PC audio or portable application like [31]. In particular, this device [31] can be a good comparison for specifications because it used by professional sound technicians to check the quality of an audio production on a commercial system. The design specs are

- $P_{O_{MAX}} \geq 2.5\,\mathrm{W}$;
- $V_{DD} = 5\,\mathrm{V}$: digital part and ADC power supply;
- $R_L = 8\,\Omega$, load nominal resistance;
- $f_{sw} \simeq 500 - 1000\,\mathrm{kHz}$;
- $f_{clk_{SYS}} \simeq 16\,\mathrm{MHz}$.

Moreover, as suggested by [5], a good audio amplifier should have $SNR > 80\,\mathrm{dB}$ and $THD > 0.1\%$, $PSRR > 80\,\mathrm{dB}$ and a power efficiency $\eta > 80\%$. A table with some reference designs taken from [5] is shown in 3.2.

Parameters	A	B	C	D	E
Supply (V_{DD}) [V]	5	5	5	5	3.6
I_Q [mA]	7	1.42	6.5	4.2	2.7
P_Q [mW]	35	7.1	32.5	21	9.7
Efficiency [%]	85	90	85	90	85
THD+N [%]	0.65	0.02	0.02	0.08	0.01
$P_{o,\ max}$ [W]	1.2	1.7	1.4	1.7	2.3
PSRR [dB]	65	88	85	93	88
SNR [dB]	83	98	96	89	97
F_{sw} [kHz]	250	192	420	300	300

Figure 3.2: *Typical specifications for commercial class-D audio amplifiers*[5]

3.2.1 Design of output stage power supply

The power supply of the output stage depends on the output power requested on the load. Considering equation 2.20, for a bridge-tied load amplifier and a pure sinewave on the load (neglecting switching frequency ripple and distortion) the single-ended stage output power is given by

$$P_{o_{SE}} = \frac{V_{o_{SE_{pk}}}^2}{2R_L} \Rightarrow V_{o_{SE_{pk}}} = \sqrt{2P_{o_{SE}}R_L} \tag{3.1}$$

where $V_{o_{SE_{pk}}}$ is the peak voltage of the sinewave at the output of output filter. As it has been seen in 2.1.4, the gain of open-loop single-ended CDA with single positive power supply is given by

$$G_M A = \frac{V_{PS}}{V_{DD}} \tag{3.2}$$

where G_M is the modulator gain and A the output stage gain. As a consequence, the peak voltage of the output sinewave is given by

$$V_{o_{SE_{pk}}} = \frac{V_{PS}}{V_{DD}} V_{in_{pk}} \tag{3.3}$$

Substituting 3.3 in 3.1, then in 2.20 and inverting the formula can be obtained

$$P_{o_{BTL}} = 4 \underbrace{\left(\frac{V_{PS}}{V_{DD}} V_{in_{pk}}\right)^2}_{V_{o_{SE_{pk}}}} \frac{1}{2R_L}$$

$$\Rightarrow \boxed{V_{PS} = \sqrt{\frac{P_{o_{MAX}} R_L}{2}} \frac{V_{DD}}{V_{in_{pk}}}} \tag{3.4}$$

Because of $V_{in_{pk}}$ depends on the input dynamic of ADC can be done some considerations:

- ADC has a V_{DD} single power supply, so the input has to be biased with an offset of $V_{DD}/2$ to obtain the largest dynamic range;

- without considering CMIR limitations and non-linearities, the input dynamic range is $[0, V_{DD}]$.

A graphical explanation of this fact is qualitatively shown in figure 3.5. As a consequence

$$V_{in_{pk}} < \frac{V_{DD}}{2} \tag{3.5}$$

Substituting in equation 3.4 the specifications can be obtained

$$\begin{aligned} V_{PS} &= \sqrt{\frac{2.5\,\mathrm{W} \cdot 8\,\Omega}{2} \frac{5\,\mathrm{V}}{2\,\mathrm{V}}} \simeq 7.9\,\mathrm{V} \\ &\Rightarrow \boxed{V_{PS} = 10\,\mathrm{V}} \end{aligned} \tag{3.6}$$

where $V_{in_{pk}} = \frac{V_{DD}}{2}$ is considered equals to $2\,\mathrm{V}$, roughly estimating $0.5\,\mathrm{V}$ of CMIR.

3.3 Closed-loop system structure and advantages

Even if open-loop class D amplifiers are relatively easy to implement and have low cost and high efficiency, the main drawback is the high distortion of the output signal and presence of non-linearities that most of the time are not acceptable, in particular in high-fidelity audio system. For this reason open-loop amplifiers are typically used in systems that implement alarms, buzzers, toys [5]. To achieve acceptable audio performances, closed-loop CDA are typically implemented, for example for audio section of cell phones, .
Generally speaking, a closed-loop system can be described with the following transfer function

$$\frac{V_o}{V_{in}} = \frac{A(s)}{1 + A(s)\beta(s)} = \frac{A(s)}{1 + T(s)} \tag{3.7}$$

where $A(s)$ is the *open-loop transfer function*, $\beta(s)$ the *feedback transfer function* and $T(s)$ the *loop-transfer function.* Two of the main characteristics of this kind of systems are

1. if $\beta(s)$ is chosen as a linear block, then the closed-loop system will have a linear behaviour;

2. the higher is $|T(s)|$, the lower are non linearities.

Two possible examples of closed-loop analog systems are shown in figures 3.3 and 3.4. In both cases, the compensator filters the error signal, modulating the command signal in order to get a null error signal. For this reason the compensator is most of

the time designed as an integrator.
Then, the command is sampled with PWM signal so that

$$V_{sw} = \begin{cases} \text{HIGH if} \quad V_{o_{comp}} - V_{tr} > 0 \\ \text{LOW if} \quad V_{o_{comp}} - V_{tr} < 0 \end{cases} \tag{3.8}$$

where V_{sw} is the output signal from *Relay* [32] block that simulates the power output stage, $V_{o_{comp}}$ is the output signal from compensator block of figure 3.3 and 3.4 and the value HIGH and LOW are respectively the positive and negative power supply that the output voltage of the output stage can assume.

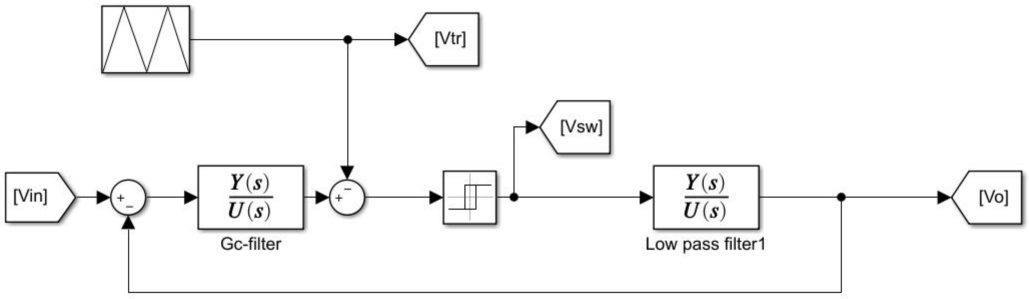


Figure 3.3: Closed-loop system with output filter in loop

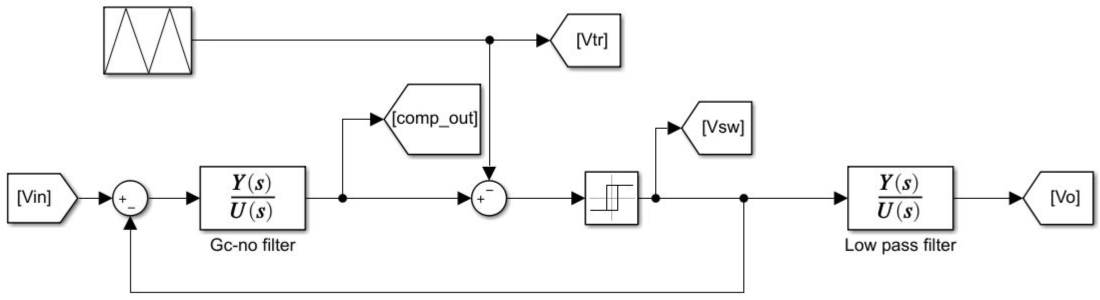


Figure 3.4: Closed-loop system without output filter in the loop

3.3.1 Differences and limits of the models

The main difference between the two schemes stays in the presence or not of the output filter in the loop. If the scheme in figure 3.3 can give better performances in terms of distortion and SNR because of the PWM signal is filtered before comes back to the feedback path, the main drawback is the complexity of loop design transfer function, in particular for stabilization and control of command dynamic;

on the other hand, the design in 3.4 is less complex because the double pole of the output filter is not present in the loop. Anyway, the PWM signal is not filtered, so the performances of the amplifier are worse than previous case. Moreover, the effect of aliasing due to the ripple sampling [33] is less attenuated respect to the solution with output filter in the feedback, because the attenuation depends only on the compensator filter.
Both models have some limitations:

- the deadtime is not considered, so the distortion due to this effect can not be evaluated;
- the triangular waveform is not affected by non linear effect due to the real exponential behaviour of analog triangular generator [5];
- the drivers are not considered and the output stage is considered with infinite bandwidth so the parasitic capacitances related to the gate are not considered;
- elements like body diodes and R_{dsON} are not modelled.

3.4 Amplifier models for simulation purposes

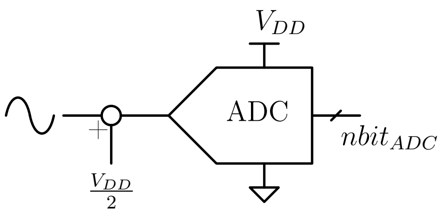
To design the closed-loop system, reaching a good stability and good performances in terms of THD and SNR, first it is necessary knowing the models of all the blocks in the loop.
In this section is described the models of each block shown in the circuit in figure 3.1. In particular the models considered are

- ADC;
- compensator;
- DPWM;
- output stage;
- output filter;

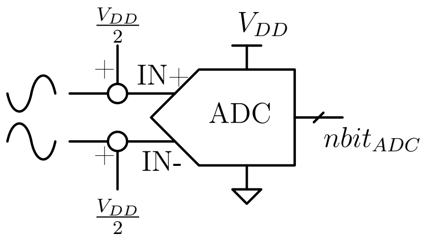
The model of final stage is the same shown in 2.1.4; the compensator is modeled with its digital transfer function using a sampling time $fclk_{SYS}$, the same of ADC.

3.4.1 Analog to digital converter

The aim of this block is to convert the analog signal into a digital one. Depending on the considered topology (open- or closed-loop) the ADC samples the input signal or the error one. Independently on the type of ADC chosen for the physical implementation (e.g. Sigma Delta, SAR, Flash), the effect of this block is to apply a *quantization* on the sampled signal that depends on the number of bits. In particular, the characteristic can be represented as in figure 3.6.



(a) *Single-ended input ADC*



(b) *Differential input ADC*

Figure 3.5: ADC models

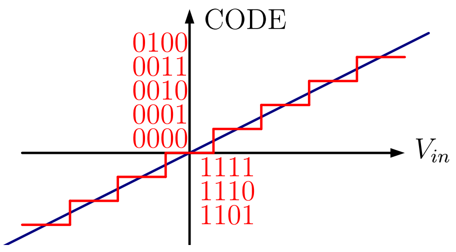


Figure 3.6: ADC characteristic

It can be seen that the representation of the numbers is considered in 2's complement. This means that the output dynamic of the generated code is

$$[-2^{nbit_{ADC-1}}, 2^{nbit_{ADC-1}} - 1] \tag{3.9}$$

where $nbit_{ADC}$ is the number of bit of ADC. Considering a positive voltage power supply V_{DD} and a differential ADC, the differential input voltage can assume values in the range

$$V_d = V_+ - V_- \in [-V_{DD}, V_{DD}] \tag{3.10}$$

with V_+ and V_- are respectively the positive and negative input of ADC.
So the transfer function of the ADC (neglecting the error offset, quantization error and other non-linearities) can be considered as a gain function, that is the angular

coefficient G_{ADC} of the blue line in figure 3.6. Mathematically

$$G_{ADC} = \frac{2^{nbit_{ADC}-1} - 1 + 2^{nbit_{ADC-1}}}{V_{DD} - (-V_{DD})} \simeq 2 \cdot \frac{2^{nbit_{ADC}-1}}{2V_{DD}}$$
$$\Rightarrow \boxed{G_{ADC} = \frac{2^{nbit_{ADC}}}{2V_{DD}}} \tag{3.11}$$

To take into account the sampling, a *zero-order hold* filter that works at $fclk_{SYS}$ is implemented.

3.4.2 Output filter

The analog output filter is implemented as a maximum-flat second-order filter with a double pole at 20 kHz. Its transfer function $F(s)$ is the following

$$F(s) = \frac{1}{1 + \frac{s}{2\pi f_{pf}}} \tag{3.12}$$

where $f_{pf} = 20\,\text{kHz}$ is the pole of the filter.

3.4.3 DPWM

The implementation of DPWM is based on a *free-running counter* that counts from 0 to $2^{nbit_{DPWM}} - 1$, where $nbit_{DPWM}$ is the number of bits of the counter (see figure 2.30). Moreover, the output code from ADC (in case of open-loop system) and from compensator (in case of closed-loop system) could be represented on $nbit > nbit_{DPWM}$; as a consequence, the output signal from ADC/compensator has to be truncated on $nbit_{DPWM}$. The truncation has to be made on the LSBs, so the $nbit_{DPWM}$ MSBs of the output signal from ADC/compensator has to be send to digital modulator. To understand the block that makes the truncation, consider an example of truncation from 4 bits to 2 bits, shown in tables 3.7.
In the table 3.7a can be seen that the truncation consists in taking the N MSBs, where N is the number of bits which represents the truncated code. On the other hand, in the table 3.7b there is the decimal conversion of truncated binary numbers: it can be seen that in decimal representation, the operation of truncation consists in the following operation

$$n_t = \left\lfloor n \cdot \frac{2^{N_t}}{2^N} \right\rfloor \tag{3.13}$$

where N is the number of bits on which the number to truncate n is represented, N_t the number of bits on which the truncated number n_t is represented. To show an example taken from the tables, let's consider the following one

- $(N=4)\ \ 0101 \rightarrow 5$

  $(N=3)\ \ \triangleright 010 \rightarrow 2 = \left\lfloor 5\dfrac{2^3}{2^4} \right\rfloor = \left\lfloor 5/2 \right\rfloor = \left\lfloor 2.5 \right\rfloor$

  $(N=2)\ \ \triangleright 01 \rightarrow 1 = \left\lfloor 5\dfrac{2^2}{2^4} \right\rfloor = \left\lfloor 5/4 \right\rfloor = \left\lfloor 1.25 \right\rfloor$

In practice, the truncation block can be modelled has a gain block with a gain of $\frac{2^M}{2^N}$, where N and M are respectively the number of bits of the input and output code of truncation gain block and $M < N$. Similarly to the analog PWM gain ($1/V_{DD}$, as shown in 2.1.4), DPWM gain G_M is related to the numeric dynamic of the counter

$$G_M = \frac{1}{2^{nbit_{PWM}}} \tag{3.14}$$

N=4	N=3	N=2
0000	000	00
0001		
0010	001	
0011		
0100	010	01
0101		
0110	011	
0111		
1000	100	10
1001		
1010	101	
1011		
1100	110	11
1101		
1110	111	
1111		

(a) *Binary representation of truncated code*

N=4	N=3	N=2
0	0	0
1		
2	1	
3		
4	2	1
5		
6	3	
7		
8	4	2
9		
10	5	
11		
12	6	3
13		
14	7	
15		

(b) *Decimal conversion of truncated number*

Figure 3.7: Example of truncation from 4 to 3 and 2 bits

3.4.4 Proposed block diagram for class D amplifier

The overall block diagram of open- and closed-loop class D amplifier are shown in figures 3.8a and 3.8b.
It can be seen that in closed-loop system an anti-aliasing filter and a feedback attenuation are added. In particular

- *feedback attenuation* is necessary because the output voltage of the output stage is a PWM signal that can assume two values: V_{PS} and 0 V. This signal has to be conditioned for the input dynamic range of the differential ADC.

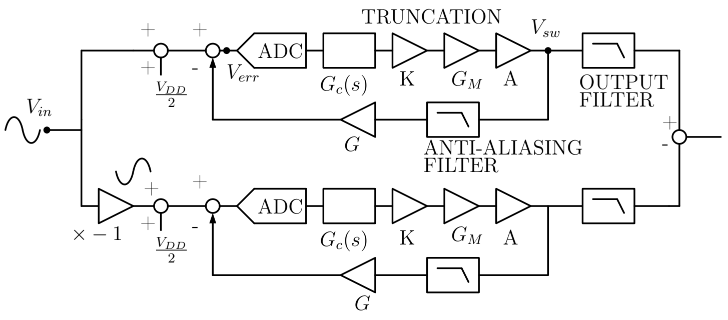
Ideally, the maximum input voltage that can be applied to IN+ and IN- of ADC is V_{DD}, as shown in figure 3.5. Because of $V_{DD} = 5\,\mathrm{V} = \frac{V_{PS}}{2} = \frac{10\,\mathrm{V}}{2}$, the attenuation gain G has to be $G = \frac{1}{2}$;

- the anti-aliasing filter is added to remove the effect of higher harmonics of PWM that cannot be correctly sampled by ADC, in particular the ones larger than $\sim \frac{f_{clkSYS}}{2} = \frac{16\,\mathrm{MHz}}{2} = 8\,\mathrm{MHz}$; the frequency of lower pole is set to $f_p = 225\,\mathrm{kHz}$;
- the truncation gain is the block K.

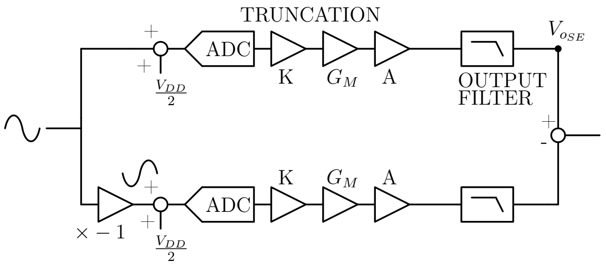
3.4.5 Open-loop gain and binary representation of ADC code

Considering the models of each block of the open-loop system in figure 3.8b, the overall gain related to the single-ended output can be computed as

$$\frac{V_{oSE}}{V_{in}} = G_{ADC}KG_MA = \frac{2^{nbit_{ADC}}}{2V_{DD}}\frac{2^{nbit_{PWM}}}{2^{nbit_{ADC}}}\frac{1}{2^{nbit_{PWM}}}V_{PS} = \frac{V_{PS}}{2V_{DD}} \neq \frac{V_{PS}}{V_{DD}} \tag{3.15}$$



(a) *Closed-loop block diagram*



(b) *Open-loop block diagram*

Figure 3.8: Closed- and open-loop models

This added terms 2 is due to the 2's complement representation of the output code of ADC. Considering that the input signal of ADC in always positive, only $2^{nbit_{ADC}-1}$ codes are used for the representation of the positive dynamic range, while the others $2^{nbit_{ADC}-1}$ are used for the negative dynamic of the signal. As a consequence, the effective number of bits used for the conversion of the input signal are $nbit_{ADC}-1$. Consequently, the truncation K factor has to become $K=\frac{2^{nbit_{PWM}}}{2^{nbit_{ADC}-1}}$. With this correction the overall gain becomes the one expected

$$\frac{V_{o_{SE}}}{V_{in}} = G_{ADC}KG_MA = \frac{2^{nbit_{ADC}}}{2V_{DD}}\frac{2^{nbit_{PWM}}}{2^{nbit_{ADC}-1}}\frac{1}{2^{nbit_{PWM}}}V_{PS} = \frac{V_{PS}}{V_{DD}} \tag{3.16}$$

3.5 Closed-loop design methodology

Once defined the model of each block in the closed- and open-loop systems, the *compensator transfer function* has to be designed. The first aim of the compensator transfer function is to obtain null error after the transient (fig. 3.8a)

$$\frac{V_{err}}{V_{in}} = \frac{1}{1+G_c(s)G_MAKG_{ADC}}$$
$$\rightarrow \lim_{s\to 0} sV_{err}(s) = \lim_{s\to 0} s\frac{1}{s}\frac{1}{1+G_c(s)G_MAKG_{ADC}} = 0$$
$$\Rightarrow G_c(s) = \frac{1}{s^n}$$

A *first order integrator* ($n=1$) has strong trade-off between unit gain frequency (UGF or ω_u) and switching frequency (f_{sw}). In particular

- if UGF is reduced, THD increases (more power for higher harmonics in band) because the integrator effect happens for narrower frequency range;
- if UGF is increased, IMD increases because of the attenuation of f_{sw} tone, its harmonics and related side-bands (due to intermodulation with input tone) are less attenuated[5]; this intermodulation in band increases the THD;

As a consequence, the switching frequency should be increased to allow larger UGF and higher linearity; the drawbacks are that

- in analog design, increasing switching frequency is not so easy due to limitation of oscillators;
- higher switching frequency means larger power dissipation;

To improve the performance of the integrator and overcome the trade-off between switching frequency and UGF, the integrator order can be increased. This choice requires to add a zero to reach the correct stability; moreover, increasing integrator order means increasing overshoot and settling time. In particular, larger overshoot can bring problems about the linearity of the system: if overshoot is too large, it can exceed the maximum value that command (the compensator output) can reach, triggering an instability. This is the main problem of designing closed-loop system with output filter in the loop.

3.5.1 Loop compensator filter design without output filter in the loop

Referring to the methodology design shown in [5], the compensator transfer function of the system in figure 3.8a is designed by *shaping* the loop transfer function, starting from the *non-compensated loop* transfer function $T_{NC}(j\omega)$ that is given by

$$T_{NC}(j\omega) = G_{ADC}KG_MAG\frac{1}{1+j\frac{\omega}{\omega_{ph}}} \tag{3.17}$$

where $\omega_{ph} > \omega_u$ is the pole of the anti-aliasing filter. Because the goal is to obtain an integrator transfer function, the idea is to recover the transfer function of compensator $C(j\omega)$ knowing the desired loop transfer function $T(j\omega)$. Mathematically speaking this means

$$C(j\omega) = \frac{T(j\omega)}{T_{NC}(j\omega)} \tag{3.18}$$

This is shown graphically in figure 3.9 where the non-compensated loop, compensator and desired loop transfer functions are shown: multiplying the red and black curve the blue loop transfer function is obtained.

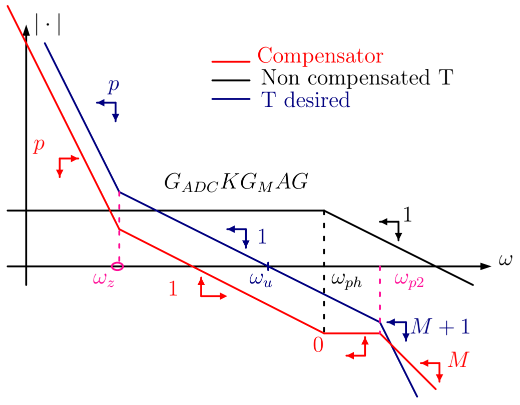


Figure 3.9: Loop, compensator and non-compensated loop transfer functions

Some observations and considerations about this graph:

- the slopes indicated are considered as $\pm n \cdot 20$dB/dec, where n is the read number and $\pm$ depends on the positivity of the slope. For example, on T *desired* transfer function, the p slope has to be read $-p \cdot 20$dB/dec;
- ω_z is the zero to add at the compensator to obtain the correct stability and in particular is related to the phase margin that one wants to obtain;
- because of the order of compensator is p and the transfer function has to cross the 0 dB level with a slope of -20 dB/dec, the order of the zero has to be $p-1$. For example, if $p = 3$ (-60 dB/dec), two zeros are needed: in fact, -60 dB/dec $+ 40$ dB/dec $= -20$ dB/dec;

- the pole of anti-aliasing filter is compensated by the compensator transfer function; this brings an advantage in terms of stability because this pole has no effect on the phase, but increases the IMD due to the f_{sw} carrier, harmonics and side-bands tones; to reduce the effect of this drawback an M-th order high frequency pole is added to the compensator.

The loop transfer function with the compensator can be written in this way

$$T(j\omega) = G_{ADC}G_M AGK \underbrace{\frac{1}{1+\frac{s}{\omega_{ph}}}}_{F_{AA}(s)} \underbrace{\frac{(1+\frac{s}{\omega_{ph}})(1+\frac{s}{\omega_z})^{(p-1)}}{(\tau_p s)^p} \frac{1}{(1+\frac{s}{\omega_{p2}})^M}}_{C(s)} \underbrace{\frac{1}{1+s\frac{T}{2}}}_{\text{ZOH register}} = \tag{3.19}$$

$$= G_{ADC}G_M AGK \frac{(1+\frac{s}{\omega_z})^{(p-1)}}{(\tau_p s)^p} \frac{1}{(1+\frac{s}{\omega_{p2}})^M} \frac{1}{1+s\frac{T}{2}} \tag{3.20}$$

where $F_{AA}(s)$ is the transfer function of anti-aliasing filter. From this equation it can be inferred that the compensation of the anti-aliasing pole avoids the impact of this pole on the compensator design, in particular on stability and integrator gain. Moreover, the *ZOH register* transfer function models the down-sampling before the PWM modulator that allows the *Uniform PWM* modulation; $T = \frac{1}{f_{sw}}$.
The first parameter to design is the *crossover frequency* ω_u. According to the literature [4][5][14] and considering the trade-off between THD and IMD

$$\omega_u = \frac{1}{10}\omega_{sw} \tag{3.21}$$

where $\omega_{sw} = 2\pi f_{sw}$. Once this frequency has been chosen, the position of the zero is decided in order to obtain the desired *phase margin PM*. Mathematically, neglecting the high frequency pole ω_{p2} of the compensator

$$PM = 180^\circ + \angle T(j\omega_u) = 180^\circ - p90^\circ + (p-1)\tan^{-1}\frac{\omega_u}{\omega_z} - \underbrace{\tan^{-1}\frac{\omega_u}{2f_{sw}}}_{\substack{\text{phase contribution}\\ \text{due to ZOH REG}}}$$

$$\rightarrow (p-1)\tan^{-1}\frac{\omega_u}{\omega_z} = PM - 180^\circ + p90^\circ + \tan^{-1}\frac{\omega_u}{2f_{sw}}$$

$$\rightarrow \tan^{-1}\frac{\omega_u}{\omega_z} = \frac{PM - 180^\circ + p90^\circ + \tan^{-1}\frac{\omega_u}{2f_{sw}}}{p-1}$$

$$\Rightarrow \boxed{\omega_z = \frac{\omega_u}{\tan\left(\frac{PM-180^\circ+p90^\circ+\tan^{-1}\frac{\omega_u}{2f_{sw}}}{p-1}\right)}} \tag{3.22}$$

It can be observed that increasing the integrator order p, ω_z decreases. This means that at increase of compensator order, the zero tends to be inside the audio band, reducing the p order integrator effect to first order inner in audio band. This means that for high frequency audio range, the effect of integrator is of the first order and not of the p order; as a consequence, for this range of frequency, THD and SNR are worsened. To better explain this fact, a qualitative graph is shown in figure 3.10. The integrator time constant τ_p can be found by the following calculation, observing figure 3.9.

$$\frac{|T(j\omega_z)|}{1} = \frac{\omega_u}{\omega_z} \Rightarrow |T(j\omega_z)| = \frac{\omega_u}{\omega_z}$$
$$\Rightarrow |C(j\omega_z)| = \frac{|T(j\omega_z)|}{G_{ADC}KG_MAG} = \frac{\omega_u}{\omega_z}\frac{1}{G_{ADC}KG_MAG}$$

$$\frac{|C(j\omega_z)|}{1} = \left(\frac{\frac{1}{\tau_p}}{\omega_z}\right)^p \Rightarrow |C(j\omega_z)|\omega_z^p = \left(\frac{1}{\tau_p}\right)^p \rightarrow |C(j\omega_z)|^{\frac{1}{p}}\omega_z = \frac{1}{\tau_p}$$

$$\rightarrow \left(\frac{\omega_u}{\omega_z}\frac{1}{G_{ADC}KG_MAG}\right)^{\frac{1}{p}}\omega_z = \frac{1}{\tau_p} \Rightarrow \boxed{\tau_p = \left(\frac{G_{ADC}KG_MAG}{\omega_u}\right)^{\frac{1}{p}}\frac{1}{\omega_z^{1-\frac{1}{p}}}} \tag{3.23}$$

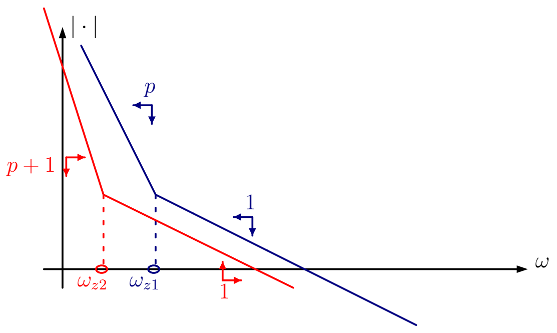


Figure 3.10: Qualitative loop transfer function at varying of p

where

$$G_{ADC}KG_MAG = \frac{2^{nbit_{ADC}}}{V_{DD}}\frac{2^{nbit_{PWM}}}{2^{nbit_{ADC}}}\frac{1}{2^{nbit_{PWM}}}V_{PS}\frac{1}{2} = \frac{V_{PS}}{V_{DD}}\frac{1}{2} \tag{3.24}$$

From equation 3.24 can be seen that the integrator time constant does not depends on the number of bits. As a consequence, varying the number of bits of DPWM the performances of the compensator do not change: it makes sense, because the performances of compensator have to be related only to the switching frequency (that is the reference to choice the crossover frequency ω_u) and to the gain of the modulator and output stage.

3.5.2 Zero-order hold model and PWM spectrum

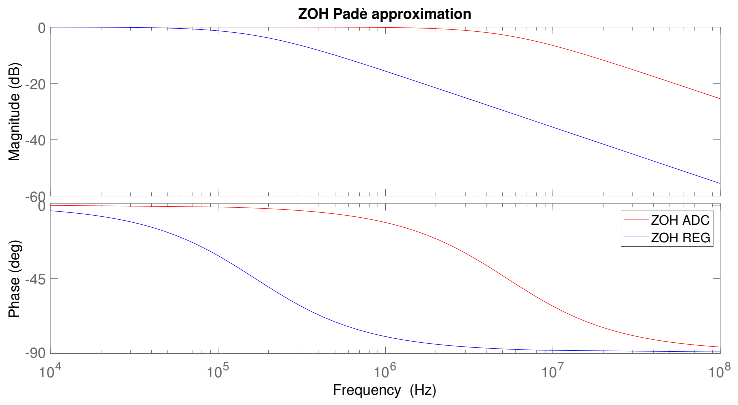
The effect of *zero-order hold* (*ZOH*) filter models the sampling effect of

1. ADC;
2. Uniform sampling of DPWM;
3. controller transfer function discretization[34].

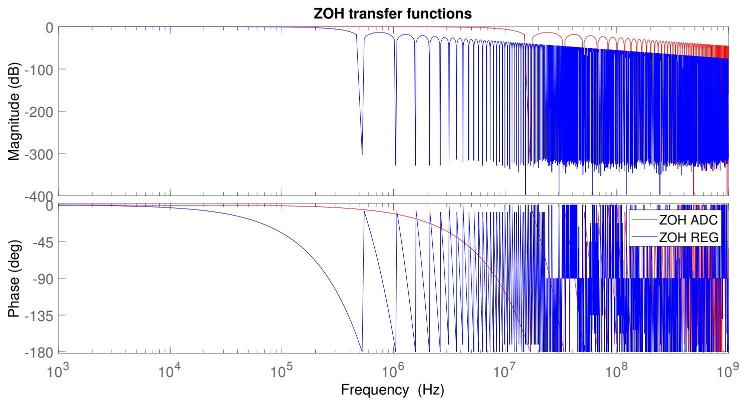
As shown in [35], the transfer function of the ZOH is given by

$$ZOH(s) = \frac{1}{T}\frac{1-e^{-Ts}}{s} \underbrace{\simeq}_{\substack{\textit{Pade'}\\ \textit{approximation}\\ \textit{[36]}}} \frac{1}{T}\frac{T}{1+s\frac{T}{2}} = \frac{1}{1+s\frac{T}{2}} \tag{3.25}$$

where T is the sampling period of the considered block, in particular for ADC and compensator $T = \frac{1}{f_{clkSYS}}$ and for DPWM uniform sampling $T = \frac{1}{f_{sw}}$. In practical case, the ZOH for uniform sampling of DPWM can be implemented by mean of a register that works at f_{sw}.



(a) *Padè approximation ZOH transfer functions*



(b) *ZOH transfer function*

Figure 3.11: ZOH transfer functions; ZOH REG is referred to DPWM register ($f_{sw} \sim 500\,\text{kHz}$)

The Padè approximation is necessary because the non-approximated transfer function is not linear, so it cannot be taken into account in the loop design. From previous equation can be observed that

- below the sample frequency the gain is constant;
- the phase starts to decrease at $\sim \frac{1}{100}\frac{1}{T}$.
- ZOHs influence the phase of the system, not the gain; in particular, ZOH that models the input register of DPWM should be considered for stability and phase margin;

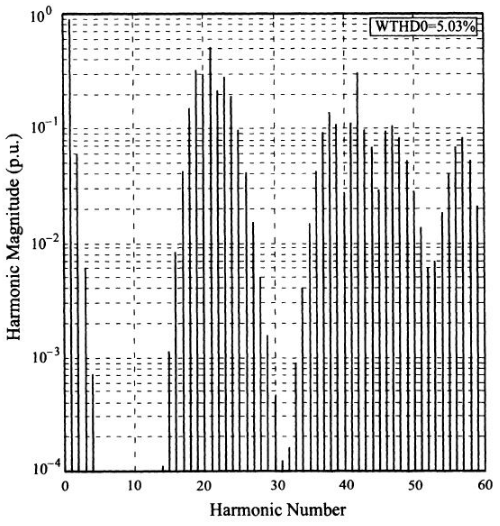


Figure 3.12: Uniform PWM spectra generated with trailing edge triangular waveform[37]

ZOH that models the ADC has no substantial effect on phase margin and loop gain in audio band.

A graphical representation of the transfer functions is shown in figure 3.11. This is the reason why in the design it is considered also the effect of DPWM ZOH: to recover the phase which is lost by the sampling and holding.
Moreover, observing the transfer function in figure 3.11b, the effect of uniform sampling can be better understood: ZOH filter has notch zeros with infinite attenuation at the switching frequency and its harmonics; this is a strong advantage in closed-loop system. Let's consider the system in figure 3.8a: the PWM signal generated by the output stage is filtered by the anti-aliasing filter, then the filtered signal is sampled by the ADC, after is filtered by the compensator and then down-sampled by DPWM. The command signal generated by the compensator has side-bands and residual PWM harmonics; thanks to ZOH, all the PWM harmonics of the command

are infinitely attenuated. Because of the Nyquist theorem, the maximum frequency that can be correctly sampled by DPWM is $f_{sw}/2$: as a consequence, the residual side-bands are aliased in audio band, increasing harmonic distortion, but thanks to ZOH (and so, uniform sampling) the harmonics of PWM are eliminated before the PWM sampling of the command, avoiding their aliasing in audio band.
The spectrum of uniform sampling PWM with trailing edge triangular waveform is shown in figure 3.12, where the side-bands due to the intermodulation with the input tones are generated and could be aliased in audio band when re-sampled by PWM in a closed-loop system. Once the phase margin has been designed, the integrator time constant and the stabilization zeros are computed by implementing the equations 3.23 and 3.22. What is missing are

- high frequency pole of the compensator;
- consideration on the aliasing due to harmonics of PWM sampled by ADC.

High frequency poles The high frequency pole (figure 3.9) is chosen to be of the second order ($M = 2$) with an angular frequency of

$$\omega_{p2} = 9\omega_u \tag{3.26}$$

This pole is not considered during the design of the stabilization zero: this means that it reduces the phase margin, obtaining $PM^* < PM = 60°$, where PM^* is the effective obtained phase margin. The aim of this pole is to reduce the aliasing effect due to the PWM sidebands, before the command is sampled by DPWM (see 3.5.2). Higher attenuation means reducing the stability of the overall system because ω_{p2} should be reduced.

PWM aliasing due to ADC sampling The ADC samples a PWM signal filtered by a low pass filter. In particular, the maximum frequency components that ADC can correctly sample is $f_{clkSYS}/2$, the Nyquist frequency. This means that this component and all the higher ones of PWM signal, have to be characterized by a level which must be under the quantization noise of the ADC. Considering figure 3.8a, an additional pole has to be added to the anti-aliasing filter. The spectrum of PWM signal is very complex because depends on Bessel functions, modulation index and, as a consequence, on the input signal amplitude. To roughly estimate the second pole it is assumed that the PWM is a square wave. The Fourier series of a square wave can be written as

$$V_{PWM} = \frac{4}{\pi} \sum_{k=1,3,5\ldots}^{\infty} \frac{1}{k} \sin 2\pi n f \tag{3.27}$$

From this equation, can be seen that only odd components are presents in square-wave spectrum and they decrease with a factor of $\frac{4}{\pi k}$ where k is the k-th odd harmonic. This means that given a sampling frequency f_{clkSYS} of ADC and a switching sampling frequency f_{sw} the harmonic to consider is

$$k = \frac{\frac{f_{clkSYS}}{2}}{f_{sw}} \tag{3.28}$$

As a consequence, the second pole of anti-aliasing filter can be computed considering the following inequality

$$\frac{4}{\pi k}\frac{V_{PS}}{2}\alpha_c\alpha_p < \frac{V_{DD}}{2^{nbit_{ADC}-1}} \tag{3.29}$$

where

- $\alpha_c = \frac{f_c}{kf_{sw}}$;
- $\alpha_p = \left(\frac{f_p}{kf_{sw}}\right)^q$, with q the degree of the pole;
- $\frac{V_{PS}}{2} = V_{DD} = 5\,\mathrm{V}$;

Solving the equation 3.29 for f_p with $q = 2$ can be obtained

$$\frac{4}{\pi k}\frac{V_{PS}}{2}\frac{f_c}{kf_{sw}}\left(\frac{f_p}{kf_{sw}}\right)^2 < \frac{V_{DD}}{2^{nbit_{ADC}-1}}$$
$$\frac{4}{\pi k}\frac{f_p^2}{k^3 f_{sw}^3} < \frac{2}{2^{nbit_{ADC}}}$$
$$f_p < k^2 f_{sw}\sqrt{\frac{\pi f_{sw}}{2f_c 2^{nbit_{ADC}}}} \tag{3.30}$$

where $k = 9$ for $f_{sw} = 1\,\mathrm{MHz}$, $k = 17$ for $f_{sw} = 500\,\mathrm{kHz}$, $V_{PS}/2 = V_{DD} = 5\,\mathrm{V}$ and $f_c = 225\,\mathrm{kHz}$ is the first pole of anti-aliasing filter. Placing these parameters in equation 3.30 can be obtained

- $f_{sw} = 500\,\mathrm{kHz} \rightarrow$
$$f_p < 17^2 \cdot 500\,\mathrm{kHz} \cdot \sqrt{\frac{\pi \cdot 500\,\mathrm{kHz}}{2 \cdot 225\,\mathrm{kHz} \cdot 2^{16}}} \simeq 1.05\,\mathrm{MHz}$$
- $f_{sw} = 1\,\mathrm{MHz} \rightarrow$
$$f_p < 9^2 \cdot 1\,\mathrm{MHz} \cdot \sqrt{\frac{\pi \cdot 1\,\mathrm{MHz}}{2 \cdot 225\,\mathrm{kHz} \cdot 2^{16}}} \simeq 836\,\mathrm{kHz}$$

Because of the estimation with square wave assummption is too conservative, the estimated harmonics are higher than the real one of PWM and a pole of 1 MHz is sufficient to attenuate 8-th harmonic of PWM signal at $f_{sw} \simeq 1\,\mathrm{MHz}$.
This filter has another effect on the aliasing and distortion in band: it attenuates the side-bands due to intermodulation with input tone, reducing the aliasing effect

due to the PWM sampling of command. For this reason f_p is designed $f_p = 500\,\text{kHz}$ for $f_{sw} \simeq 500\,\text{kHz}$ and $f_p = 1\,\text{MHz}$ for $f_{sw} = 1\,\text{MHz}$. In this way, the results are comparable because in both cases the first harmonic of PWM over Nyquist frequency of ADC ($f_{clk_{SYS}}/2$) is under quantization noise and the most problematic side-bands related to the fundamental of PWM (f_{sw}) are attenuated of the same amount by the filter.

3.5.3 Design of loop compensator filter with output filter in the loop

Differently from previous design where the output filter is left out from the loop, another tested design consists in leaving the output filter inside the loop. Theoretically, this solution has better performances in terms of SNR and THD[5], but it has problems related to the dynamic of the command.
The adopted design methodology is the same used for the solution without output filter in the loop: the compensator transfer function is designed by starting from non compensated transfer function and the loop transfer function desired. The reference block diagram is shown in figure 3.13.
The transfer functions in figure 3.14 are parametrized as in figure 3.9. Can be observed that

- the double pole of the output filter ω_{pf} is compensated with a double zero by the compensator; the exact zero-pole cancellation is not necessarly required, but due to the stability of the overall system compensator zeros have to be placed in proximity of the output filter poles;
- ω_z is necessary to obtain the correct phase margin, as explained for the previous design;
- the transfer function of compensator results with a p-order pole in the origin, a zero of order 2 and a zero of order $p-1$; as a consequence there are p poles and $p+1$ zeros: the transfer function is improper and not feasible; at least one extra pole at high frequency is needed to make the transfer function proper.

The designed compensator transfer function, considering the high frequency extra poles, is the one shown in figure 3.15. Depending on the order of poles ω_{ph1} and ω_{ph2}, the slopes of the last two segments of transfer function change. In particular:

- ω_{ph1}, M-th order pole $\Rightarrow$ slope $= (M-1)\cdot 20\,\text{dB/dec}$;
- ω_{ph2}, N-th order pole $\Rightarrow$ slope $= (M+N-1)\cdot 20\,\text{dB/dec}$;

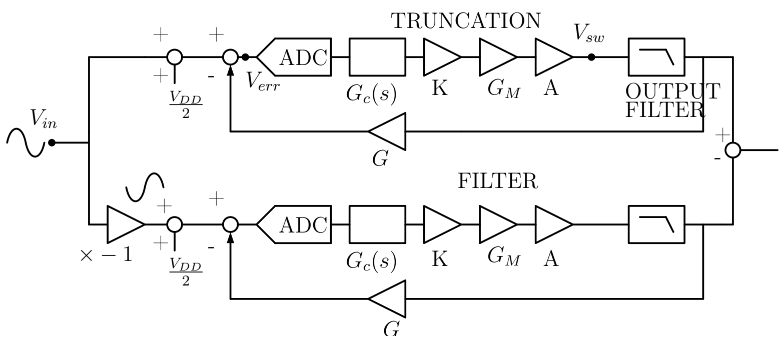


Figure 3.13: Block diagram of closed-loop system with output filter in the loop

The position and the order of the poles influence different parameters of the system

- if the order of the poles is high and/or the frequencies of the poles are too low, they reduce the phase margin, leading to instability of control loop: in fact, the effect on the phase due to the poles, starts to act more than one decade before pole and this effect is much strong at increasing of the order pole. Schematically

$$\text{if } \omega_{ph1}(\omega_{ph2}) \downarrow \Rightarrow PM \downarrow$$
$$\text{if } M(N) \uparrow \Rightarrow PM \downarrow$$

- if the poles are placed at too high frequencies, the compensator band increases due to the effect of the zeros; as a consequence, the overshoot of the compensator increases, exceeding the dynamic range.

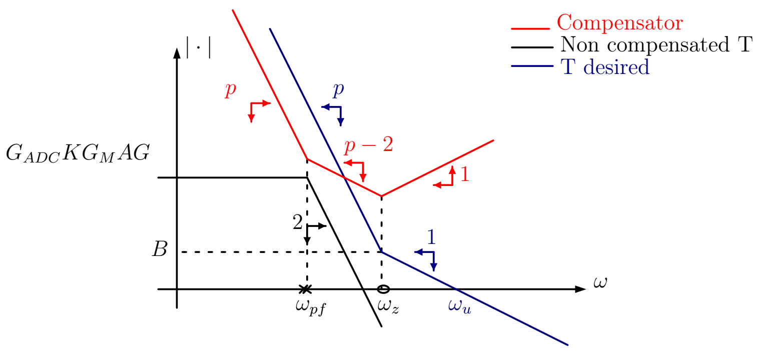


Figure 3.14: Compensator, loop non-compensated and loop compensated transfer functions with output filter in feedback path

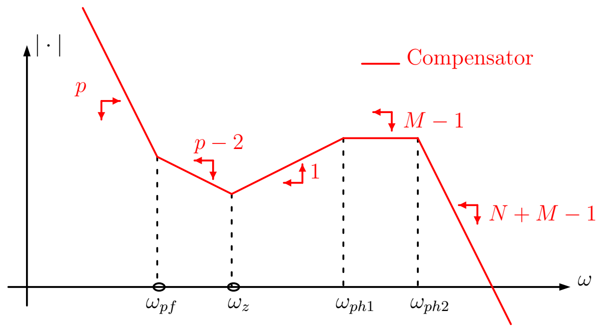


Figure 3.15: Compensator transfer function with high frequency poles

Referring to figure 3.14, neglecting the ZOH of ADC that models the sampling event, the loop transfer function $T(j\omega)$ is

$$T(s) = \underbrace{G_{ADC}KG_MAG\frac{1}{(1+s/\omega_{pf})^2}}_{F(s)}\underbrace{(1+s/\omega_{pf})^2\frac{(1+s/\omega_z)^{(p-1)}}{(\tau_p s)^p}}_{C(s)}\underbrace{\frac{1}{1+s\frac{T}{2}}}_{\text{ZOH register}} \tag{3.31}$$

The compensator zero ω_z should be placed to obtain the properly phase margin PM

$$PM = 180° + \angle T(j\omega_u) = 180° - p90° + (p-1)\tan^{-1}\frac{\omega_u}{\omega_z} - \underbrace{\tan^{-1}\frac{\omega_u}{2f_{sw}}}_{\substack{\text{phase contribution}\\ \text{due to ZOH REG}}}$$

$$\rightarrow (p-1)\tan^{-1}\frac{\omega_u}{\omega_z} = PM - 180° + p90° + \tan^{-1}\frac{\omega_u}{2f_{sw}}$$

$$\rightarrow \tan^{-1}\frac{\omega_u}{\omega_z} = \frac{PM - 180° + p90° + \tan^{-1}\frac{\omega_u}{2f_{sw}}}{p-1}$$

$$\Rightarrow \boxed{\omega_z = \frac{\omega_u}{\tan\left(\frac{PM-180°+p90°+\tan^{-1}\frac{\omega_u}{2f_{sw}}}{p-1}\right)}} \tag{3.32}$$

where $\boxed{\omega_u = \frac{1}{10}2\pi f_{sw}}$ is the crossover frequency.
Now, τ_p has to be properly designed. Observing fig. 3.14 can be done the following computations

$$\frac{|F(j\omega_z)|}{G_{ADC}KG_MAG} = \left(\frac{\omega_{pf}}{\omega_z}\right)^2 \Rightarrow |F(j\omega_z)| = G_{ADC}KG_MAG\left(\frac{\omega_{pf}}{\omega_z}\right)^2$$

$$\frac{|T(j\omega_z)|}{1} = \frac{\omega_u}{\omega_z} \Rightarrow |T(j\omega_z)| = \frac{\omega_u}{\omega_z}$$

$$\Rightarrow |C(j\omega_z)| = \frac{|T(j\omega_z)|}{|F(j\omega_z)|} = \frac{\omega_u}{\omega_z}\frac{1}{G_{ADC}KG_MAG}\frac{\omega_z^2}{\omega_{pf}^2} = \frac{\omega_u\omega_z}{\omega_{pf}^2}\frac{1}{G_{ADC}KG_MAG}$$

$$\frac{|C(j\omega_{pf})|}{|C(j\omega_z)|} = \left(\frac{\omega_z}{\omega_{pf}}\right)^{p-2} \Rightarrow |C(j\omega_{pf})| = |C(j\omega_z)|\left(\frac{\omega_z}{\omega_{pf}}\right)^{p-2} =$$

$$= \frac{\omega_u\omega_z}{\omega_{pf}^2}\frac{1}{G_{ADC}KG_MAG}\frac{\omega_z^{p-2}}{\omega_{pf}^{p-2}} = \frac{\omega_u}{G_{ADC}KG_MAG}\frac{\omega_z^{p-1}}{\omega_{pf}^p}$$

$$\frac{|C(j\omega_{pf})|}{1} = \left(\frac{\frac{1}{\tau_p}}{\omega_{pf}}\right)^p \Rightarrow |C(j\omega_{pf})|\omega_{pf}^p = \left(\frac{1}{\tau_p}\right)^p \tag{3.33}$$

$$\rightarrow |C(j\omega_{pf})|^{\frac{1}{p}}\omega_{pf} = \frac{1}{\tau_p}$$

$$\left(\frac{\omega_u}{G_{ADC}KG_MAG}\frac{\omega_z^{p-1}}{\omega_{pf}^p}\right)^{\frac{1}{p}}\omega_{pf} = \frac{1}{\tau_p}$$

$$\Rightarrow \boxed{\tau_p = \left(\frac{G_{ADC}KG_MAG}{\omega_u}\right)^{\frac{1}{p}}\frac{1}{\omega_z^{1-\frac{1}{p}}}} \tag{3.34}$$

Comparing equations 3.23 and 3.34 can be seen that there's no differences between them: this because all the extra poles due to analog filters (respectively the anti-aliasing and the output filter) are compensated with a zero-pole compensation, nullifying their effect on the loop transfer function. Similar considerations, can be formulated about the ω_z (equations 3.32 and 3.22): the phase effect of anti-aliasing filter and output filter poles is nullified by the zero-pole compensation, so they do not appear in the formula of the zero and do not have any effect on system stability. For this design no anti-aliasing filter is added in feedback path because this function is already implemented by the output filter.

3.6 Verification of design strategies

The methodology applied for the two different solutions is implemented in MATLAB (see *Appendix, 5.2.2, 5.2.4, 5.2.1*), verifying that the transfer functions desired are the ones obtained. Once computed the compensator, the sensitivity, input/output and input/compensator output transfer functions, the step response of the command and the output are computed to observe the transient and the difference between analog and digital compensator response. Then, noises derived from quantization of ADC and DPWM are computed, evaluating even the effect of the closed-loop on the reduction of the DPWM one. Finally, SNR due to quantization noises is computed using trapezoidal numerical integration.

The phase margin PM to obtain a good stability is chosen to be $PM = 60°$; the order of the compensator is chosen $p = 3$. From now on, to simplify the notation, the values of switching frequencies $f_{sw_{GaN}}$ and $f_{sw_{MOS}}$ will be referred to $f_{sw} \sim 500\,\text{kHz}$ and $f_{sw} \sim 1\,\text{MHz}$ respectively. The compensator transfer function is design in s domain and then bilinearly transformed in z domain by means of *c2d* MATLAB function, using *matched* option. A comparison between analog and digital results is made for all the different transfer functions and step responses, to make in evidence that the difference between the two is negligible.

$\mathbf{f_{sw_{GaN}}}$	1 048 576 Hz
$\mathbf{f_{sw_{MOS}}}$	524 288 Hz
#bit ADC	16
$\mathbf{f_{clkSYS}}$	16 777 216 Hz
Triangular waveform	*Trailing Edge*
Output stage power supply ($\mathbf{V_{PS}}$)	10 V
DPWM and ADC power supply ($\mathbf{V_{DD}}$)	5 V
Tone test	1 kHz
Amplitude tone test	2.25 V $(90\% \frac{V_{DD}}{2})$

(a) *Main parameters of simulations*

Compensator	$\tau_p = \left(\frac{G_{ADC}KG_MAG}{\omega_u}\right)^{\frac{1}{p}} \frac{1}{\omega_z^{1-\frac{1}{p}}}$
	$\omega_z = \frac{\omega_u}{\tan\left(\frac{PM-180^\circ+p90^\circ+\tan^{-1}\frac{\omega_u}{2fsw}}{p-1}\right)}$
	$\omega_{p2} = 9\omega_u \ (M = 2)$
	$\omega_u = \frac{1}{10}\omega_{sw}$
Anti-aliasing filter	$f_{ph} = 225\,\text{kHz}$
	$\omega_{ph2} \simeq \omega_{sw} \ (q = 2)$

(b) *Filters parameters in closed-loop system*

Figure 3.16: Parameters of the class D amplifier

3.6.1 Implementation of loop design of the amplifier with anti-aliasing filter in feedback path

The solution with anti-aliasing filter in feedback path (shown in figure 3.8a) is implemented in MATLAB. The main parameters of the closed-loop system are shown in table 3.16a. The input voltage signal is taken at 90% of the maximum possible dynamic to test the amplifier at maximum of its capability, without exiting from the linear condition. The integrator time constant, zeros and poles frequency of compensator and anti-aliasing filter are shown in the table 3.16b.

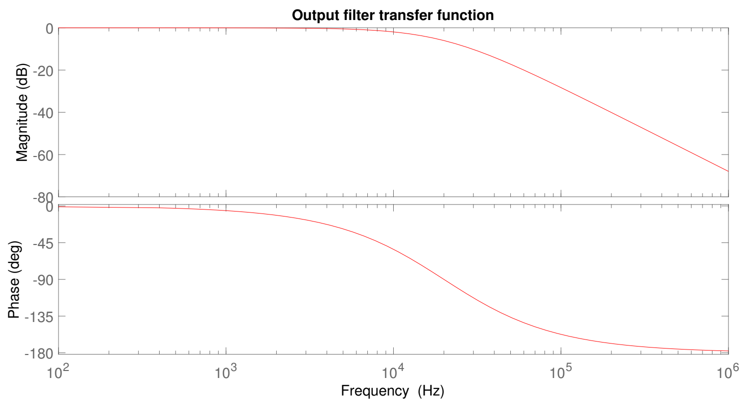


Figure 3.17: Output filter transfer function

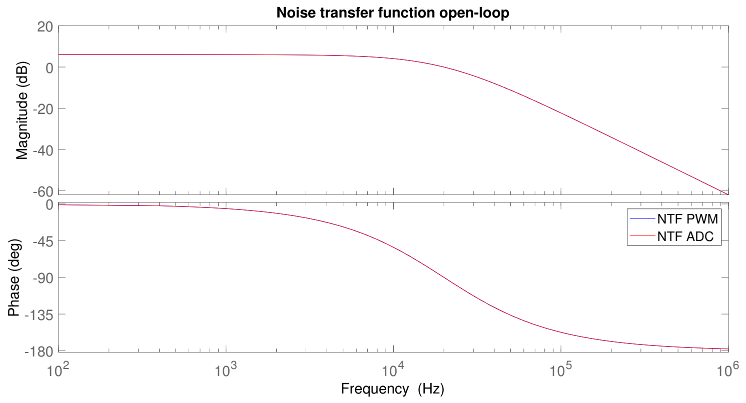
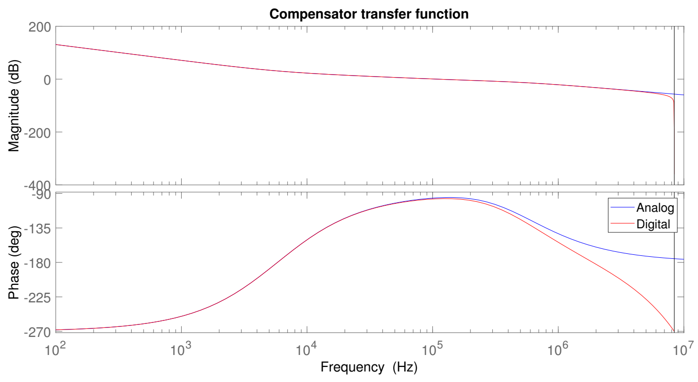


Figure 3.18: Open-loop transfer function (without compensator)

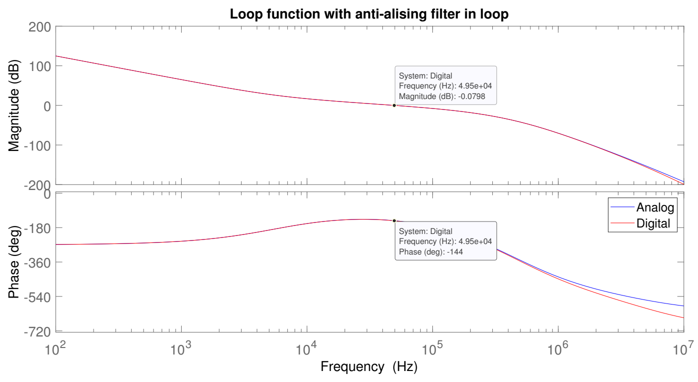
The notation $f_{sw_{GaN}}$ and $f_{sw_{MOS}}$ are related to the fact that the switching frequency are more suitable respectively for GaN and MOS transistors [15], but there are MOS devices that allow to reach $f_{sw} \simeq f_{sw_{GaN}}$[4][16].
In practice, because $p = 3$ and compensator needs $p - 1 = 2$ zeros, they are taken around the designed one ω_z: in this way the phase does not change so abruptly. In particular the zeros are taken

$$\omega_{z1} = \frac{9}{10}\omega_z$$
$$\omega_{z2} = \frac{11}{10}\omega_z$$

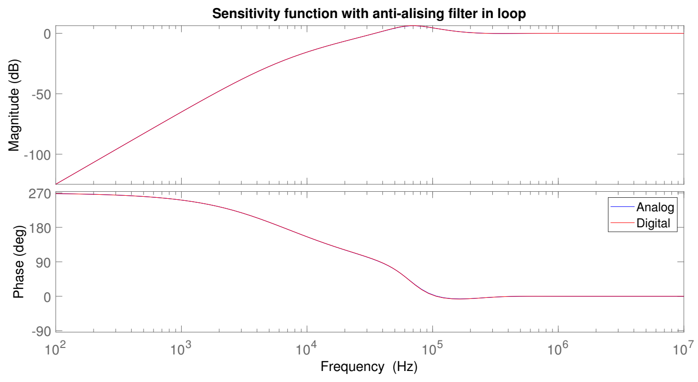
The resulting transfer functions and step responses are shown in the figures 3.19, 3.21 and 3.20, 3.22, both for $f_{sw} \sim 500\,\text{kHz}$ and $f_{sw} \sim 1\,\text{MHz}$. In particular from figures 3.20b and 3.19b can be seen that the crossover frequency is very similar to the one expected ($f_u = \frac{\omega_u}{2\pi} = \frac{1}{10} f_{sw}$); the phase margins result lower than the goal (PM): this is due to the fact that during the design the effect of high frequency poles is neglected; the step responses does not exceed the dynamic range, both for the output and compensator; in fact, the maximum reachable value for the output stage is the power supply (10 V), while for the compensator is $2^{nbit_{ADC}} = 2^{16} = 65536 > 20000$. In table 3.24 are shown the phase margins, crossover frequencies and step response parameters for systems working at both switching frequencies, evaluated respectively with the MATLAB functions *margin* and *stepinfo*. As expected gain margins are equal between them and $f(\angle 180°)$ of system designed with $f_{sw} \sim 1\,\text{MHz}$ is twice the $f(\angle 180°)$ of system working at $f_{sw} \sim 500\,\text{kHz}$. Moreover, also the time responses (figures 3.22, 3.21) are coherent: because of the crossover frequency of the system working at $f_{sw} \sim 1\,\text{MHz}$ is twice the crossover frequency of the system working at $f_{sw} \sim 500\,\text{kHz}$, the rise time of the faster system is half of the slower one; similarly, the settling time of faster system is half of the slower one and the same for the peak time; on the other hand, peak voltages and overshoots are the same.



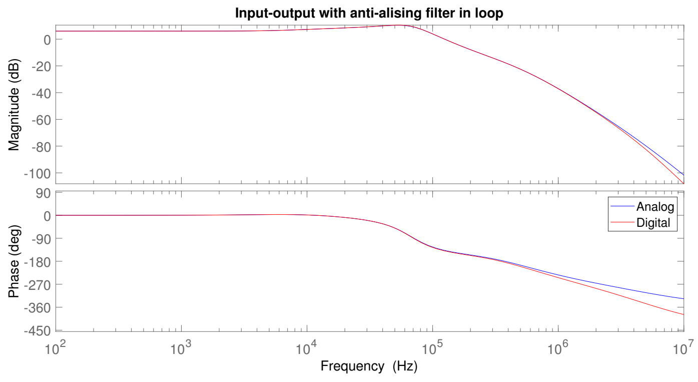
(a) *Compensator transfer function*



(b) *Loop transfer function*

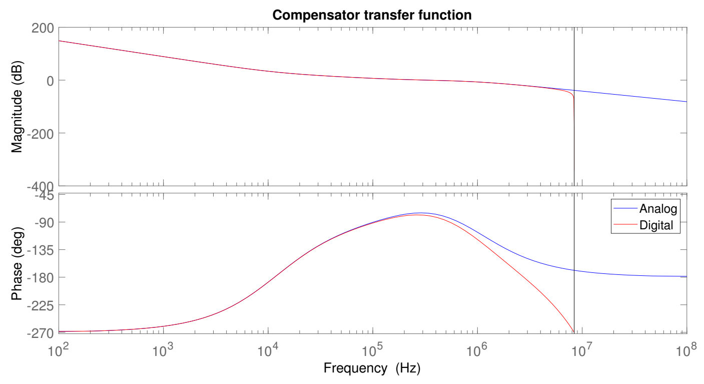


(c) *Sensitivity transfer function*

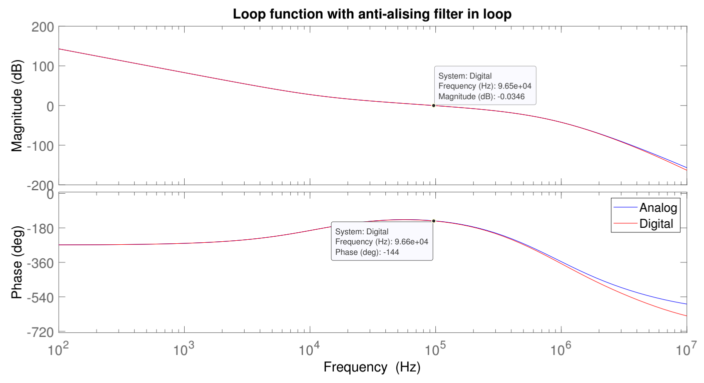


(d) *Input/output transfer function*

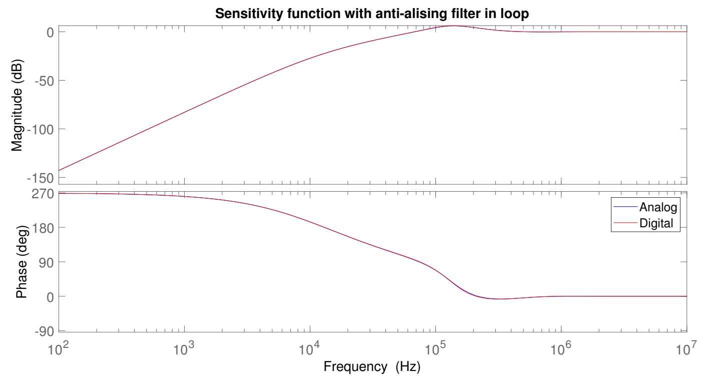
Figure 3.19: Closed-loop transfer functions of the system working at $f_{sw} \sim 500\,\text{kHz}$



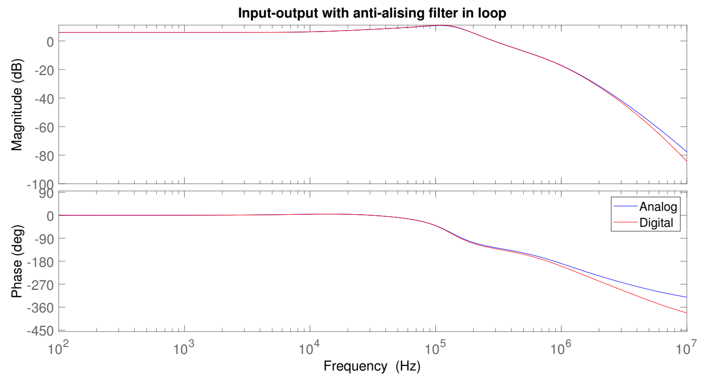
(a) *Compensator transfer function*



(b) *Loop transfer function*

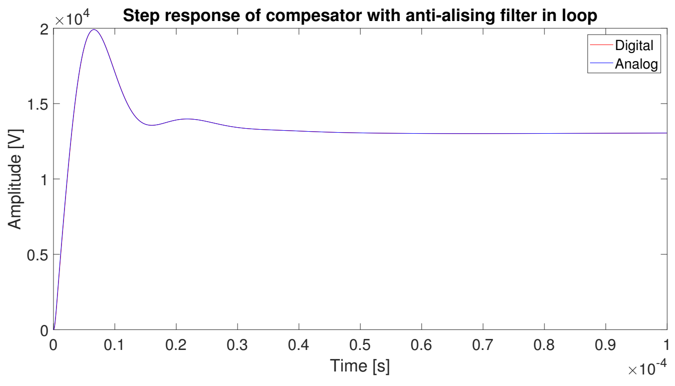


(c) *Sensitivity transfer function*

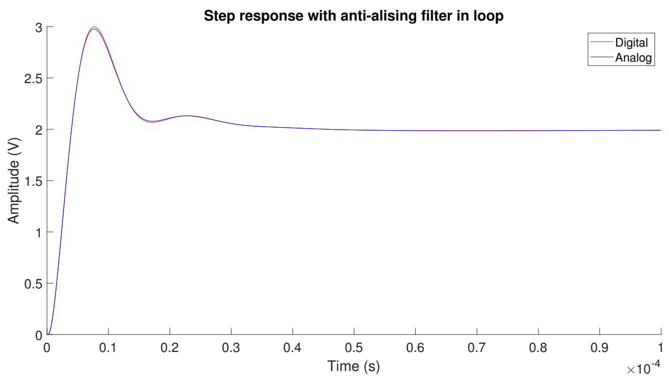


(d) *Input/output transfer function*

Figure 3.20: Closed-loop transfer functions of the system working at $f_{sw} \sim 1\,\text{MHz}$

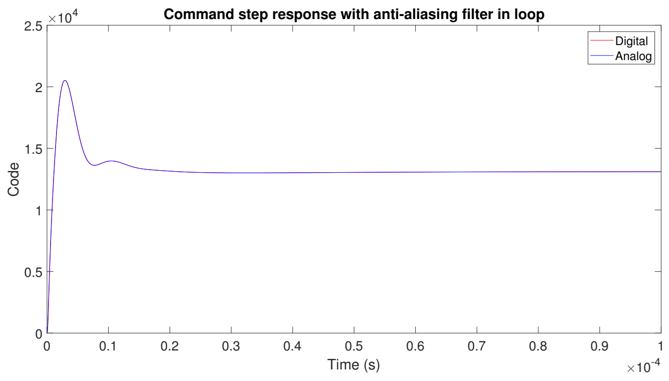


(a) *Step response of compensator*

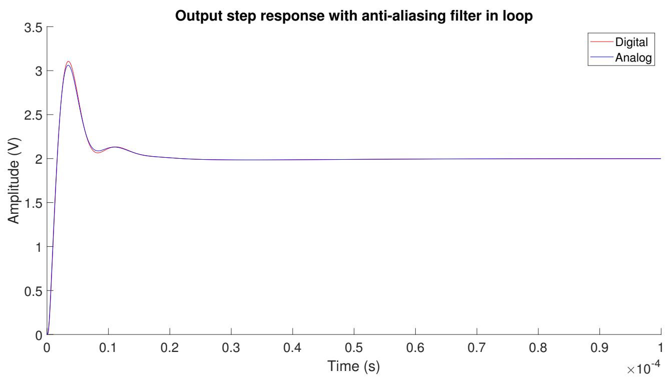


(b) *Step response of system output*

Figure 3.21: Step responses of compensator and system output ($f_{sw} \sim 500\,\mathrm{kHz}$)

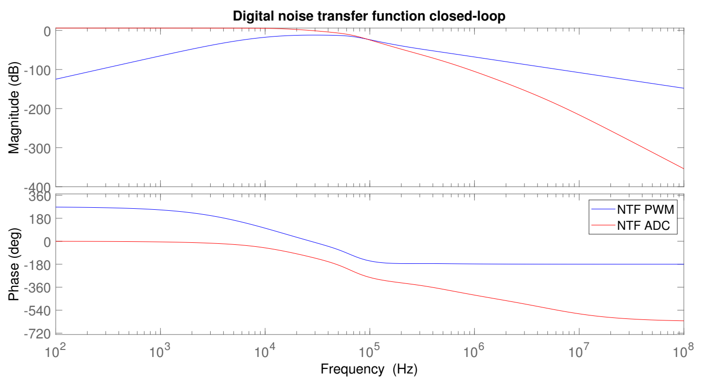


(a) *Step response of compensator*

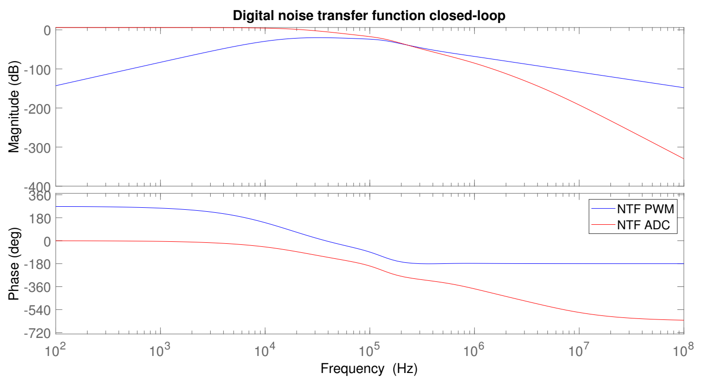


(b) *Step response of system output*

Figure 3.22: Step responses of compensator and system output ($f_{sw} \sim 1\,\mathrm{MHz}$)



(a) *Noise transfer functions of system with $f_{sw} \sim 500\,\mathrm{kHz}$*



(b) *Noise transfer functions of system with $f_{sw} \sim 1\,\mathrm{MHz}$*

Figure 3.23: Noise transfer functions of closed-loop system

	f_{sw}∼500 kHz	**f_{sw}∼1 MHz**
PM	36.2°	36.38°
UGF	49 077 Hz	96 216 Hz
GM (gain margin)	2.7 dB	2.63 dB
f(∠180°)	106 562 Hz	209 246 Hz

(a) *Loop transfer functions parameters*

Step responses system output		
	f_{sw}∼500 kHz	**f_{sw}∼1 MHz**
Rise time	2.6 μs	1.1 μs
Settling time (1%)	33.7 μs	15.6 μs
Overshoot	50.6%	55.34%
Peak	2.997 V	3.106 V
Peak time	7.69 μs	3.52 μs

(b) *Step responses system output*

Step responses compensator output		
	f_{sw}∼500 kHz	**f_{sw}∼1 MHz**
Rise time	2.18 μs	0.82 μs
Settling time (1%)	32.8 μs	15.1 μs
Overshoot	52.6%	56.6%
Peak	19913	20521
Peak time	6.61 μs	2.92 μs

(c) *Step responses compensator*

Figure 3.24: Systems parameters

An observation about the sensitivity and noise transfer functions (figures 3.19c, 3.20c, 3.23b, 3.23a): the positive slope in audio band depends only on the compensator order; this means that the reduction of PWM quantization noise in band depends only on the performances of the integrator: larger the integrator gain in audio band, stronger reduction of PWM quantization noise can be obtained. Anyway, larger integrator gain means larger crossover frequency and, as a consequence, larger switching frequency ($\omega_u = \frac{1}{10}\omega_{sw}$), requiring transistors that are suitable to be driven at high frequency and reducing power efficiency.

3.6.2 Quantization noises

There are two sources of quantization noises: AD converter and DPWM. In particular, considering a uniform distribution of quantization noise, the power noises can be evaluated as

$$\sigma^2_{n_{ADC}} = \frac{(2V_{DD})^2}{12 \cdot 2^{2nbit_{ADC}}} = \frac{4 \cdot 25\,\mathrm{V}^2}{12 \cdot 2^{2 \cdot 16}} \simeq 5.15 \times 10^{-10}\mathrm{V}^2 \Rightarrow \sigma^2_n\Big|_{\mathrm{dB}} \simeq -87\,\mathrm{dB} \quad (3.35)$$

$$\sigma^2_{n_{PWM}} = \frac{V^2_{DD}}{12 \cdot 2^{2 \cdot nbit_{PWM}}} = \frac{25\,\mathrm{V}^2}{12 \cdot 2^{2 \cdot 10}} \simeq 1.98 \cdot 10^{-6}\mathrm{V}^2 \Rightarrow \sigma^2_n\Big|_{\mathrm{dB}} \simeq -57.01\,\mathrm{dB}^1 \quad (3.36)$$

To evaluate what is the most problematic noise, it is needed extract the *power spectral density*, multiply it with the square of the transfer function that describes the noise path and then integrate in audio band. Mathematically

$$P_n = \int_{20\,\mathrm{Hz}}^{20\,\mathrm{kHz}} S_n |H(jf)|^2 df \quad (3.37)$$

where S_n is the considered power spectral density and $|H(jf)|$ the transfer function of noise path. Considering the *noise transfer functions (NTF)* of open-loop system for ADC and DPWM (figure 3.18) can be obtained

$$NTF(s)\Big|_{DPWM} = \frac{V_{PS}}{V_{DD}} \frac{1}{\left(1 + \frac{s}{s_{pf}}\right)} = NTF(s)\Big|_{ADC} \quad (3.38)$$

The power spectral density can be obtained by the following formula

$$S_n = \frac{V^2_{DD}}{6 \cdot 2^{2N} f_s} \quad (3.39)$$

where N and f_s are respectively $nbit_{ADC}$ and f_{clkSYS} if power spectral density of ADC noise is considered or $nbit_{PWM}$ and f_{sw} is DPWM quantization noise is considered. Combining 3.39 and 3.38 in 3.37, the theoretical noises of open-loop system and SNR are evaluated using trapezoidal numerical integration in MATLAB (*trapz*

[1]The best case is with $nbit_{PWM}$=*10*, when DPWM quantization noise is the minimum

#bitPWM	$\mathbf{P_{n_{ADC}}}$	$\mathbf{P_{n_{PWM}}}$
10	-87.12	-57.01
8	-87.12	-44.98
7	-87.12	-38.96
6	-87.12	-32.94
5	-87.12	-26.92

(a) *Quantization noise of ADC and PWM*

#bitPWM	$\mathbf{f_{sw}}\sim$ 500 kHz		$\mathbf{f_{sw}}\sim$ 1 MHz	
	$\mathbf{P_{n_{ADC}}}$	$\mathbf{P_{n_{PWM}}}$	$\mathbf{P_{n_{ADC}}}$	$\mathbf{P_{n_{PWM}}}$
10	-108.32	-84.79	-108.92	-97.71
8	-108.32	-72.75	-108.92	-85.68
7	-108.32	-66.73	-108.92	-79.66
6	-108.32	-60.71	-108.92	-73.64
5	-108.32	-54.68	-108.92	-67.61

(b) *Noises in audio band in closed-loop systems*

#bitPWM	$\mathbf{f_{sw}}\sim$ 500 kHz		$\mathbf{f_{sw}}\sim$ 1 MHz	
	$\mathbf{P_{n_{ADC}}}$	$\mathbf{P_{n_{PWM}}}$	$\mathbf{P_{n_{ADC}}}$	$\mathbf{P_{n_{PWM}}}$
10	-109.25	-64.09	-109.25	-67.10
8	-109.25	-52.05	-109.25	-55.06
7	-109.25	-46.03	-109.25	-49.04
6	-109.25	-40.01	-109.25	-43.02
5	-109.25	-34.00	-109.25	-37.00

(c) *Noises in audio band in open-loop systems*

Figure 3.25: Noises of systems with anti-aliasing filter in feedback path

function).
In a similar way, the integral of equation 3.37 is applied for closed-loop system. In this case the noise transfer functions are the ones represented in figures 3.23b and 3.23a. In particular

$$NTF(s)\Big|_{PWM} = \frac{V_{PS}}{V_{DD}} \frac{1}{1+T(s)} F_{LP}(s) \tag{3.40}$$

$$NTF(s)\Big|_{ADC} = \frac{V_{PS}}{V_{DD}} \frac{G_c(s)}{1+T(s)} F_{LP}(s) \tag{3.41}$$

where $G_c(s)$ and $T(s) = G_{ADC}G_M A G_c(s) F_{AA}(s) G$ are respectively the compensator and the loop transfer functions and $F_{LP}(s)$ is the output filter transfer function. Remembering that the number of bits of ADC is fixed at 16 and the $f_{clkSYS} \sim$ 16 MHz, tables 3.25b, 3.25c and 3.25a show the MATLAB theoretical results. The results are coherent with the ones expected:

- quantization noise of PWM increases of 6 dB reducing of one the number of bits (table 3.25a);
- in open-loop system, in audio band the limitation is given by the PWM noise that decreases of 6 dB thus reducing by one the number bits of DPWM; moreover, the DPWM noise results 3 dB lower at $f_{sw} \sim$ 1 MHz than the case at $f_{sw} \sim$ 500 kHz because the frequency is doubled;
- in closed-loop system, PWM noise is reduced thanks to the effect of the integrator (see figure 3.23b, 3.23a); in particular there's a reduction of about 20 dB

for the case at $f_{sw} \sim 500\,\text{kHz}$ and of about $30\,\text{dB}$ for the case at $f_{sw} \sim 1\,\text{MHz}$; this is due to the fact that the crossover frequency for the second case is twice the first and, as a consequence, the integrator effect last for a large range of frequency;

- on the other hand, for closed-loop system, the noise of ADC in band is increased of about $1\,\text{dB}$; this is related to the roll-off presents in audio band (see figures 3.20d, 3.19d); this roll-off is not clearly visible in NTFs of figures 3.23b and 3.23a because the output filter pole attenuates the magnitude increase around $20\,\text{kHz}$;
- in all these cases, open- and closed-loop systems, the performances noise limitation are due to DPWM resolutions.

3.6.3 Implementation of loop design of system with output filter in feedback path

Similarly to the previous solution discussed in section 3.6.1, a MATLAB code is written to evaluate the transfer functions and the step responses of the solution with output filter in feedback path (see figure 3.13). The main parameters of the system are the same in table 3.16a; on the other hand, the compensator has different high frequency poles.

Compensator	$\tau_p = \left(\frac{G_{ADC}KG_MAG}{\omega_u}\right)^{\frac{1}{p}} \frac{1}{\omega_z^{1-\frac{1}{p}}}$
	$\omega_z = \frac{\omega_u}{\tan\left(\frac{PM-180^\circ+p90^\circ+\tan^{-1}\frac{\omega_u}{2f_{sw}}}{p-1}\right)}$
	$\omega_{zx} = \frac{9}{10}\omega_{pf}$ $\omega_{zy} = \omega_{pf}$
	$\omega_{ph1} = 9\omega_u\ (N=1)$ $\omega_{ph1} = 10\omega_u\ (M=1)$

Figure 3.26: Compensator filter parameters

In table 3.26, ω_{zx} and ω_{zy} are the zeros that compensate the double pole of the output filter. The two high frequency poles inserted to make the transfer function proper (ω_{ph1}) and to close the band (ω_{ph2}) are chosen to be of the first order to reduce their effect on the phase margin. The compensation zeros are chosen as in previous design (section 3.26), so $\omega_{z1} = \frac{9}{10}\omega_z$ and $\omega_{z2} = \frac{11}{10}\omega_z$. The resulting transfer function and step responses at $f_{sw} \sim 500\,\text{kHz}$ and $f_{sw} \sim 1\,\text{MHz}$ are shown in figures 3.28, 3.30, 3.29 and 3.31. The overshoot of the compensator is very large (fig. 3.30a, 3.31a) with respect to the previous solution: this is due to the fact that the bandwidth of the compensator is very large and in particular it does not reach the zero magnitude level and high frequency range is amplified (fig. 3.28a, 3.29a). This is a problem because the number of bits of the compensator are limited and saturation

of the command signal can be obtained, exiting from the linearity condition.
Theoretical noises are evaluated with the same methodology used for previous design. What changes here are the NTFs, in particular

$$NTF\Big|_{PWM} = \frac{V_{PS}}{V_{DD}} \frac{F_{LP}(s)}{1+T(s)} \tag{3.42}$$

$$NTF\Big|_{ADC} = \frac{V_{PS}}{V_{DD}} \frac{G_c(s)F_{LP}(s)}{1+T(s)} \tag{3.43}$$

where $T(s) = G_{ADC}KG_MGF_{LP}(s)G_c(s)$.

#bit	$f_{sw}\sim$500 kHz		$f_{sw}\sim$1 MHz	
	$P_{n_{ADC}}$	$P_{n_{PWM}}$	$P_{n_{ADC}}$	$P_{n_{PWM}}$
8	-106.43	-71.90	-106.68	-81.89
7	-106.43	-65.89	-106.68	-75.87
6	-106.43	-59.87	-106.68	-69.85
5	-106.43	-53.85	-106.68	-63.83

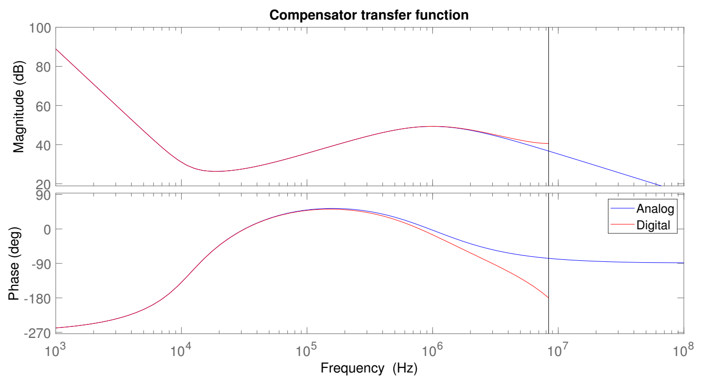
Figure 3.27: Noises in audio band in closed-loop

Theoretical results of noises shown in table 3.27 are similar to the ones found in table 3.25b: this similarity is due to the fact that the noise in audio band depends on the integrator gain that is the same of previous solution (τ_p for the two solutions has the same formula, see 3.23 and 3.34); the difference is due to the presence of zero to compensate the output filter pole, that in previous solution with anti-aliasing filter in feedback path is at higher frequency, over ω_u ($f_p = 225\,\text{kHz}$). This consideration further emphasizes the fact that the main parameter to reach high SNR is the integrator gain.

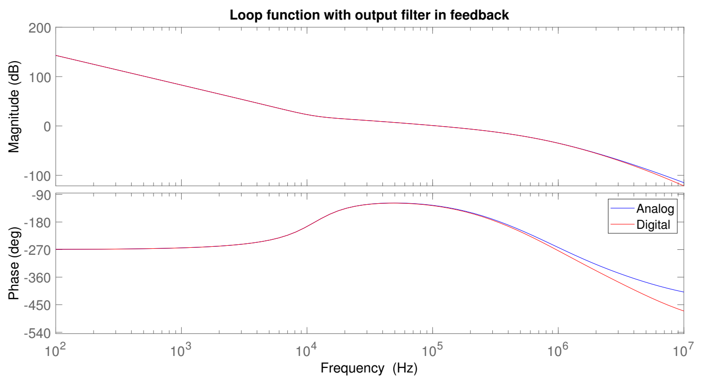
Effect of increasing order of compensator high frequency poles

As shown in subsection 3.5.3, increasing the order of high frequency poles of the compensator reduces the overshoot of the command signal (compensator output) but at the same time reduces the phase margin of the loop transfer function. A demonstration of this fact is shown in the results of the following design, caried out in the same way it has been shown in subsection 3.6.3 at $f_{sw} \sim 500\,\text{kHz}$, with the difference that the order of high frequency poles ω_{ph1} and ω_{ph2} is set to $M = N = 3$. Comparing this solution with the previous one with $N = M = 1$, can be observed that (figures 3.34, 3.35):

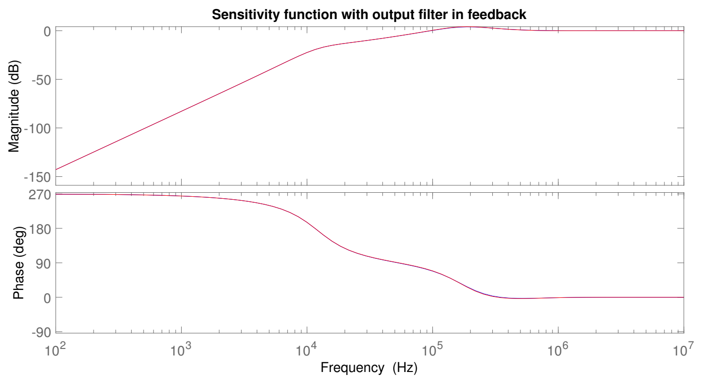
- phase margin of the higher order system results to be lower than the previous one, as expected at increasing the order of high frequency poles;
- the overshoot of command signal (fig. 3.35b) is reduced of about 50% about (even if overcomes the maximum dynamic range);
- the overshoot of the output signal is increased because the phase margin is reduced;



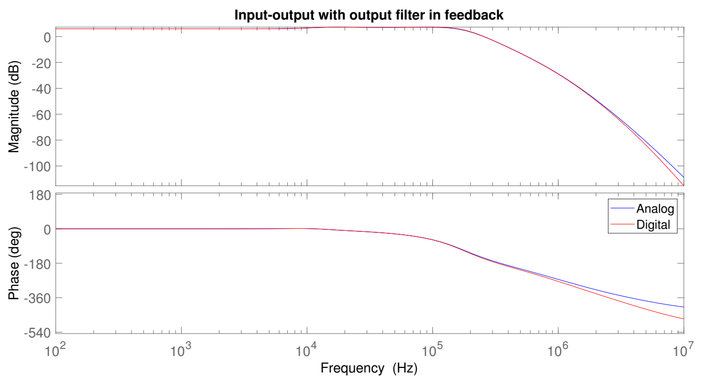
(a) *Compensator transfer function*



(b) *Loop transfer function*

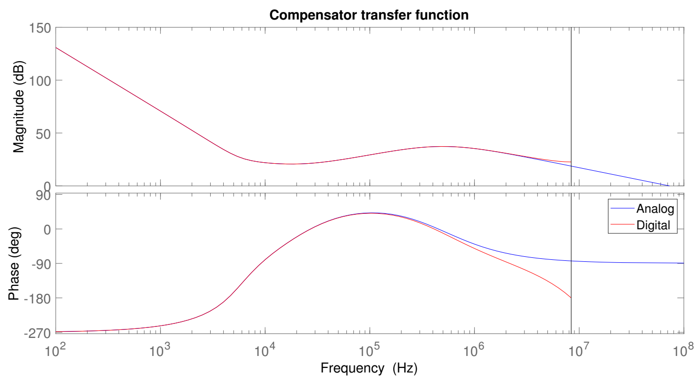


(c) *Sensitivity transfer function*

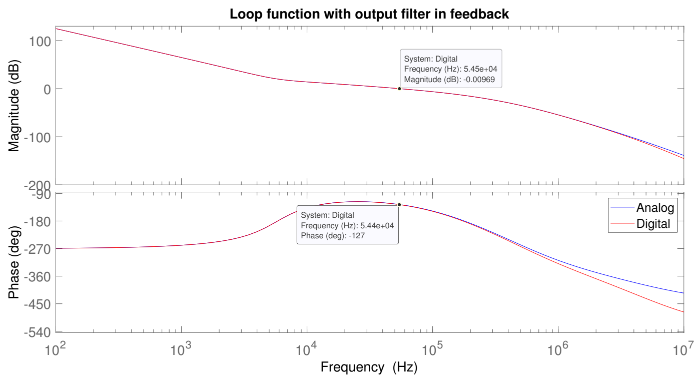


(d) *Input/output transfer function*

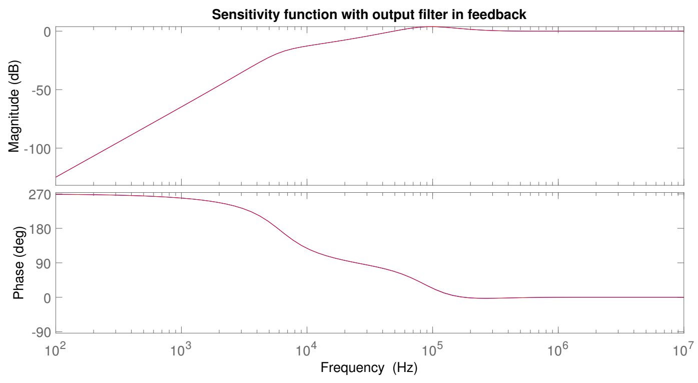
Figure 3.29: Closed-loop transfer function of the system working at $f_{sw} \sim 1\,\text{MHz}$



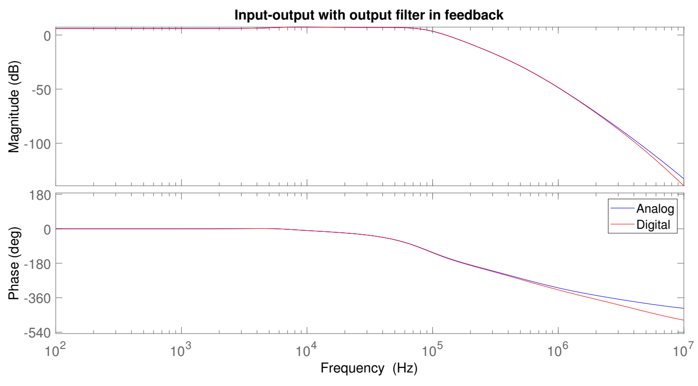
(a) *Compensator transfer function*



(b) *Loop transfer function*

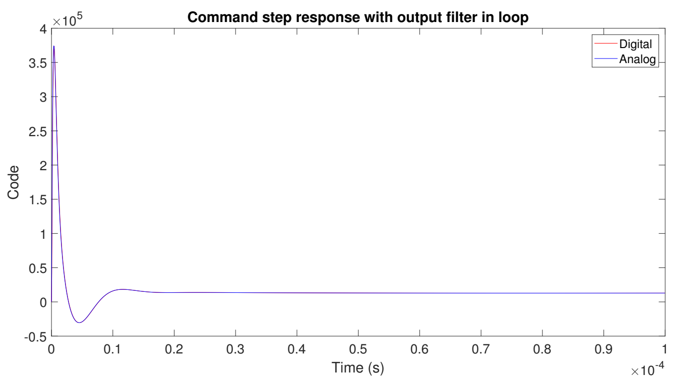


(c) *Sensitivity transfer function*

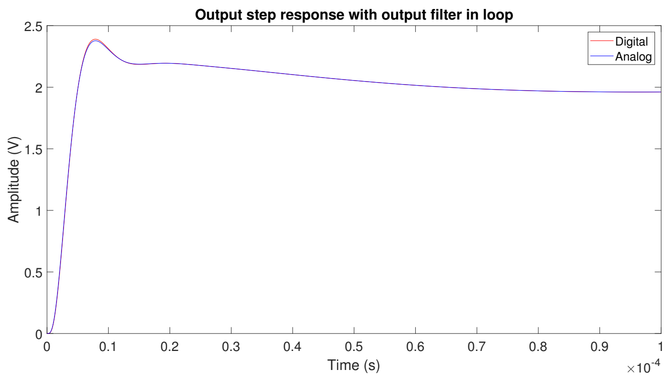


(d) *Input/output transfer function*

Figure 3.28: Closed-loop transfer function of the system working at $f_{sw} \sim 500\,\text{kHz}$

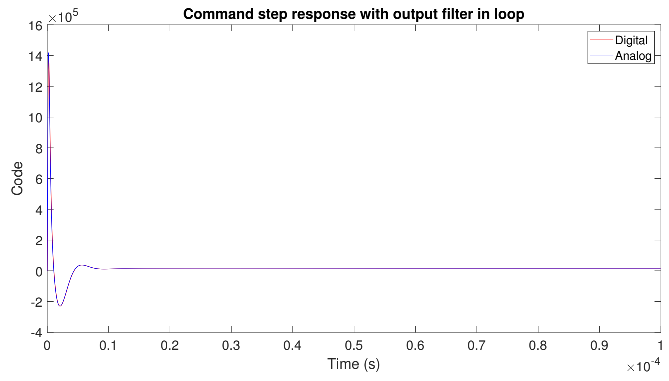


(a) *Step response of compensator*

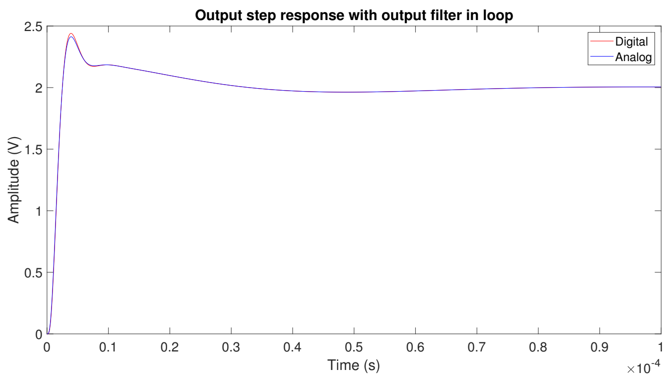


(b) *Step response of system output*

Figure 3.30: Step responses of compensator and system output ($f_{sw} \sim 500\,\text{kHz}$)

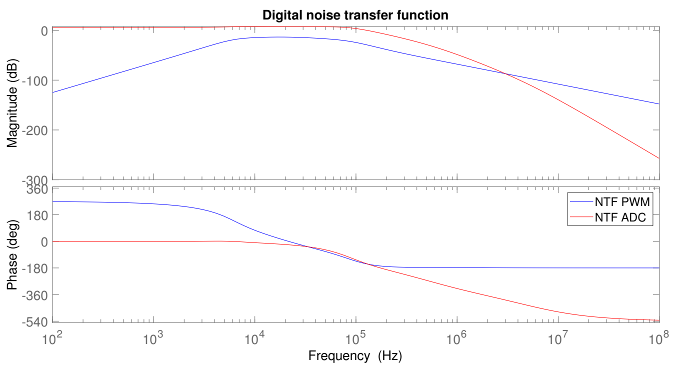


(a) *Step response of compensator*

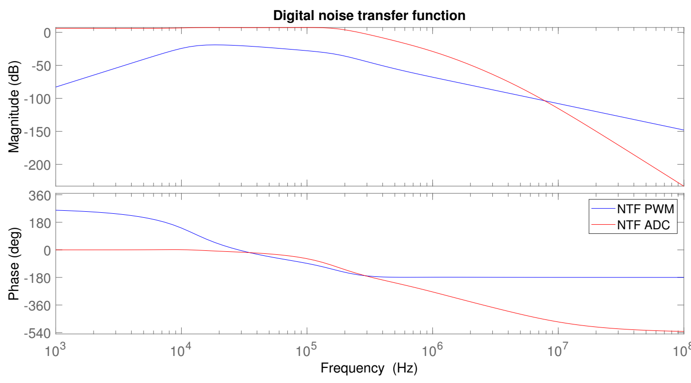


(b) *Step response of system output*

Figure 3.31: Step responses of compensator and system output ($f_{sw} \sim 1\,\text{MHz}$)



(a) *Noise transfer functions of system with* $f_{sw} \sim 500\,\text{kHz}$



(b) *Noise transfer functions of system with* $f_{sw} \sim 1\,\text{MHz}$

Figure 3.32: Noise transfer functions of closed-loop system

	f_{sw}~500 kHz	f_{sw}~1 MHz
PM	53.42°	51.2°
UGF	54 411 Hz	109 612 Hz
GM (gain margin)	4.8 dB	4.36 dB
f(∠180°)	173 940 Hz	328 301 Hz

(a) *Loop transfer functions parameters*

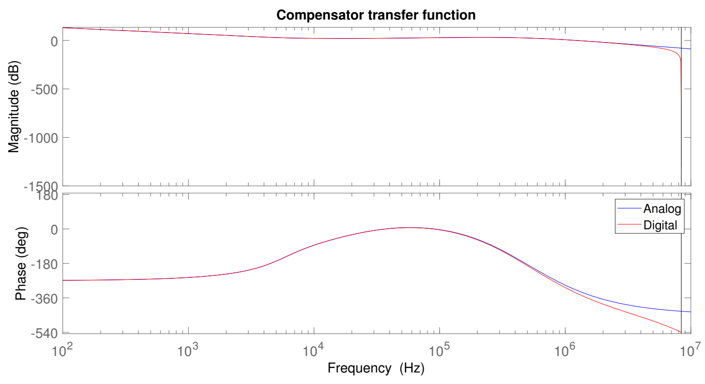
Step responses system output		
	f_{sw}~500 kHz	f_{sw}~1 MHz
Rise time	3.01 µs	1.51 µs
Settling time (1%)	64.8 µs	53.5 µs
Overshoot	21.8%	21.7%
Peak	2.39 V	2.44 V
Peak time	7.87 µs	3.94 µs

(b) *Step responses system output*

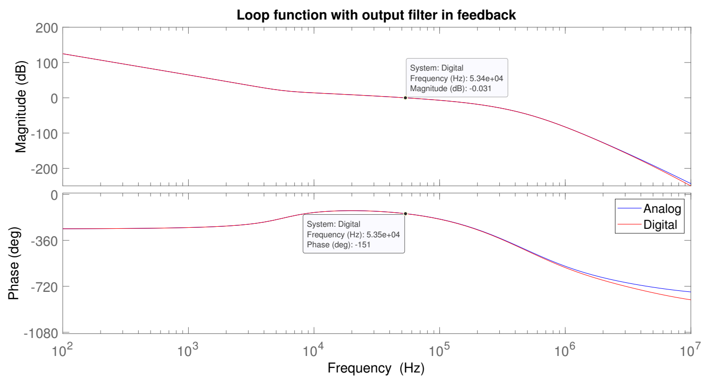
Step responses compensator output		
	f_{sw}~500 kHz	f_{sw}~1 MHz
Rise time	14.1 ns	2 ns
Settling time (1%)	8.2 µs	4.1 µs
Overshoot	2797%	10647%
Undershoot	235%	1752%
Peak	373820	1410272
Peak time	0.41 µs	0.23 µs

(c) *Step responses compensator*

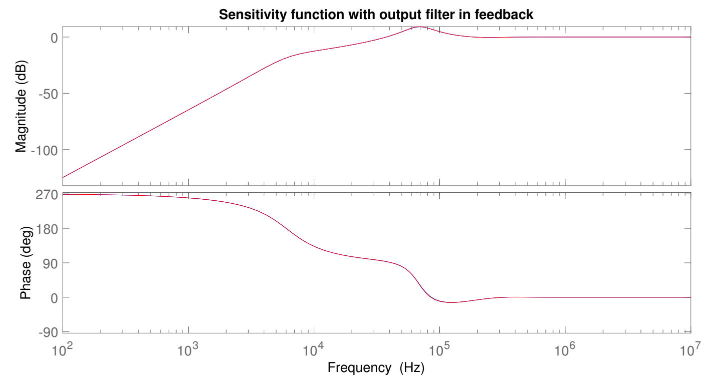
Figure 3.33: Systems parameters



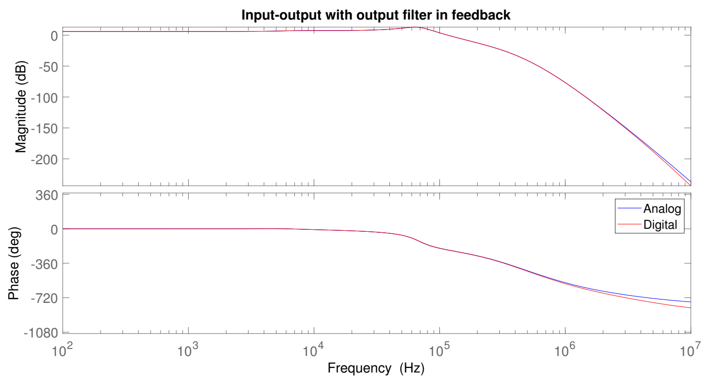
(a) *Compensator transfer function*



(b) *Loop transfer function*

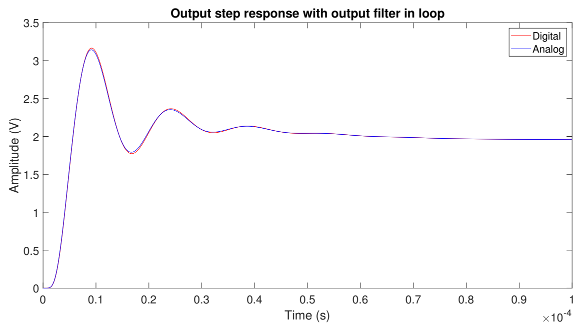


(c) *Sensitivity transfer function*

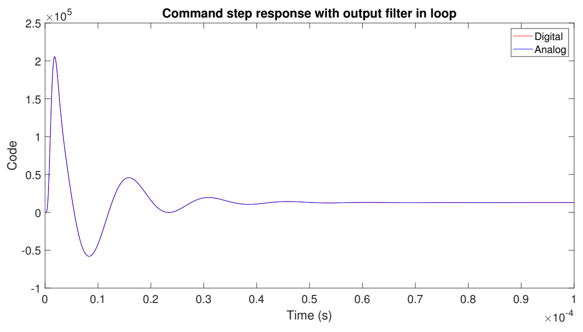


(d) *Input/output transfer function*

Figure 3.34: Closed-loop transfer function of the system working at $f_{sw} \sim 500\,\text{kHz}$, $M = N = 3$



(a) *Step response of compensator*



(b) *Step response of system output*

Figure 3.35: Step responses of compensator and system output ($f_{sw} \sim 500\,\mathrm{kHz}$, $M = N = 3$)

In conclusion, high frequency poles in loop transfer function have negative effect on stability of the loop. This is the reason why in these designs high frequency poles are set first or, at least, second order and with angular frequency around $10\omega_u$, since in this way the phase margin is reduced about $10°$ with respect to the one desired.

PM	29.45°
UGF	53 276 Hz
GM (gain margin)	1.77 dB
f(∠180°)	84 909 Hz

(a) *Loop transfer functions parameters*

Step response system output	
Rise time	2.78 µs
Settling time (1%)	61.74 µs
Overshoot	61.3%
Peak	3.16 V
Peak time	9.18 µs

(b) *Step response system output*

Step response compensator output	
Rise time	201.18 ns
Settling time (1%)	33.4 µs
Overshoot	1492%
Undershoot	450%
Peak	205579
Peak time	1.87 µs

(c) *Step response compensator*

Figure 3.36: Systems parameters with $f_{sw} \sim 500\,\mathrm{kHz}$, $N = M = 3$

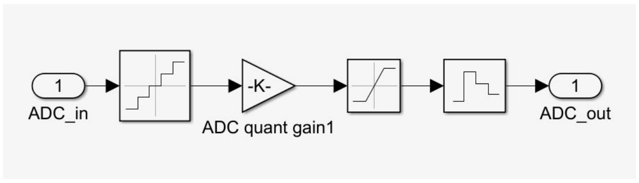


Figure 3.37: ADC Simulink® model blocks

3.7 Simulink® models

In this section are shown the Simulink® models of different blocks, in particular

- ADC;
- DPWM;
- output stage;
- truncation gain;
- dead time model.

3.7.1 ADC model

As shown in section 3.4.1, the main blocks that model the ADC are the *gain block* (to map analog value in digital world) and the *zero-order hold* (to model the sampling event). Moreover, has to be taken in consideration that the ADC introduces a quantization (as shown in graph 3.6) so a *quantizer* has to be added before the amplification block. The Simulink® model is shown in figure 3.37. The *quantizer* has the function to quantize the input signal depending on the number of bits and on the power supply of the ADC; in particular

$$q = \frac{V_{DD}}{2^{nbit_{ADC}}} \quad \text{if power supply is single and positive}$$
$$q = \frac{2V_{DD}}{2^{nbit_{ADC}}} \quad \text{if power suppply is symmetrical (positive and negative)}$$

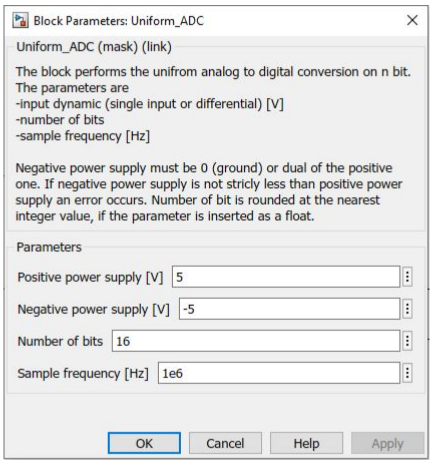
where q is the quantization interval. As a consequence the parameter *Quantization interval* of the *quantizer* is set to

$$\frac{V_{PS_p} - V_{PS_n}}{2^{nbit_{ADC}}} \tag{3.44}$$

where V_{PS_p} is the positive power supply and V_{PS_n} is the negative power supply. In particular, $V_{PS_n} = -V_{PS_p}$ if the ADC has dual input dynamic or is differential, otherwise, if $V_{PS_n} = 0$, the ADC works correctly only with input signal in range $[V_{PS_p}, 0]$. Can be observed that in case of differential ADC, the negative power

ADC_in
ADC_out
Uniform_ADC

(a) *Subsystem of modelled Uniform ADC*



Uniform_ADC (mask) (link)

The block performs the unifrom analog to digital conversion on n bit.
The parameters are
-input dynamic (single input or differential) [V]
-number of bits
-sample frequency [Hz]

Negative power supply must be 0 (ground) or dual of the positive one. If negative power supply is not stricly less than positive power supply an error occurs. Number of bit is rounded at the nearest integer value, if the parameter is inserted as a float.

Parameters

Positive power supply [V]	5
Negative power supply [V]	-5
Number of bits	16
Sample frequency [Hz]	1e6

(b) *Mask of modelled Uniform ADC*

Figure 3.38: Simulink® ADC subsystem model

supply really corresponds to the minimum differential voltage input that can be applied.
The output signal of the quantizer is an *analog quantized signal.* To be mapped in digital word, an amplifier has to be added. In particular this amplification factor (parameter *gain* of the *Gain* block) is given by

$$\frac{2^{nbit_{ADC}}}{V_{PS_p} - V_{PS_n}} \tag{3.45}$$

Comparing this formula with the ADC gain one (equation 3.11) they are the same if power supplies are considered $V_{PS_p} = -V_{PS_n} = V_{DD}$. The output of the gain block provides the digital code that represents the quantized input signal.
To take into account that when a number of bits is given, the maximum and the minimum value are respectively $2^{nbit_{ADC}-1} - 1$ and $-2^{nbit_{ADC}-1}$, a *limiter* is added. In this way, when the input signal overcomes the power supply, there is a saturation of the output code. Moreover, if there's only a single power supply, the saturation values are $2^{nbit_{ADC}} - 1$ and 0. The choice of the these limiting values implies that the representation of the output code is done in 2's complement if ADC is differential, otherwise, if the ADC is single-ended, the representation of output code is unsigned Finally, a *zero-order hold* working at f_{clkSYS} models the sampling.
All these blocks are grouped in a *subsystem* that through a *mask* asks to the user to insert positive and negative power supply, number of bits and sampling frequency (figure 3.38).

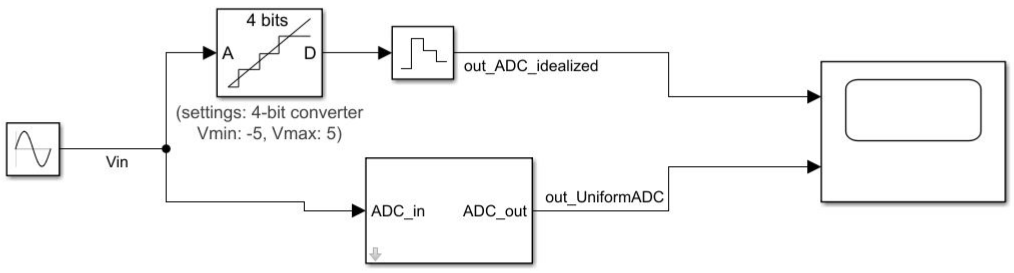


Figure 3.39: Test scheme of Uniform ADC

ADC test

To test the model of ADC, it is compared with the *Idealized ADC quantizer* of Simulink® native library. Because this block only performs only the quantization, a ZOH block samples its output. The system tested is shown in figure 3.39. The system is tested in three ways with a sinewave signal and V_{DD} ADC power supply:

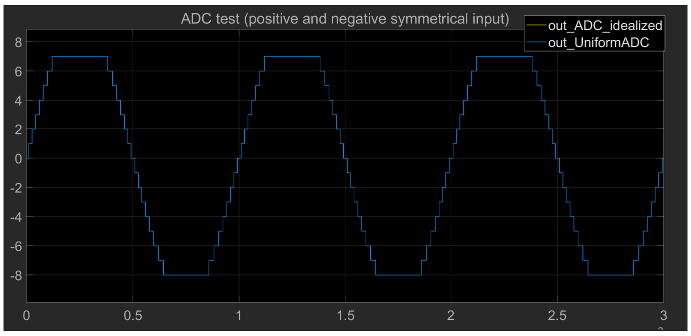
1. input signal is set to be strictly positive and it must cover the entire dynamic of ADCs; in this case amplitude of input signal is set to $V_{DD}/2$ and it is biased with $V_{DD}/2$;
2. the input signal can be positive or negative and it has to cover the entire dynamic of ADCs; the amplitude of the input signal is V_{DD}, with no bias;
3. input signal overcomes the input dynamic of the ADCs and the saturation to the maximum/minimum value is verified

The parameters of ADC are the following:

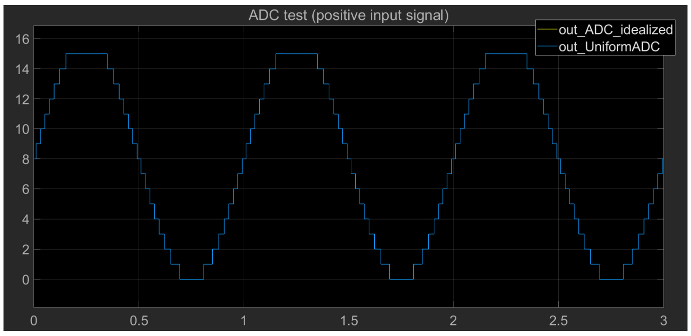
- positive power supply: 5 V
- negative power supply: −5 V (if ADC has dual power supply) or 0 V (in case of ADC can sample only strictly positive signals);
- sample frequency: $f_s = 500\,\text{kHz}$;
- bits number: 4.

The figure 3.40 shows the three tests. The first result to observe is that in all simulations the *Idealized ADC quantizer* output is superimposed with the *Unifrom ADC* one, that is a first check of the ADC model correctness. Then, analyzing in detail each test, the codes obtained are the expected ones. In fact, from figure 3.40a, the maximum and minimum reached values are respectively $2^{nbit_{ADC}-1} - 1 = 2^{(4-1)} - 1 = 7$ and $-2^{nbit_{ADC}-1} = -2^{(4-1)} = -8$; in a similar way in figures 3.40b and 3.40c the minimum value is 0 and the maximum one is $2^{nbit_{ADC}} - 1 = 2^4 - 1 = 15$

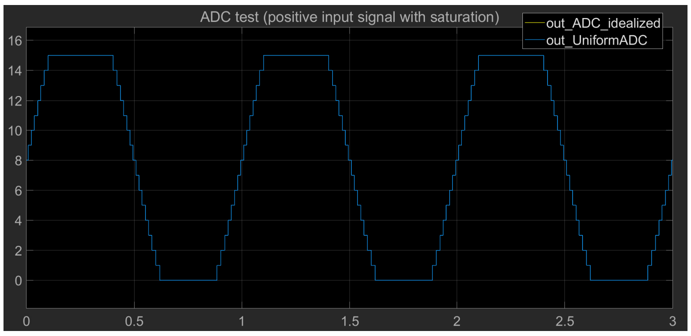
with the only difference that in the case where input signal overcomes the ADC input dynamic, the time for which the maximum/minimum code is generated by ADC is larger: the saturation is reached.



(a) *Test with input signal with positive and negative dynamic*



(b) *Test with input signal with positive dynamic*



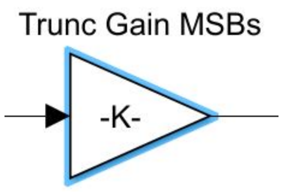
(c) *Test with positive signal that exceed the ADCs dynamic*

Figure 3.40: ADCs tests results

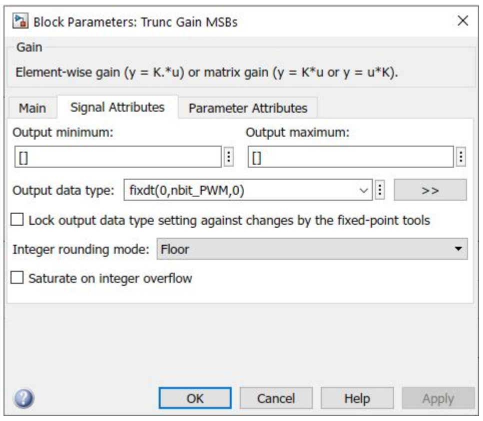
3.7.2 Truncation gain

As shown in subsection 3.4.3, the operation of truncation is necessary to adapt the dynamic of the output code of ADC/comparator to the one of DPWM, which generally has a lower number of bits than ADC/comparator. This operation can easily done with a *gain* block with gain of

$$\frac{2^{nbit_{ADC}}}{2^{nbit_{PWM}}} \tag{3.46}$$



(a) *Truncation gain block*


Block Parameters: Trunc Gain MSBs

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Main | Signal Attributes | Parameter Attributes

Output minimum: []

Output maximum: []

Output data type: fixdt(0,nbit_PWM,0)

☐ Lock output data type setting against changes by the fixed-point tools

Integer rounding mode: Floor

☐ Saturate on integer overflow

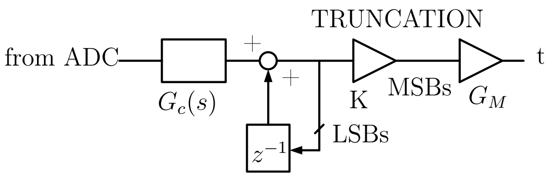
(b) *Truncation gain block parameters*

Figure 3.41: Truncation gain block and parameters

where $nbit_{ADC}$ becomes $nbit_{ADC} - 1$ in case of open-loop system for the motivations shown in subsection 3.4.5. Moreover, to make the floor operation of equation 3.13 and to limit the representation of the number on $nbit_{PWM}$, the parameters *Signal attributes/Integer rounding mode* and *Signal attributes/Output data type* are respectively set to *Floor* and *fixdt(0, nbit_PWM, 0)*.

3.7.3 Noise shaping

To increase the SNR, instead to increase the number of bit of DPWM (which require larger clock frequency), should be applied a *noise shaping* on the truncated signal, before it is sampled by DPWM. A block diagram of the noise shaping algorithm [38], is shown in figure 3.42a. The idea is to recover the LSBs that are truncated before DPWM sampling at n-1th step and sum them to the nth output code from compensator.



(a) *Noise shaping applied to the closed-loop model*

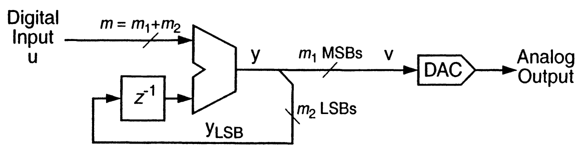


Figure 2.17: A first-order digital modulator used in a ΔΣ DAC.

(b) *Noise shaping of [38]*

Figure 3.42: Noise shaping technique

In the simulated circuit, the problem is to recover the LSBs. The idea that has been implemented is the following one: let's consider to recover M LSBs from a number n represented on $N > M$ bits; if the representation of the numbers is finite and with *non-saturated representation* (so overflow is accepted), multiplying n by $2^{(N-M)}$ the resulting number n_x has the M MSBs equal to the LSBs of n and the remaining $N-M$ LSBs are all zeros. Then, dividing n_x by $2^{(N-M)}$ the resulting number n_t is composed by $N-M$ MSBs as zeros and the LSBs are the MSBs of n_x. As a consequence n_t can be represented on M bits because all the MSBs are zeros and the LSBs are recovered. Let's consider and example where $M = 2$ LSBs are recovered from a number (11, decimal) represented on $N = 4$ bits

- $$\begin{aligned} & 11\big|_{dec} \rightarrow 1011\big|_{bin} \\ & \qquad \Rightarrow 1100\big|_{bin} \rightarrow 12\big|_{dec} = \left(11 \cdot \frac{2^N}{2^M} = 11 \cdot \frac{2^4}{2^2}\right) \\ & \qquad \Rightarrow 0011\big|_{bin} \rightarrow 3\big|_{dec} = \left(12 \cdot \frac{2^M}{2^N} = 12 \cdot \frac{2^2}{2^4}\right) \end{aligned} \tag{3.47}$$

Limiting the representation of 0011 on two bits, the LSBs 11 can be represented on $M = 2$ bits.

3.7.4 Truncation and noise shaping test

To test the truncation and the LSBs recovering operations, a fully positive sinusoidal signal is converted in digital by the *Uniform ADC*. Then, the generated code is truncated and LSBs are extracted, obtaining the timing diagram shown in figure 3.44. The tested block diagram is shown in figure 3.43, where the *gain* blocks for the truncation and LSBs recovering are shown. In particular the gain blocks have the following parameters

- *Trunc Gain MSBs*:
  - *Gain:* $\frac{2^{nbit\_PWM}}{2^{nbit\_ADC}}$;
  - *Output data type: fixdt(0, nbit_PWM, 0)*;
  - *Rounding mode: Floor.*
- *Trunc Gain 3:*
  - *Gain:* $\frac{2^{nbit\_ADC}}{2^{nbit\_PWM}}$;
  - *Output data type: fixdt(0, nbit_ADC, 0)*;
  - *Rounding mode: Floor.*
- *Trunc Gain 4:*

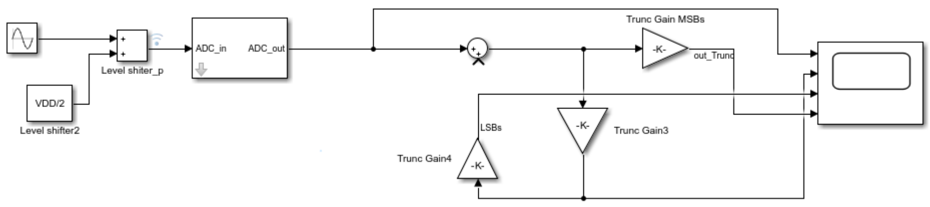


Figure 3.43: Block diagram to test truncation and LSBs recovering

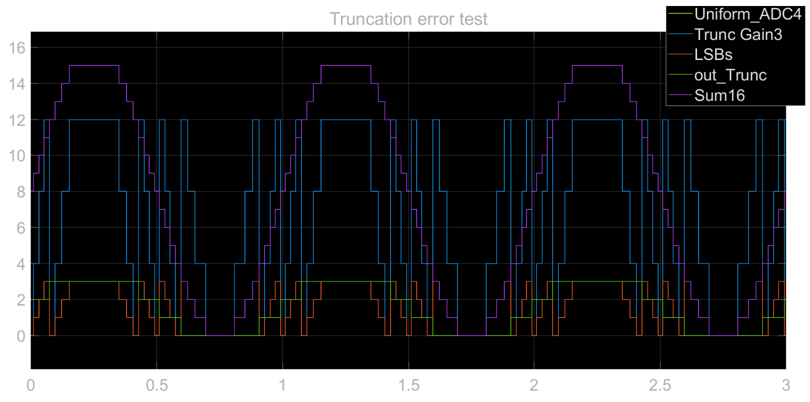


Figure 3.44: Timing diagram of truncation and LSBs recovering

- *Gain:* $\frac{2^{nbit\_PWM}}{2^{nbit\_ADC}}$;
- *Output data type: fixdt(0, nbit_ PWM, 0)*;
- *Rounding mode: Floor.*

- *Sum*:

  - *Output Data Type: fixdt(0, nbit_ ADC, 0).*

In the test $nbit_{ADC} = 4$ and $nbit_{PWM} = 2$ and observing the timing diagram 3.44 when the output code of ADC is 11 the LSBs and the MSBs are exactly the ones shown in example 3.47. Moreover, to test noise shaping and to verify that no signal saturates, the block diagram of figure 3.45 is simulated, obtaining the timing diagram in figure 3.46. Can be seen that the output of the sum node (*Sum16*) does not exceed the maximum possible dynamic ($2^3 = 8$) and consists in the same signal generated by ADC with a superimposition of an high frequency error signal (LSBs). Moreover, when the output of ADC reaches the maximum value, the output of the *Sum* block comes back to lowest values: this is due to the fact the adder works on $nbit_{ADC} = 4$ bits, obtaining overflow when the maximum value is reached. The delay line works at the same frequency of ADC (100 kHz).

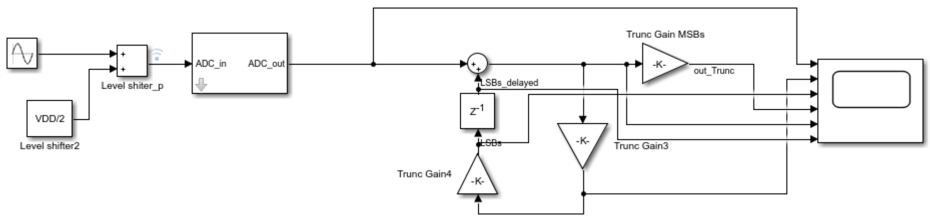


Figure 3.45: Noise shaping test scheme

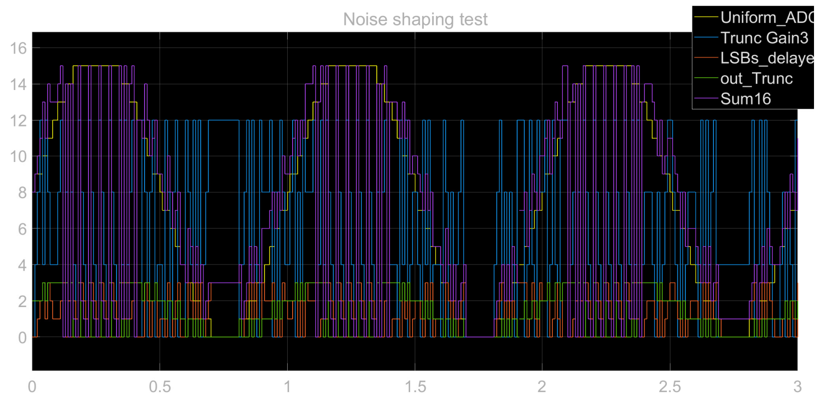


Figure 3.46: Noise shaping test results ($nbit_{ADC} = 4$, $LSBs = 2$, $MSBs = nbit_{PWM} = 2$)

3.7.5 DPWM blocks scheme

The block diagram of DPWM is implemented in Simulink® in a way which is similar with the model in figure 3.3 and 3.4. The only difference is that the triangular waveform is not a dual edge, but a trailing edge generated by means of a *free-running counter* where the resolution and the clock frequency are the parameters of the block. It can be noted that the effective PWM signal comes from output stage, because after the sum node there's a signal with no physical meaning that has the only function to drive the output stage (the *Relay* block). The block diagram and results of a simulation tested for ten PWM periods at $f_{sw} \sim 500\,\text{kHz}$ and $nbit_{PWM} = 5$ are shown in figures 3.47 and 3.48. As expected, the DPWM counter is reset after $2^{nbit_{PWM}} - 1 = 2^5 = 31$ and the output of *Relay* is high until the PWM counter is lower that the input DPWM code, while it becomes low when the DPWM counter is larger than the input code. Moreover can be observed the uniform sampling: the input code (in_DPWM) remains constant for all the switching period. Because of the *free-running counter* has the number of bits and the clock frequency as parameters, the clock frequency is derived as

$$f_{clk_{DPWM}} = f_{sw} \cdot 2^{nbit_{DPWM}} \tag{3.48}$$

The block *convert* is needed to have same data type at the inputs of the sum node.

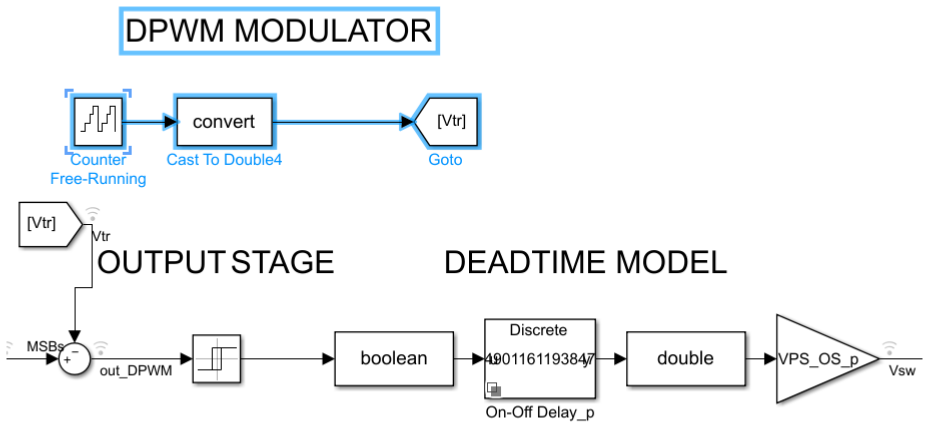


Figure 3.47: Block diagram of DPWM and output stage

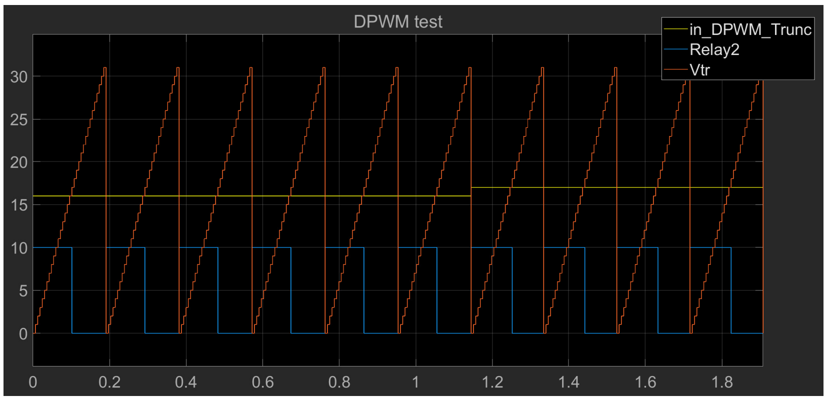
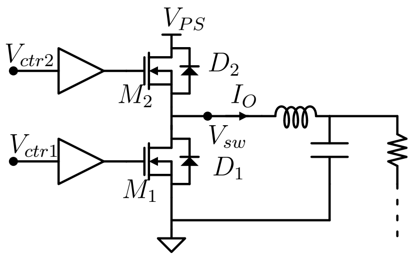


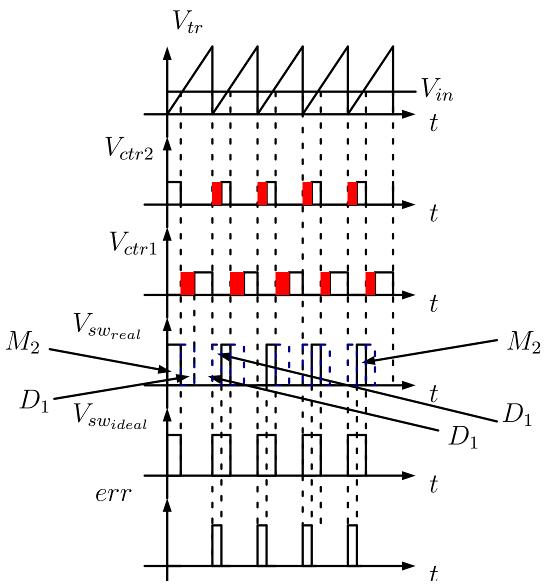
Figure 3.48: DPWM test ($nbit_{PWM} = 5$, $f_{sw} \sim 500\,\text{kHz}$)

3.7.6 Dead time model

In figure 3.47 is also shown the block that simulates the dead time. Dead time is needed to avoid the short circuit between power supply and ground and it is an amount of time where both transistors do not conduct. The duration of dead time influences efficiency and distortion: in particular, if dead time is large the power dissipation is reduced, but distortion increases; on the other hand, reducing the dead time duration, THD is reduced, but power consumptions increase.
Consider a bridge-tied load class D amplifier and what happens at the output of each half-bridge stage. In particular, let's consider a half-bridge H_{HS} (*high-side* stage) the stage that generates the voltage on the positive node of the load and H_{LS} (*low-side* stage) the stage that generates the voltage on negative node of the load. Moreover, for the sake of simplicity, let's consider constant the input voltage of the amplifier V_{in} and the current $I_O > 0$ at the output of H_{HS} stage. The situation is shown in figure 3.49, where architecture of H_{HS} and a qualitative timing diagram



(a) *H_{HS} output stage*



(b) *Qualitative timing diagram of conductive devices of H_{HS}*

Figure 3.49

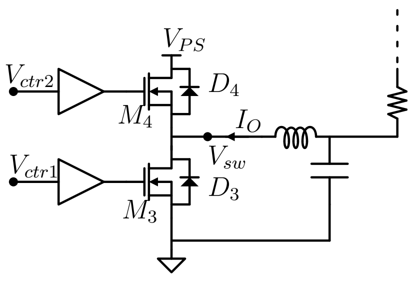
of the gate voltages and output stage voltage are considered. The timing diagram shows the different behaviour of output stage voltage in ideal condition and in case of dead time.

- **Ideal condition**
  - $V_{ctr2} = H$, $V_{ctr1} = L$: M_2 is on, so it is the conductive device; current flows from power supply through M_2 and goes to the output;
  - $V_{ctr2} = L$, $V_{ctr1} = H$: M_2 is off; because of $I_O > 0$, M_1 cannot conduct, so it's the body diode that makes the current flow to the output.
- **Real condition**
  in this case a dead time (red regions) is removed
  - $V_{ctr2} = H$, $V_{ctr1} = L$: M_2 is on, so it is the conductive device; the condition is the same of the ideal case;
  - $V_{ctr2} = L$, $V_{ctr1} = L$: both M_1 and M_2 are off but the continuity condition of the inductor currents makes conduct D_1;
  - $V_{ctr2} = L$, $V_{ctr1} = H$: M_2 is off, D_1 is conductive, M_1 is on, same condition of ideal case;

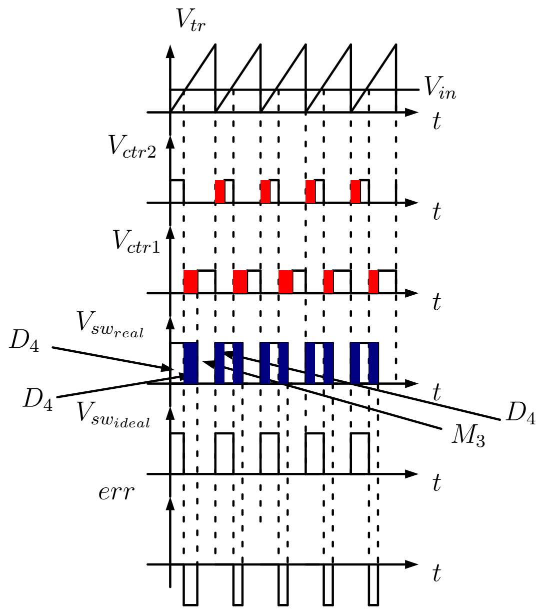
The *error* signal is defined as

$$error = V_{sw_{ideal}} - V_{sw_{real}} \tag{3.49}$$

and occurs on rising edge of V_{sw}: in particular $V_{sw_{real}}$ *has the rising edge shifted ahead by the dead time with respect to the rising edge of* $V_{sw_{ideal}}$.



(a) *H_{LS} output stage*



(b) *Qualitative timing diagram of conductive devices of H_{LS}*

Figure 3.50

Now, if the output current of H_{HS} is $I_O > 0$ then the output current of H_{LS} is $I_O < 0$. Considering this fact, a similar analysis made for H_{HS} can be made for conductive devices H_{LS}, referring to figure 3.50. Both *ideal* and *real* (with dead time, highlighted with blue region) cases are considered

- **Ideal condition**
  - $V_{ctr2} = H$, $V_{ctr1} = L$: M_4 is on, but because $I_O < 0$ the conductive device is D_4; M_3 is off;
  - $V_{ctr2} = L$, $V_{ctr1} = H$: M_2 is off, M_3 is on and is the conductive device because the current flows from the inductor and enters in the drain of M_3;
- **Real condition**
  - $V_{ctr2} = H$, $V_{ctr1} = L$: M_4 is on, D_4 is the conductive devices, M_3 is off; same situation of the ideal case;
  - $V_{ctr2} = L$, $V_{ctr1} = L$: both M_4 and M_3 are off but the continuity condition of the inductor current makes conduct D_4;

- $V_{ctr2} = L$, $V_{ctr1} = H$: M_4 is off, D_3 is on, M_3 is on; same condition of ideal case;

Similarly to H_{HS}, the *error* occurs only on falling edge and in particular *falling edge of $V_{sw_{real}}$ is shifted ahead by the dead time with respect to the falling edge of $V_{sw_{ideal}}$*.

3.7.7 Open-loop DC error

Depending on the current direction, considering continuous input voltage V_{in}, the output voltage is affected by an error on DC component that basically depends on the duty cycle of *error* signal. In particular, the DC component $\bar{V}_{error}$ of *error* signal $V_{error}(t)$ is given by

$$\bar{V}_{error} = \int_0^T V_{error}(t)dt = \int_0^{\Delta t} V_{err_{pk}} dt = \tag{3.50}$$

$$= \frac{\Delta t}{T} V_{err_{pk}} \tag{3.51}$$

where $V_{err_{pk}} = V_{PS}$ if $I_O > 0$ or $V_{err_{pk}} = -V_{PS}$ if $I_O < 0$, Δt is the dead time and V_{PS} the power supply of output stage. As a consequence, the error on DC component is given by

$$\bar{V}_{error} = \begin{cases} \frac{\Delta t}{T} V_{PS} & \text{if } I_O > 0 \\ -\frac{\Delta t}{T} V_{PS} & \text{if } I_O < 0 \end{cases} \tag{3.52}$$

Therefore, the real DC component due to dead time presence is lower by a factor $\frac{\Delta t}{T} V_{PS}$ with respect to the ideal one if $I_O > 0$, otherwise is $\frac{\Delta t}{T} V_{PS}$ larger than ideal condition. Finally if $\Delta t \ll T$ the dead time can be considered negligible.

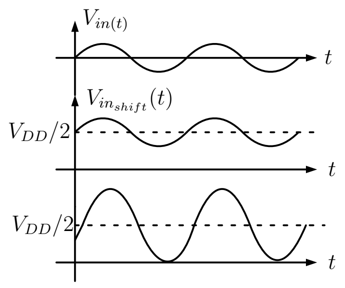


Figure 3.51: Input signal and output signal of the half-bridge output stages

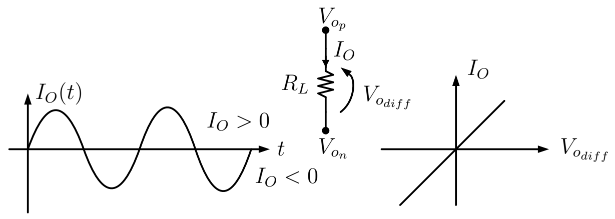


Figure 3.52: Current and characteristic I_O-$V_{o_{diff}}$ on the load

Load current

Ideally, by considering only a sinusoidal signal as input of the amplifier $V_{in}(t)$, the output differential voltage on the load $V_{o_{diff}}(t)$ is still a sinusoidal waveform at the same frequency. Mathematically, considering open-loop stage and single power supply for the half-bridge output stages, results

$$V_{in}(t) = V_{in}\sin(\omega t + \Phi)$$

$$\Rightarrow V_{in_{shift}} = V_{in}\sin(\omega t + \Phi) + \frac{V_{DD}}{2} \tag{3.53}$$

$$\Rightarrow V_{o_p} = \frac{V_{PS}}{V_{DD}}\Big[V_{in}\sin(\omega t + \Phi)\Big] + \frac{V_{PS}}{V_{DD}}\frac{V_{DD}}{2} \tag{3.54}$$

$$= \frac{V_{PS}}{V_{DD}}\Big[V_{in}\sin(\omega t + \Phi)\Big] + \frac{V_{PS}}{2} \tag{3.55}$$

$$V_{o_n} = -\frac{V_{PS}}{V_{DD}}\Big[V_{in}\sin(\omega t + \Phi)\Big] + \frac{V_{PS}}{V_{DD}}\frac{V_{DD}}{2} = -\frac{V_{PS}}{V_{DD}}\Big[V_{in}\sin(\omega t + \Phi)\Big] + \frac{V_{PS}}{2}$$

$$\Rightarrow V_{o_{diff}} = V_{o_p} - V_{o_n} = 2\frac{V_{PS}}{V_{DD}}\Big[V_{in}\sin(\omega t + \Phi)\Big] \tag{3.56}$$

As a consequence, considering purely resistive load, the current on the load as the following characteristics

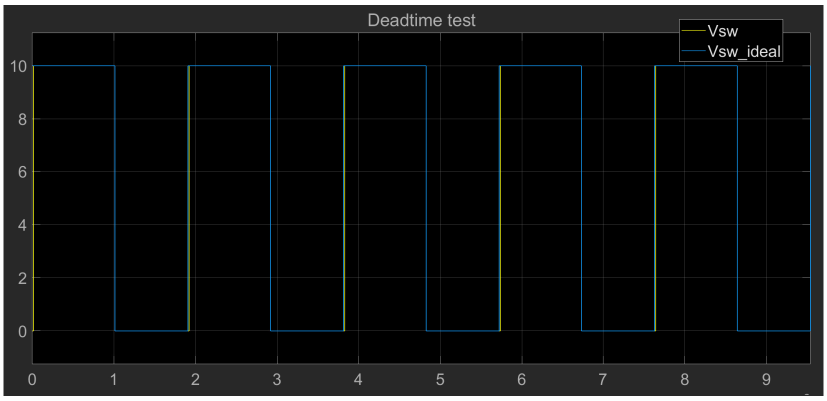
- has no DC components;
- sign changes in time;

Putting together the considerations on half-bridge stages and load current, can be concluded that for half period of the output signal the current I_O comes out from the output stage H_{HS} and comes into H_{LS} ($I_O > 0$), while for the other half period of output signal, when $I_O < 0$, the output current comes out from H_{LS} and comes into H_{HS}. In this way, the average current on the load is zero.

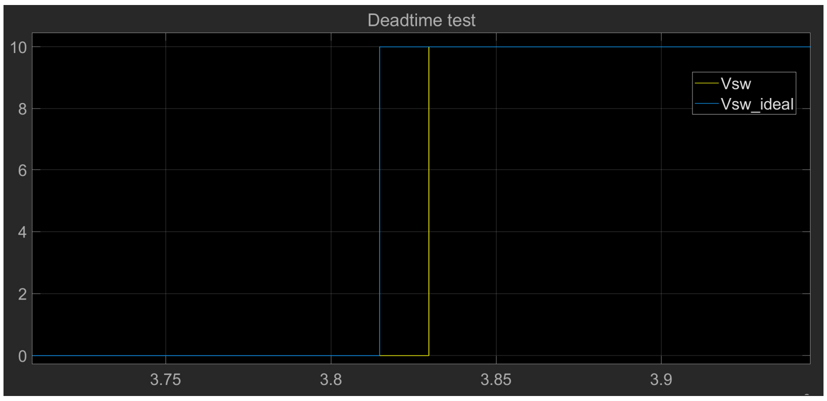
Dead time model limitations

Due to the change of sign in time, even the effect of the deadtime changes in time on the half-bridge stages. In particular, when the current on the load $I_O > 0$

(fig. 3.52), the situation is the one described in 3.7.6, while when $I_O < 0$ the behaviour is inverted so H_{LS} works as H_{HS} when $I_O > 0$ and H_{HS} works as H_{LS} when $I_O > 0$. This means that in order to know the behaviour of each half-bridge stage, the information of the current and in particular the sign is required: this is a limitation in Simulink® model because only voltage signal can be simulated. So an approximated model is used. In particular, the current is considered for both stages with the same sign $I_O > 0$. This creates only a shift on the DC components in open-loop system and no substantial differences in the output of the closed-loop system due to the infinite attenuation at $s \to 0$ of the NTFs 3.32 and 3.23.



(a) *Dead time test*



(b) *Dead time test detail of rising edge shifting*

Figure 3.53: Dead time test ($f_{sw} \sim 500\,\mathrm{kHz}$, $nbit_{PWM} = 5$, $t_{\Delta} \simeq 15\,\mathrm{ns}$)

Dead time model description and test

The idea of the deadtime model (fig. 3.47) is to chose a current direction out from the half-bridge stages and then set the dead time by means of the block *On-Off delay*: in particular, if the output current is considered $I_O > 0$, *On-delay* parameter is set with the dead time chosen and the *Off delay* one is set to 0; on the other hand, if $I_O < 0$, the *On delay* is set to 0 while the *Off delay* is set to the dead time.

Because of the *On-Off delay* block can receive only boolean data input and generate boolean data output, the output signal of *Relay* is converted in boolean data: if the *Relay* output is high then the output of boolean converter is 1, otherwise is 0. After the conversion the rising edge is delayed by the dead time (i.e. $I_O > 0$ is considered) and then re-converted in double format. Finally, because of the output signal of double converter is a two-level signal than can assume only 1 or 0 values, a *Gain* block with V_{PS} gain is added, to obtain at the output a two-level signal that can assume V_{PS} and 0 values. In figures 3.53a and 3.53b is shown the result of simulation at $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 5$, $t_{\Delta} \simeq 15\,\text{ns}$, comparing the ideal waveform with the real one (with dead time). In particular, in figure 3.53b can be clearly observed the dead time shifting of the rising edge, as expected from the analysis done previously.

3.8 Simulation data management

In this section are described the parameters of the simulations to obtain good spectral estimation and analysis of the main amplifiers parameters, like distortion and SNR.

3.8.1 Simulation stop time parameter of the simulations

To obtain a good estimation of power spectra from simulations data, the frequency resolution in audio band should be at least of 1 Hz or lower. In terms of simulation time, this means that the *Stop time* parameter of the simulations must be at least of 1 s; because of the fft can be applied only to periodic signals, the initial transient of the simulation has to be excluded: the duration of the transient is mainly related to the larger time constant of the system, which depends on the poles of the blocks. In particular, the block with the lower pole is the output filter ($f_{pf} = 20\,\text{kHz}$) from which can be derived a time constant $\tau_{pf} = \frac{1}{2\pi f_{pf}} \simeq 8\,\mu\text{s}$. To take in consideration this transient, the *Stop time* is taken 1.5 s: the first 500 ms are excluded from the frequency analysis because the transient is included, while the other second of signals is used to make the frequency analysis at 1 Hz resolution. 500 ms are reasonably enough to consider that the initial transient (estimated as $10\tau_{pf} = 80\,\mu\text{s}$) is ended.

3.8.2 Simulation integration time

The *integration time step* t_{int} (set in Simulink® in *Model Settings/Solver/Fixed-step size (fundamental sample time)* menu) is set *fixed* and it is related to the minimum sample period in the system. In particular two main sample frequencies are present: $f_{clk_{SYS}}$ and $f_{clk_{PWM}}$, where the second one is the clock frequency of the *free-running counter*. To be sure to reach at least $2f_{clk_{SYS}}$ in the frequency spectrum, the integration time is set as

$$t_{int} = \min\left(\frac{1}{4f_{clk_{SYS}}}, \frac{1}{f_{clk_{PWM}}}\right) \tag{3.57}$$

where $f_{clk_{PWM}} = f_{sw}2^{nbit_{PWM}}$ changes at varying of PWM number of bits. As a consequence, at reducing of the number of bits, t_{int} is limited at $\frac{1}{4f_{clk_{SYS}}}$ if $f_{clk_{PWM}} < 4f_{clk_{SYS}}$.

3.8.3 Decimation factor and data storage

Because of the *Stop time* and t_{int} are different of about 8-9 orders of magnitude, the number of samples for each is of the order of $\sim 10^8$ when $t_{int} = t_{int_{max}} = \frac{1}{4f_{clk_{SYS}}}$ and has to be saved in a file. To do so, in menu *Model Setting/Data Import/Export/Configure Signal Log* are selected the data to log out and decimation factor DF is applied to the analog signals. For these output and input signals, the frequency analysis has to be done in audio band and around the PWM harmonics. In particular, because of the maximum $f_{sw} \simeq 1\,\text{MHz}$, making the frequency analysis up to $2f_{clk_{SYS}}$ is sufficient. As a consequence, the analog input and output signals of the systems (in particular V_{in}, $V_{in_{shift}}$, V_{o_p} and $V_{o_{diff}}$, $out_{AAfilter}$, see figures 3.55, 3.56, 3.57) can be sampled and saved at $4f_{clk_{SYS}}$ in the datastore. This means that if the $t_{int} < \frac{1}{4f_{clk_{SYS}}}$, a decimation of data output has to be performed and in particular the *decimation factor* DF is

$$DF = \frac{\frac{1}{t_{int}}}{4fclk_{SYS}} \tag{3.58}$$

It can be seen that from equation 3.57, if $t_{int} = \frac{1}{4f_{clk_{SYS}}}$ the decimation factor $DF = 1$. These settings of integration time and *Stop time* are managed by means of MATLAB code (see *Appendix, 5.2.2, 5.2.4, 5.2.1, 5.2.7*).
Actually, the analog signals could be sampled and saved in datastore [39] at t_{int}, but this creates two performance problems:

- the size of the file in uselessly enlarged because saving larger amount of samples allows to make frequency analysis up to $\frac{1}{2t_{int}} > 2f_{clk_{SYS}} \simeq 32\,\text{MHz}$ that is useless;

- frequency analysis requires more time and the size of *timetables* of each signal could overloading RAM.

As a consequence, outputs like V_{in}, $V_{in_{shift}}$, V_{o_p} and $V_{o_{diff}}$, $out_{AAfilter}$ (see figures 3.56, 3.57,3.55) are sampled at $4f_{clk_{SYS}}$ while the other signals are sampled at $1/t_{int}$ (that could be equals to $4f_{clk_{SYS}}$ for equation 3.57). The resulting file is a *Datastore* with the data extracted from the nodes shown in figures 3.55, 3.56, 3.57 with a like-wifi symbol.

$\mathbf{f_{clk_{SYS}}}$=16777216 Hz				
	$\mathbf{f_{sw}}$=1048576 Hz			
#nbit$_{\mathbf{PWM}}$	**int. time [ns]**	**dec. int. time [ns]**	**dec. fac**	**$\mathbf{f_{clk_{PWM}}}$ [MHz]**
10	0.931	14.9	16	~ 1074
8	3.73	14.9	4	268435456
7	7.45	14.9	2	134217728
6	14.9	14.9	1	67108864
5	14.9	14.9	1	33554432

(a) *Simulation parameters for* $f_{sw} \sim 1\,\mathrm{MHz}$

$\mathbf{f_{clk_{SYS}}}$=16777216 Hz				
	$\mathbf{f_{sw}}$=524288			
#nbit$_{\mathbf{PWM}}$	**int. time [ns]**	**dec. int. time [ns]**	**dec. fac**	**$\mathbf{f_{clk_{PWM}}}$ [MHz]**
10	1.81	14.9	8	536870912
8	7.45	14.9	2	134217728
7	14.9	14.9	1	67108864
6	14.9	14.9	1	33554432
5	14.9	14.9	1	16777216

(b) *Simulation parameters for* $f_{sw} \sim 500\,\mathrm{kHz}$

Figure 3.54: Simulation parameters at varying of f_{sw} and $nbit_{PWM}$

Each signal stored in the file has a *timetable*, with *Time* and *Data* fields. Each simulation generates a file that can overcome 40 GB size, that is not possible to manage by means of RAM. As a consequence, a code that manages these kind of files (*Datastore*) is written (*Appendix, subsection 5.2.9, 5.2.8, 5.2.10*), allowing the reading of portion of file and signals, without overloading RAM. In the tables 3.54 are shown the integration time and the time at which analog signals are sampled and saved in datastore: these times are evaluated by means of the script shown *Appendix, subsections 5.2.4, 5.2.1, 5.2.2* and depend on the number of bits of DPWM and switching frequency because $f_{clk_{PWM}} = 2^{nbit_{PWM}} f_{sw}$ (see equation 3.57). Can be seen that the integration time (*int. time*) coincides with the sampling time of analog outputs (*dec. int. time*) when the clock frequency of DPWM ($f_{clk_{PWM}}$) becomes equal or lower than the sampling frequency of ADC and comensator ($f_{clk_{SYS}}$); as a consequence, the decimation factor (*dec. fac.*) becomes equal to 1. Moreover, the down-sampling time (*dec. int. time*) is always the same and coincides with $\frac{1}{4f_{clk_{SYS}}}$, obtaining the same number of samples for each simulation at varying of PWM bit number. As a consequence, the resolution and the Nyquist frequency for the frequency analysis is the same for all analog signals, making comparable the different spectra.

Because of Simulink® accepts only multiple sampling time of the integrator time and because of $f_{clk_{PWM}} \propto 2^N$ all the sampling times in the system are chosen as multiple of 2^N: this is the reason why the switching frequency, the sampling time of ADC and compensator $f_{clk_{SYS}}$ and the deadtime $\Delta t \sim 15\,\mathrm{ns}$ are chosen with these numbers (see table 3.16a).

3.9 Class D audio amplifier model with DPWM modulation

The designed models of closed-loop (both with and without output filter in the feedback path) and open-loop class D amplifiers are shown in figures 3.55, 3.56, and 3.57. In particular, can be seen the differential output node which makes the difference of the two single-ended output voltages of the half-bridge models. The input signal is biased with $V_{DD}/2$, as shown in figure 3.5, to be sure that input signal (in case of open-loop system) or error signal (in case of closed-loop one) are inside dynamic input range of ADC. Moreover, an inverting amplification is applied to the input signal for the low-side half-bridge; in figure 3.56 can be seen also the commented noise shaping block diagram: this block can be uncommented to activate the noise shaping effect on the loop.
Simulations are performed with the parameters listed in table 3.16a, varying the PWM number of bits between 5 and 8 for open-loop systems and adding a simulation with 10 bit for closed-loop ones. The dead time is set to $\Delta t \sim 15\,\mathrm{ns}$.

3.9.1 Open-loop results

From open-loop datastores are extracted different data to plot. From figures 3.58 it can be seen that at varying of PWM bits number the peak value of triangular waveforms changes and reach the value $2^{nbit_{PWM}} - 1$, as expected from the behaviour of a *free-running counter*. About the output analog signals, both single-ended and differential voltages are shown (fig. 3.59 and 3.60): the single-ended output is the amplified input sinusoidal signal with an offset, as expected from equation 3.55; moreover, also the differential voltage is the one expected from equation 3.56 where the amplitude of sinusoidal signal is twice the single-ended one and the DC component is zero. In particular, considering the parameters of table 3.16a the theoretical

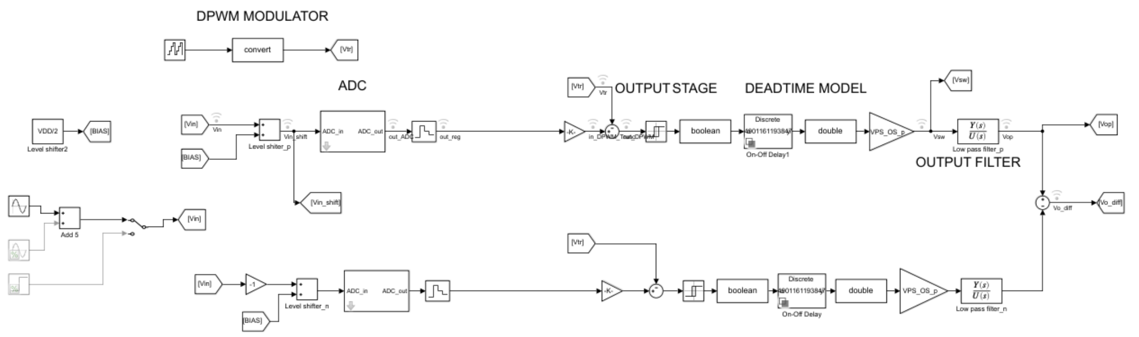


Figure 3.55: Open-loop class D amplifier model

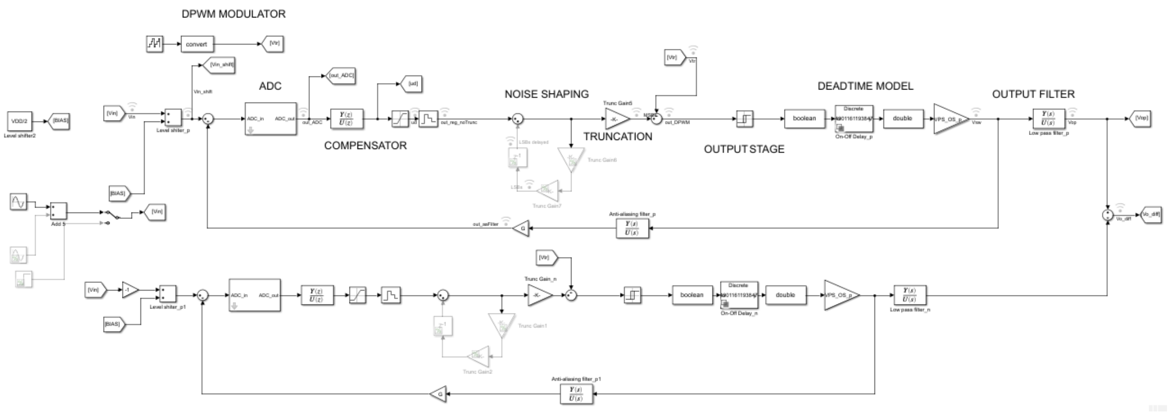


Figure 3.56: Closed-loop class D amplifier model with anti-aliasing filter in feedback path

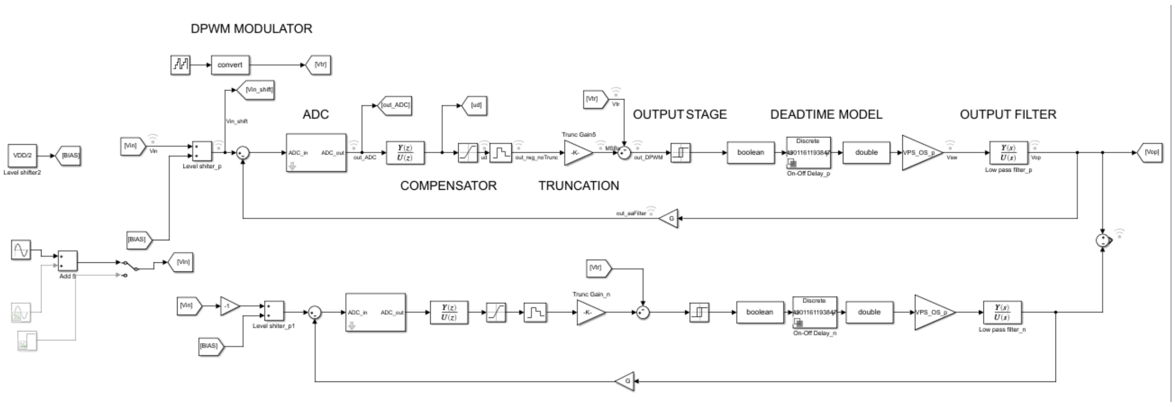
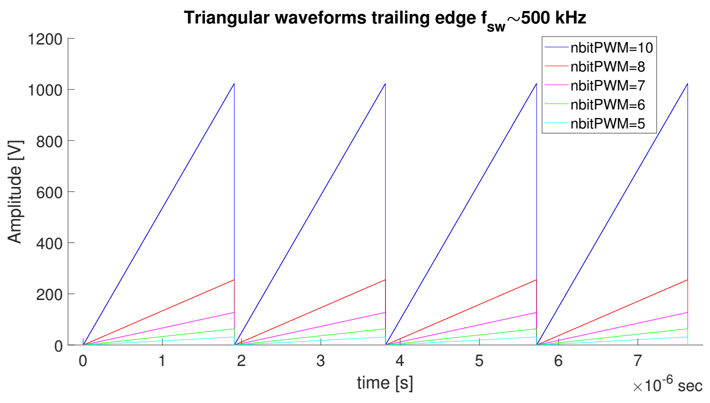
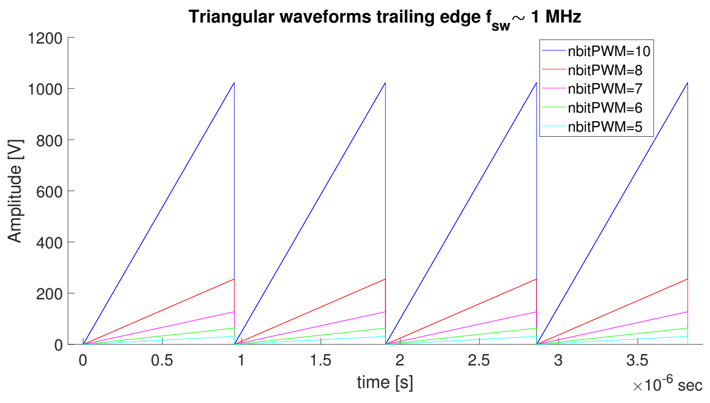


Figure 3.57: Closed-loop class D amplifier model with output filter in feedback path



(a) $f_{sw} \sim 500\,\mathrm{kHz}$



(b) $f_{sw} \sim 1\,\mathrm{MHz}$

Figure 3.58: Trailing edge triangular waveform

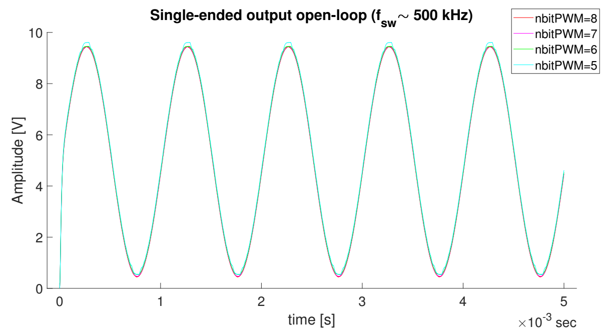
output peak voltage is given by

$$\begin{aligned} V_{o_{pk}} &= V_{in_{pk}} \frac{V_{PS}}{V_{DD}} + \frac{V_{DD}}{2}\frac{V_{PS}}{V_{DD}} = V_{in_{pk}} G_M A + \frac{V_{PS}}{2} \\ &= \underbrace{2.25\,\mathrm{V}\frac{10\,\mathrm{V}}{5\,\mathrm{V}}}_{V_{o_{SEpk}}} + 5\,\mathrm{V} = 9.5\,\mathrm{V}^2 \\ &\Rightarrow V_{o_{diff,peak}} = 2V_{o_{SEpk}} = 9\,\mathrm{V} \end{aligned}$$

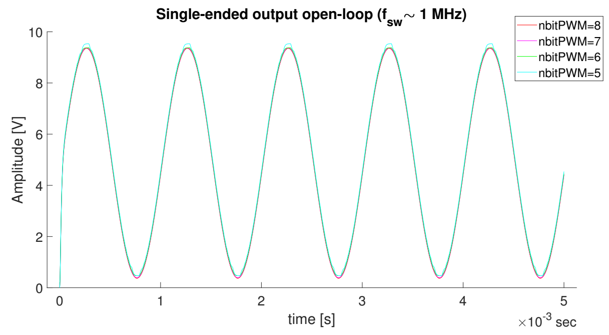
Moreover, it can be observed that at reducing of PWM bits number, the distortion increases and it is lower for higher switching frequency. Open-loop differential spectra are shown in *Appendix, at DPWM open-loop system spectra* . Basing on these spectra, THD and SNR are evaluated by means of *snr* and *thd* functions, obtaining the results shown in table 3.61. Some observations can be done:

- theoretical SNR is reduced by 6 dB increasing by one the DPWM bit number and by 3 dB when switching frequency is doubled, as expected from the theory;

[2]The dead time effect is considered negligible because $\Delta t \sim 15\,\mathrm{ns} \ll \frac{1}{f_{sw_{max}}} \sim \frac{1}{1\,\mathrm{MHz}} < 0.9\,\mu\mathrm{s}$

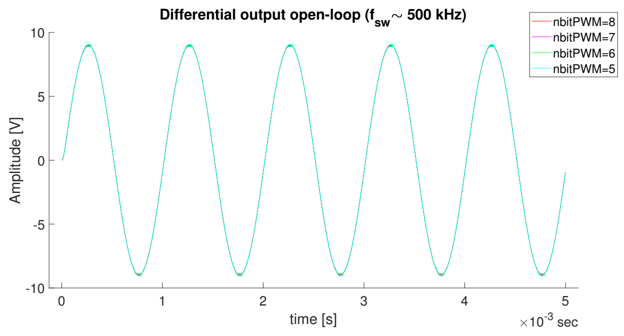


(a) $f_{sw} \sim 500\,\mathrm{kHz}$

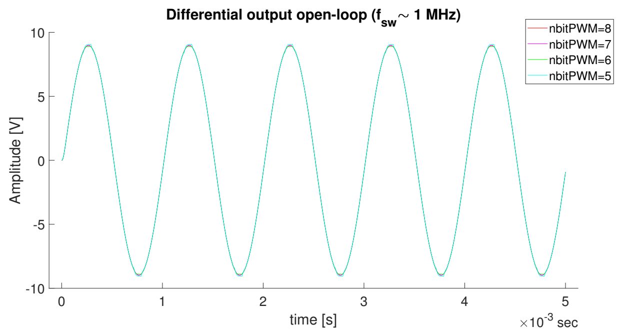


(b) $f_{sw} \sim 1\,\mathrm{MHz}$

Figure 3.59: Open-loop single-ended outputs



(a) $f_{sw} \sim 500\,\mathrm{kHz}$



(b) $f_{sw} \sim 1\,\mathrm{MHz}$

Figure 3.60: Open-loop differential outputs

# bits	f_{sw}~500 kHz			f_{sw}~1 MHz		
	SNR [dB]	SNR theory [dB]	THD [dB]	SNR [dB]	SNR theory [dB]	THD [dB]
8	64.09	66.08	-70.67	70.40	68.09	-47.26
7	61.41	59.06	-75.74	64.66	62.07	-57.40
6	55.23	53.04	-52.47	62.45	56.05	-51.27
5	53.87	47.02	-45.83	60.92	50.03	-45.53

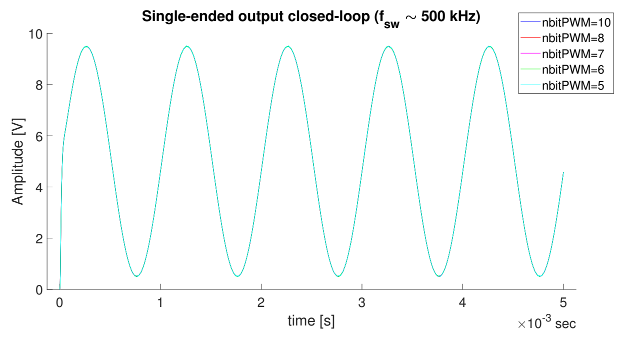
Figure 3.61: Open-loop results DPWM (differential output)

- theoretical and simulated SNRs are different among them because the non-linearities due to the PWM are not attenuated in band;
- non-linearities increase by reducing the PWM bits; in fact, comparing theoretical and simulation SNR, the difference between the two is larger when the number of these bits is reduced;
- as expected, the even harmonics included in the spectrum of the differential output are attenuated a lot, so only odd harmonics contribute to THD;
- THD increases by reducing the PWM bits number; this behaviour means that the linearity of the system is related to the number of PWM bits.

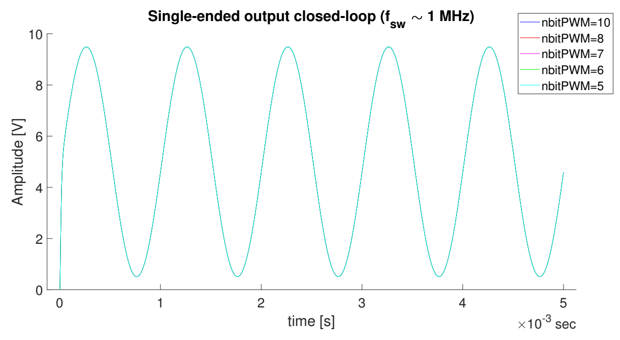
Theoretical SNR on single-ended output is computed as shown in subsection 3.6.2, making the ratio between the input power signal ($\frac{V_{o_{pk}}^2}{2}$) and the total power noise (see *Appendix*, *Code for theoretical SNR evaluation*). The SNR of differential output is derived adding 3 dB to the single-ended one, because the differential tone is twice the single-ended one (6 dB larger) and power noise increases by 3 dB, therefore the net theoretical SNR increases of 3 dB.

3.9.2 Closed-loop results of system with anti-aliasing filter in the loop

Similarly to open-loop model, even with closed-loop configuration with anti-aliasing filter in the feedback path, the simulations are performed by using the parameters in table 3.16a by varying the number of PWM bits from 10 to 5. The designed transfer functions of compensator, anti-aliasing filter and output filter are discussed in subsection 3.6.1. The single ended and differential output voltages in time domain are shown in figures 3.62 and 3.63 . As expected, the DC error is zero due to triple zero in the origin of the noise transfer function (fig. 3.23b, 3.23a). On the other, dead time cannot be increased too much, otherwise the compensator saturates. The simulations of differential output voltage spectra are shown in the *Appendix, DPWM closed-loop spectra* and from them SNR and THD of simulated systems are computed; similarly to open-loop system, theoretical SNR is evaluated too.

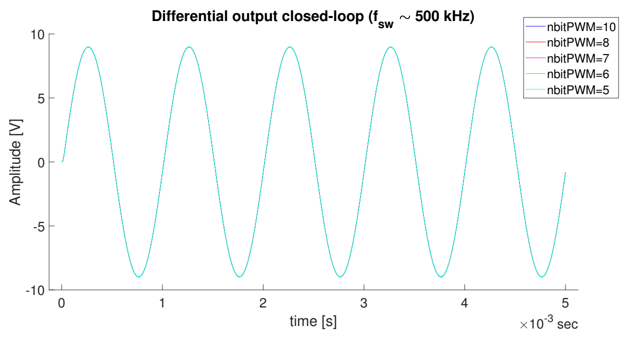


(a) $f_{sw} \sim 500\,\mathrm{kHz}$

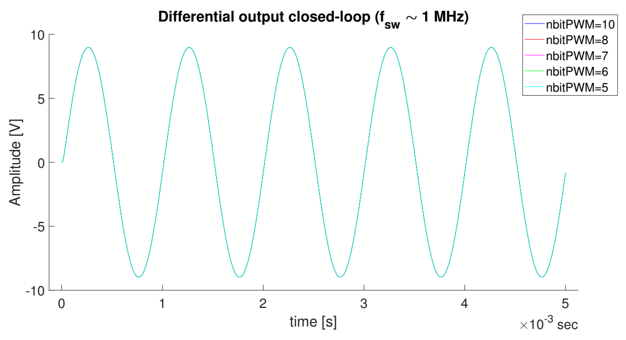


(b) $f_{sw} \sim 1\,\mathrm{MHz}$

Figure 3.62: Closed-loop single-ended outputs



(a) $f_{sw} \sim 500\,\mathrm{kHz}$



(b) $f_{sw} \sim 1\,\mathrm{MHz}$

Figure 3.63: Closed-loop differential outputs

# bits	f_{sw}~500 kHz			f_{sw}~1 MHz		
	SNR [dB]	SNR theory [dB]	THD [dB]	SNR [dB]	SNR theory [dB]	THD [dB]
10	92.69	97.82	-90.58	105.67	110.45	-102.64
8	80.36	85.80	-87.25	94.24	98.71	-101.73
7	75.57	79.78	-87.52	87.69	92.70	-102.32
6	68.39	73.76	-83.45	81.40	86.68	-89.22
5	62.18	67.74	-77.60	73.85	80.66	-86.61

Figure 3.64: Closed-loop results DPWM (differential output)

The results obtained for differential output voltages are shown in table 3.64. Some considerations can be done on these data:

- there's a constant difference of 6 dB about between theoretical and simulated SNR: this is due to the PWM aliasing; in fact the output signal from compensator has a band $[0, f_{clk_{SYS}}/2]$ while $f_{sw} \sim 1\,\text{MHz}/f_{sw} \sim 500\,\text{kHz}$; as a consequence ADC quantization noise over $f_{sw}/2$ is aliased; to reduce this non-linear effect can be increased the number and/or the order of the high frequency poles of the compesator trasfer function (reducing stability) and/or the loop gain of the feedback system (increasing ω_u and f_{sw}, obtaining larger power dissipation);

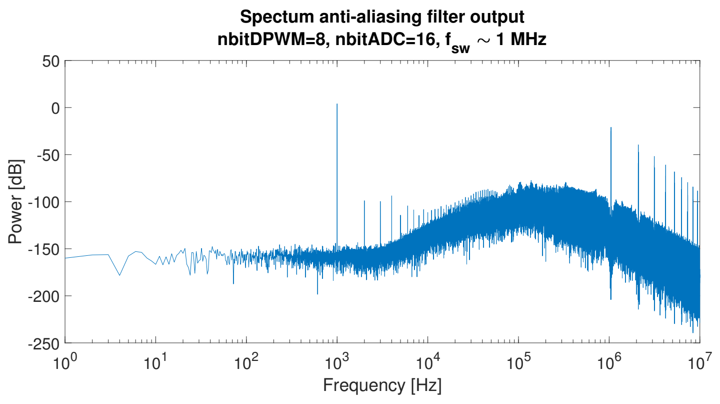
- non-linearities due to the PWM resolution are reduced by the compensator filter and their effect is shown in figures 3.66a and 3.88b, where the spectra of output anti-aliasing filter is shown for simulation at $f_{sw} \sim 1\,\text{MHz}$ with $nbit_{PWM} = 5$, $nbit_{PWM} = 8$ and $nbit_{PWM} = 10$; it can be observed that in the range between 10 kHz and 100 kHz the distribution of noise is different among the spectra and in particular spurious tones have higher power (over $-100\,\text{dB}$) when $nbit_{PWM} = 8$, while, when $nbit_{PWM} = 10$, their power does not overcome $-100\,\text{dB}$; similarly, when $nbit_{PWM} = 5$ the high frequency spurious tones are very large, reaching almost $-50\,\text{dB}$; therefore, varying the PWM bits, not only quantization noise, but even non-linear effects increase;

- as expected, THD is lower for system working at larger f_{sw}; low THDs are reached in both systems thanks to the effect of anti-aliasing filter that attenuates high frequency PWM harmonics, reducing the aliasing due to ADC and PWM sampling.

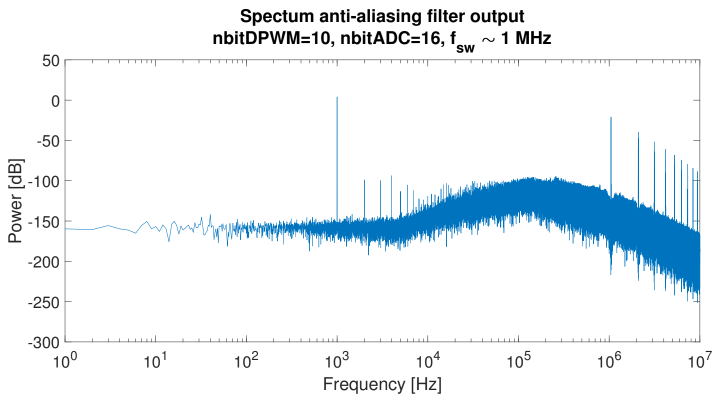
Compensator output is shown in figures 3.67 and 3.68, both for systems working at $f_{sw} \sim 500\,\text{kHz}$ and $f_{sw} \sim 1\,\text{MHz}$. The behaviour at changing of PWM bits number can be observed in the detail (fig. 3.67b and 3.68b): reducing the resolution of PWM, compensator command signal has more oscillations. This is a non-linear effect due to the different resolution between ADC and DPWM. Moreover, the oscillations are reduced for $f_{sw} \sim 1\,\text{MHz}$.

f_{sw}=500 kHz		f_{sw}=1 MHz	
SNR [dB]	**THD [dB]**	**SNR [dB]**	**THD [dB]**
94.45	-90.80	112.29	-103.53

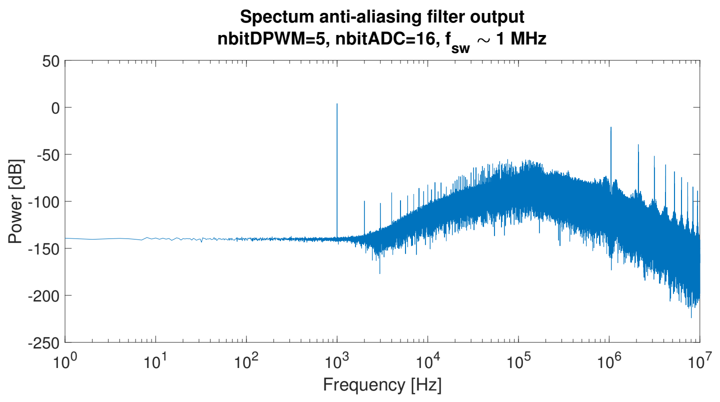
Figure 3.65: Differential output voltages performances with noise shaping ($nbit_{PWM} = 8$)



(a) $nbit_{PWM} = 8$



(b) $nbit_{PWM} = 10$



(c) $nbit_{PWM} = 5$

Figure 3.66: Output anti-aliasing filter ($f_{sw} \sim 1\,\text{MHz}$)

However, command signal never exceeds the possible dynamic range, both during the transient and at steady-state.
In conclusions, the performances of these systems reach the specifications suggested by [5] when the number of PWM bits is higher then 7; the problem is that $f_{clk_{PWM}}$ is very large, in particular for the solution at $f_{sw} \sim 1\,\mathrm{MHz}$ with 7 and 8 DPWM bits, increasing power dissipation. Moreover, the low resolution of PWM reduces the SNR, requiring the increase of compensator loop gain in band, at the price of complicated design [25] or switching frequency increasing [16], again loosing efficiency.

Noise shaping

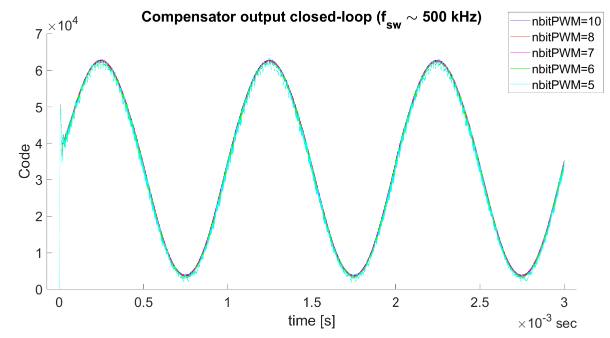
Simulations at $f_{sw} \sim 500\,\mathrm{kHz}$ and $f_{sw} \sim 1\,\mathrm{MHz}$ with $nbit_{PWM} = 8$ and first-order noise shaping is performed. As shown in figure 3.56, the truncated LSBs are recovered and delayed by T_{sw} and summed to the next sample processed by DPWM. This solution has the effect to increase the SNR in audio band, moving to high frequency the quantization noise. Due to the presence of loop, this additional high frequency noise can be injected in audio band due to aliasing effect; therefore, the theoretical increase of SNR is higher than the results of simulations (table 3.65). Spectrum results are shown *Appendix, DPWM closed-loop spectra.*

3.9.3 Closed-loop system with output filter in the feedback path

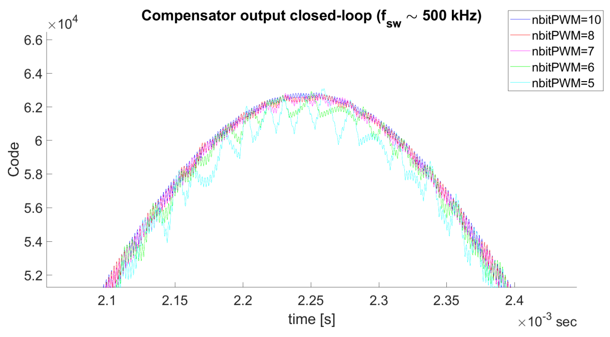
The solution in figure 3.57, which has been designed as shown in subsection 3.6.3, is tested with the parameters simulation in table 3.16a. As previuosly pointed out, the large compensator bandwidth has the consequence to increase the overshoot of the command signal, creating saturation problem during the transient that can bring instability. This is what happens for system working at $f_{sw} \sim 1\,\mathrm{MHz}$: the overshoot is so high that the compensator cannot recover from non-linear condition. On the other hand, the solution at $f_{sw} \sim 500\,\mathrm{kHz}$ results stable after the initial transient.
This instability can be reduced in two ways:

- reducing the integrator order: this means reducing the effect on PWM quantization noise in band;
- increasing the order of the high frequency pole: this means reducing the stability of the overall system.

In the designed case (where integrator order $p = 3$) SNR and THD parameters are evaluated only for the case at $f_{sw} \sim 500\,\mathrm{kHz}$ with $nbit_{PWM} = 8$. Comparing table 3.69 with table 3.64, it can be seen that the SNRs are very similar: this is due

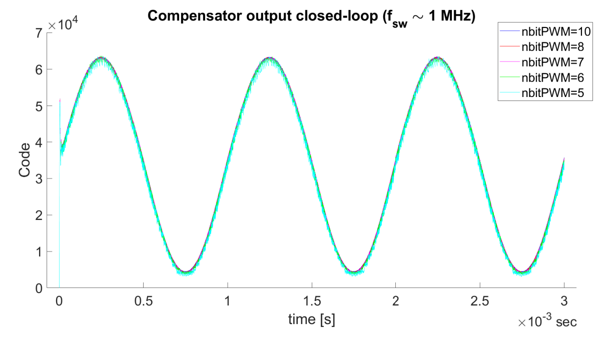


(a) *Command signal*

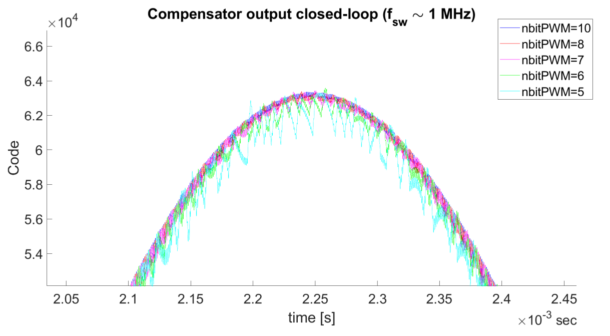


(b) *Detail around the peak of command signal*

Figure 3.67: Command signal ($f_{sw} \sim 500\,\text{kHz}$)



(a) *Command signal*

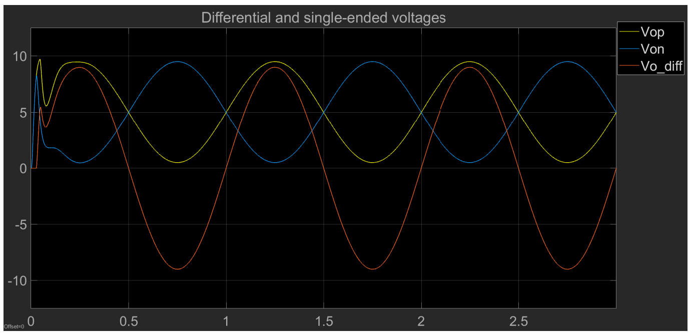


(b) *Detail around the peak of command signal*

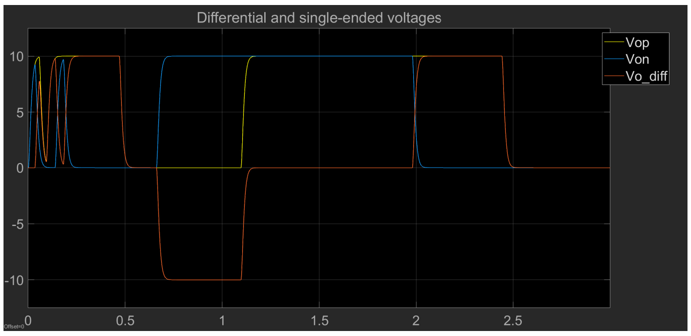
Figure 3.68: Command signal ($f_{sw} \sim 1\,\text{MHz}$)

$f_{sw}\sim$ 500 kHz			
#bit$_{PWM}$	**SNR [dB]**	**SNR theory [dB]**	**THD [dB]**
8	79.20	84.94	-69.51

Figure 3.69: SNR and THD of differential voltage of solution with output filter in feedback path



(a) *Output voltages* $f_{sw} \sim 500\,\mathrm{kHz}$



(b) *Output voltages* $f_{sw} \sim 1\,\mathrm{MHz}$

Figure 3.70: Output voltages with $nbit_{PWM} = 8$

to the fact that the integrator design strategies are the same in the two solutions. On the other hand, the difference on THD is due to the absence of high frequency pole of anti-aliasing filter: in fact, this double pole allows to attenuate higher PWM harmonics, further reducing PWM aliasing effect.
The spectrum of differential output of simulated solution at $f_{sw} \sim 500\,\mathrm{kHz}$, $nbit_{PWM} = 8$ (in figure 3.70a) is shown in *Appendix, DPWM closed-loop spectra*: as in the spectra of differential output in the solution with anti-aliasing filter in feedback path, the even harmonics result have low amplitudes, making THD strongly dependent on the power of the odd harmonics.

3.9.4 Comparison between the two solutions

Comparing the proposed solutions in subsections 3.9.2 and 3.9.3, the one with anti-aliasing filter in feedback path results with lower THD. Actually, this parameter can

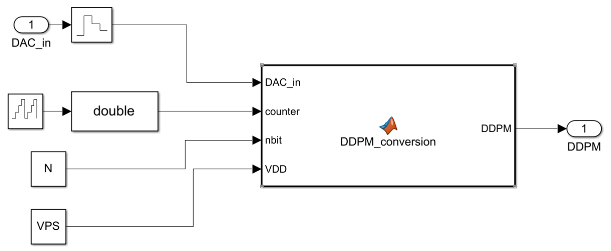


Figure 3.71: DDPM model

be improved also for the solution in subsection 3.9.3 adding the same anti-aliasing filter in the feedback path, with a weak reduction of phase margin. Anyway, the problem of the overshoot that trigger non-linearities, makes the design of solution with output filter in feedback path strongly dependent on the high frequency poles and integrator order of the compensator and, as a consequence, on the switching frequency f_{sw}. On the other hand, the solution of subsection 3.9.2 results more stable at varying of f_{sw}, integrator order and non-linearities, obtaining a command signal that does not exceed the possible dynamic range. Therefore, this solution is the one used from now on to investigate the effect of different modulation type on the designed class D amplifier model.

3.10 Class D amplifier model with DDPWM modulation

The implementation of in Simulink® environment of the model of the DDPM and DPWM modulators basing respectively on [20] and [23] is described in this section. Then the models are simulated and results are compared with those reported in [20],[21],[23]; after that, DDPWM is inserted into the model of class D amplifier with anti-aliasing filter in feedback path and performances are compared with those obtained by the configuration that include the DPWM modulator reported in section 3.9.

3.10.1 DDPM model

The DDPM modulator is implemented with a *MATLAB function* block that mimics the operation of the hardware architecture based on priority multiplexer shown in [20]. The code of the function is shown and commented in *Appendix, 5.2.6.* The tested model is shown in figure 3.71 and the testbench scheme is shown in figure 3.72 The *Mask* parameters of DDPM block are

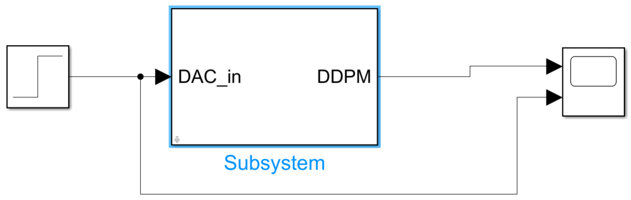


Figure 3.72: DDPM testbench

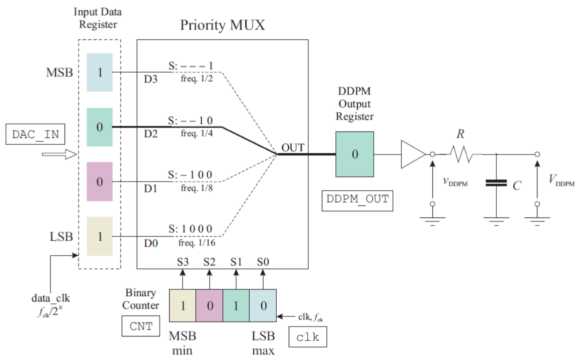


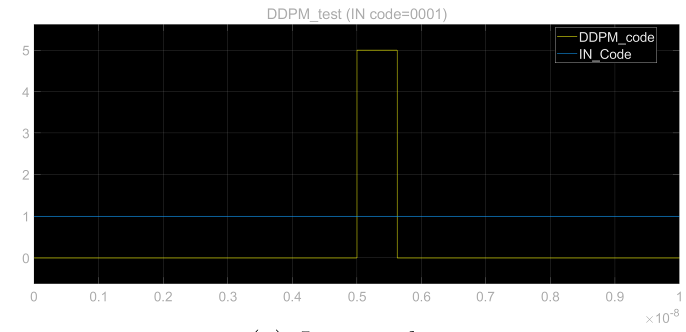
Figure 3.73: DDPM architecture with priority multiplexer [20]

- *switching frequency* f_{sw} of the free-running counter, from which the clock frequency $f_{clk} = 2^{nbit_{DDPM}} f_{sw}$ is set;
- *Number of bits* $nbit_{DDPM}$ of the free-running counter;
- *power supply* V_{PS} to obtain at the output a DDPM signal which can be assumed 0 or V_{PS} values.

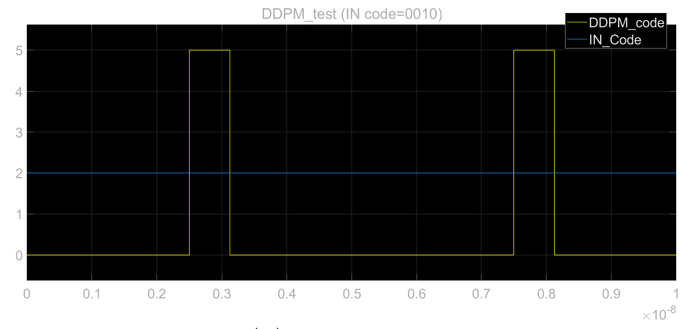
Given these parameters, the *DDPM converter MATLAB function block* works as a priority multiplexer, generating the DDPM pattern corresponding to the input code (*Step function*). DDPM modulator is tested setting 4 bits resolution and a clock frequency $f_{clk} = 100\,\text{MHz}$. In order to make a comparison with DDPM code shown in [20] (fig. 2.36) the simulations are performed for 8, 4, 2, 1, 10 input code. As shown in figures 3.74 the results fully agree with those reported of literature, with the only difference that the last zero of DDPM pattern of [20] becomes the first in the simulation results.

3.10.2 DDPWM model

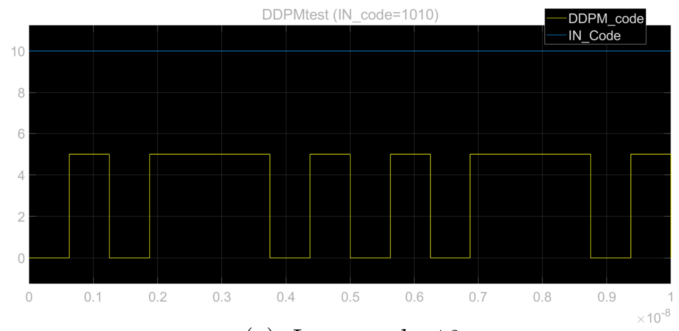
The DDPWM Simulink® model (figure 3.75b) is based on the architecture shown in figure 3.75a and requires the DDPM subsystem (figure 3.75b). The architecture in [23] and the Simulink® model are the same with the only difference that the input register (modelled by a ZOH) is left out the modulator and the MSBs and the



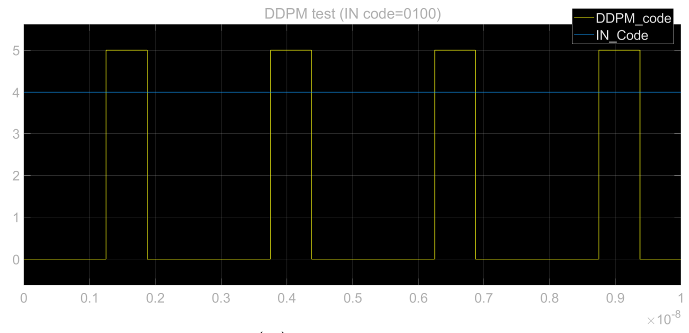
(a) *Input code 1*



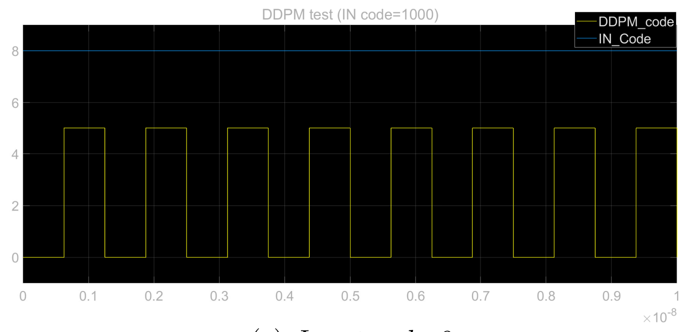
(b) *Input code 2*



(c) *Input code 10*



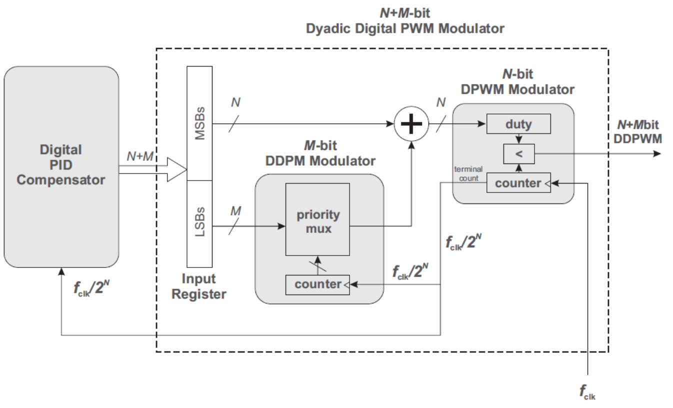
(d) *Input code 4*



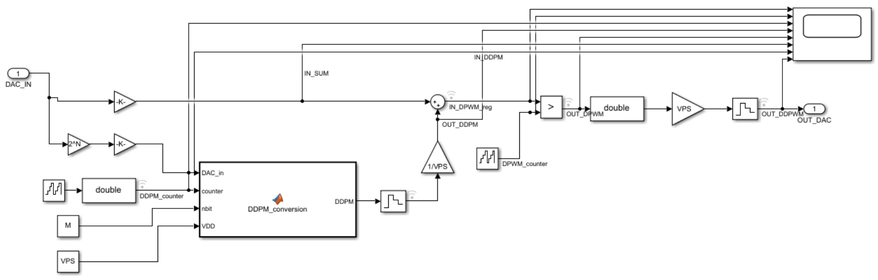
(e) *Input code 8*

Figure 3.74: DDPM code at varying of input code

recover of LSBs is made by using the *gain* block as shown in subsection 3.7.2. In particular, given N MSBs and M LSBs, the MSBs are recovered by multiplying the input code by $\frac{1}{2^M}$, with the output number represented on N bits unsigned, while LSBs are recovered multiplying input code by 2^N and limiting the representation on $N + M$ bits and then multiplying the resulting number by $\frac{1}{2^N}$, limiting the representation of the input code to *DDPM converter* to M bits. The ZOH at the output of DDPM converter works at $\frac{f_{clk_{PWM}}}{2^N}$, the same frequency of DDPM counter and the *gain* block with $gain = \frac{1}{V_{PS}}$ allows to scale the dynamic of DDPM sequence between 1 and 0 (so in digital value).



(a) *DDPWM architecture of* [23][21]



(b) *DDPWM Simulink® model*

Figure 3.75: Comparison between DDPWM architecture and model

The parameters of DDPWM model are

- *Power supply*;
- *Clock frequency* $f_{clk_{PWM}}$ of DPWM counter;
- *LSBs number* M, from which the DDPM clock frequency is internally computed;
- *MSBs number* N, that are sent to DPWM section.

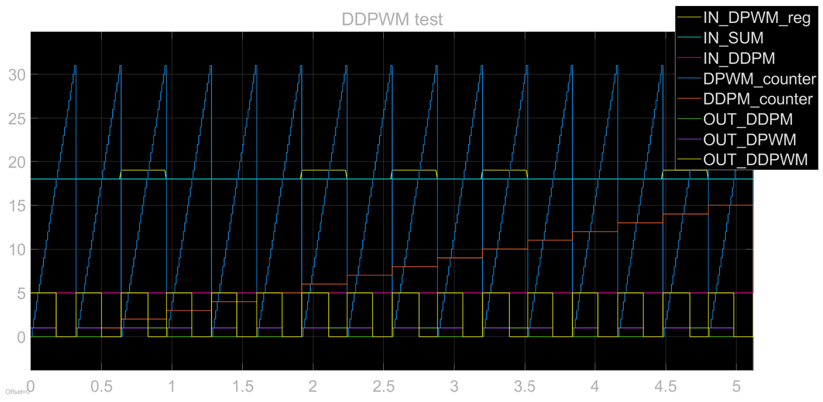


Figure 3.76: *Timing diagram of DDPWM test ($M = 4$, $N = 5$, $f_{clk_{PWM}} = 100\,\mathrm{MHz}$)*

In particular, to test the DDPWM model, M is set to 4, $N = 5$, *power supply* is set to 5 V and DPWM clock frequency $f_{clk_{PWM}} = 100\,\mathrm{MHz}$. To compare the results with the ones shown in [23], the input code is set to 100100101 (i.e. 293 in decimal representation) and the *Stop time* of simulation is set to $\frac{1}{f_{clk_{PWM}}}2^{N+M}$ to visualize an entire period of DDPWM conversion. The timing diagram of the simulation is shown in figure 3.76. Can be observed that the DPWM counter works effectively at $f_{clk_{PWM}}$ and the DDPM counter reaches the value of $2^M - 1$ after 2^M periods of DPWM counter, as expected from [23]. Moreover, the DDPWM pattern is the same of the one shown in figure 2.38a, confirming the correctness of the model.

3.10.3 DDPWM modulator applied to CDA model

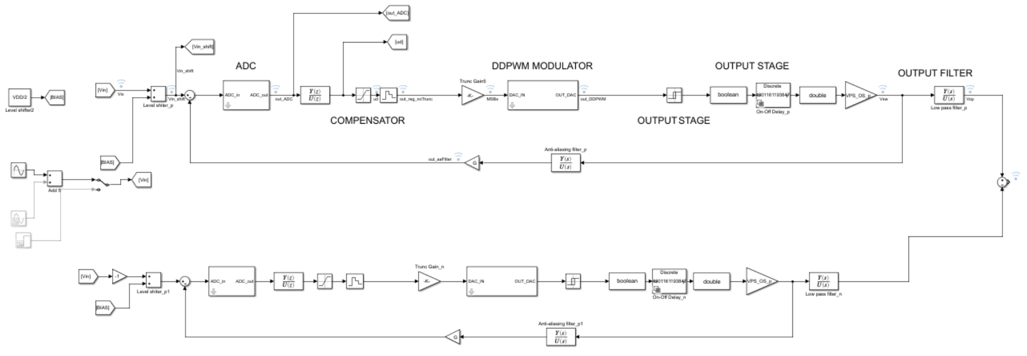
The DPWM modulator is replaced by the DDPWM one (figures 3.77a, 3.77b) and simulations are performed. Nothing changes about the loop design: the gain model of DDPWM remains the same of the DPWM ($G_M = 1/2^{nbit_{DPWM}}$). The only parameter that changes is the truncation gain which becomes $\frac{2^{N+M}}{2^{nbit_{ADC}}}$ for closed-loop and $\frac{2^{N+M}}{2^{nbit_{ADC}-1}}$. The results of simulations at $f_{sw} \sim 500\,\mathrm{kHz}$ for closed- and open-loop are shown in table 3.78, where the number of LSBs is fixed to $M = 4$ and N is 5 or 8. Therefore, the DDPWM resolution results respectively of 9 and 12 bits.
Comparing with tables 3.64 and 3.61 there is a reduction of performances, both in open- and closed loop configurations

- SNR is reduced in both configurations; this effect can be explained observing the timing diagram of figure 3.79; because of DDPWM input changes each T_{sw}, it changes too much rapidly to be modulated by the DDPM. In fact input code has to be constant at least one period of DDPM, otherwise the DDPWM modulation lost its advantages;

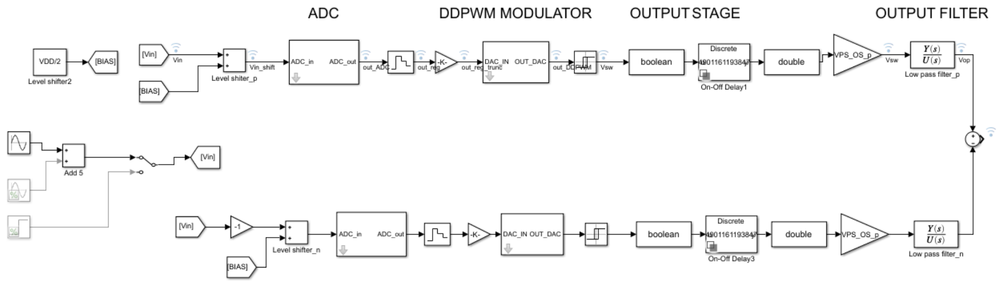
- the THD of open loop CDA is increased instead, and higher harmonics results under the noise level;

- in closed-loop system, there are no substantial differences with respect to DPWM solution.

Moreover, simulation with $n_{bit_{ADC}} = 7$ and $N = 7$ and $M = 2$ are performed applying a constant input signal. The results in time domain are shown in figure 3.80a, where it is visible the LCO behaviour on output voltages: higher resolution has not solved the LCO problem. This is due to the fact that the three conditions to eliminate LCO suggested in [24] are derived for a PID controller, while, in this case, the transfer function of designed controller is an IIR; moreover, the condition on integrator gain ($K_i < 1$) is not respected: integrator gain of class D amplifier has to be as larger as possible to reduce distortion and non-linearities in audio band. Even on compensator signal LCO phenomena is clearly visible (figure 3.80b).



(a) *Closed-loop CDA model with DDPWM modulator*



(b) *Open-loop CDA model with DDPWM modulator*

Figure 3.77: CDA model with DDPWM modulator

MSBs (N)	**SNR [dB]**	**THD[dB]**
5	43.14 dB	−79.13 dB
8	60.25 dB	−93.29 dB

(a) *DDPWM open-loop system (M=4)*

MSBs (N)	**SNR [dB]**	**THD[dB]**
5	59.72 dB	−75.64 dB
8	77.28 dB	−88.65 dB

(b) *DDPWM closed-loop system (M=4)*

Figure 3.78: Noise and distortion parameters of CDA with DDPWM simulations (differential output voltages, $f_{sw} \sim 500\,\text{kHz}$)

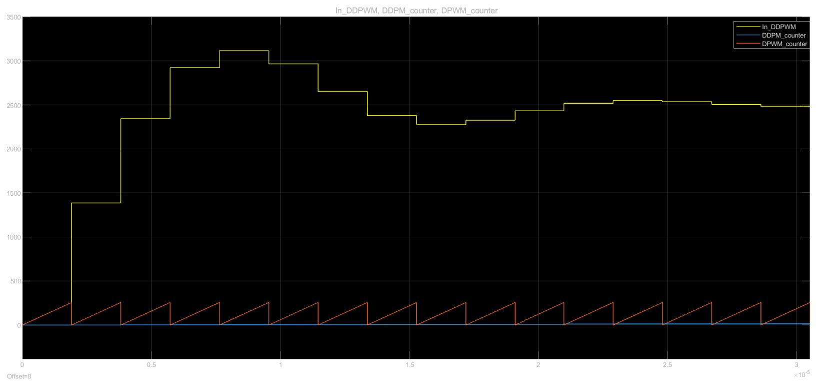
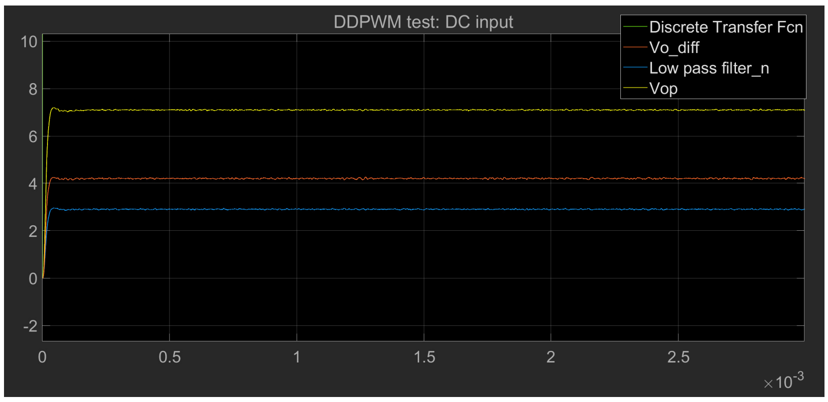
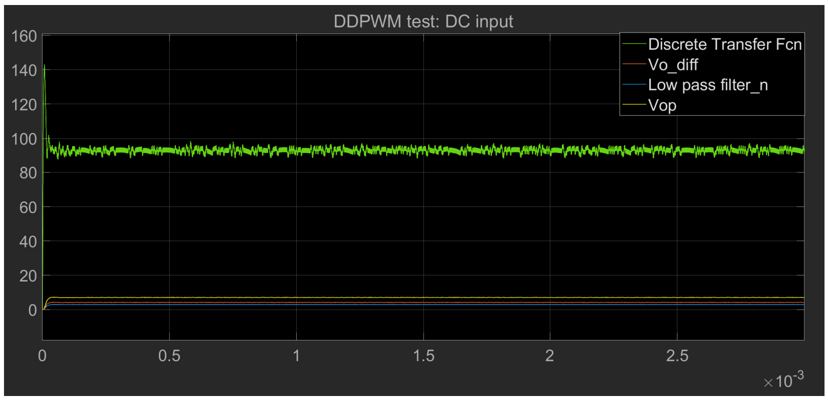


Figure 3.79: Detail of DDPWM modulation



(a) *DC output voltage*



(b) *DC output compensator*

Figure 3.80: DC simulation results ($n_{bit_{ADC}} = 7$, $M = 5$, $N = 5$)

3.11 DDPM-DPWM combination

To overcome the limitation of DDPWM modulation in class D amplifier, a combination between DDPM and DPWM modulations is proposed: the idea is to modulate the N MSBs of the input code with DPWM modulator working at switching fre-

quency $f_{sw} = 2^N f_{clk_{PWM}}$ and the N LSBs with a DDPM modulator working with a clock frequency $f_{clk_{PWM}}$; the output stream of DPWM modulator drives the final stage, assuming 0 or V_{PS} (output stage power supply) values, while the DDPM stream can assume 0 and $V_{PS}/2^N$ values, driving a low-power power stage (e.g. output logic port of an FPGA/microcontroller). Then the two streams are summed and sent to the output filter and feedback path (in case of closed-loop system). In this way the effective resolution is increased of 2^N working at the same clock frequency of DPWM; moreover, differently from DDPWM modulation the DDPM stream is concluded each T_{sw} period.
To test this modulator the testbench shown in figure 3.82 is used with the parameters listed in table 3.81.
The gain blocks are used to recover the LSBs and the MSBs. The results of timing digram in figure 3.83 are coherent with the ones expected: because the input code is 10101010 the MSBs and the LSBs are the same number (1010, 10 in decimal representation) and in the timing diagram they appear superimposed. The DDPM sequence is exactly the same shown in figure 3.74c and the maximum value reached is $V_{PS}/2^N = 5\,\mathrm{V}/16 = 0.3125\,\mathrm{V}$. Moreover, the DDPM sequence ends each T_{sw}, showing that DPWM and DDPM sections are synchronized.

$\mathbf{f_{sw}}$	100 Hz
$\mathbf{f_{clk_{PWM}}}$	1600 Hz
$\mathbf{n_{bit_{DPWM}}}$	4
$\mathbf{n_{bit_{DDPM}}}$	4
IN code	10101010 ($170\|_{dec}$)

Figure 3.81: DDPM-DPWM combination test parameters

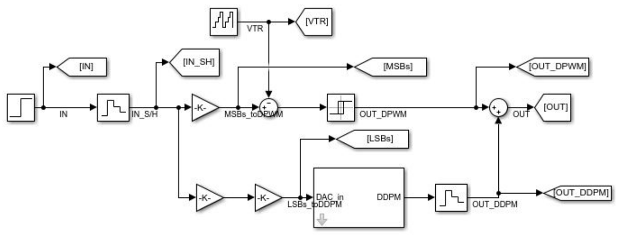


Figure 3.82: Testbench of DDPM-DPWM combination modulation

3.11.1 DDPM-DPWM combination applied to class D amplifier model

The modulation section of the class D amplifier model in closed-loop configuration is implemented with DDPM-DPWM combination and simulations with different

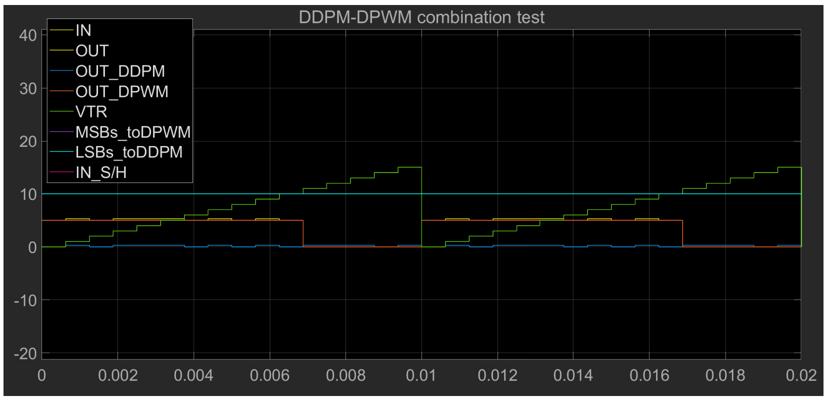


Figure 3.83: Testbench results

number of bits of DPWM section (8, 7, 6, 5) and switching frequencies ($f_{sw} \sim 500\,\mathrm{kHz}$, $f_{sw} \sim 1\,\mathrm{MHz}$) are performed. Because of the DDPM section works with the same LSBs number of DPWM section, the effective resolution of the modulator is twice the number of bits of DPWM. Therefore, with 8, 7, 6, 5 number of bits of DPWM the total resolution is respectively 16, 14, 12, 10 bits. The simulated model is shown in figure 3.85 and the results in terms of SNR and THD are shown in table 3.87. Differential output spectra are shown in *Appendix, 5.1.3.*
The first gain block, in the gain chain before DDPM, is needed to truncate input code on 2^{N+N} bits and then, the next two gain blocks, recover the LSBs as shown in figure 3.82. Can be seen that the final stage is only driven by the DPWM modulator, so works at f_{sw}. To further test the correctness of the modulator model, a short simulation is performed on open-loop CDA model with $n_{bit_{ADC}} = 8$ and $n_{nbi_{PWM}} = n_{bit_{DDPM}} = 4$. The results are the one expected, shown in figure 3.86

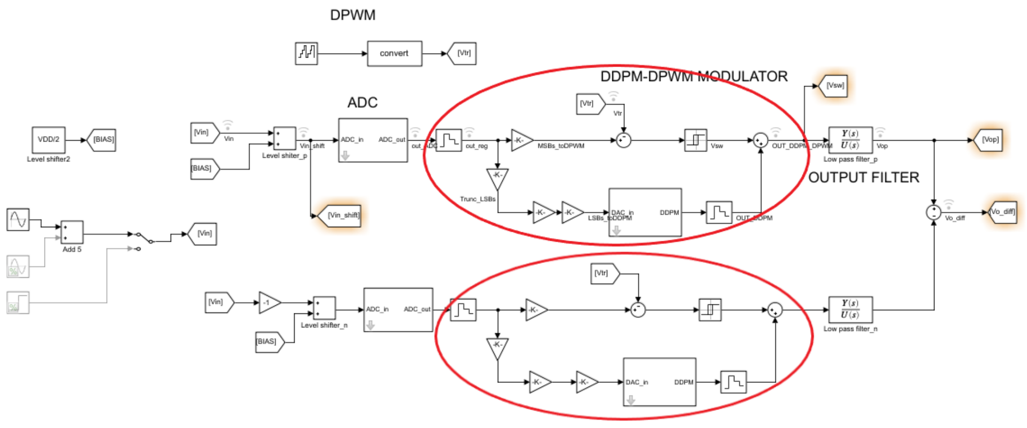


Figure 3.84: Open-loop CDA with DDPM-DPWM combination modulator

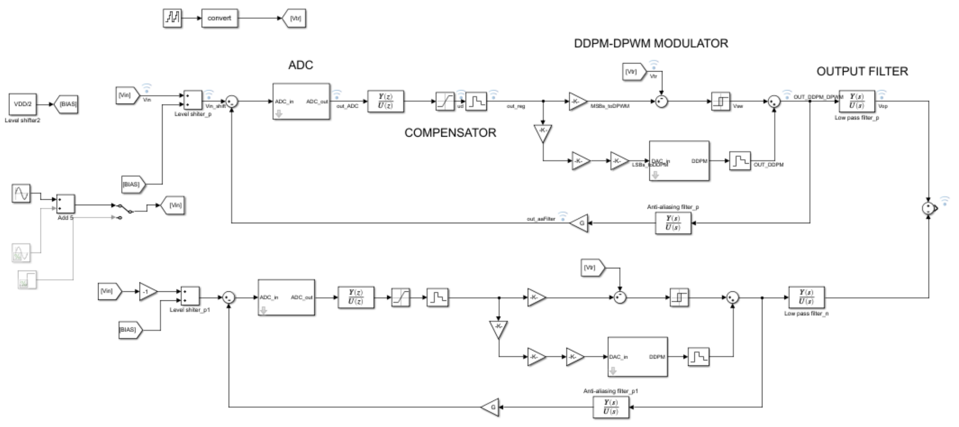


Figure 3.85: Closed-loop CDA with DDPM-DPWM combination modulator

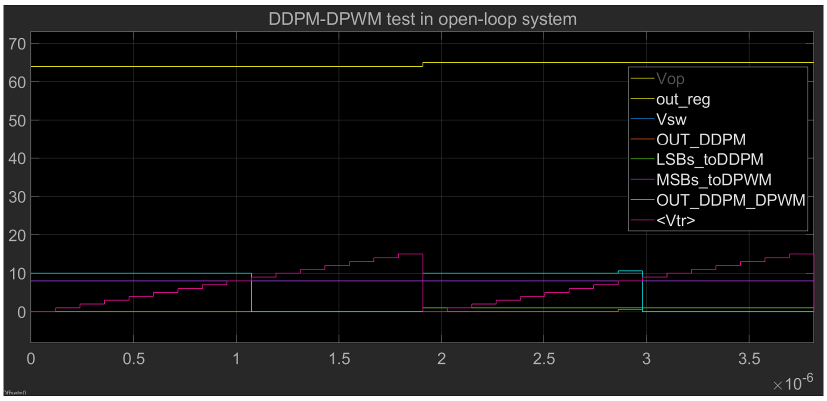


Figure 3.86: Test of DDPM-DPWM combination modulator applied on open-loop CDA ($n_{bit_{ADC}} = 8$, $n_{nbi_{PWM}} = n_{bit_{DDPM}} = 4$)

# bits	f_{sw}~500 kHz			f_{sw}~1 MHz		
	SNR [dB]	**SNR theory [dB]**	**THD [dB]**	**SNR [dB]**	**SNR theory [dB]**	**THD [dB]**
8	102.91	121.14	-92.58	119.78	121.96	-105.88
7	100.64	118.63	-92.80	118.31	121.75	-106.03
6	98.07	109.58	-93.16	113.66	119.36	-106.00
5	86.15	97.82	-69.01	100.43	110.45	-80.34

Figure 3.87: Closed-loop results DDPM-DPWM combination

Results of closed-loop system on differential output show a strong increase of performances (tabel 3.87):

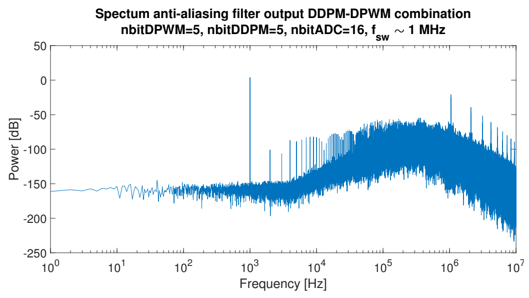
- for $f_{sw} \sim 1\,\text{MHz}$ and high number of bits of modulator (8, 7) the performances are very similar to the theoretical one; reducing the number of bits the non-linearities of DPWM resolution becomes not negligible and the difference between theoretical and simulation SNR increases;

- for $f_{sw} \sim 500\,\mathrm{kHz}$ the difference between theoretical and simulation results are larger because the compensator integrator gain is lower than the case working at high switching frequency, therefore non-linearities are less attenuated;
- comparing the case at 5 bits with DDPM-DPWM combination (so 10 bits resolution) with the case at 10 bits DPWM in table 3.64 the difference between simulation results is about 5 dB. This effect can be explained observing the spectra of anti-aliasing filter output of DDPM-DPWM combination and DPWM both at 10 bits resolution (figures 3.88); it can be observed that in DDPM-DPWM combination case spurious tones are much larger than DPWM case; this is due to the fact that the resolution of DPWM in DDPM-DPWM combination case is of 5 bits, so non-linearities are much stronger than a pure DPWM modulation working at 10 bits; therefore the SNR is limited by the non-linearities DPWM resolution;
- THD is reduced of about 4 dB for 6, 7, 8 DPWM number of bits; for 10 bit resolution (5 bits for DPWM and 5 bit for DDPM) THD dramatically decreases with respect to DPWM solution at 10 bits; again, this is due to the non-linear effect of low resolution DPWM, which inject spurious tones in audio band.

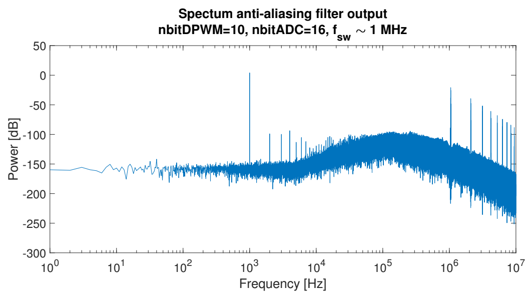
Concluding, this new modulation that combines DDPM and DPWM has brought lot of advantages in terms of noise and distortion because the resolution can be easily increased without increasing switching and clock frequency. As a consequence, even power dissipation due to turn on and turn off events remains basically the same.

3.12 Results conclusion

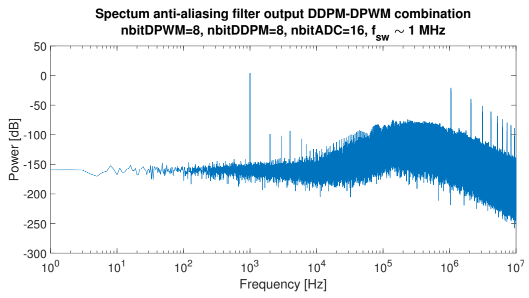
In this chapter are shown the models used to implement a class D amplifier with global digital feedback, from ADC to DPWM, from output stage to ZOH, compensator and analog filters. Then a design methodology described step by step is shown and applied to two possible feedback architectures: one with the output filter in feedback path, taking the feedback signal at the output of the filter, and the other which consists into taking the feedback signal from output stage and filtering it by an anti-aliasing filter. For both solutions are illustrated advantages and disadvantages, showing that the first one have problem of stability due to the large overshoot of the command while the second one results more independent from switching frequency parameter and the command never reaches the maximum value ($2^{n_{bit_{ADC}}}$). At the same time, the bottleneck of this kind of systems is shown: resolution of DPWM reduces the performances in terms of SNR and THD. In particular, due to non linear effects, DPWM resolution influences both THD and SNR. Due to the



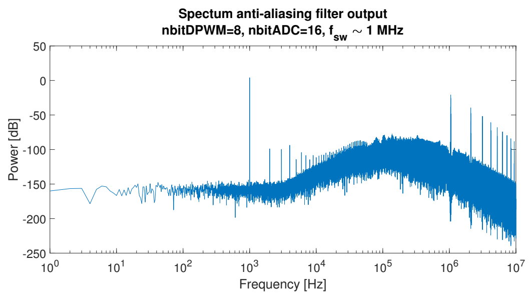
(a) $n_{bit_{DPWM}} = 5$, $n_{bit_{DDPM}} = 5$



(b) $n_{bit_{PWM}} = 10$



(c) $n_{bit_{DPWM}} = 8$, $n_{bit_{DDPM}} = 8$



(d) $n_{bit_{PWM}} = 8$

Figure 3.88: Anti-aliasing filter output spectra (DPWM vs DDPM-DPWM combination)

limitation in switching frequency, compensator gain cannot be increased so much, as done in [16] where the switching frequency is over 2 MHz; as a consequence different type of modulations are investigated and applied to this class D amplifier structure. In particular, DDPWM, which is applied to buck converter overcoming the trade-off between switching frequency and resolution of DPWM [23][21], in class D amplifier model has not produced any particular increase of performances because of the different compensator design methodology and large variation of input signal (sinusoidal wave), differently from buck (DC input voltage). To overcome the limitation of DDPWM a combination between DDPM and DPWM is suggested, showing that advantages in terms of SNR and THD can be reached without increasing switching and clock frequency of modulator and, therefore, without increasing power dissipation.

Chapter 4

Conclusions

By means of simulations and Simulink® models, a new kind of modulation scheme based on DDPM and DPWM is implemented, reaching high performances without increasing switching frequency and with a relative low loop gain in audio band.
In particular, starting from results and designs found in the literature, it is shown that class D amplifiers have lot of advantages in terms of portability and power efficiency with respect to linear classes. This allows to use this kind of amplifiers in many applications like cell phones, medical acoustic applications, personal computers and portable devices, reducing the waste of energy. Anyway the intrinsic non-linearity of these amplifiers requires the careful design of a closed-loop system using integrator as the compensator filter in order to reach high-fidelity audio performances. This compensator has to be characterized by a very large integrator gain in the bandwidth of audio signals, which in analog architecture is limited by the characteristic of the employed operational amplifier [4][3][5]. In recent years, the development of digital architectures have brought the possibility to implement very complex transfer functions, moving this application in the field of class D amplifiers [14][16]. From this state of the art of class D amplifiers, this thesis investiagted and implemented a model of CDA by discussing several issues and bottlenecks and illustrate how to overcome them, by describing a methodology design of the closed-loop inspired by [5]. In particular is found that the two main drawbacks of these kinds of systems are

- requirement of large loop gain which has as a consequence the requirement of large switching frequency [16] or very complex loop design [14];
- DPWM quantization noise is the most problematic noise and introduces non-linearities in band;

Then, the new DDPWM modulation introduced in literature in the field of power converter, which aim is to increase DPWM resolution overcoming the trade-off between resolution and switching frequency, is applied to CDA model substituting the traditional DPWM one: the results have shown that no particular improvement of performances are introduced because the variation of the input signal is too fast with respect to the DDPM pattern period, nullifying the advantage of DDPWM modulation. Therefore, an alternative modulation scheme based on DDPM and DPWM combination is proposed, bringing a strong increase of performances, in particular on modulator resolution which is increased by 2^N quantization levels, where $2N$ is the modulator input code number of bits. As a consequence, SNR is strongly increased with respect to previous solutions and also THD is slightly increased. In conclusions, this new proposed combination of DDPM and DPWM modulations allows to

- increase resolution of modulator, which in DPWM traditional solution is the bottleneck of the system in terms of SNR and THD;
- maintain the same switching and clock frequencies of DPWM solution, avoiding an increasing in power dissipation;
- reduce the requirement of integrator loop gain to reduce quantization DPWM noise.

Starting from these results, a physical implementation using GaN and MOS devices could be designed with the further goal to investigate the effect of GaN devices in a closed-loop fashion class D amplifier. Digital section (compensator and modulator) could be implemented by mean of FPGA. Moreover, psychoacoustic study to compare the implemented class D amplifier with one of the traditional classes (A, B or AB) can be done, observing the perception of audio quality by human hearing.

Chapter 5

Appendix

In this chapter are shown all the spectra evaluated by the simulations results and the codes used to simulate, design the systems and to make analysis on data obtained by the simulations, in particular waveforms and spectra.
About the codes, they have to be used modifying the interesting variables (e.g. switching frequency, number of bits of modulator); therefore the variables where the results are saved, changes name depending on the parameters used. For example, in code 5.2.1, the number of bits of DPWM is $nbit_{PWM} = 8$ and $f_{sw} \sim 1\,\mathrm{MHz}$, so the code is the following one

```
Fsw=1e6;
Fsw=2^round(log2(Fsw));%Hz; switching frequency
nbit_PWM=8; %number of bits for the quantized triangular waveform
nbit_ADC=16; %number of bits for ADC
fclk_PWM=Fsw*2^nbit_PWM; %Hz clock frequency of PWM generator
```

⋮

```
èè
%%
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_PWM_dB=Pn_PWM_dB;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_ADC_dB=Pn_ADC_dB;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_PWM_cl=Pn_dB_PWM_cl;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_ADC_cl=Pn_dB_ADC_cl;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_tot_cl=Pn_tot_dB_cl;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.SNR_dB_cl=SNR_dB_cl;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_PWM_ol=Pn_dB_PWM_ol;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_ADC_ol=Pn_dB_ADC_ol;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_tot_ol=Pn_tot_dB_ol;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.SNR_dB_ol=SNR_dB_ol;
save('DatastoreFiles/SNRandPn_nbitADC16_TrailingEdge.mat',...
    'SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath', '-append')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%save parameters simulations
TE_PWM8_fsw1M.Fsw=Fsw;%save switching frequency
TE_PWM8_fsw1M.fclk_PWM=fclk_PWM;%save PWM TE clock
TE_PWM8_fsw1M.t_int=t_int;%save integration time
```

```
TE_PWM8_fsw1M.dec_fac=dec_fac;%save decimation factor
TE_PWM8_fsw1M.t_int_dec=t_int_dec; %save sample time of system outputs (except
    triangular waveform)
[Gm, Pm, Wcg, Wpg]=margin(T_filter_z);%evaluate gain and phase margins with
    correspondent frequency
marginPWM8_fsw1M.Gm=Gm;
marginPWM8_fsw1M.Pm=Pm;
marginPWM8_fsw1M.Fcg=Wcg/(2*pi);
marginPWM8_fsw1M.Fpg=Wpg/(2*pi);
StepInfo_fsw1M_out=stepinfo(step_Hfilter_z,t_step);
StepInfo_fsw1M_comp=stepinfo(step_Gfilter_z,t_step);
save('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat',...
    'TE_PWM8_fsw1M', 'marginPWM8_fsw1M', 'fclk_SYS', 'StepInfo_fsw1M_out',...
    'StepInfo_fsw1M_comp', '-append')%save simulation and loop tf parameters
```

To save the variables related to SNR, phase margin and step responses of the theoretical design for $nbit_{PWM} = 5$ and $f_{sw} \sim 500\,\text{kHz}$, the code has to change in the following way

```
    Fsw=1e6;
Fsw=2^round(log2(Fsw));%Hz; switching frequency
nbit_PWM=8; %number of bits for the quantized triangular waveform
nbit_ADC=16; %number of bits for ADC
fclk_PWM=Fsw*2^nbit_PWM; %Hz clock frequency of PWM generator
```

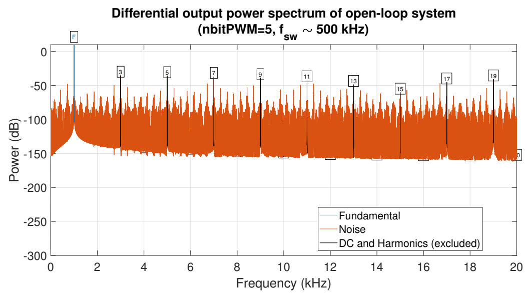
⋮

```
    %%
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_PWM_dB=Pn_PWM_dB;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_ADC_dB=Pn_ADC_dB;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_dB_PWM_cl=Pn_dB_PWM_cl;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_dB_ADC_cl=Pn_dB_ADC_cl;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_dB_tot_cl=Pn_tot_dB_cl;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.SNR_dB_cl=SNR_dB_cl;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_dB_PWM_ol=Pn_dB_PWM_ol;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_dB_ADC_ol=Pn_dB_ADC_ol;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.Pn_dB_tot_ol=Pn_tot_dB_ol;
SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath.SNR_dB_ol=SNR_dB_ol;
save('DatastoreFiles/SNRandPn_nbitADC16_TrailingEdge.mat',...
    'SNRandPn_PWM5_fsw500k_AAfilterInFeedbackPath', '-append')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%save parameters simulations
TE_PWM5_fsw500k.Fsw=Fsw;%save switching frequency
TE_PWM5_fsw500k.fclk_PWM=fclk_PWM;%save PWM TE clock
TE_PWM5_fsw500k.t_int=t_int;%save integration time
TE_PWM5_fsw500k.dec_fac=dec_fac;%save decimation factor
TE_PWM5_fsw500k.t_int_dec=t_int_dec; %save sample time of system outputs (except
    triangular waveform)
[Gm, Pm, Wcg, Wpg]=margin(T_filter_z);%evaluate gain and phase margins with
    correspondent frequency
marginPWM5_fsw500k.Gm=Gm;
marginPWM5_fsw500k.Pm=Pm;
marginPWM5_fsw500k.Fcg=Wcg/(2*pi);
marginPWM5_fsw500k.Fpg=Wpg/(2*pi);
StepInfo_fsw500k_out=stepinfo(step_Hfilter_z,t_step);
StepInfo_fsw500k_comp=stepinfo(step_Gfilter_z,t_step);
save('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat',...
    'TE_PWM5_fsw500k', 'marginPWM5_fsw500k', 'fclk_SYS', 'StepInfo_fsw500k_out',...
    'StepInfo_fsw500k_comp', '-append')%save simulation and loop tf parameters
```

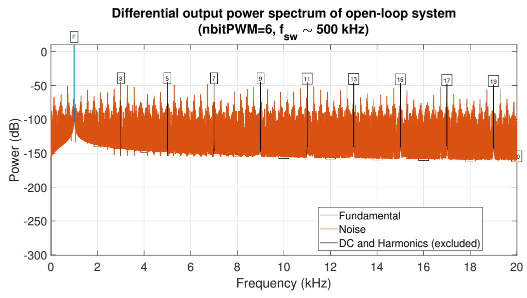
Moreover, for simulation with anti-aliasing filter, the high frequency pole has to be set to 1 MHz for $f_{sw} \sim 1$ MHz and 500 kHz for simulation working at $f_{sw} \sim 500$ kHz. Finally, the files where the parameters are saved has to be created when the Workspace is empty and then, at each running of the codes for different parameters, the results found will be *appended*.

5.1 Spectra

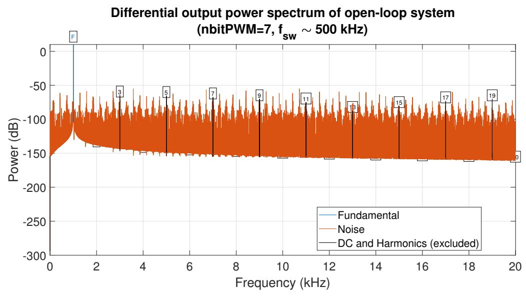
5.1.1 DPWM open-loop system spectra



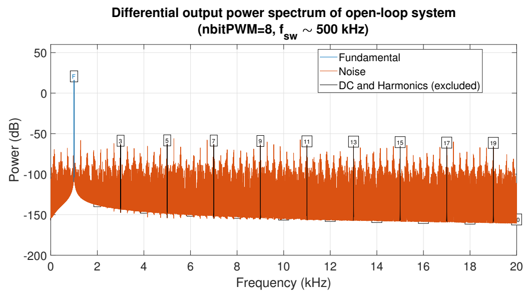
(a) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 5$



(b) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 6$

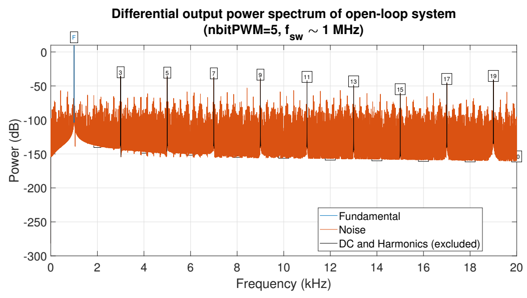


(c) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 7$

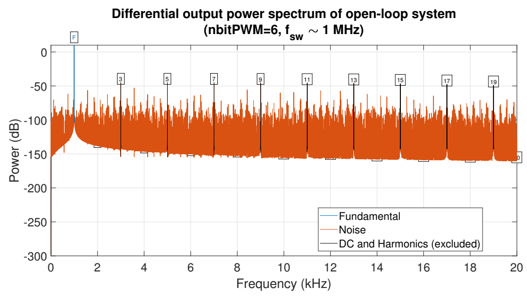


(d) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 8$

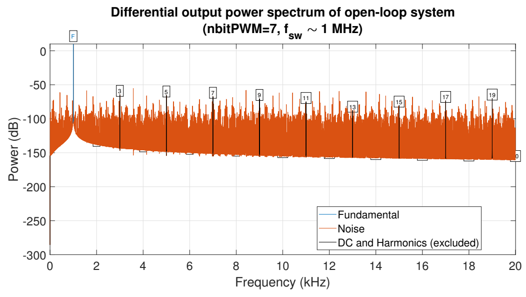
Figure 5.1: Differential output power spectrum of open-loop system ($f_{sw} \sim 500\,\text{kHz}$)



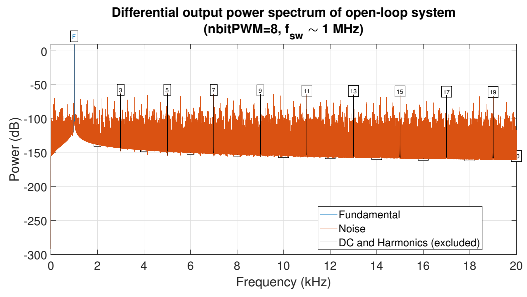
(a) $f_{sw} \sim 1\,\text{MHz}$, $nbit_{PWM} = 5$



(b) $f_{sw} \sim 1\,\text{MHz}$, $nbit_{PWM} = 6$



(c) $f_{sw} \sim 1\,\text{MHz}$, $nbit_{PWM} = 7$



(d) $f_{sw} \sim 1\,\text{MHz}$, $nbit_{PWM} = 8$

Figure 5.2: Differential output power spectrum of open-loop system ($f_{sw} \sim 1\,\text{MHz}$)

5.1.2 DPWM closed-loop spectra

Output filter in feedback path

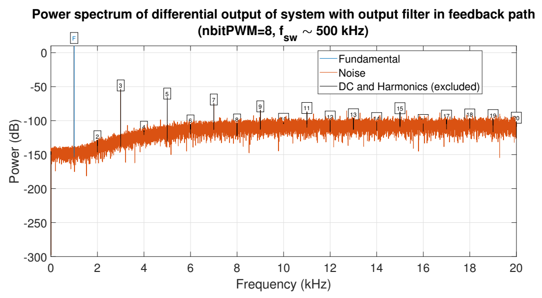
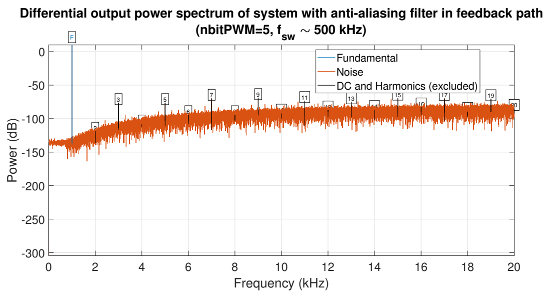
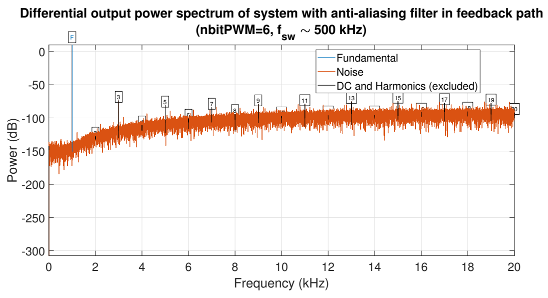


Figure 5.3: Differential output power spectrum of closed-loop system with output filter in feedback path ($f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 8$)

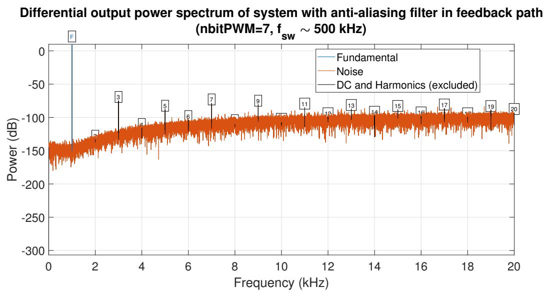
Anti-aliasing filter in feedback path



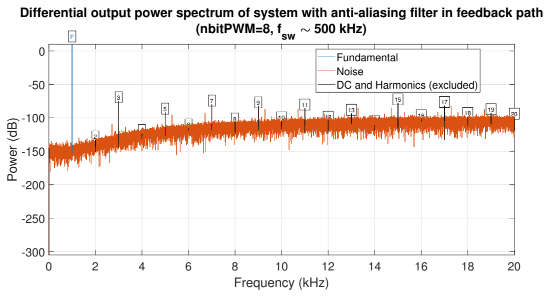
(a) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 5$



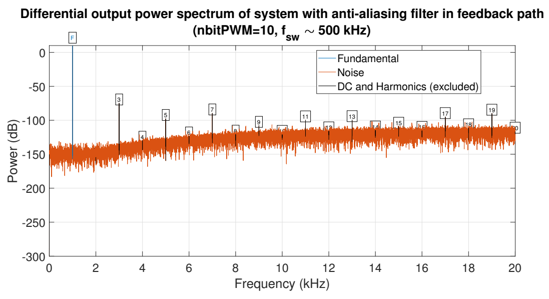
(b) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 6$



(c) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 7$

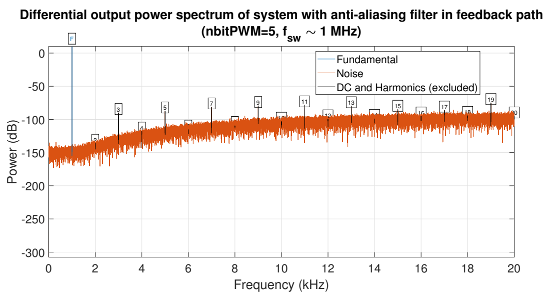


(d) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 8$

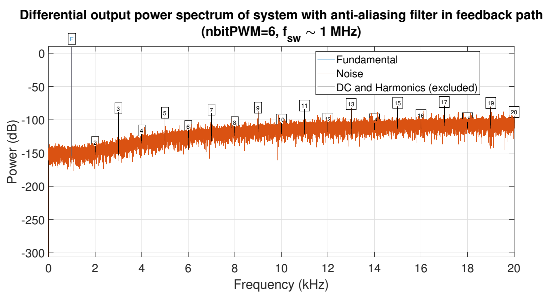


(e) $f_{sw} \sim 500\,\text{kHz}$, $nbit_{PWM} = 10$

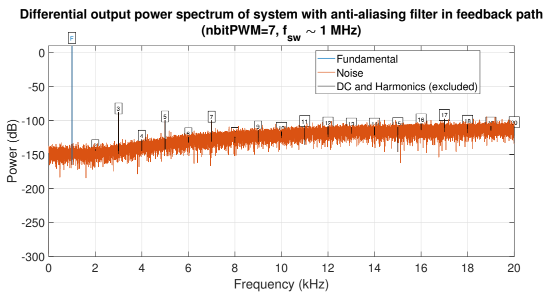
Figure 5.4: Differential output power spectrum of closed-loop system with anti-aliasing filter in feedback path ($f_{sw} \sim 500\,\text{kHz}$)



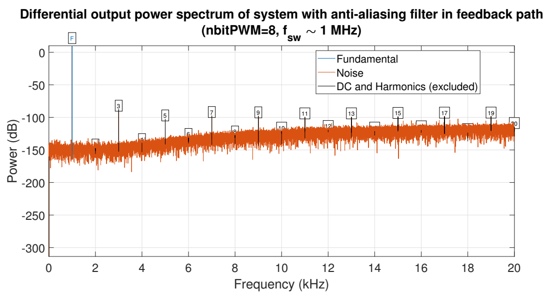
(a) $f_{sw} \sim 1\,\mathrm{MHz}$, $nbit_{PWM} = 5$



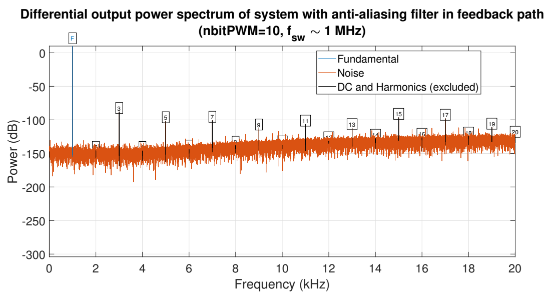
(b) $f_{sw} \sim 1\,\mathrm{MHz}$, $nbit_{PWM} = 6$



(c) $f_{sw} \sim 1\,\mathrm{MHz}$, $nbit_{PWM} = 7$



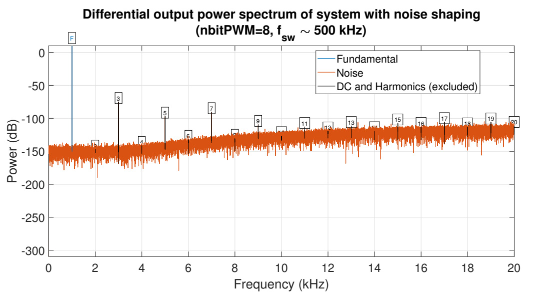
(d) $f_{sw} \sim 1\,\mathrm{MHz}$, $nbit_{PWM} = 8$



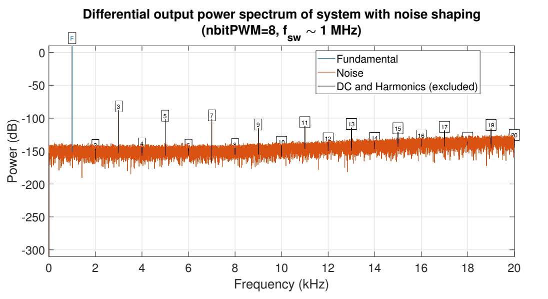
(e) $f_{sw} \sim 1\,\mathrm{MHz}$, $nbit_{PWM} = 10$

Figure 5.5: Differential output power spectrum of closed-loop system with anti-aliasing filter in feedback path ($f_{sw} \sim 1\,\mathrm{MHz}$)

Noise shaping



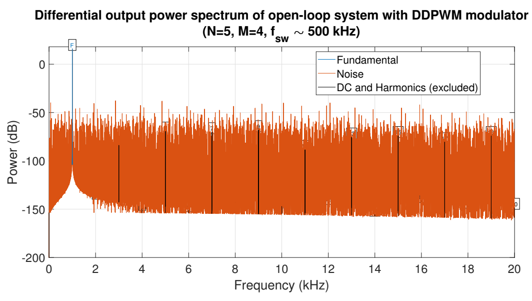
(a) $f_{sw} \sim 500\,\mathrm{kHz}$, $nbit_{PWM} = 8$



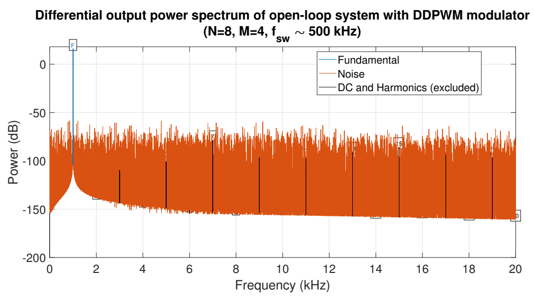
(b) $f_{sw} \sim 1\,\mathrm{MHz}$, $nbit_{PWM} = 8$

Figure 5.6: Differential output power spectrum of closed-loop system with first-order noise shaping with $nbit_{PWM} = 8$

5.1.3 DDPWM spectra

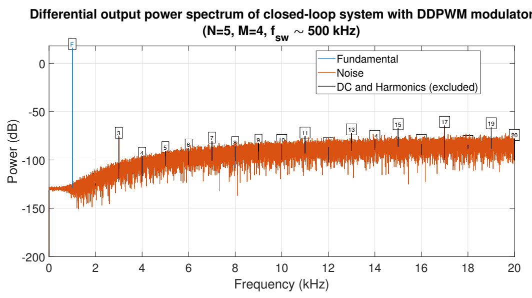


(a) $N = 5,\ M = 4$

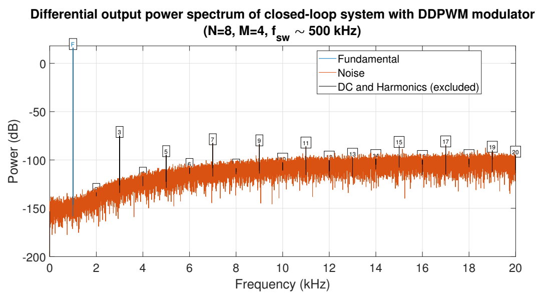


(b) $N = 8,\ M = 4$

Figure 5.7: Differential output power spectrum of open-loop system with DDPWM modulator ($f_{sw} \sim 500\,\text{kHz}$)



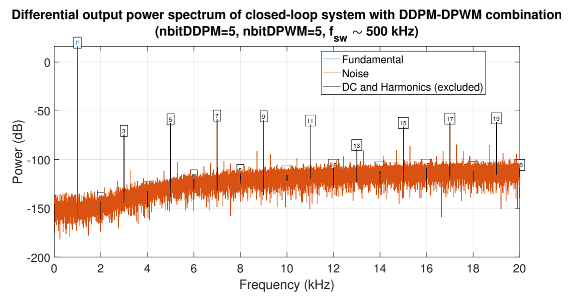
(a) $N = 5,\ M = 4$



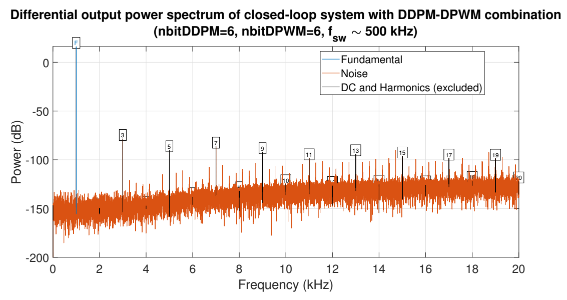
(b) $N = 8,\ M = 4$

Figure 5.8: Differential output power spectrum of closed-loop system with DDPWM modulator ($f_{sw} \sim 500\,\text{kHz}$)

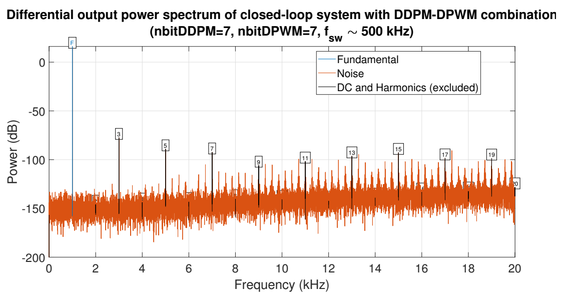
5.1.4 DDPM-DPWM combination closed-loop system spectra



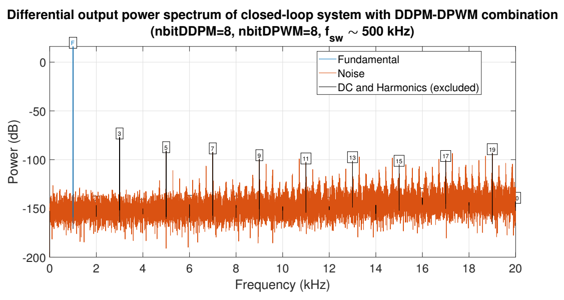
(a) $nbit_{DDPM} = 5,\ nbit_{DPWM} = 5$



(b) $nbit_{DDPM} = 6,\ nbit_{DPWM} = 6$

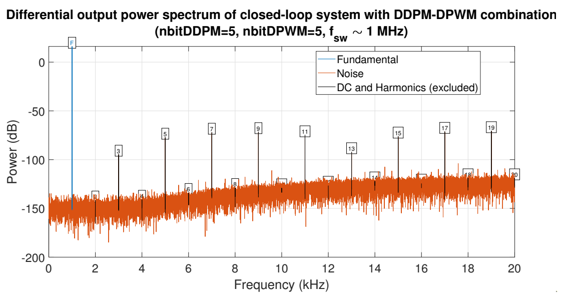


(c) $nbit_{DDPM} = 7,\ nbit_{DPWM} = 7$

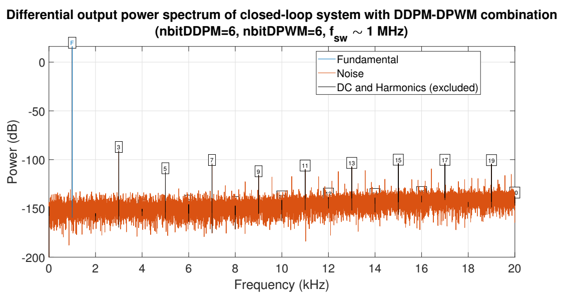


(d) $nbit_{DDPM} = 8,\ nbit_{DPWM} = 8$

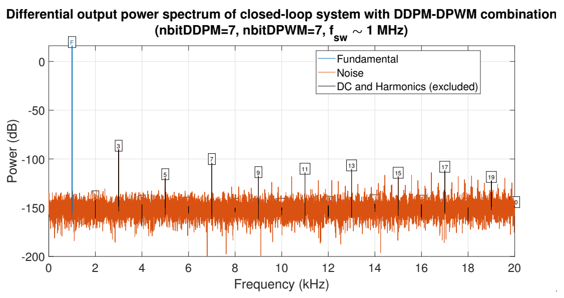
Figure 5.9: Differential output power spectrum of closed-loop system with DDPM-DPWM combination modulator ($f_{sw} \sim 500\,\text{kHz}$)



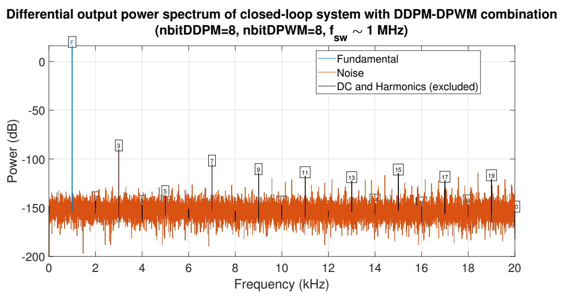
(a) $nbit_{DDPM} = 5$, $nbit_{DPWM} = 5$



(b) $nbit_{DDPM} = 6$, $nbit_{DPWM} = 6$



(c) $nbit_{DDPM} = 7$, $nbit_{DPWM} = 7$



(d) $nbit_{DDPM} = 8$, $nbit_{DPWM} = 8$

Figure 5.10: Differential output power spectrum of closed-loop system with DDPM-DPWM combination modulator ($f_{sw} \sim 1\,\text{MHz}$)

5.2 Codes

5.2.1 Code for design of system with DPWM modulator and anti-aliasing filter in feedback path

```
clc;
clear;
close all;
%%
%Data
n=1;% use this variable to number plots consequentially
f1=1e3; %Hz; frequency of input signal
f2=300; %Hz; frequency of second component input signal to test IMD
fclk_SYS=20e6; %Hz; clock frequency of the system
fclk_SYS=2^floor(log2(fclk_SYS)); %Hz; clock frequency of the system
VPS_OS_p=10;% $V positive power supply of output stage
VPS_OS_n=0;% $V negative power supply of output stage
Vtr_p=5;%V triangular positive peak
Vtr_n=0; %V triangular waveform positive peak
Vtr_pp=Vtr_p-Vtr_n; %V triangular waveform negative peak
VPS_PWM=Vtr_p; %V power supply triangular waveform generator
Fsw=1e6;
Fsw=2^round(log2(Fsw));%Hz; switching frequency
nbit_PWM=8; %number of bits for the quantized triangular waveform
nbit_ADC=16; %number of bits for ADC
fclk_PWM=Fsw*2^nbit_PWM; %Hz clock frequency of PWM generator
G=1/2;%attenuator in feedback path
%digital blocks parameters
VDD=VPS_PWM; %V; power supply of input ADC; I consider the supply of digital
                %part the same of the modulator because
                %implemented in the same digital hardware. Moreover, I
                %consider Vtr_p=VPS_PWM
Amp=.9*VDD/2; %amplitude of sinewave input signal
A=VPS_OS_p-VPS_OS_n;
Adigital=2^nbit_PWM/2^nbit_ADC; %truncation gain

if Adigital>1%in case the nbit_PWM>nbit_ADC
    Adigital=1;
end

GM=1/2^nbit_PWM; % gain of PWM
G_ADC=2^nbit_ADC/(2*VDD); %ADC  gain
G_PWM=2^nbit_PWM/VDD; %digital to analog scale factor; it is realted to the
                        %quatization noise of DPWM
Pn_PWM=VDD^2/(12*2^(2*nbit_PWM)); %PWM power noise
Pn_PWM_dB=pow2db(Pn_PWM); %PWM power noise in dB
Sn_PWM=VDD^2/2^(2*nbit_PWM)/(6*Fsw); %quantization noise power spectral density of
    DPWM
Pn_ADC=(2*VDD)^2/(12*2^(2*nbit_ADC)); %ADC power noise
Pn_ADC_dB=pow2db(Pn_ADC); %ADC power noise in dB
Sn_ADC=(2*VDD)^2/2^(2*nbit_ADC)/(6*fclk_SYS); %quantization noise power spectral
    density of ADC
PM=pi/3; %rad; phase margin chosen
%Simulation parameters
t_int=min(1/fclk_PWM, 1/(4*fclk_SYS)); %integration time for Simulink simulation
dec_fac=1/t_int/(4*fclk_SYS); %decimation factor to sample the output signal always
     at 4*fclk_SYS
```

```
if dec_fac<1
    dec_fac=1;
end
t_int_dec=t_int*dec_fac; %sample time of output signals
t_sim=1.5;%s
deadtime=2^ceil(log2(10e-9)); %s: deadtime to simulate the no shortcircuit of ouput
     stage

s=tf('s');
%design output LPF
fpf=20e3; %Hz; LPF pole frequency
omega_pf=2*pi*fpf; %rad/s; pole frequency of output filter
LPF=zpk(minreal(1/(1+s/omega_pf)^2)); %tf of output filter
[num_LPF, den_LPF]=tfdata(LPF, 'v');% num and den coefficients of output frequency
    transfer function

%set options parameters like font size and units for time graph
PropGraphTime.XAxis.Label.String='Time (s)';
PropGraphTime.YAxis.Label.String='Amplitude (V)';
PropGraphTime.FontSize=17;
PropGraphTime.YAxis.FontSize=17;
PropGraphTime.XAxis.FontSize=17;

plot_LPF=bodeplot(LPF,'r'); %define output filter plot structure
PropBode=getoptions(plot_LPF); %take options of graph
%set options parameters like font size and units for bode plot
PropBode.Title.FontSize=17;
PropBode.Xlabel.String='Frequency';
PropBode.Xlabel.FontSize=17;
PropBode.Ylabel.FontSize=17;
PropBode.TickLabel.FontSize=17;
PropBode.FreqUnits='Hz';
%output filter
figure(n)
n=n+1;
plot_LPF=bodeplot(LPF,'r'); %define output filter plot structure
PropBode.Title.String='Output filter transfer function'; %set title
setoptions(plot_LPF, PropBode); %set option of bode plot

%ZOH filter
T1=1/(fclk_SYS);%sample frequency of ADC
ZOH_ADC=1/(1+s*T1/2);%Sample&Hold related to ADC
ZOH_COMP=ZOH_ADC;%because of ADC and FPGA works at the same frequency, the two ZOH
    tf are equal
T2=1/(Fsw);%switching sample time
ZOH_REG=1/(1+s*T2/2);%Sample&Hold related to the hold register before the PWM
    sampling

figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_ZOH=bodeplot(ZOH_ADC, 'r', ZOH_REG, 'b');
PropBode.Title.String='ZOH èPad approximation';
setoptions(plot_ZOH, PropBode)
legend('ZOH ADC', 'ZOH REG')
%Transfer function of ZOH with èPad approximation
ZOH_ADC_NoApprox=(1-exp(-T1*s))/s*1/T1;
ZOH_REG_NoApprox=(1-exp(-T2*s))/s*1/T2;

```

```
figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_ZOH=bodeplot(ZOH_ADC_NoApprox, 'r', ZOH_REG_NoApprox, 'b');
PropBode.Title.String='ZOH transfer functions';
setoptions(plot_ZOH, PropBode)
legend('ZOH ADC', 'ZOH REG')
%%
%compensator with first order feedback filter (DPWM); feedback signal from output
    stage
p=3;
omega_u=2*pi*Fsw*1/10;
GBW=omega_u/(2*pi);
omega_p2=225e3*2*pi; %rad/s; pole of the feedback filter
omega_p2_H=1e6*2*pi; %rad/s; second pole to reduce aliasing of DPWM;
Gf_HF=zpk(minreal(1/(1+s/omega_p2)*1/(1+s/omega_p2_H)^2));%feedback filter transfer
     function
[num_GfHF, den_GfHF]=tfdata(Gf_HF, 'v');
omega_ph=9*omega_u; %pole to close the band
M=2; %order of last pole
extra_PM=atan(omega_u/(2*Fsw));%phase lost due to DPWM ZOH
omega_z1=omega_u/tan((PM-pi+p*pi/2+extra_PM)/(p-1));%zero to obtain the correct
    phase margin
tau_p=(GM*A*G*Adigital*G_ADC/(omega_u*omega_z1^(p-1)))^(1/p);%integrator
                                                     %time constant
if omega_z1<=0 && p>1 %verify that the zero is in HLP
    error('The compensation zero is negative')
end
if p==1 %set tau_p for p=1 (the zero is not necessary
    tau_p=(GM*A*G*G_ADC*Adigital/omega_u)^1/p;%
    omega_z1=1;
end
Gc_new=zpk(minreal((1+s/(9/10*omega_z1))*(1+s/(11/10*omega_z1)*...
    (1+s/omega_p2))*1/(tau_p*s)^p*1/((1+s/omega_ph)^M)));
    %continuos compensator transfer function
[num_Gcnew, den_Gcnew]=tfdata(Gc_new, 'v');%extract the coefficient
                                              %of continuos transfer function
T_filter=ZOH_ADC*ZOH_REG*GM*A*Gc_new*G*Gf_HF*Adigital*G_ADC;
            %Loop transfer function with output filter in feedback
P_filter=1/(1+T_filter);%senitivity function with output filter in feedback
H_filter=ZOH_ADC*ZOH_REG*Gc_new*GM*A*Adigital*G_ADC/(1+T_filter);
            %in-out transfer function with output filter in feedback loop
G_filter=ZOH_ADC*Gc_new*G_ADC/(1+T_filter);
            %in-out_compensator transfer function; it is useful to see the
            %overshoot of the command
%digital domain
Gc_new_z=zpk(minreal(c2d(Gc_new, 1/fclk_SYS, 'matched')));%digital compensator
[numGc_new_z, denGc_new_z]=tfdata(Gc_new_z, 'v');
T_filter_z=zpk(minreal(...
    Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*G*Gf_HF*Adigital*G_ADC));
P_filter_z=1/(1+T_filter_z);
H_filter_z=zpk(minreal(...
    Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*Adigital*G_ADC/(1+T_filter_z)));
G_filter_z=ZOH_ADC*ZOH_COMP*Gc_new*G_ADC/(1+T_filter);

figure(n)%plot digital and analog compensator transfer function
n=n+1;
Gcnew_plot=bodeplot(Gc_new, 'b', Gc_new_z, 'r');
PropBode.Title.String='Compensator transfer function';
```

```
setoptions(Gcnew_plot, PropBode)
legend('Analog', 'Digital')


figure(n)%plot obtained loop transfer function (analog and digital)
n=n+1;
hold on
PropBode.Title.String='Loop function with anti-aliasing filter in loop';
plot_Tfilter=bodeplot(T_filter, 'b', T_filter_z, 'r');
setoptions(plot_Tfilter, PropBode);
xlim([100 10e6])
legend('Analog', 'Digital')

figure(n)%plot sensitivity transfer function (analog and digital)
n=n+1;
hold on
PropBode.Title.String='Sensitivity function with anti-aliasing filter in loop';
plot_Pfilter=bodeplot(P_filter, 'b', P_filter_z, 'r');
setoptions(plot_Pfilter,PropBode);
xlim([100 10e6])
legend('Analog', 'Digital')

figure(n)%in-out transfer function
n=n+1;
hold on
PropBode.Title.String='Input-output with anti-aliasing filter in loop';
plot_Hfilter=bodeplot(H_filter, 'b', H_filter_z, 'r');
setoptions(plot_Hfilter, PropBode);
xlim([100 10e6])
legend('Analog', 'Digital')

%step responses
t_step=0:1/fclk_SYS:100e-6;
step_Hfilter_z=step(H_filter_z, t_step);
step_Hfilter=step(H_filter, t_step);

step_Gfilter_z=step(G_filter_z, t_step);
step_Gfilter=step(G_filter, t_step);

%step respnse in-out
figure(n)
n=n+1;
hold on
plot(t_step, step_Hfilter_z, 'r', t_step, step_Hfilter, 'b');
ax=gca;
ax.FontSize=PropGraphTime.FontSize;
ax.XAxis.FontSize=PropGraphTime.XAxis.FontSize;
ax.YAxis.FontSize=PropGraphTime.YAxis.FontSize;
ax.YAxis.Label.String=PropGraphTime.YAxis.Label.String;
ax.XAxis.Label.String=PropGraphTime.XAxis.Label.String;
ax.Title.String='Output step response with anti-aliasing filter in loop';
legend('Digital', 'Analog')

%step response input/compensator
figure(n)
n=n+1;
plot(t_step, step_Gfilter_z, 'r', t_step, step_Gfilter, 'b')
ax=gca;
ax.FontSize=PropGraphTime.FontSize;
ax.XAxis.FontSize=PropGraphTime.XAxis.FontSize;
ax.YAxis.FontSize=PropGraphTime.YAxis.FontSize;
ax.YAxis.Label.String='Code';
ax.XAxis.Label.String=PropGraphTime.XAxis.Label.String;
ax.Title.String='Command step response with anti-aliasing filter in loop';
legend('Digital', 'Analog')
%%
%SNR theoretical
NTF_PWM_filter_z=GM*A*G*G_PWM/(1+T_filter_z)*LPF; %noise transfer function
                                                   %of DPWM quantization noise
NTF_ADC_filter_z=H_filter_z*LPF; %noise transfer function of ADC
                                 %quantization noise
NTF_ADC_ol=2*G_ADC*GM*A*LPF*Adigital;%open-loop noise transfer function of ADC
NTF_PWM_ol=GM*A*G_PWM*LPF;%open loop noise transfer function of ADC

figure('Name', 'Digital noise transfer function closed-loop') %bode blot of NTF
hold on
plot_NTF_PWM=bodeplot(NTF_PWM_filter_z, 'b', NTF_ADC_filter_z, 'r');
setoptions(plot_NTF_PWM,PropBode);
legend('NTF PWM', 'NTF ADC')
title('Digital noise transfer function closed-loop')

figure('Name', 'Noise transfer function open-loop') %bode blot of NTF
hold on
plot_NTF_PWM=bodeplot(NTF_PWM_ol, 'b', NTF_ADC_ol, 'r');
setoptions(plot_NTF_PWM,PropBode);
legend('NTF PWM', 'NTF ADC')
title('Noise transfer function open-loop')
%evaluate power and SNR of PWM filtered by the system
DCgain=1/G;
Amp_out=DCgain*Amp;
[SNR_dB_PWM_cl, Pn_dB_PWM_cl, Ps_dB]=...
    SNRtheory(Amp_out, Sn_PWM, NTF_PWM_filter_z, 20e3);
%evaluate power and SNR of ADC filtered by the system
[SNR_dB_ADC_cl, Pn_dB_ADC_cl, ~]=...
    SNRtheory(Amp_out, Sn_ADC, NTF_ADC_filter_z, 20e3);
%evaluate power and SNR of PWM of open-loop system
[SNR_dB_PWM_ol, Pn_dB_PWM_ol, ~]=...
    SNRtheory(Amp_out, Sn_PWM, NTF_PWM_ol, 20e3);
%evaluate power and SNR of ADC of open-loop system
[SNR_dB_ADC_ol, Pn_dB_ADC_ol, ~]=...
    SNRtheory(Amp_out, Sn_ADC, NTF_ADC_ol, 20e3);
%Compute total power...
%...of closed-loop system...
Pn_tot_cl=10^(Pn_dB_PWM_cl/10)+10^(Pn_dB_ADC_cl/10);%V^2
Pn_tot_dB_cl=pow2db(Pn_tot_cl); %dB
Ps=10^(Ps_dB/10); %power signal (V^2)
SNR_dB_cl=10*log10(Ps/Pn_tot_cl); %SNR due to noise in system
%...and open-loop one
Pn_tot_ol=10^(Pn_dB_PWM_ol/10)+10^(Pn_dB_ADC_ol/10);%V^2
Pn_tot_dB_ol=pow2db(Pn_tot_ol); %dB
SNR_dB_ol=10*log10(Ps/Pn_tot_ol); %SNR due to noise in system
%save power noises and SNR in a structure
%%
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_PWM_dB=Pn_PWM_dB;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_ADC_dB=Pn_ADC_dB;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_PWM_cl=Pn_dB_PWM_cl;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_ADC_cl=Pn_dB_ADC_cl;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_tot_cl=Pn_tot_dB_cl;
```

```
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.SNR_dB_cl=SNR_dB_cl;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_PWM_ol=Pn_dB_PWM_ol;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_ADC_ol=Pn_dB_ADC_ol;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.Pn_dB_tot_ol=Pn_tot_dB_ol;
SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath.SNR_dB_ol=SNR_dB_ol;
save('DatastoreFiles/SNRandPn_nbitADC16_TrailingEdge.mat',...
    'SNRandPn_PWM8_fsw1M_AAfilterInFeedbackPath', '-append')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%save parameters simulations
TE_PWM8_fsw1M.Fsw=Fsw;%save switching frequency
TE_PWM8_fsw1M.fclk_PWM=fclk_PWM;%save PWM TE clock
TE_PWM8_fsw1M.t_int=t_int;%save integration time
TE_PWM8_fsw1M.dec_fac=dec_fac;%save decimation factor
TE_PWM8_fsw1M.t_int_dec=t_int_dec; %save sample time of system outputs (except
    triangular waveform)
[Gm, Pm, Wcg, Wpg]=margin(T_filter_z);%evaluate gain and phase margins with
    correspondent frequency
marginPWM8_fsw1M.Gm=Gm;
marginPWM8_fsw1M.Pm=Pm;
marginPWM8_fsw1M.Fcg=Wcg/(2*pi);
marginPWM8_fsw1M.Fpg=Wpg/(2*pi);
StepInfo_fsw1M_out=stepinfo(step_Hfilter_z,t_step);
StepInfo_fsw1M_comp=stepinfo(step_Gfilter_z,t_step);
save('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat',...
    'TE_PWM8_fsw1M', 'marginPWM8_fsw1M', 'fclk_SYS', 'StepInfo_fsw1M_out',...
    'StepInfo_fsw1M_comp', '-append')%save simulation and loop tf parameters
```

5.2.2 Code for design of system with DPWM modulator and output filter in feedback path

```
clc;
clear;
close all;
%%
%Data
n=1;% use this variable to number plots consequentially
f1=1e3; %Hz; frequency of input signal
f2=300; %Hz; frequency of second component input signal to test IMD
fclk_SYS=20e6; %Hz; clock frequency of the system
fclk_SYS=2^floor(log2(fclk_SYS)); %Hz; clock frequency of the system
VPS_OS_p=10;% $V positive power supply of output stage
VPS_OS_n=0;% $V negative power supply of output stage
Vtr_p=5;%V triangular positive peak
Vtr_n=0; %V triangular waveform positive peak
Vtr_pp=Vtr_p-Vtr_n; %V triangular waveform negative peak
VPS_PWM=Vtr_p; %V power supply triangular waveform generator
Fsw=1e6; %Hz;  frequency of sawtooth (trailing edge) PWM and frequency of natural PWM
Fsw=2^round(log2(Fsw))/2;
nbit_PWM=8; %number of bits for the quantized triangular waveform
nbit_ADC=16; %number of bits for ADC
fclk_PWM=Fsw*2^nbit_PWM; %Hz clock frequency of PWM generator
G=1/2;%attenuator in feedback path
%digital blocks parameters
VDD=VPS_PWM; %V; power supply of input ADC; I consider the supply of digital
                %part the same of the modulator because
                %implemented in the same digital hardware. Moreover, I
                %consider Vtr_p=VPS_PWM
Amp=.9*VDD/2; %amplitude of sinewave input signal
A=VPS_OS_p-VPS_OS_n;
Adigital=2^nbit_PWM/2^nbit_ADC; %truncation gain

if Adigital>1%in case the nbit_PWM>nbit_ADC
    Adigital=1;
end

GM=1/2^nbit_PWM; % gain of PWM
G_ADC=2^nbit_ADC/(2*VDD); %ADC  gain
G_PWM=2^nbit_PWM/VDD; %digital to analog scale factor; it is realted to the
                      %quatization noise of DPWM

Pn_PWM=VDD^2/(12*2^(2*nbit_PWM)); %PWM power noise
Pn_PWM_dB=pow2db(Pn_PWM); %PWM power noise in dB
Sn_PWM=VDD^2/2^(2*nbit_PWM)/(6*Fsw); %quantization noise power spectral density of DPWM
Pn_ADC=(2*VDD)^2/(12*2^(2*nbit_ADC)); %ADC power noise
Pn_ADC_dB=pow2db(Pn_ADC); %ADC power noise in dB
Sn_ADC=(2*VDD)^2/2^(2*nbit_ADC)/(6*fclk_SYS); %quantization noise power spectral density of ADC
Pn_PWM_fclk_SYS=Sn_PWM*fclk_SYS/2; %PWM quantization power noise evaluate from 0 to fclk_SYS
Pn_PWM_fclk_SYS_dB=pow2db(Pn_PWM_fclk_SYS); %(the previous one in dB)
PM=pi/3; %rad; phase margin chosen
%Simulation parameters
t_int=min(1/fclk_PWM, 1/(4*fclk_SYS)); %integration time for Simulink simulation
```

```
dec_fac=1/t_int/(4*fclk_SYS); %decimation factor to sample the output signal always at 1/t_int
if dec_fac<1
    dec_fac=1;
end
t_int_dec=t_int*dec_fac; %sample time of output signals
t_sim=1.5;%s
deadtime=2^ceil(log2(10e-9)); %s: deadtime to simulate the no shortcircuit of ouput stage
s=tf('s');
%design output LPF
fpf=20e3; %Hz; LPF pole frequency
omega_pf=2*pi*fpf; %rad/s; pole frequency of output filter
LPF=zpk(minreal(1/(1+s/omega_pf)^2)); %tf of output filter
[num_LPF, den_LPF]=tfdata(LPF, 'v');% num and den coefficients of output frequency transfer function
PropGraphTime.XAxis.Label.String='Time (s)';
PropGraphTime.YAxis.Label.String='Amplitude (V)';
PropGraphTime.FontSize=17;
PropGraphTime.YAxis.FontSize=17;
PropGraphTime.XAxis.FontSize=17;

plot_LPF=bodeplot(LPF,'r'); %define output filter plot structure
PropBode=getoptions(plot_LPF); %take options of graph
%set options parameters like font size and units for bode plot
PropBode.Title.FontSize=17;
PropBode.Xlabel.String='Frequency';
PropBode.Xlabel.FontSize=17;
PropBode.Ylabel.FontSize=17;
PropBode.TickLabel.FontSize=17;
PropBode.FreqUnits='Hz';
%output filter
figure(n)
n=n+1;
plot_LPF=bodeplot(LPF,'r');
PropBode.Title.String='Output filter transfer function';
setoptions(plot_LPF,PropBode);

%ZOH filter
T1=1/(fclk_SYS);%sample frequency of ADC
ZOH_ADC=1/(1+s*T1/2);%Sample&Hold related to ADC
ZOH_COMP=ZOH_ADC;%because of ADC and FPGA works at the same frequency, the two ZOH tf are equal
T2=1/(Fsw);%switching sample time
ZOH_REG=1/(1+s*T2/2);%Sample&Hold related to the hold register before the PWM sampling

figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_ZOH=bodeplot(ZOH_ADC, 'r', ZOH_REG, 'b');
PropBode.Title.String='ZOH èPad approximation';
setoptions(plot_ZOH, PropBode)
legend('ZOH ADC', 'ZOH REG')

ZOH_ADC_NoApprox=(1-exp(-T1*s))/s*1/T1;
ZOH_REG_NoApprox=(1-exp(-T2*s))/s*1/T2;
figure(n)%plot transfer functions of ZOH
n=n+1;
```

```
hold on
plot_ZOH=bodeplot(ZOH_ADC_NoApprox, 'r', ZOH_REG_NoApprox, 'b');
PropBode.Title.String='ZOHs transfer function';
setoptions(plot_ZOH, PropBode)
legend('ZOH ADC', 'ZOH REG')
%%
%compensator with feedback filter (DPWM). In this solution the output
%filter works as anti-aliasing filter
p=3; %integrator order
omega_u=2*pi*Fsw/10;
GBW=omega_u/(2*pi);
omega_ph1=9*omega_u;%poles to realized proper feedback transfer function...
omega_ph2=10*omega_u;%...and close the band
omega_zx=omega_pf*9/10;%zeros to compensate the double pole of feedback filter
omega_zy=omega_pf;
M=1; %order of first high frequency pole
N=1; %order of second high frequency pole
extra_PM=atan(omega_u/(2*Fsw));
omega_z1=omega_u/tan((PM-pi+p*pi/2+extra_PM)/(p-1));%zero to obtain the correct
    phase margin;
if omega_z1<=0
    error('The zero is negative')
end
%I consider
%   -the ZOH_REG
%   -neglect the ZOH_ADC
tau_p=(GM*A*G*Adigital*G_ADC/(omega_u))^(1/p)*(1/omega_z1)^(1-1/p);
%analog domain
Gc_new=zpk(minreal(((1+s/omega_zx)*(1+s/omega_zy)*(1+s/(9/10*omega_z1)*(1+s/(11/10*
    omega_z1)))...
    *1/(tau_p*s)^p*1/((1+s/omega_ph1)^M*(1+s/omega_ph2)^N)));
[num_Gcnew, den_Gcnew]=tfdata(Gc_new, 'v');
T_filter=GM*A*ZOH_ADC*ZOH_REG*Gc_new*LPF*G*G_ADC*Adigital;%Loop transfer function
    with output filter in feedback
P_filter=1/(1+T_filter);%senitivity function with output filter in feedback
H_filter=ZOH_ADC*ZOH_REG*Gc_new*GM*A*LPF*G_ADC*Adigital/(1+T_filter);%in-out
    transfer function with output filter in feedback loop
G_filter=ZOH_ADC*Gc_new*G_ADC/(1+T_filter);
%digital domain
Gc_new_z=zpk(minreal(c2d(Gc_new, 1/fclk_SYS, 'matched')));%digital compensator
[numGc_new_z, denGc_new_z]=tfdata(Gc_new_z, 'v');
T_filter_z=zpk(minreal(Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*LPF*G*G_ADC*Adigital));
P_filter_z=1/(1+T_filter_z);
H_filter_z=zpk(minreal(Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*LPF*G_ADC*Adigital/(1+
    T_filter_z)));
G_filter_z=ZOH_ADC*ZOH_COMP*Gc_new*G_ADC/(1+T_filter);

figure(n)%compensator transfer function plot (digital and analog)
n=n+1;
Gcnew_plot=bodeplot(Gc_new, 'b', Gc_new_z, 'r');
PropBode.Title.String='Compensator transfer function';
setoptions(Gcnew_plot, PropBode)
legend('Analog', 'Digital')

figure(n)%loop filter transfer function plot (digital and analog)
n=n+1;
hold on
plot_Tfilter=bodeplot(T_filter, 'b', T_filter_z, 'r');
PropBode.Title.String='Loop function with output filter in feedback';
```

```
setoptions(plot_Tfilter, PropBode);
xlim([100 10e6])
legend('Analog', 'Digital')

figure(n)%plot sensitivity transfer function
n=n+1;
hold on
plot_Pfilter=bodeplot(P_filter, 'b', P_filter_z, 'r');
PropBode.Title.String='Sensitivity function with output filter in feedback';
setoptions(plot_Pfilter, PropBode);
xlim([100 10e6])

figure(n)%plot in-out transfer function
n=n+1;
hold on
plot_Hfilter=bodeplot(H_filter, 'b', H_filter_z, 'r');
PropBode.Title.String='Input-output with output filter in feedback';
setoptions(plot_Hfilter, PropBode);
xlim([100 10e6])
legend('Analog', 'Digital')

%step response input/output
t_step=0:1/fclk_SYS:100e-6;
step_Hfilter_z=step(H_filter_z, t_step);
step_Hfilter=step(H_filter, t_step);
%step response input/compensator
step_Gfilter_z=step(G_filter_z, t_step);
step_Gfilter=step(G_filter, t_step);

figure(n)%output step response
n=n+1;
plot(t_step, step_Hfilter_z, 'r', t_step, step_Hfilter, 'b')
legend('Digital', 'Analog')
ax=gca;
ax.FontSize=PropGraphTime.FontSize;
ax.XAxis.FontSize=PropGraphTime.XAxis.FontSize;
ax.YAxis.FontSize=PropGraphTime.YAxis.FontSize;
ax.YAxis.Label.String=PropGraphTime.YAxis.Label.String;
ax.XAxis.Label.String=PropGraphTime.XAxis.Label.String;
ax.Title.String='Output step response with output filter in loop';

figure(n)%command step response
n=n+1;
plot(t_step, step_Gfilter_z, 'r', t_step, step_Gfilter, 'b')
legend('Digital', 'Analog')
ax=gca;
ax.FontSize=PropGraphTime.FontSize;
ax.XAxis.FontSize=PropGraphTime.XAxis.FontSize;
ax.YAxis.FontSize=PropGraphTime.YAxis.FontSize;
ax.YAxis.Label.String='Code';
ax.XAxis.Label.String=PropGraphTime.XAxis.Label.String;
ax.Title.String='Command step response with output filter in loop';
%%
%SNR theoretical
NTF_PWM_filter_z=GM*A*LPF*G*G_PWM/(1+T_filter_z); %noise transfer function of DPWM
    quantization noise
NTF_ADC_filter_z=H_filter_z; %noise transfer function of ADC quantization noise
figure(n) %bode blot of NTF
n=n+1;
```

```
hold on
plot_NTF_PWM=bodeplot(NTF_PWM_filter_z, 'b', NTF_ADC_filter_z, 'r');
PropBode.Title.String='Digital noise transfer function';
setoptions(plot_NTF_PWM, PropBode);
legend('NTF PWM', 'NTF ADC')

Amp_out=Amp*1/G;
%evaluate power and SNR of PWM filtered by the system
[SNR_dB_PWM, Pn_dB_PWM_InLoop, Ps_dB]=...
    SNRtheory(Amp_out, Sn_PWM, NTF_PWM_filter_z, 20e3);
%evaluate power and SNR of ADC filtered by the system
[SNR_dB_ADC, Pn_dB_ADC_InLoop, ~]=...
    SNRtheory(Amp_out, Sn_ADC, NTF_ADC_filter_z, 20e3);
%Compute total power...
Pn_tot=10^(Pn_dB_PWM_InLoop/10)+10^(Pn_dB_ADC_InLoop/10); %V^2
Pn_tot_dB=pow2db(Pn_tot); %dB
Ps=10^(Ps_dB/10); %power signal (V^2)
SNR_dB_theory=10*log10(Ps/Pn_tot); %SNR due to noise in system
%save power noises and SNR in a structure
SNRandPn_PWM8_fsw500k_M1N1_OFinFeedebackPath.Pn_PWM_dB=Pn_PWM_dB;
SNRandPn_PWM8_fsw500k_M1N1_OFinFeedebackPath.Pn_ADC_dB=Pn_ADC_dB;
SNRandPn_PWM8_fsw500k_M1N1_OFinFeedebackPath.Pn_dB_PWM_LoopEffect=Pn_dB_PWM_InLoop;
SNRandPn_PWM8_fsw500k_M1N1_OFinFeedebackPath.Pn_dB_ADC_LoopEffect=Pn_dB_ADC_InLoop;
SNRandPn_PWM8_fsw500k_M1N1_OFinFeedebackPath.Pn_dB_tot_LoopEffect=Pn_tot_dB;
SNRandPn_PWM8_fsw500k_M1N1_OFinFeedebackPath.SNR_dB_theory=SNR_dB_theory;
save('DatastoreFiles/SNRandPn_OFinFP',...
    'SNRandPn_PWM8_fsw500k_M1N1_OFinFeedebackPath', '-append')
%%
%save parameters simulations
TE_PWM8_fsw500k_M1N1.Fsw=Fsw;%save switching frequency
TE_PWM8_fsw500k_M1N1.fclk_PWM=fclk_PWM;%save PWM TE clock
TE_PWM8_fsw500k_M1N1.t_int=t_int;%save integration time
TE_PWM8_fsw500k_M1N1.dec_fac=dec_fac;%save decimation factor
TE_PWM8_fsw500k_M1N1.t_int_dec=t_int*dec_fac; %save sample time of system outputs (
    except triangular waveform)
t_int_PWM8=t_int;
[Gm, Pm, Wcg, Wpg]=margin(T_filter_z);%evaluate gain and phase margins with
    correspondet frequency
marginPWM8_fsw500k_M1N1.Gm=Gm;
marginPWM8_fsw500k_M1N1.Pm=Pm;
marginPWM8_fsw500k_M1N1.Fcg=Wcg/(2*pi);
marginPWM8_fsw500k_M1N1.Fpg=Wpg/(2*pi);
StepInfo_fsw500k_out_M1N1=stepinfo(step_Hfilter_z,t_step);
StepInfo_fsw500k_comp_M1N1=stepinfo(step_Gfilter_z,t_step);
save('DatastoreFiles/SimParameters_nbitADC16_OFinFP.mat',...
    'TE_PWM8_fsw500k_M1N1', 'marginPWM8_fsw500k_M1N1', 'fclk_SYS', '
    StepInfo_fsw500k_out_M1N1', 'StepInfo_fsw500k_comp_M1N1', '-append')%save
    simulation and loop tf parameters
```

5.2.3 Code for design of system with DDPWM modulator

```
clc;
clear;
close all;
%%
%Data
n=1;% use this variable to number plots consequentially
f1=1000; %Hz; frequency of input signal
f2=300; %Hz; frequency of second component input signal to test IMD
fclk_SYS=20e6; %Hz; clock frequency of the system
fclk_SYS=2^floor(log2(fclk_SYS)); %Hz; clock frequency of the system
nbit_ADC=16; %ADC number of bit
Fsw=1e6;
Fsw=2^ceil(log2(Fsw))/2;
VPS_OS_p=10;% $V positive power supply of output stage
VPS_OS_n=0;% $V negative power supply of output stage
Vtr_p=5;%V triangular positive peak
Vtr_n=0; %V triangular waveform positive peak
Vtr_pp=Vtr_p-Vtr_n; %V triangular waveform negative peak
VPS_PWM=Vtr_p; %V power supply triangular waveform generator
%DDPWM
M_DDPWM=4; %LSB of input register
N_DDPWM=8; %MSB of input register
fclk_DDPWM=2^(N_DDPWM)*Fsw;%Hz; clock frequency of DPWM
nbit_DDPWM=N_DDPWM+M_DDPWM;%Hz; total number of bit of DDPWM modulator
G_Trunc_cl=2^nbit_DDPWM/2^nbit_ADC;%truncation gain of closed-loop system
G_Trunc_ol=2^nbit_DDPWM/2^(nbit_ADC-1);%truncation gain of open-loop system
if G_Trunc_cl>1
    G_trunc_cl=1;
end
if G_Trunc_ol>1
    G_trunc_ol=1;
end
G=1/2;%attenuator in feedback path
%digital blocks parameters
VDD=VPS_PWM; %V; power supply of input ADC; I consider the supply of digital
                %part the same of the modulator because
                %implemented in the same digital hardware. Moreover, I
                %consider Vtr_p=VPS_PWM
Amp=.9*VDD/2; %amplitude of sinewave input signal
A=VPS_OS_p-VPS_OS_n;

Adigital=2^N_DDPWM/2^nbit_ADC; %gain to remapping the output of compensator on
    nbit_PWM
if Adigital>1%in case the resPWM>resADC
    Adigital=1;
end
GM=1/2^N_DDPWM; % gain of PWM
G_ADC=2^nbit_ADC/(2*VDD); %analog to digital scale factor
G_PWM=2^N_DDPWM/VDD; %digital to analog scale factor

Pn_PWM=VDD^2/(12*2^(2*nbit_DDPWM)); %PWM power noise
Pn_PWM_dB=pow2db(Pn_PWM); %PWM power noise in dB
Sn_PWM=VDD^2/2^(2*nbit_DDPWM)/(6*Fsw); %quantization noise power spectral density
    of DPWM
Pn_ADC=(2*VDD)^2/(12*2^(2*nbit_ADC)); %ADC power noise
Pn_ADC_dB=pow2db(Pn_ADC); %ADC power noise in dB
Sn_ADC=(2*VDD)^2/2^(2*nbit_ADC)/(6*fclk_SYS); %quantization noise power spectral
```

```
    density of ADC
Pn_PWM_fclk_SYS=Sn_PWM*fclk_SYS/2; %PWM quantization power noise evaluate from 0 to
     fclk_SYS/2
Pn_PWM_fclk_SYS_dB=pow2db(Pn_PWM_fclk_SYS); %(the previous one in dB)
PM=pi/3; %rad; phase margin chosen
%Simulation parameters
t_int=min([1/fclk_DDPWM 1/(4*fclk_SYS)]); %integration time for Simulink simulation
dec_fac=1/t_int/(4*fclk_SYS); %decimation factor to sample the output signal always
     at 1/t_int
if dec_fac<1
    dec_fac=1;
end
t_int_dec=t_int*dec_fac; %sample time of output signals
t_sim=1.5;%s
deadtime=2^ceil(log2(10e-9)); %s: deadtime to simulate the no shortcircuit of ouput
     stage

s=tf('s');
%design output LPF
fpf=20e3; %Hz; LPF pole frequency
omega_pf=2*pi*fpf; %rad/s; pole frequency of output filter
LPF=zpk(minreal(1/(1+s/omega_pf)^2)); %tf of output filter
[num_LPF, den_LPF]=tfdata(LPF, 'v');% num and den coefficients of output frequency
    transfer function

%output filter
figure(n)
n=n+1;
plot_LPF=bodeplot(LPF,'r');
setoptions(plot_LPF,'FreqUnits','Hz');
title('Output filter transfer function')

%ZOH filter
T1=1/(fclk_SYS);%sample frequency of ADC
ZOH_ADC=1/(1+s*T1/2);%Sample&Hold related to ADC
ZOH_COMP=ZOH_ADC;%because of ADC and FPGA works at the same frequency, the two ZOH
    tf are equal
T2=1/Fsw;%switching sample time
ZOH_REG=1/(1+s*T2/2);%Sample&Hold related to the hold register before the PWM
    sampling
figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_ZOHadc=bodeplot(ZOH_ADC, 'r');
plot_ZOHreg=bodeplot(ZOH_REG, 'b');
setoptions(plot_ZOHadc, 'FreqUnits','Hz')
setoptions(plot_ZOHreg, 'FreqUnits','Hz')
legend('ZOH ADC', 'ZOH REG')
ZOH_ADC_NoApprox=(1-exp(-T1*s))/s*1/T1;
ZOH_REG_NoApprox=(1-exp(-T2*s))/s*1/T2;
figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_ZOHadc=bodeplot(ZOH_ADC_NoApprox, 'r');
plot_ZOHreg=bodeplot(ZOH_REG_NoApprox, 'b');
setoptions(plot_ZOHadc, 'FreqUnits','Hz')
setoptions(plot_ZOHreg, 'FreqUnits','Hz')
title('ZOHs transfer function')
legend('ZOH ADC', 'ZOH REG')
```

```
%%
%compensator with first order feedback filter (DPWM); feedback signal from output
    stage
p=3;
omega_u=2*pi*Fsw*1/10;
GBW=omega_u/(2*pi);
omega_p2=225e3*2*pi; %rad/s; pole of the feedback filter
omega_p2_H=1e6*2*pi/2; %rad/s; second pole to reduce aliasing of DPWM;
Gf_HF=zpk(minreal(1/(1+s/omega_p2)*1/(1+s/omega_p2_H)^2));%feedback filter transfer
     function
[num_GfHF, den_GfHF]=tfdata(Gf_HF, 'v');
omega_ph=9*omega_u; %pole to close the band
M=2; %order of last pole
extra_PM=atan(omega_u/(2*Fsw));
omega_z1=omega_u/tan((PM-pi+p*pi/2+extra_PM)/(p-1));%zero to obtain the correct
    phase margin
tau_p=(GM*A*G*Adigital*G_ADC/(omega_u*omega_z1^(p-1)))^(1/p);
if omega_z1<=0 && p>1
    error('The compensation zero is negative')
end
if p==1
    tau_p=(GM*A*G*G_ADC*Adigital/omega_u)^1/p;
    omega_z1=1;
end
%compensator transfer function
Gc_new=zpk(minreal((1+s/(9/10*omega_z1))*(1+s/(11/10*omega_z1)*(1+s/omega_p2))*1/(
    tau_p*s)^p*1/((1+s/omega_ph)^M)));
[num_Gcnew, den_Gcnew]=tfdata(Gc_new, 'v');
T_filter=ZOH_ADC*ZOH_REG*GM*A*Gc_new*G*Gf_HF*Adigital*G_ADC;%Loop transfer function
     with output filter in feedback
P_filter=1/(1+T_filter);%senitivity function with output filter in feedback
H_filter=ZOH_ADC*ZOH_REG*Gc_new*GM*A*Adigital*G_ADC/(1+T_filter);%in-out transfer
    function with output filter in feedback loop
G_filter=ZOH_ADC*Gc_new*G_ADC/(1+T_filter);%in-command gransfer function
%digital domain
Gc_new_z=zpk(minreal(c2d(Gc_new, 1/fclk_SYS, 'matched')));%digital compensator
[numGc_new_z, denGc_new_z]=tfdata(Gc_new_z, 'v');
%digital loop transfer function
T_filter_z=zpk(minreal(...
    Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*G*Gf_HF*Adigital*G_ADC));
%digital sensitivity transfer functio
P_filter_z=1/(1+T_filter_z);
%in-out transfer function
H_filter_z=zpk(minreal(...
    Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*Adigital*G_ADC/(1+T_filter_z)));
G_filter_z=ZOH_ADC*ZOH_COMP*Gc_new*G_ADC/(1+T_filter);

figure(n)%plot compensator transfer function (analog and digital)
n=n+1;
Gcnew_plot=bodeplot(Gc_new, Gc_new_z);
setoptions(Gcnew_plot, 'FreqUnits', 'Hz')
legend('Analog', 'Digital')
title('Compensator transfer function')

figure(n)%plot loop transfer function (analog and digital)
n=n+1;
hold on
plot_Tfilter=bodeplot(T_filter, 'b');
setoptions(plot_Tfilter,'FreqUnits','Hz');
```

```
plot_Tfilter_z=bodeplot(T_filter_z, 'r');
setoptions(plot_Tfilter_z,'FreqUnits','Hz');
xlim([100 10e6])
legend('Analog', 'Digital')
title('Loop function with anti-alising filter in loop')

figure(n)%plot sensitivity transfer function (analog and digital)
n=n+1;
hold on
plot_Pfilter=bodeplot(P_filter, 'b');
setoptions(plot_Pfilter,'FreqUnits','Hz');
plot_Pfilter_z=bodeplot(P_filter_z, 'r');
setoptions(plot_Pfilter_z,'FreqUnits','Hz');
xlim([100 10e6])
legend('Analog', 'Digital')
title('Sensitivity function with anti-alising filter in loop')
figure(n)
n=n+1;
hold on
plot_Hfilter=bodeplot(H_filter, 'b');
setoptions(plot_Hfilter,'FreqUnits','Hz');
plot_Hfilter_z=bodeplot(H_filter_z, 'r');
setoptions(plot_Hfilter_z,'FreqUnits','Hz');
xlim([100 10e6])
legend('Analog', 'Digital')
title('Input-output with anti-alising filter in loop')

%step response input/output
t_step=0:1/fclk_SYS:100e-6;
step_Hfilter_z=step(H_filter_z, t_step);
step_Hfilter=step(H_filter, t_step);
%step response input/compensator
step_Gfilter_z=step(G_filter_z, t_step);
step_Gfilter=step(G_filter, t_step);

figure(n)%output step response
n=n+1;
plot(t_step, step_Hfilter_z, 'r', t_step, step_Hfilter, 'b')
legend('Digital', 'Analog')
xlabel('Time [s]')
ylabel('Amplitude [V]')
title('Step response with anti-alising filter in loop')

figure(n)%command step response
n=n+1;
plot(t_step, step_Gfilter_z, 'r', t_step, step_Gfilter, 'b')
legend('Digital', 'Analog')
xlabel('Time [s]')
ylabel('Amplitude [V]')
title('Step response of compesator with anti-alising filter in loop')
%%
%SNR theoretical
NTF_PWM_filter_z=GM*A*G*G_PWM/(1+T_filter_z)*LPF; %noise transfer function of DPWM
    quantization noise
NTF_ADC_filter_z=H_filter_z*LPF; %noise transfer function of ADC quantization noise
NTF_ADC_ol=2*G_ADC*GM*A*LPF*Adigital;
NTF_PWM_ol=GM*A*G_PWM*LPF;
figure('Name', 'Digital noise transfer function closed-loop') %bode blot of NTF
hold on
```

```
plot_NTF_PWM=bodeplot(NTF_PWM_filter_z, 'b');
setoptions(plot_NTF_PWM,'FreqUnits','Hz');
plot_NTF_ADC=bodeplot(NTF_ADC_filter_z, 'r');
setoptions(plot_NTF_ADC,'FreqUnits','Hz');
legend('NTF PWM', 'NTF ADC')
title('Digital noise transfer function closed-loop')
figure('Name', 'Noise transfer function open-loop') %bode blot of NTF
hold on
plot_NTF_PWM=bodeplot(NTF_PWM_ol, 'b');
setoptions(plot_NTF_PWM,'FreqUnits','Hz');
plot_NTF_ADC=bodeplot(NTF_ADC_ol, 'r');
setoptions(plot_NTF_ADC,'FreqUnits','Hz');
legend('NTF PWM', 'NTF ADC')
title('Noise transfer function open-loop')
%evaluate power and SNR of PWM filtered by the system
[SNR_dB_PWM_cl, Pn_dB_PWM_cl, Ps_dB]=...
    SNRtheory(Amp, Sn_PWM, NTF_PWM_filter_z, 20e3);
%evaluate power and SNR of ADC filtered by the system
[SNR_dB_ADC_cl, Pn_dB_ADC_cl, ~]=...
    SNRtheory(Amp, Sn_ADC, NTF_ADC_filter_z, 20e3);
%evaluate power and SNR of PWM of open-loop system
[SNR_dB_PWM_ol, Pn_dB_PWM_ol, ~]=...
    SNRtheory(Amp, Sn_PWM, NTF_PWM_ol, 20e3);
%evaluate power and SNR of ADC of open-loop system
[SNR_dB_ADC_ol, Pn_dB_ADC_ol, ~]=...
    SNRtheory(Amp, Sn_ADC, NTF_ADC_ol, 20e3);
%Compute total power...
%...of closed-loop system...
Pn_tot_cl=10^(Pn_dB_PWM_cl/10)+10^(Pn_dB_ADC_cl/10);%V^2
Pn_tot_dB_cl=pow2db(Pn_tot_cl); %dB
Ps=10^(Ps_dB/10); %power signal (V^2)
SNR_dB_cl=10*log10(Ps/Pn_tot_cl); %SNR due to noise in system
%...and open-loop one
Pn_tot_ol=10^(Pn_dB_PWM_ol/10)+10^(Pn_dB_ADC_ol/10);%V^2
Pn_tot_dB_ol=pow2db(Pn_tot_ol); %dB
SNR_dB_ol=10*log10(Ps/Pn_tot_ol); %SNR due to noise in system
%save power noises and SNR in a structure
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_PWM_dB=Pn_PWM_dB;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_ADC_dB=Pn_ADC_dB;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_dB_PWM_cl=Pn_dB_PWM_cl;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_dB_ADC_cl=Pn_dB_ADC_cl;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_dB_tot_cl=Pn_tot_dB_cl;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.SNR_dB_cl=SNR_dB_cl;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_dB_PWM_ol=Pn_dB_PWM_ol;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_dB_ADC_ol=Pn_dB_ADC_ol;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.Pn_dB_tot_ol=Pn_tot_dB_ol;
SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath.SNR_dB_ol=SNR_dB_ol;
save('DatastoreFiles/SNRandPn_nbitADC16_TrailingEdge.mat',...
    'SNRandPn_PWM8_fsw500k_AAfilterInFeedbackPath', '-append')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%save parameters simulations
TE_PWM8_fsw500k.Fsw=Fsw;%save switching frequency
TE_PWM8_fsw500k.fclk_PWM=fclk_PWM;%save PWM TE clock
TE_PWM8_fsw500k.t_int=t_int;%save integration time
TE_PWM8_fsw500k.dec_fac=dec_fac;%save decimation factor
TE_PWM8_fsw500k.t_int_dec=t_int_dec; %save sample time of system outputs (except
    triangular waveform)
[Gm, Pm, Wcg, Wpg]=margin(T_filter_z);%evaluate gain and phase margins with
```

```
    correspondet frequency
marginPWM8_fsw500k.Gm=Gm;
marginPWM8_fsw500k.Pm=Pm;
marginPWM8_fsw500k.Fcg=Wcg/(2*pi);
marginPWM8_fsw500k.Fpg=Wpg/(2*pi);
save('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat',...
    'TE_PWM8_fsw500k', 'marginPWM8_fsw500k', 'fclk_SYS', '-append')%save simulation
     and loop tf parameters
```

5.2.4 Code for design of system with DDPM-DPWM combination modulator

```
clc;
clear;
close all;
%%
%Data
n=100;% use this variable to number plots consequentially
f1=1000; %Hz; frequency of input signal
f2=300; %Hz; frequency of second component input signal to test IMD
fclk_SYS=5e6; %Hz; clock frequency of the system
fclk_SYS=2^floor(log2(fclk_SYS)); %Hz; clock frequency of the system
nbit_ADC=16; %ADC number of bit
Fsw=1e6;
Fsw=2^ceil(log2(Fsw));%Hz; switching frequency
VPS_OS_p=10;% $V positive power supply of output stage
VPS_OS_n=0;% $V negative power supply of output stage
Vtr_p=5;%V triangular positive peak
Vtr_n=0; %V triangular waveform positive peak
Vtr_pp=Vtr_p-Vtr_n; %V triangular waveform negative peak
VPS_PWM=Vtr_p; %V power supply triangular waveform generator
%DPWM
nbit_DPWM=8;%number of bits of DPWM
fclk_DPWM=Fsw*2^nbit_DPWM;%Hz; clock frequency of DPWM counter

G=1/2;%attenuator in feedback path
%digital blocks parameters
VDD=VPS_PWM; %V; power supply of input ADC; I consider the supply of digital
                %part the same of the modulator because
                %implemented in the same digital hardware. Moreover, I
                %consider Vtr_p=VPS_PWM
Amp=.9*VDD/2; %amplitude of sinewave input signal
A=VPS_OS_p-VPS_OS_n;
Adigital_ol=2^nbit_DPWM/2^(nbit_ADC-1); %gain to remapping the output of ADC on
    nbit_PWM
Adigital_cl=2^nbit_DPWM/2^nbit_ADC; %gain to remapping the output of compensator on
     nbit_PWM
if Adigital_cl>1%in case the resPWM>resADC
    Adigital_cl=1;
end
if Adigital_ol>1%in case the resPWM>resADC
    Adigital_ol=1;
end
GM=1/2^nbit_DPWM; % gain of PWM
G_ADC=2^nbit_ADC/(2*VDD); %analog to digital scale factor
G_PWM=2^nbit_DPWM/VDD; %digital to analog scale factor
G_Trunc_LSBs_cl=2^(2*nbit_DPWM)/2^(nbit_ADC);%truncation gain in CL system
G_Trunc_LSBs_ol=2^(2*nbit_DPWM)/2^(nbit_ADC-1);%truncation gain in OL
if G_Trunc_LSBs_cl>1
    G_Trunc_LSBs_cl=1;
end
if G_Trunc_LSBs_ol>1
    G_Trunc_LSBs_ol=1;
end
Pn_PWM=VDD^2/(12*2^(2*2*nbit_DPWM)); %PWM power noise
Pn_PWM_dB=pow2db(Pn_PWM); %PWM power noise in dB
Sn_PWM=VDD^2/2^(2*2*nbit_DPWM)/(6*Fsw); %quantization noise power spectral density
```

```
    of DPWM
Pn_ADC=(2*VDD)^2/(12*2^(2*nbit_ADC)); %ADC power noise
Pn_ADC_dB=pow2db(Pn_ADC); %ADC power noise in dB
Sn_ADC=(2*VDD)^2/2^(2*nbit_ADC)/(6*fclk_SYS); %quantization noise power spectral
    density of ADC
PM=pi/3; %rad; phase margin chosen
%Simulation parameters
t_int=min([1/(4*fclk_SYS) 1/fclk_DPWM]); %integration time for Simulink simulation
dec_fac=1/t_int/(4*fclk_SYS); %decimation factor to sample the output signal always
     at 1/t_int
if dec_fac<1
    dec_fac=1;
end
t_int_dec=t_int*dec_fac; %sample time of output signals
t_sim=1.5;%s
deadtime=2^ceil(log2(10e-9)); %s: deadtime to simulate the no shortcircuit of ouput
     stage

s=tf('s');
%design output LPF
fpf=20e3; %Hz; LPF pole frequency
omega_pf=2*pi*fpf; %rad/s; pole frequency of output filter
LPF=zpk(minreal(1/(1+s/omega_pf)^2)); %tf of output filter
[num_LPF, den_LPF]=tfdata(LPF, 'v');% num and den coefficients of output frequency
    transfer function

%output filter
figure(n)
n=n+1;
plot_LPF=bodeplot(LPF,'r');
setoptions(plot_LPF,'FreqUnits','Hz');
title('Output filter transfer function')

%ZOH filter
T1=1/(fclk_SYS);%sample frequency of ADC
ZOH_ADC=1/(1+s*T1/2);%Sample&Hold related to ADC
ZOH_COMP=ZOH_ADC;%because of ADC and FPGA works at the same frequency, the two ZOH
    tf are equal
T2=1/Fsw;%switching sample time
ZOH_REG=1/(1+s*T2/2);%Sample&Hold related to the hold register before the PWM
    sampling
figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_ZOHadc=bodeplot(ZOH_ADC, 'r');
plot_ZOHreg=bodeplot(ZOH_REG, 'b');
setoptions(plot_ZOHadc, 'FreqUnits','Hz')
setoptions(plot_ZOHreg, 'FreqUnits','Hz')
legend('ZOH ADC', 'ZOH REG')
ZOH_ADC_NoApprox=(1-exp(-T1*s))/s*1/T1;
ZOH_REG_NoApprox=(1-exp(-T2*s))/s*1/T2;
figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_ZOHadc=bodeplot(ZOH_ADC_NoApprox, 'r');
plot_ZOHreg=bodeplot(ZOH_REG_NoApprox, 'b');
setoptions(plot_ZOHadc, 'FreqUnits','Hz')
setoptions(plot_ZOHreg, 'FreqUnits','Hz')
title('ZOHs transfer function')
```

```
legend('ZOH ADC', 'ZOH REG')
%%
%compensator with first order feedback filter (DPWM); feedback signal from output
    stage
p=3;
omega_u=2*pi*Fsw*1/10;
GBW=omega_u/(2*pi);
omega_p2=225e3*2*pi; %rad/s; pole of the feedback filter
omega_p2_H=1e6*2*pi; %rad/s; second pole to reduce aliasing of DPWM;
Gf_HF=zpk(minreal(1/(1+s/omega_p2)*1/(1+s/omega_p2_H)^2));%feedback filter transfer
     function
figure(n)%plot transfer functions of ZOH
n=n+1;
hold on
plot_AAfilter=bodeplot(Gf_HF, 'r');
setoptions(plot_AAfilter,'FreqUnits','Hz');
title('Anti-aliasing filter transfer function')
[num_GfHF, den_GfHF]=tfdata(Gf_HF, 'v');
omega_ph=9*omega_u; %pole to close the band
M=2; %order of last pole
extra_PM=atan(omega_u/(2*Fsw));
omega_z1=omega_u/tan((PM-pi+p*pi/2+extra_PM)/(p-1));%zero to obtain the correct
    phase margin
tau_p=(GM*A*G*Adigital_cl*G_ADC/(omega_u*omega_z1^(p-1)))^(1/p);
if omega_z1<=0 && p>1
    error('The compensation zero is negative')
end
if p==1
    tau_p=(GM*A*G*G_ADC*Adigital_cl/omega_u)^1/p;
    omega_z1=1;
end
Gc_new=zpk(minreal((1+s/(9/10*omega_z1))*(1+s/(11/10*omega_z1)*(1+s/omega_p2))*1/(
    tau_p*s)^p*1/((1+s/omega_ph)^M)));
[num_Gcnew, den_Gcnew]=tfdata(Gc_new, 'v');
T_filter=ZOH_ADC*ZOH_REG*GM*A*Gc_new*G*Gf_HF*Adigital_cl*G_ADC;%Loop transfer
    function with output filter in feedback
P_filter=1/(1+T_filter);%senitivity function with output filter in feedback
H_filter=ZOH_ADC*ZOH_REG*Gc_new*GM*A*Adigital_cl*G_ADC/(1+T_filter);%in-out
    transfer function with output filter in feedback loop
G_filter=ZOH_ADC*Gc_new*G_ADC/(1+T_filter);
%digital domain
Gc_new_z=zpk(minreal(c2d(Gc_new, 1/fclk_SYS, 'matched')));%digital compensator
[numGc_new_z, denGc_new_z]=tfdata(Gc_new_z, 'v');
T_filter_z=zpk(minreal(...
    Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*G*Gf_HF*Adigital_cl*G_ADC));%digital loop
    transfer function
P_filter_z=1/(1+T_filter_z);%sensitovity transfer function
H_filter_z=zpk(minreal(...
    Gc_new*ZOH_ADC*ZOH_COMP*ZOH_REG*GM*A*Adigital_cl*G_ADC/(1+T_filter_z)));
%in-out transfer function
G_filter_z=ZOH_ADC*ZOH_COMP*Gc_new*G_ADC/(1+T_filter);
%in-command transfer function

figure(n)%plot compensator transfer function (digital and analog)
n=n+1;
Gcnew_plot=bodeplot(Gc_new, Gc_new_z);
setoptions(Gcnew_plot, 'FreqUnits', 'Hz')
legend('Analog', 'Digital')
title('Compensator transfer function')
```

```

figure(n)%plot loop transfer function (digital and analog)
n=n+1;
hold on
plot_Tfilter=bodeplot(T_filter, 'b');
setoptions(plot_Tfilter,'FreqUnits','Hz');
plot_Tfilter_z=bodeplot(T_filter_z, 'r');
setoptions(plot_Tfilter_z,'FreqUnits','Hz');
xlim([100 10e6])
legend('Analog', 'Digital')
title('Loop function with anti-alising filter in loop')

figure(n)%plot sensitivity transfer function (digital and analog)
n=n+1;
hold on
plot_Pfilter=bodeplot(P_filter, 'b');
setoptions(plot_Pfilter,'FreqUnits','Hz');
plot_Pfilter_z=bodeplot(P_filter_z, 'r');
setoptions(plot_Pfilter_z,'FreqUnits','Hz');
xlim([100 10e6])
legend('Analog', 'Digital')
title('Sensitivity function with anti-alising filter in loop')

figure(n)%in-out transfer function
n=n+1;
hold on
plot_Hfilter=bodeplot(H_filter, 'b');
setoptions(plot_Hfilter,'FreqUnits','Hz');
plot_Hfilter_z=bodeplot(H_filter_z, 'r');
setoptions(plot_Hfilter_z,'FreqUnits','Hz');
xlim([100 10e6])
legend('Analog', 'Digital')
title('Input-output with anti-alising filter in loop')
%step response input/output
t_step=0:1/fclk_SYS:100e-6;
step_Hfilter_z=step(H_filter_z, t_step);
step_Hfilter=step(H_filter, t_step);
%step response input/compensator
step_Gfilter_z=step(G_filter_z, t_step);
step_Gfilter=step(G_filter, t_step);

figure(n)%plot output setp response
n=n+1;
plot(t_step, step_Hfilter_z, 'r', t_step, step_Hfilter, 'b')
legend('Digital', 'Analog')
xlabel('Time [s]')
ylabel('Amplitude [V]')
title('Step response with anti-alising filter in loop')

figure(n)%plot command step response
n=n+1;
plot(t_step, step_Gfilter_z, 'r', t_step, step_Gfilter, 'b')
legend('Digital', 'Analog')
xlabel('Time [s]')
ylabel('Amplitude [V]')
title('Step response of compesator with anti-alising filter in loop')
%%
%SNR theoretical
NTF_PWM_filter_z=GM*A*G*G_PWM/(1+T_filter_z)*LPF; %noise transfer function of DPWM
```

```
    quantization noise
NTF_ADC_filter_z=H_filter_z*LPF; %noise transfer function of ADC quantization noise
NTF_ADC_ol=2*G_ADC*GM*A*LPF*Adigital_cl;
NTF_PWM_ol=GM*A*G_PWM*LPF;

figure('Name', 'Digital noise transfer function closed-loop') %bode blot of NTF
hold on
plot_NTF_cl=bodeplot(NTF_PWM_filter_z , 'b', NTF_ADC_filter_z , 'r');
setoptions(plot_NTF_PWM,'FreqUnits','Hz');
legend('NTF PWM', 'NTF ADC')
title('Digital noise transfer function closed-loop')
figure('Name', 'Noise transfer function open-loop') %bode blot of NTF
hold on
plot_NTF_ol=bodeplot(NTF_PWM_ol , 'b', NTF_ADC_ol , 'r');
setoptions(plot_NTF_PWM_ol,'FreqUnits','Hz');
legend('NTF PWM', 'NTF ADC')
title('Noise transfer function open-loop')
DCgain=1/G;
Amp_out=DCgain*Amp;
%evaluate power and SNR of PWM filtered by the system
[SNR_dB_PWM_cl, Pn_dB_PWM_cl, Ps_dB]=...
    SNRtheory(Amp_out, Sn_PWM, NTF_PWM_filter_z, 20e3);
%evaluate power and SNR of ADC filtered by the system
[SNR_dB_ADC_cl, Pn_dB_ADC_cl, ~]=...
    SNRtheory(Amp_out, Sn_ADC, NTF_ADC_filter_z, 20e3);
%evaluate power and SNR of PWM of open-loop system
[SNR_dB_PWM_ol, Pn_dB_PWM_ol, ~]=...
    SNRtheory(Amp_out, Sn_PWM, NTF_PWM_ol, 20e3);
%evaluate power and SNR of ADC of open-loop system
[SNR_dB_ADC_ol, Pn_dB_ADC_ol, ~]=...
    SNRtheory(Amp_out, Sn_ADC, NTF_ADC_ol, 20e3);
%Compute total power...
%...of closed-loop system...
Pn_tot_cl=10^(Pn_dB_PWM_cl/10)+10^(Pn_dB_ADC_cl/10);%V^2
Pn_tot_dB_cl=pow2db(Pn_tot_cl); %dB
Ps=10^(Ps_dB/10); %power signal (V^2)
SNR_dB_cl=10*log10(Ps/Pn_tot_cl); %SNR due to noise in system
...and open-loop one
Pn_tot_ol=10^(Pn_dB_PWM_ol/10)+10^(Pn_dB_ADC_ol/10);%V^2
Pn_tot_dB_ol=pow2db(Pn_tot_ol); %dB
SNR_dB_ol=10*log10(Ps/Pn_tot_ol); %SNR due to noise in system
save power noises and SNR in a structure
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_PWM_dB=Pn_PWM_dB;
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_ADC_dB=Pn_ADC_dB;
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_dB_PWM_cl=Pn_dB_PWM_cl;
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_dB_ADC_cl=Pn_dB_ADC_cl;
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_dB_tot_cl=Pn_tot_dB_cl;
SNRandPn_DDPM8_DPWM8_fsw1M.SNR_dB_cl=SNR_dB_cl;
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_dB_PWM_ol=Pn_dB_PWM_ol;
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_dB_ADC_ol=Pn_dB_ADC_ol;
SNRandPn_DDPM8_DPWM8_fsw1M.Pn_dB_tot_ol=Pn_tot_dB_ol;
SNRandPn_DDPM8_DPWM8_fsw1M.SNR_dB_ol=SNR_dB_ol;
save('DatastoreFiles/SNRandPn_nbitADC16_DDPM_DPWM_comb_fclkSYS_LOW.mat',...
    'SNRandPn_DDPM8_DPWM8_fsw1M', '-append')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%save parameters simulations
DDPM8_DPWM8_fsw1M.Fsw=Fsw;%save switching frequency
DDPM8_DPWM8_fsw1M.fclk_PWM=fclk_DPWM;%save PWM TE clock
DDPM8_DPWM8_fsw1M.t_int=t_int;%save integration time
```

```
DDPM8_DPWM8_fsw1M.dec_fac=dec_fac;%save decimation factor
DDPM8_DPWM8_fsw1M.t_int_dec=t_int_dec; %save sample time of system outputs (except
    triangular waveform)
[Gm, Pm, Wcg, Wpg]=margin(T_filter_z);%evaluate gain and phase margins with
    correspondet frequency
marginDDPM8_DPWM8_fsw1M.Gm=Gm;
marginDDPM8_DPWM8_fsw1M.Pm=Pm;
marginDDPM8_DPWM8_fsw1M.Fcg=Wcg/(2*pi);
marginDDPM8_DPWM8_fsw1M.Fpg=Wpg/(2*pi);
StepInfo_fsw1M_out=stepinfo(step_Hfilter_z,t_step);
StepInfo_fswfsw1M_comp=stepinfo(step_Gfilter_z,t_step);
save('DatastoreFiles/SimParameters_nbitADC16_DDPM_DPWM_fclkSYS_LOW.mat',...
    'DDPM8_DPWM8_fsw1M', 'marginDDPM8_DPWM8_fsw1M', 'fclk_SYS', 'StepInfo_fsw1M_out
    ',...
    'StepInfo_fswfsw1M_comp', '-append')%save simulation and loop tf parameters
```

5.2.5 Code for theoretical SNR evaluation

```
function [SNR_dB_theory, Pn_dB, Ps_dB]=...
    SNRtheory(Amp, Sn, NTF, fband)
%evaluate and compare SNR and power noise of a uniform distributed
%quantization noise and input sinusoidal signal with amplitude Amp,
%starting from the transfer function of the system
%NTF and noise power spectral density Sn;
%extract magnitude and frequency with a df=1 Hz

%magnitude and frequency vectore extraction
[magNTF, ~,freqNTF]=bode(NTF, 1*2*pi:1*2*pi:(fband+fband/2)*2*pi);
freqNTF=freqNTF/(2*pi); %bode gives freq in rad/s; convert to Hz
magNTF=magNTF(1, 1, 1:fband); %select magnitude between 0 and 20 kHz
%power noise computation with trapezoidal numerical integral
power_noise=trapz(freqNTF(1:fband), Sn.*magNTF(:).^2);
Pn_dB=pow2db(power_noise); %evaluate power noise in dB
%power signal and SNR evaluation
power_signal=Amp^2/2; %evaluate signal power
Ps_dB=pow2db(power_signal); %evaluate signal power in dB
SNR_dB_theory=pow2db(power_signal/power_noise); %evaluate SNR in dB
end
```

5.2.6 Code for DDPM algorithm

```
function DDPM=DDPM_conversion(DAC_in, counter, nbit, VDD)
%DAC_in: number related to the ADC conversion (encode)
%counter: output of a free running counter
%nbit: number of bit for conversion
    DAC_in=flip(dec2bin(cast(DAC_in, 'double'), nbit));
    counter=dec2bin(counter, nbit);
    DDPM=real(VDD*str2double(DAC_in(find((counter=='1'), 1, 'last'))));
        if(isnan(DDPM))
            DDPM=0;
        end
end
```

5.2.7 Code to set simulation parameters

```
%open loop simulation
sys='CDA_openloop_DDPWM';
open_system(sys)
%Set the PWM with sawtooth waveform and step as input signal
%strcture data for set parametr for simulation
paramNameValStruct.SaveTime='off';
paramNameValStruct.ReturnWorkspaceOutputs='off';
paramNameValStruct.SaveState='off';
paramNameValStruct.SaveFinalState='off';
paramNameValStruct.SaveOutput='off';
paramNameValStruct.DSMLogging='off';
paramNameValStruct.ReturnWorkspaceOutputsName='simOut_input1kHzsinewave_cl';
paramNameValStruct.SignalLogging='on';
paramNameValStruct.SignalLoggingName='outs_openloop';
paramNameValStruct.LoggingToFile='on';
paramNameValStruct.LoggingFileName='DatastoreFiles/
    simOut_ol_1kHzsinewave_fsw500k_tsim1.5_nbitADC16_nbitPWM9_DDPWM_N8M4.mat';
paramNameValStruct.FixedStep='t_int';
paramNameValStruct.StopTime='t_sim';
paramNameValStruct.Solver='FixedStepAuto';
paramNameValStruct.SolverType='Fixed-step';
set_param(sys, 'SaveTime', paramNameValStruct.SaveTime,...
               'ReturnWorkspaceOutputs', paramNameValStruct.ReturnWorkspaceOutputs
    ,...
               'FixedStep', paramNameValStruct.FixedStep,...
               'StopTime', paramNameValStruct.StopTime, ...
               'Solver', paramNameValStruct.Solver,...
               'SolverType', paramNameValStruct.SolverType,...
               'SignalLogging', paramNameValStruct.SignalLogging,...
               'SignalLoggingName', paramNameValStruct.SignalLoggingName,...
               'LoggingToFile', paramNameValStruct.LoggingToFile,...
               'LoggingFileName', paramNameValStruct.LoggingFileName,...
               'SaveState', paramNameValStruct.SaveState,...
               'SaveFinalState', paramNameValStruct.SaveFinalState,...
               'SaveOutput', paramNameValStruct.SaveOutput,...
               'DSMLogging', paramNameValStruct.DSMLogging)

%%
%closed loop simulation
sys='CDA_closedloop_OFinFP';
open_system(sys)
%Set the PWM with saw tooth waveform and step as input signal
paramNameValStruct.SaveTime='off';
paramNameValStruct.ReturnWorkspaceOutputs='off';
paramNameValStruct.SaveState='off';
paramNameValStruct.SaveFinalState='off';
paramNameValStruct.SaveOutput='off';
paramNameValStruct.DSMLogging='off';
paramNameValStruct.SignalLogging='on';
paramNameValStruct.SignalLoggingName='outs_closedloop';
paramNameValStruct.LoggingToFile='on';
paramNameValStruct.LoggingFileName='DatastoreFiles/
    simOut_cl_1kHzsinewave_OFinFP_fsw500k_tsim1.5_nbitADC16_nbitPWM8_TrailingEdge.
    mat';
paramNameValStruct.FixedStep='t_int';
paramNameValStruct.StopTime='t_sim';
paramNameValStruct.Solver='FixedStepAuto';
paramNameValStruct.SolverType='Fixed-step';
set_param(sys, 'SaveTime', paramNameValStruct.SaveTime,...
               'ReturnWorkspaceOutputs', paramNameValStruct.ReturnWorkspaceOutputs
    ,...
               'FixedStep', paramNameValStruct.FixedStep,...
               'StopTime', paramNameValStruct.StopTime, ...
               'Solver', paramNameValStruct.Solver,...
               'SolverType', paramNameValStruct.SolverType,...
               'SignalLogging', paramNameValStruct.SignalLogging,...
               'SignalLoggingName', paramNameValStruct.SignalLoggingName,...
               'LoggingToFile', paramNameValStruct.LoggingToFile,...
               'LoggingFileName', paramNameValStruct.LoggingFileName,...
               'SaveState', paramNameValStruct.SaveState,...
               'SaveFinalState', paramNameValStruct.SaveFinalState,...
               'SaveOutput', paramNameValStruct.SaveOutput,...
               'DSMLogging', paramNameValStruct.DSMLogging)
```

5.2.8 Code for the time analysis of system with DPWM modulator working at $f_{sw} \sim 500\,\mathrm{kHz}$

```
%set datastores and times
close all
clear
clc
t_sim=1.5;%s simulation time (Stop Time)
t_win=1;%s; time window to obtain 1 Hz resolution in fft analysis
n=1;
%closed loop trailing edge datastore
ds_cl_nbitADC16_nbitPWM10_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM10_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM8_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM7_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM7_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM6_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM6_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM5_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM5_TrailingEdge.mat', 'outs_closedloop');
% open loop trailing edge datastore
ds_ol_nbitADC16_nbitPWM8_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge.mat', 'outs_openloop');
ds_ol_nbitADC16_nbitPWM7_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM7_TrailingEdge.mat', 'outs_openloop');
ds_ol_nbitADC16_nbitPWM6_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM6_TrailingEdge.mat', 'outs_openloop');
ds_ol_nbitADC16_nbitPWM5_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM5_TrailingEdge.mat', 'outs_openloop');
%load all t_int
load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat',...
    'TE_PWM10_fsw500k', 'TE_PWM8_fsw500k', 'TE_PWM5_fsw500k',...
    'TE_PWM6_fsw500k', 'TE_PWM7_fsw500k', 'fclk_SYS');
%import t_int_dec
t_int_dec_PWM10=TE_PWM10_fsw500k.t_int_dec;
t_int_dec_PWM8=TE_PWM8_fsw500k.t_int_dec;
t_int_dec_PWM7=TE_PWM7_fsw500k.t_int_dec;
t_int_dec_PWM6=TE_PWM6_fsw500k.t_int_dec;
t_int_dec_PWM5=TE_PWM5_fsw500k.t_int_dec;
%import t_int
t_int_PWM10=TE_PWM10_fsw500k.t_int;
t_int_PWM8=TE_PWM8_fsw500k.t_int;
t_int_PWM7=TE_PWM7_fsw500k.t_int;
t_int_PWM6=TE_PWM6_fsw500k.t_int;
t_int_PWM5=TE_PWM5_fsw500k.t_int;
%%
%Trailing Edge
%define time vector; define the read
```

```
%number of sample to read 4 period of traingular waveform
read_size_tr_PWM10=uint64(4*(t_int_PWM10*2^10)/t_int_PWM10+1);
read_size_tr_PWM8=uint64(4*(t_int_PWM8*2^8)/t_int_PWM8+1);
read_size_tr_PWM7=uint64(4*(t_int_PWM7*2^7)/t_int_PWM7+1);
read_size_tr_PWM6=uint64(4*(t_int_PWM6*2^6)/t_int_PWM6+1);
read_size_tr_PWM5=uint64(4*(t_int_PWM5*2^5)/t_int_PWM5+1);

%set ReadSize to read first 4 periods of triangular waveform
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM10;
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM5;

%read first 4 periods of triangular waveform and extract Time data
time_PWM10_tr=read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values).Time;
time_PWM8_tr=read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values).Time;
time_PWM7_tr=read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values).Time;
time_PWM6_tr=read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values).Time;
time_PWM5_tr=read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values).Time;

%reset read pointer to the first sample of the signal
%(necessary to read fromn the start each signal)
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values);

%set ReadSize lenght again
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM10;
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM5;

figure(n)%plot triangular waveform
n=n+1;
hold on
stairs(time_PWM10_tr, read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values).Data,
    'b')
stairs(time_PWM8_tr, read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values).Data, 'r
    ')
stairs(time_PWM7_tr, read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values).Data, 'm
    ')
stairs(time_PWM6_tr, read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values).Data, 'g
    ')
stairs(time_PWM5_tr, read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values).Data, 'c
    ')
legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Triangular waveforms trailing edge f_{sw}\sim500 kHz')
xlabel('time [s]')
ylabel('Amplitude [V]')

%reset read pointer to the first sample of the signal
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values);
```

```
%%
%set time vector to plot five period of sinewave
T=1e-3;%s period of input/output signal (sinewave)
n_period=5; %number of output signal periods that we want to see
read_size_PWM8=uint64(T/t_int_dec_PWM8*n_period);% #samples in n_period
read_size_PWM7=uint64(T/t_int_dec_PWM7*n_period);
read_size_PWM6=uint64(T/t_int_dec_PWM6*n_period);
read_size_PWM5=uint64(T/t_int_dec_PWM5*n_period);
read_size_PWM10=uint64(T/t_int_dec_PWM10*n_period);
%%set ReadSize to read first n_period of single-ended output
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values.ReadSize=read_size_PWM10;
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values.ReadSize=read_size_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values.ReadSize=read_size_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values.ReadSize=read_size_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values.ReadSize=read_size_PWM5;
%extract Time vector
time_PWM10=read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values).Time;
time_PWM8=read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values).Time;
time_PWM7=read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values).Time;
time_PWM6=read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values).Time;
time_PWM5=read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values).Time;

%reset read length
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values);

figure(n)%plot single-ended output of open-loop system
n=n+1;
hold on
plot(time_PWM8, read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values).Data, 'r')
plot(time_PWM7, read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values).Data, 'm')
plot(time_PWM6, read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values).Data, 'g')
plot(time_PWM5, read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values).Data, 'c')

legend('nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Single-ended output open-loop (f_{sw}\sim 500 kHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')
%reset read pointer to the first sample of the signal
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values);

%set ReadSize to read first n_period of differential output
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{7}.Values.ReadSize=read_size_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{7}.Values.ReadSize=read_size_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{7}.Values.ReadSize=read_size_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{7}.Values.ReadSize=read_size_PWM5;
figure(n)%plot differential output (time vector is the same of single ended
        %becuase the sample time is the same)
n=n+1;
hold on
plot(time_PWM8, read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{7}.Values).Data, 'r')
plot(time_PWM7, read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{7}.Values).Data, 'm')
plot(time_PWM6, read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{7}.Values).Data, 'g')
plot(time_PWM5, read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{7}.Values).Data, 'c')
```

```

legend('nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Differential output open-loop (f_{sw}\sim 500 kHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')
%reset read pointer to the first sample of the signal
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{7}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{7}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{7}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{7}.Values);
%%
%plot single-ended voltages of closed-loop system
%set read size to read the first n_period of signle-ended signals
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{6}.Values.ReadSize=read_size_PWM10;
ds_cl_nbitADC16_nbitPWM8_TrailingEdge{6}.Values.ReadSize=read_size_PWM8;
ds_cl_nbitADC16_nbitPWM7_TrailingEdge{6}.Values.ReadSize=read_size_PWM7;
ds_cl_nbitADC16_nbitPWM6_TrailingEdge{6}.Values.ReadSize=read_size_PWM6;
ds_cl_nbitADC16_nbitPWM5_TrailingEdge{6}.Values.ReadSize=read_size_PWM5;

figure(n)%plot single-ended voltages of closed-loop system
n=n+1;
hold on
plot(time_PWM10, read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{6}.Values).Data, 'b')
plot(time_PWM8, read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{6}.Values).Data, 'r')
plot(time_PWM7, read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{6}.Values).Data, 'm')
plot(time_PWM6, read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{6}.Values).Data, 'g')
plot(time_PWM5, read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{6}.Values).Data, 'c')

legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Single-ended output closed-loop (f_{sw} \sim 500 kHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')
%reset read pointer to the first sample of the signal
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{6}.Values);
%%
%plot differential output of closed-loop system
%reset read pointer to the first sample of the signal
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values);
%set read size to read the forst n_period of the signals
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values.ReadSize=read_size_PWM10;
ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values.ReadSize=read_size_PWM8;
ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values.ReadSize=read_size_PWM7;
ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values.ReadSize=read_size_PWM6;
ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values.ReadSize=read_size_PWM5;

figure(n)%plot differential voltages of closed-loop system for different
        %number of DPWM bits
n=n+1;
hold on
plot(time_PWM10, read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values).Data, 'b')
plot(time_PWM8, read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values).Data, 'r')
plot(time_PWM7, read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values).Data, 'm')
plot(time_PWM6, read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values).Data, 'g')
plot(time_PWM5, read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values).Data, 'c')

legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Differential output closed-loop (f_{sw} \sim 500 kHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')
%reset read pointer to the first sample of the signal
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values);
%%
%Compensator output
n_period=3;
t_int_comp=1/fclk_SYS;%s; sample period of the compensator signal
read_size_PWM8_comp=uint64(T/t_int_comp*n_period);% #samples in n_period
read_size_PWM7_comp=uint64(T/t_int_comp*n_period);
read_size_PWM6_comp=uint64(T/t_int_comp*n_period);
read_size_PWM5_comp=uint64(T/t_int_comp*n_period);
read_size_PWM10_comp=uint64(T/t_int_comp*n_period);

%set read size of compensator data to read the first n_period
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values.ReadSize=read_size_PWM10_comp;
ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values.ReadSize=read_size_PWM8_comp;
ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values.ReadSize=read_size_PWM7_comp;
ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values.ReadSize=read_size_PWM6_comp;
ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values.ReadSize=read_size_PWM5_comp;
%read time vector of each compensator signal and save them in different
%variables
time_PWM10_comp=read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values).Time;
time_PWM8_comp=read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values).Time;
time_PWM7_comp=read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values).Time;
time_PWM6_comp=read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values).Time;
time_PWM5_comp=read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values).Time;
%reset read pointer to the first sample of the signal
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values);

figure(n)%plot compensator output signal (command) at varying pf DPWM number
        %of bits
n=n+1;
hold on
stairs(time_PWM10_comp, read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values).Data
    , 'b')
stairs(time_PWM8_comp, read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values).Data,
    'r')
stairs(time_PWM7_comp, read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values).Data,
    'm')
stairs(time_PWM6_comp, read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values).Data,
    'g')
stairs(time_PWM5_comp, read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values).Data,
    'c')

```

```
legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Compensator output closed-loop (f_{sw} \sim 500 kHz)')
xlabel('time [s]')
ylabel('Code')
%reset read pointer to the first sample of the signal
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values);
```

5.2.9 Code for the time analysis of system with DPWM modulator working at $f_{sw} \sim 1\,\mathrm{MHz}$

```
%set datastores and times
close all
clear
clc
t_sim=1.5;%s; simulation time (Stop Time)
t_win=1;%s; time window to obtain 1 Hz resolution in fft analysis
n=1;
% closed loop trailing edge datastore
ds_cl_nbitADC16_nbitPWM10_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM10_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM8_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM7_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM7_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM6_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM6_TrailingEdge.mat', 'outs_closedloop');
ds_cl_nbitADC16_nbitPWM5_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM5_TrailingEdge.mat', 'outs_closedloop');
% open loop trailing edge datastore
ds_ol_nbitADC16_nbitPWM8_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge.mat', 'outs_openloop');
ds_ol_nbitADC16_nbitPWM7_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM7_TrailingEdge.mat', 'outs_openloop');
ds_ol_nbitADC16_nbitPWM6_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM6_TrailingEdge.mat', 'outs_openloop');
ds_ol_nbitADC16_nbitPWM5_TrailingEdge=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM5_TrailingEdge.mat', 'outs_openloop');
%load all t_int
load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat',...
    'TE_PWM10_fsw1M', 'TE_PWM8_fsw1M', 'TE_PWM5_fsw1M',...
    'TE_PWM6_fsw1M', 'TE_PWM7_fsw1M', 'fclk_SYS');
%import t_int_dec
t_int_dec_PWM10=TE_PWM10_fsw1M.t_int_dec;
t_int_dec_PWM8=TE_PWM8_fsw1M.t_int_dec;
t_int_dec_PWM7=TE_PWM7_fsw1M.t_int_dec;
t_int_dec_PWM6=TE_PWM6_fsw1M.t_int_dec;
t_int_dec_PWM5=TE_PWM5_fsw1M.t_int_dec;
%import t_int
t_int_PWM10=TE_PWM10_fsw1M.t_int;
t_int_PWM8=TE_PWM8_fsw1M.t_int;
t_int_PWM7=TE_PWM7_fsw1M.t_int;
t_int_PWM6=TE_PWM6_fsw1M.t_int;
t_int_PWM5=TE_PWM5_fsw1M.t_int;
%%
%Trailing Edge
%define time vector; define the read
```

```
%number of sample to read 4 periods of traingular waveform
read_size_tr_PWM10=uint64(4*(t_int_PWM10*2^10)/t_int_PWM10+1);
read_size_tr_PWM8=uint64(4*(t_int_PWM8*2^8)/t_int_PWM8+1);
read_size_tr_PWM7=uint64(4*(t_int_PWM7*2^7)/t_int_PWM7+1);
read_size_tr_PWM6=uint64(4*(t_int_PWM6*2^6)/t_int_PWM6+1);
read_size_tr_PWM5=uint64(4*(t_int_PWM5*2^5)/t_int_PWM5+1);

%set ReadSize to read first 4 periods of triangular waveform
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM10;
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM5;

%read first 4 periods of triangular waveform and extract Time data
time_PWM10_tr=read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values).Time;
time_PWM8_tr=read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values).Time;
time_PWM7_tr=read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values).Time;
time_PWM6_tr=read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values).Time;
time_PWM5_tr=read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values).Time;

%reset read pointer to the position of the first sample (necessary to read from the
    start each signal)
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values);

%set ReadSize lenght again
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM10;
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values.ReadSize=read_size_tr_PWM5;

figure(n)%plot triangular waveform
n=n+1;
hold on
stairs(time_PWM10_tr, read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{2}.Values).Data,
    'b')
stairs(time_PWM8_tr, read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values).Data, 'r
    ')
stairs(time_PWM7_tr, read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values).Data, 'm
    ')
stairs(time_PWM6_tr, read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values).Data, 'g
    ')
stairs(time_PWM5_tr, read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values).Data, 'c
    ')
legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Triangular waveforms trailing edge f_{sw}\sim 1 MHz')
xlabel('time [s]')
ylabel('Amplitude [V]')

%reset read pointer to position of the first sample
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{2}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{2}.Values);
```

```
%%
%set time vector to plot five period of sinewave
T=1e-3;%s period of input/output signal (sinewave)
n_period=5; %number of output signal periods that we want to see
read_size_PWM8=uint64(T/t_int_dec_PWM8*n_period);% #samples in n_period
read_size_PWM7=uint64(T/t_int_dec_PWM7*n_period);
read_size_PWM6=uint64(T/t_int_dec_PWM6*n_period);
read_size_PWM5=uint64(T/t_int_dec_PWM5*n_period);
read_size_PWM10=uint64(T/t_int_dec_PWM10*n_period);
%set ReadSize to read first n_period of single-ended output
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values.ReadSize=read_size_PWM10;
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values.ReadSize=read_size_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values.ReadSize=read_size_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values.ReadSize=read_size_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values.ReadSize=read_size_PWM5;
%extract Time vector
time_PWM10=read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values).Time;
time_PWM8=read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values).Time;
time_PWM7=read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values).Time;
time_PWM6=read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values).Time;
time_PWM5=read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values).Time;

%reset read pointer to position of the first sample
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values);

figure(n)%plot single-ended output
n=n+1;
hold on
plot(time_PWM8, read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values).Data, 'r')
plot(time_PWM7, read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values).Data, 'm')
plot(time_PWM6, read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values).Data, 'g')
plot(time_PWM5, read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values).Data, 'c')

legend('nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Single-ended output open-loop (f_{sw}\sim 1 MHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')

%reset read pointer to position of the first sample
reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{5}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{5}.Values);

%%set ReadSize to read first n_period of differential output
ds_ol_nbitADC16_nbitPWM8_TrailingEdge{7}.Values.ReadSize=read_size_PWM8;
ds_ol_nbitADC16_nbitPWM7_TrailingEdge{7}.Values.ReadSize=read_size_PWM7;
ds_ol_nbitADC16_nbitPWM6_TrailingEdge{7}.Values.ReadSize=read_size_PWM6;
ds_ol_nbitADC16_nbitPWM5_TrailingEdge{7}.Values.ReadSize=read_size_PWM5;
figure(n)%plot differential output (time vector is the same of single ended
        %becuase the sample time is the same)
n=n+1;
hold on
plot(time_PWM8, read(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{7}.Values).Data, 'r')
plot(time_PWM7, read(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{7}.Values).Data, 'm')
plot(time_PWM6, read(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{7}.Values).Data, 'g')
plot(time_PWM5 , read(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{7}.Values).Data , 'c')

legend('nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Differential output open-loop (f_{sw}\sim 1 MHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')

reset(ds_ol_nbitADC16_nbitPWM8_TrailingEdge{7}.Values);
reset(ds_ol_nbitADC16_nbitPWM7_TrailingEdge{7}.Values);
reset(ds_ol_nbitADC16_nbitPWM6_TrailingEdge{7}.Values);
reset(ds_ol_nbitADC16_nbitPWM5_TrailingEdge{7}.Values);
%%
%plot Vop closed-loop
%set read size
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{6}.Values.ReadSize=read_size_PWM10;
ds_cl_nbitADC16_nbitPWM8_TrailingEdge{6}.Values.ReadSize=read_size_PWM8;
ds_cl_nbitADC16_nbitPWM7_TrailingEdge{6}.Values.ReadSize=read_size_PWM7;
ds_cl_nbitADC16_nbitPWM6_TrailingEdge{6}.Values.ReadSize=read_size_PWM6;
ds_cl_nbitADC16_nbitPWM5_TrailingEdge{6}.Values.ReadSize=read_size_PWM5;

figure(n)
n=n+1;
hold on
plot(time_PWM10 , read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{6}.Values).Data , 'b')
plot(time_PWM8 , read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{6}.Values).Data , 'r')
plot(time_PWM7 , read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{6}.Values).Data , 'm')
plot(time_PWM6 , read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{6}.Values).Data , 'g')
plot(time_PWM5 , read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{6}.Values).Data , 'c')

legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Single-ended output closed-loop (f_{sw} \sim 1 MHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')

%reset read pointer to position of the first sample
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{6}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{6}.Values);
%%
%reset read pointer to position of the first sample
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values);

%plot differential output closed-loop
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values.ReadSize=read_size_PWM10;
ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values.ReadSize=read_size_PWM8;
ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values.ReadSize=read_size_PWM7;
ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values.ReadSize=read_size_PWM6;
ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values.ReadSize=read_size_PWM5;

figure(n)
n=n+1;
hold on
plot(time_PWM10 , read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values).Data , 'b')
plot(time_PWM8, read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values).Data, 'r')
plot(time_PWM7, read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values).Data, 'm')
plot(time_PWM6, read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values).Data, 'g')
plot(time_PWM5, read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values).Data, 'c')

legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
title('Differential output closed-loop (f_{sw} \sim 1 MHz)')
xlabel('time [s]')
ylabel('Amplitude [V]')

reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{9}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{9}.Values);
%%
%Compensator output
n_period=3;
t_int_comp=1/fclk_SYS;
read_size_PWM8_comp=uint64(T/t_int_comp*n_period);% #samples in n_period
read_size_PWM7_comp=uint64(T/t_int_comp*n_period);
read_size_PWM6_comp=uint64(T/t_int_comp*n_period);
read_size_PWM5_comp=uint64(T/t_int_comp*n_period);
read_size_PWM10_comp=uint64(T/t_int_comp*n_period);

%set read size of compensator data
ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values.ReadSize=read_size_PWM10_comp;
ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values.ReadSize=read_size_PWM8_comp;
ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values.ReadSize=read_size_PWM7_comp;
ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values.ReadSize=read_size_PWM6_comp;
ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values.ReadSize=read_size_PWM5_comp;
%set time vector to plot compensator output
time_PWM10_comp=read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values).Time;
time_PWM8_comp=read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values).Time;
time_PWM7_comp=read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values).Time;
time_PWM6_comp=read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values).Time;
time_PWM5_comp=read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values).Time;

reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values);

figure(n)
n=n+1;
hold on
stairs(time_PWM10_comp, read(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values).Data
    , 'b')
stairs(time_PWM8_comp, read(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values).Data,
    'r')
stairs(time_PWM7_comp, read(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values).Data,
    'm')
stairs(time_PWM6_comp, read(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values).Data,
    'g')
stairs(time_PWM5_comp, read(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values).Data,
    'c')

legend('nbitPWM=10', 'nbitPWM=8', 'nbitPWM=7', 'nbitPWM=6', 'nbitPWM=5')
```

```
title('Compensator output closed-loop (f_{sw} \sim 1 MHz)')
xlabel('time [s]')
ylabel('Code')
reset(ds_cl_nbitADC16_nbitPWM10_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM8_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM7_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM6_TrailingEdge{7}.Values);
reset(ds_cl_nbitADC16_nbitPWM5_TrailingEdge{7}.Values);
```

5.2.10 Code to compute spectra

```
clc
clear
close all
%%
n=1;
ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5_nbitADC16_nbitPWM10_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5_nbitADC16_nbitPWM8_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5_nbitADC16_nbitPWM7_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5_nbitADC16_nbitPWM6_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5_nbitADC16_nbitPWM5_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5_nbitADC16_nbitPWM10_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5_nbitADC16_nbitPWM8_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5_nbitADC16_nbitPWM7_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5_nbitADC16_nbitPWM6_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5_nbitADC16_nbitPWM5_TrailingEdge.mat',...
    'outs_closedloop');
%Noise shaping
ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS=Simulink.SimulationData.
```

```
    DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM8_NS_TrailingEdge.mat',...
    'outs_closedloop');
ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_NS_TrailingEdge.mat',...
    'outs_closedloop');
%open-loop datastores
%fsw=1 MHz
ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge.mat',...
    'outs_openloop');
ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM7_TrailingEdge.mat',...
    'outs_openloop');
ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM6_TrailingEdge.mat',...
    'outs_openloop');
ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM5_TrailingEdge.mat',...
    'outs_openloop');
%fsw=500 kHz
ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge.mat',...
    'outs_openloop');
ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM7_TrailingEdge.mat',...
    'outs_openloop');
ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM6_TrailingEdge.mat',...
    'outs_openloop');
ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5=Simulink.SimulationData.
    DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM5_TrailingEdge.mat',...
    'outs_openloop');
%DDPWM
ds_ol_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM12_DDPWM_N8M4.mat',...
    'outs_openloop');
ds_ol_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM9_DDPWM_N5M4.mat',...
    'outs_openloop');
ds_cl_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM12_DDPWM_N8M4.mat',...
    'outs_closedloop');
ds_cl_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM9_DDPWM_N5M4.mat',...
    'outs_closedloop');
%DDPM-DPWM combination
ds_ol_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_ol_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_nbitDDPM8.mat',...
    'outs_openloop');
ds_cl_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_nbitDDPM8.mat',...
    'outs_closedloop');
ds_cl_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM7_nbitDDPM7.mat',...
    'outs_closedloop');
ds_cl_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM6_nbitDDPM6.mat',...
    'outs_closedloop');
ds_cl_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM5_nbitDDPM5.mat',...
    'outs_closedloop');
ds_cl_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw1M_tsim1.5
    _nbitADC16_nbitPWM8_nbitDDPM8.mat',...
    'outs_closedloop');
ds_cl_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw1M_tsim1.5
    _nbitADC16_nbitPWM7_nbitDDPM7.mat',...
    'outs_closedloop');
ds_cl_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw1M_tsim1.5
    _nbitADC16_nbitPWM6_nbitDDPM6.mat',...
    'outs_closedloop');
ds_cl_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw1M_tsim1.5
    _nbitADC16_nbitPWM5_nbitDDPM5.mat',...
    'outs_closedloop');
%OFinFP
ds_cl_OFinFP_fsw500k_nbitADC16_nbitPWM8=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_OFinFP_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge.mat',...
    'outs_closedloop');

load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat', 'TE_PWM5_fsw1M');
%it is necessary import only one t_int_dec because it is constant for all
%simulations
t_int_dec=TE_PWM5_fsw1M.t_int_dec;
t_sim=1.5; %simulation time used
t_win=1; %analysis window without transient for fft
```

```
t_jump=t_sim-t_win; %window time with transient

sample_start=uint64(t_jump/t_int_dec)+1; %position of first sample in t_win
sample_tot=uint64(t_sim/t_int_dec)+1; %number of total sample in vector

window=rectwin(sample_tot-sample_start); %define the window analysis
nfft=numel(window); %define the number of samples point in the window which the fft
     is done
window_ol=rectwin(sample_tot-1-sample_start); %define the window analysis
nfft_ol=numel(window_ol); %define the number of samples point in the window which
    the fft is done
%%
%output filter in feedback path
%closed-loop
%fsw=500 kHz
Vop_OFinFP_fsw500k_nbitADC16_nbitPWM8=readall...
    (ds_cl_OFinFP_fsw500k_nbitADC16_nbitPWM8{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_OFinFP_fsw500k_nbitADC16_nbitPWM8, freq_OFinFP_fsw500k_nbitADC16_nbitPWM8
    ]=...
    pwelch(Vop_OFinFP_fsw500k_nbitADC16_nbitPWM8-mean(
    Vop_OFinFP_fsw500k_nbitADC16_nbitPWM8),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_OFinFP_fsw500k_nbitADC16_nbitPWM8',...
    'freq_OFinFP_fsw500k_nbitADC16_nbitPWM8', '-v7.3', '-append')
clear PO_DIFF_OFinFP_fsw500k_nbitADC16_nbitPWM8
    freq_OFinFP_fsw500k_nbitADC16_nbitPWM8...
    Vop_OFinFP_fsw500k_nbitADC16_nbitPWM8
%%
%CLOSED LOOP
%fsw=500 kHz closed-loop
Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8
```

```
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5
%%
%fsw=1 MHz closed-loop
Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10

Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8

Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7

Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6

Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5
%%
%noise shaping
Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS{10}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS{10}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS', '-v7.3', '-append'
    )
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS
%%
%OPEN-LOOP
%fsw=1 MHz open-loop
Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
[POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8', '-v7.3', '-append')
clear POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8

Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
[POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7', '-v7.3', '-append')
clear POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7

Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
[POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6', '-v7.3', '-append')
clear POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6

Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
[POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5,
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5-mean(
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5',...
    'freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5', '-v7.3', '-append')
clear POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5
    freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5...
    Vop_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5
%%
%fsw=500k open-loop
Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8,...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
```

```
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6

Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5=readall...
    (ds_ol_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5{7}.Values)...
    .Data(sample_start+1:sample_tot-1);
[PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5,
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5]=...
    pwelch(Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5-mean(
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5',...
    'freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5', '-v7.3', '-append')
clear PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5
    freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5...
    Vop_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5
%%
%DDPWM
Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4=readall...
    (ds_ol_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4{8}.Values)...
    .Data(sample_start+1:sample_tot-1);
[POP_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4,
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4]=...
    pwelch(Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4-...
    mean(Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    POP_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4',...
    'freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4', '-v7.3', '-append')
clear POP_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4...
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4

Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4=readall...
```

```
    (ds_cl_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4,
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4]=...
    pwelch(Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4-mean(
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PODIFF_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4',...
    'freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4', '-v7.3', '-append')
clear PODIFF_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4...
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4

Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4=readall...
    (ds_cl_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4{8}.Values)...
    .Data(sample_start+1:sample_tot);
[POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4,
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4]=...
    pwelch(Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4-mean(
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4',...
    'freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4', '-v7.3', '-append')
clear POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4...
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4

Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4=readall...
    (ds_ol_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4{8}.Values)...
    .Data(sample_start+1:sample_tot-1);
[POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4,
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4]=...
    pwelch(Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4-mean(
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4',...
    'freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4', '-v7.3', '-append')
clear POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4...
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4

Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2=readall...
    (MYSOL_ds_ol_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2{9}.Values)...
    .Data(sample_start+1:sample_tot-1);
[POP_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2,
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2]=...
    pwelch(Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2-mean(
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2),...
    window_ol, [], nfft_ol, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop_MYSOL.mat',...
    'POP_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2',...
    'freq_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2', '-v7.3', '-append')
clear POP_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2...
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2

```

```
Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2=readall...
    (MYSOL_ds_cl_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2{6}.Values)...
    .Data(sample_start+1:sample_tot);
[POP_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2,
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2]=...
    pwelch(Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2-mean(
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop_MYSOL.mat', '
    POP_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2',...
    'freq_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2', '-v7.3', '-append')
clear POP_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2
    freq_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2...
    Vop_DDPWM_fsw500k_nbitADC16_nbitPWM8_N6M2
%%
Vop_DC_fsw500k_nbitADC6_nbitPWM7=readall...
    (ds_cl_DC_nbitADC6_nbitPWM7{6}.Values)...
    .Data(sample_start+1:sample_tot);
[POP_DC_fsw500k_nbitADC6_nbitPWM7, freq_DC_fsw500k_nbitADC6_nbitPWM7]=...
    pwelch(Vop_DC_fsw500k_nbitADC6_nbitPWM7-mean(Vop_DC_fsw500k_nbitADC6_nbitPWM7)
    ,...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', 'POP_DC_fsw500k_nbitADC6_nbitPWM7'
    ,...
    'freq_DC_fsw500k_nbitADC6_nbitPWM7', '-v7.3', '-append')
clear POP_DC_fsw500k_nbitADC6_nbitPWM7 freq_DC_fsw500k_nbitADC6_nbitPWM7...
    Vop_DC_fsw500k_nbitADC6_nbitPWM7

Vop_DC_fsw500k_nbitADC7_nbitPWM6=readall...
    (ds_cl_DC_nbitADC7_nbitPWM6{6}.Values)...
    .Data(sample_start+1:sample_tot);
[POP_DC_fsw500k_nbitADC7_nbitPWM6, freq_DC_fsw500k_nbitADC7_nbitPWM6]=...
    pwelch(Vop_DC_fsw500k_nbitADC7_nbitPWM6-mean(Vop_DC_fsw500k_nbitADC7_nbitPWM6)
    ,...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', 'POP_DC_fsw500k_nbitADC7_nbitPWM6'
    ,...
    'freq_DC_fsw500k_nbitADC7_nbitPWM6', '-v7.3', '-append')
clear POP_DC_fsw500k_nbitADC7_nbitPWM6 freq_DC_fsw500k_nbitADC7_nbitPWM6...
    Vop_DC_fsw500k_nbitADC7_nbitPWM6
%%
%DC DDPWM
Vop_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2=readall...
    (ds_cl_DC_DDPWM_nbitADC6_nbitPWM7_N5M2{6}.Values)...
    .Data(sample_start+1:sample_tot);
[POP_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2,
    freq_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2]=...
    pwelch(Vop_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2-mean(
    Vop_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    POP_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2',...
    'freq_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2', '-v7.3', '-append')
clear POP_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2
    freq_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2...
    Vop_DC_DDPWM_fsw500k_nbitADC6_nbitPWM7_N5M2

Vop_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2=readall...
    (ds_cl_DC_DDPWM_nbitADC7_nbitPWM6_N4M2{6}.Values)...
    .Data(sample_start+1:sample_tot);
[POP_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2,
    freq_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2]=...
    pwelch(Vop_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2-mean(
    Vop_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    POP_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2',...
    'freq_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2', '-v7.3', '-append')
clear POP_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2
    freq_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2...
    Vop_DC_DDPWM_fsw500k_nbitADC7_nbitPWM6_N4M2
%%
%DDPM-DPWM
%open-loop
Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8=readall...
    (ds_ol_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8{6}.Values)...
    .Data(sample_start+1:sample_tot-1);
[PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8,
    freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8]=...
    pwelch(Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8-mean(
    Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8),...
    window_ol, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_OpenLoop.mat', '
    PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8',...
    'freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8', '-v7.3', '-append')
clear PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8
    freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8...
    Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8
%%
Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8=readall...
    (ds_cl_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8,
    freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8]=...
    pwelch(Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8-mean(
    Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8',...
    'freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8', '-v7.3', '-append')
clear PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8
    freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8...
    Vop_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8

Vop_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7=readall...
    (ds_cl_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7{8}.Values)...
    .Data(sample_start+1:sample_tot-1);
[PODIFF_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7,
    freq_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7]=...
    pwelch(Vop_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7-mean(
    Vop_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7),...
    window_ol, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PODIFF_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7',...
    'freq_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7', '-v7.3', '-append')
clear PODIFF_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7
    freq_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7...
    Vop_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7
```

```

Vop_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6=readall...
    (ds_cl_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6,
    freq_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6]=...
    pwelch(Vop_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6-mean(
    Vop_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PODIFF_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6',...
    'freq_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6', '-v7.3', '-append')
clear PODIFF_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6
    freq_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6...
    Vop_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6

Vop_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5=readall...
    (ds_cl_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5,
    freq_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5]=...
    pwelch(Vop_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5-mean(
    Vop_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PODIFF_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5',...
    'freq_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5', '-v7.3', '-append')
clear PODIFF_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5
    freq_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5...
    Vop_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5

Vop_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8=readall...
    (ds_cl_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8, freq_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8
    ]=...
    pwelch(Vop_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8-mean(
    Vop_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PODIFF_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8',...
    'freq_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8', '-v7.3', '-append')
clear PODIFF_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8
    freq_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8...
    Vop_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8

Vop_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7=readall...
    (ds_cl_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7, freq_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7
    ]=...
    pwelch(Vop_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7-mean(
    Vop_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PODIFF_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7',...
    'freq_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7', '-v7.3', '-append')
clear PODIFF_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7
    freq_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7...
    Vop_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7

Vop_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6=readall...
    (ds_cl_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6, freq_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6]=...
    pwelch(Vop_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6-mean(Vop_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', 'PODIFF_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6',...
    'freq_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6', '-v7.3', '-append')
clear PODIFF_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6 freq_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6...
    Vop_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6

Vop_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5=readall...
    (ds_cl_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PODIFF_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5, freq_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5]=...
    pwelch(Vop_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5-mean(Vop_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', 'PODIFF_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5',...
    'freq_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5', '-v7.3', '-append')
clear PODIFF_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5 freq_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5...
    Vop_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5
%%
%%anti-alisaing filter output
outAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8=readall...
    (ds_cl_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8{3}.Values)...
    .Data(sample_start+1:sample_tot);
[PoutAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8, ~]=...
    pwelch(outAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8-mean(outAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', 'PoutAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8',...
     '-v7.3', '-append')
clear PoutAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8 ...
   outAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8

outAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5=readall...
    (ds_cl_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5{3}.Values)...
    .Data(sample_start+1:sample_tot);
[PoutAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5, ~]=...
    pwelch(outAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5-mean(outAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', 'PoutAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5',...
     '-v7.3', '-append')
clear PoutAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5 ...
   outAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5

```

```
outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8{3}.Values)...
    .Data(sample_start+1:sample_tot);
[PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8, ~]=...
    pwelch(outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8-mean(
    outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8',...
     '-v7.3', '-append')
clear PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8...
    outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8

outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5{3}.Values)...
    .Data(sample_start+1:sample_tot);
[PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5, ~]=...
    pwelch(outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5-mean(
    outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5',...
     '-v7.3', '-append')
clear PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5...
    outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5

outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10=readall...
    (ds_cl_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10{3}.Values)...
    .Data(sample_start+1:sample_tot);
[PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10, ~]=...
    pwelch(outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10-mean(
    outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10',...
     '-v7.3', '-append')
clear PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10...
    outAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10
%%
%Low frequency flck_SYS ~ 4MHz
ds_cl_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw1M_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge_fclkSYS_LOW.mat',...
    'outs_closedloop');
ds_cl_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_AAfilterInFeedbackPath_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_TrailingEdge_fclkSYS_LOW.mat',...
    'outs_closedloop');
ds_cl_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW=Simulink.SimulationData.DatasetRef...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw1M_tsim1.5
    _nbitADC16_nbitPWM8_nbitDDPM8_fclkSYS_LOW.mat',...
    'outs_closedloop');
ds_cl_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW=Simulink.SimulationData.DatasetRef
    ...
    ('DatastoreFiles/simOut_cl_1kHzsinewave_fsw500k_tsim1.5
    _nbitADC16_nbitPWM8_nbitDDPM8_fclkSYS_LOW.mat',...
    'outs_closedloop');
load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge_fclkSYS_LOW.mat', '
    TE_PWM8_fsw1M');
```

```
%it is necessary import only one t_int_dec because it is constant for all
%simulations
t_int_dec=TE_PWM8_fsw1M.t_int_dec;
t_sim=1.5; %simulation time used
t_win=1; %analysis window without transient for fft
t_jump=t_sim-t_win; %window time with transient

sample_start=uint64(t_jump/t_int_dec)+1; %position of first sample in t_win
sample_tot=uint64(t_sim/t_int_dec)+1; %number of total sample in vector

window=rectwin(sample_tot-sample_start); %define the window analysis
nfft=numel(window); %define the number of samples point in the window which the fft
     is done
%%
%power spectra
Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW=readall...
    (ds_cl_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW,
    freq_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW]=...
    pwelch(Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW-mean(
    Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW',...
    'freq_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW', '-v7.3', '-append')
clear PO_DIFF_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW
    freq_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW...
    Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW

Vop_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW=readall...
    (ds_cl_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW{9}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW,
    freq_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW]=...
    pwelch(Vop_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW-mean(
    Vop_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW',...
    'freq_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW', '-v7.3', '-append')
clear PO_DIFF_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW
    freq_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW...
    Vop_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW

Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW=readall...
    (ds_cl_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW,
    freq_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW]=...
    pwelch(Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW-mean(
    Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW',...
    'freq_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW', '-v7.3', '-append')
clear PO_DIFF_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW
    freq_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW...
    Vop_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW
```

```

Vop_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW=readall...
    (ds_cl_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW{8}.Values)...
    .Data(sample_start+1:sample_tot);
[PO_DIFF_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW,
    freq_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW]=...
    pwelch(Vop_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW-mean(
    Vop_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW),...
    window, [], nfft, 1/t_int_dec, 'onesided', 'power');
save('SpectraData/PowerSpectra_ClosedLoop.mat', '
    PO_DIFF_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW',...
    'freq_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW', '-v7.3', '-append')
clear PO_DIFF_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW
    freq_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW...
    Vop_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW
```

5.2.11 Code to compute SNR from simulations data

```
clc
clear
close all
%%
load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat', 'TE_PWM5_fsw1M');
%it is necessary import only one t_int_dec because it is constant for all
%simulations
t_int_dec=TE_PWM5_fsw1M.t_int_dec;
t_sim=1.5; %simulation time used
t_win=1; %analysis window without transient for fft
t_jump=t_sim-t_win; %window time with transient

sample_start=uint64(t_jump/t_int_dec)+1; %position of first sample in t_win
sample_tot=uint64(t_sim/t_int_dec)+1; %number of total sample in vector
window=rectwin(sample_tot-sample_start); %define the window analysis
nfft=numel(window); %define the number of samples point in the window which the fft
     is done
window_ol=rectwin(sample_tot-1-sample_start); %define the window analysis
nfft_ol=numel(window_ol); %define the number of samples point in the window which
    the fft is done

%file and window
file_PS_cl=matfile('SpectraData/PowerSpectra_ClosedLoop.mat');
file_PS_ol=matfile('SpectraData/PowerSpectra_OpenLoop.mat');
rbw=round(enbw(window));
rbw_ol=enbw(window_ol);
%set deafult parameters of the graphs
set(0, 'defaultAxesFontSize', 20);
set(0, 'defaultLegendFontSize', 20);
%%
%SNR computation
%output filter in FP
figure('Name', 'SNR system with output filter in feedback path (nbitPWM=8, f_{sw} \
    sim 500 kHz)')
snr(file_PS_cl.PO_DIFF_OFinFP_fsw500k_nbitADC16_nbitPWM8(1:20005, 1),...
    file_PS_cl.freq_OFinFP_fsw500k_nbitADC16_nbitPWM8(1:20005, 1), rbw, 20, 'power'
    )
title({'Power spectrum of differential output of system with output filter in
    feedback path';...
    '(nbitPWM=8, f_{sw} \sim 500 kHz)'})
%closed loop
%fsw=500k
%differential spectra
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=10,
    f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10(1:20005, 1),
     rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=10, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=8,
    f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=8, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=7,
    f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=7, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=6,
    f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=6, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=5,
    f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=5, f_{sw} \sim 500 kHz)'})
%%
%fsw=1M closed-loop
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=10,
    f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10(1:20005, 1)
    ,...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=10, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=8,
    f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1)
    ,...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=8, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=7,
    f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1)
    ,...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=7, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=6,
    f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1)
    ,...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=6, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path (nbitPWM=5,
    f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1)
    ,...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1),
    rbw, 20, 'power')
title({'Differential output power spectrum of system with anti-aliasing filter in
    feedback path';...
    '(nbitPWM=5, f_{sw} \sim 1 MHz)'})
%%
%noise shaping
figure('Name', 'SNR system with anti-aliasing filter in feedback path, noise
    shaping (nbitPWM=8, f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS(1:20005, 1),
     rbw, 20, 'power')
title({'Differential output power spectrum of system with noise shaping';...
    '(nbitPWM=8, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system with anti-aliasing filter in feedback path, noise
    shaping (nbitPWM=8, f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS
    (1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS(1:20005,
    1), rbw, 20, 'power')
title({'Differential output power spectrum of system with noise shaping';...
    '(nbitPWM=8, f_{sw} \sim 500 kHz)'})
%%
%open-loop
%fsw=500k
%differential
figure('Name', 'SNR system open-loop (nbitPWM=8, f_{sw} \sim 500 kHz)')
snr(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=8, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system open-loop (nbitPWM=7, f_{sw} \sim 500 kHz)')
snr(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=7, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system open-loop (nbitPWM=6, f_{sw} \sim 500 kHz)')
snr(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=6, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system open-loop (nbitPWM=5, f_{sw} \sim 500 kHz)')
snr(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=5, f_{sw} \sim 500 kHz)'})
%%
%fsw=1M
%differential
figure('Name', 'SNR system open-loop (nbitPWM=8, f_{sw} \sim 1 MHz)')
snr(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=8, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system open-loop (nbitPWM=7, f_{sw} \sim 1 MHz)')
snr(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=7, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system open-loop (nbitPWM=6, f_{sw} \sim 1 MHz)')
snr(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=6, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system open-loop (nbitPWM=5, f_{sw} \sim 1 MHz)')
snr(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1),
    rbw_ol+.1, 20, 'power')
title({'Differential output power spectrum of open-loop system';...
    '(nbitPWM=5, f_{sw} \sim 1 MHz)'})
%%
%DDPWM
figure('Name', 'SNR system open-loop DDPWM (nbitPWM=12, f_{sw} \sim 500 kHz, N=8, M
    =4)')
snr(file_PS_ol.POP_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1),...
    file_PS_ol.freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1), rbw_ol+.1,
    20, 'power')
ylim([-200 18])
title({'Differential output power spectrum of open-loop system with DDPWM modulator
    ';...
    '(N=8, M=4, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system closed-loop DDPWM (nbitPWM=12, f_{sw} \sim 500 kHz, N=8,
     M=4)')
snr(file_PS_cl.PODIFF_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1),...
    file_PS_cl.freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1), rbw, 20, '
    power')
ylim([-200 18])
title({'Differential output power spectrum of closed-loop system with DDPWM
    modulator';...
    '(N=8, M=4, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system open-loop DDPWM (nbitPWM=9, f_{sw} \sim 500 kHz, N=5, M
    =4)')
```

```
snr(file_PS_ol.POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1),...
    file_PS_ol.freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1), rbw_ol+.1,
    20, 'power')
ylim([-200 18])
title({'Differential output power spectrum of open-loop system with DDPWM modulator
    ';...
    '(N=5, M=4, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system closed-loop DDPWM (nbitPWM=9, f_{sw} \sim 500 kHz, N=5,
    M=4)')
snr(file_PS_cl.POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1),...
    file_PS_cl.freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1), rbw, 20, '
    power')
ylim([-200 18])
title({'Differential output power spectrum of closed-loop system with DDPWM
    modulator';...
    '(N=5, M=4, f_{sw} \sim 500 kHz)'})
%%
%%DDPM-DPWM combination
%closed-loop
%fsw=500k
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =8, nbitDDPM=8, f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1), rbw, 20, '
    power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM
    combination';...
    '(nbitDDPM=8, nbitDPWM=8, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =7, nbitDDPM=7, f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1), rbw, 20, '
    power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM
    combination';...
    '(nbitDDPM=7, nbitDPWM=7, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =6, nbitDDPM=6, f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1), rbw, 20, '
    power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM
    combination';...
    '(nbitDDPM=6, nbitDPWM=6, f_{sw} \sim 500 kHz)'})
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =5, nbitDDPM=5, f_{sw} \sim 500 kHz)')
snr(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1), rbw, 20, '
    power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM
    combination';...
    '(nbitDDPM=5, nbitDPWM=5, f_{sw} \sim 500 kHz)'})
%fsw=1MHz
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =8, nbitDDPM=8, f_{sw} \sim 1 MHz)')
```

```matlab
snr(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1), rbw, 20, 'power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM combination';...
    '(nbitDDPM=8, nbitDPWM=8, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM=7, nbitDDPM=7, f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1), rbw, 20, 'power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM combination';...
    '(nbitDDPM=7, nbitDPWM=7, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM=6, nbitDDPM=6, f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1), rbw, 20, 'power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM combination';...
    '(nbitDDPM=6, nbitDPWM=6, f_{sw} \sim 1 MHz)'})
figure('Name', 'SNR system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM=5, nbitDDPM=5, f_{sw} \sim 1 MHz)')
snr(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1), rbw, 20, 'power')
ylim([-200 16.1])
title({'Differential output power spectrum of closed-loop system with DDPM-DPWM combination';...
    '(nbitDDPM=5, nbitDPWM=5, f_{sw} \sim 1 MHz)'})
%%
%Anti-aliasing spectra
figure('Name', 'Spectum anti-aliasing filter output DDPM-DPWM combination (nbitMOD=8)')
semilogx(file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8(1:10e6, 1),...
    pow2db(file_PS_cl.PoutAA_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8(1:10e6, 1)))
xlabel('Frequency [Hz]')
ylabel('Power [dB]')
title({'Spectum anti-aliasing filter output DDPM-DPWM combination',...
    'nbitDPWM=8, nbitDDPM=8, nbitADC=16, f_{sw} \sim 1 MHz'})

figure('Name', 'Spectum anti-aliasing filter output DDPM-DPWM combination (nbitMOD=5)')
semilogx(file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5(1:10e6, 1),...
    pow2db(file_PS_cl.PoutAA_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5(1:10e6, 1)))
xlabel('Frequency [Hz]')
ylabel('Power [dB]')
title({'Spectum anti-aliasing filter output DDPM-DPWM combination',...
    'nbitDPWM=5, nbitDDPM=5, nbitADC=16, f_{sw} \sim 1 MHz'})

figure('Name', 'Spectum anti-aliasing filter output DPWM (nbitPWM=8)')
semilogx(file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:10e6, 1),...
    pow2db(file_PS_cl.PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:10e6, 1)))
```

```
xlabel('Frequency [Hz]')
ylabel('Power [dB]')
title({'Spectum anti-aliasing filter output',...
    'nbitDPWM=8, nbitADC=16, f_{sw} \sim 1 MHz'})

figure('Name', 'Spectum anti-aliasing filter output DPWM (nbitPWM=5)')
semilogx(file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:10e6, 1)
    ,...
    pow2db(file_PS_cl.PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:10e6
    , 1)))
xlabel('Frequency [Hz]')
ylabel('Power [dB]')
title({'Spectum anti-aliasing filter output',...
    'nbitDPWM=5, nbitADC=16, f_{sw} \sim 1 MHz'})

figure('Name', 'Spectum anti-aliasing filter output DPWM (nbitPWM=10)')
semilogx(file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10(1:10e6,
    1),...
    pow2db(file_PS_cl.PoutAA_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10(1:10
    e6, 1)))
xlabel('Frequency [Hz]')
ylabel('Power [dB]')
title({'Spectum anti-aliasing filter output',...
    'nbitDPWM=10, nbitADC=16, f_{sw} \sim 1 MHz'})
%%
%Low frequency flck_SYS ~ 4MHz
load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge_fclkSYS_LOW.mat', '
    TE_PWM8_fsw1M');
%it is necessary import only one t_int_dec because it is constant for all
%simulations
t_int_dec=TE_PWM8_fsw1M.t_int_dec;
t_sim=1.5; %simulation time used
t_win=1; %analysis window without transient for fft
t_jump=t_sim-t_win; %window time with transient

sample_start=uint64(t_jump/t_int_dec)+1; %position of first sample in t_win
sample_tot=uint64(t_sim/t_int_dec)+1; %number of total sample in vector

window=rectwin(sample_tot-sample_start); %define the window analysis
rbw=round(enbw(window));

figure('Name', 'SNR system with low system frequency (nbitPWM=8, f_{sw} \sim 500
    kHz)')
snr(file_PS_cl.PO_DIFF_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1), rbw, 20, '
    power')

figure('Name', 'SNR system with low system frequency (nbitPWM=8, f_{sw} \sim 1 MHz)
    ')
snr(file_PS_cl.PO_DIFF_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1), rbw, 20, '
    power')

figure('Name', 'SNR system with low system frequency (nbitPWM=8, f_{sw} \sim 1 MHz)
    ')
snr(file_PS_cl.PO_DIFF_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1), rbw, 20, '
    power')

```

```
figure('Name', 'SNR system with low system frequency (nbitPWM=8, f_{sw} \sim 500
    kHz)')
snr(file_PS_cl.PO_DIFF_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1), rbw, 20,
    'power')
```

5.2.12 Code to compute THD from simulations data

```
clc
clear
close all
%%
load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge.mat', 'TE_PWM5_fsw1M');
%it is necessary import only one t_int_dec because it is constant for all
%simulations
t_int_dec=TE_PWM5_fsw1M.t_int_dec;
t_sim=1.5; %simulation time used
t_win=1; %analysis window without transient for fft
t_jump=t_sim-t_win; %window time with transient

sample_start=uint64(t_jump/t_int_dec)+1; %position of first sample in t_win
sample_tot=uint64(t_sim/t_int_dec)+1; %number of total sample in vector
window=rectwin(sample_tot-sample_start); %define the window analysis
nfft=numel(window); %define the number of samples point in the window which the fft
     is done
window_ol=rectwin(sample_tot-1-sample_start); %define the window analysis
nfft_ol=numel(window_ol); %define the number of samples point in the window which
    the fft is done

%file and window
file_PS_cl=matfile('SpectraData/PowerSpectra_ClosedLoop.mat');
file_PS_ol=matfile('SpectraData/PowerSpectra_OpenLoop.mat');
rbw=round(enbw(window));
rbw_ol=enbw(window_ol);
%%
%THD computation
%output filter in FP
figure('Name', 'THD system with output filter in feedback path (nbitPWM=8, f_{sw} \
    sim 500 kHz)')
thd(file_PS_cl.PO_DIFF_OFinFP_fsw500k_nbitADC16_nbitPWM8(1:20005, 1),...
    file_PS_cl.freq_OFinFP_fsw500k_nbitADC16_nbitPWM8(1:20005, 1), rbw, 20, 'power'
    )
%CLOSED-LOOP
%fsw=500k closed-loop
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=10,
    f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM10(1:20005, 1),
     rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=8,
    f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005, 1),
    rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=7,
    f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005,
    1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005, 1),
    rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=6,
    f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005,
1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005, 1), rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=5, f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005, 1), rbw, 20, 'power')
%%
%fsw=1MHz closed-loop
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=10, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM10(1:20005, 1), rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=8, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1), rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=7, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1), rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=6, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1), rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path (nbitPWM=5, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1), rbw, 20, 'power')
%%
%noise shaping
figure('Name', 'THD system with anti-aliasing filter in feedback path, noise shaping (nbitPWM=8, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8_NS(1:20005, 1),  rbw, 20, 'power')
figure('Name', 'THD system with anti-aliasing filter in feedback path, noise shaping (nbitPWM=8, f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS(1:20005, 1),...
    file_PS_cl.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8_NS(1:20005, 1), rbw, 20, 'power')
%%
%OPEN-LOOP
%fsw=500k open-loop
figure('Name', 'THD system open-loop (nbitPWM=8, f_{sw} \sim 500 kHz)')
```

```
thd(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM8(1:20005, 1),
    rbw_ol+.1, 20, 'power')
figure('Name', 'THD system open-loop (nbitPWM=7, f_{sw} \sim 500 kHz)')
thd(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM7(1:20005, 1),
    rbw_ol+.1, 20, 'power')
figure('Name', 'THD system open-loop (nbitPWM=6, f_{sw} \sim 500 kHz)')
thd(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM6(1:20005, 1),
    rbw_ol+.1, 20, 'power')
figure('Name', 'THD system open-loop (nbitPWM=5, f_{sw} \sim 500 kHz)')
thd(file_PS_ol.PO_DIFF_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005,
    1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw500k_nbitADC16_nbitPWM5(1:20005, 1),
    rbw_ol+.1, 20, 'power')
%%
%fsw=1M open-loop
figure('Name', 'THD system open-loop (nbitPWM=8, f_{sw} \sim 1 MHz)')
thd(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM8(1:20005, 1),
    rbw_ol+.1, 20, 'power')
figure('Name', 'THD system open-loop (nbitPWM=7, f_{sw} \sim 1 MHz)')
thd(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM7(1:20005, 1),
    rbw_ol+.1, 20, 'power')
figure('Name', 'THD system open-loop (nbitPWM=6, f_{sw} \sim 1 MHz)')
thd(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM6(1:20005, 1),
    rbw_ol+.1, 20, 'power')
figure('Name', 'THD system open-loop (nbitPWM=5, f_{sw} \sim 1 MHz)')
thd(file_PS_ol.POP_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1),...
    file_PS_ol.freq_AAfilterInFeedbackPath_fsw1M_nbitADC16_nbitPWM5(1:20005, 1),
    rbw_ol+.1, 20, 'power')
%%
%DDPWM
figure('Name', 'THD system open-loop DDPWM (nbitPWM=12, f_{sw} \sim 500 kHz, N=8, M
    =4)')
thd(file_PS_ol.POP_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1),...
    file_PS_ol.freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1), rbw_ol+.1,
    20, 'power')
figure('Name', 'THD system closed-loop DDPWM (nbitPWM=12, f_{sw} \sim 500 kHz, N=8,
     M=4)')
thd(file_PS_cl.PODIFF_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1),...
    file_PS_cl.freq_DDPWM_fsw500k_nbitADC16_nbitPWM12_N8M4(1:20005, 1), rbw, 20, '
    power')
figure('Name', 'THD system open-loop DDPWM (nbitPWM=9, f_{sw} \sim 500 kHz, N=5, M
    =4)')
thd(file_PS_ol.POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1),...
    file_PS_ol.freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1), rbw_ol+.1,
    20, 'power')
figure('Name', 'THD system closed-loop DDPWM (nbitPWM=9, f_{sw} \sim 500 kHz, N=5,
    M=4)')
thd(file_PS_cl.POP_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1),...
    file_PS_cl.freq_DDPWM_fsw500k_nbitADC16_nbitPWM9_N5M4(1:20005, 1), rbw, 20, '
    power')
```

```
%%
%DDPM-DPWM combination
%fsw=500k
figure('Name', 'THD system open-loop DDPM-DPWM combination (nbitADC=16, nbitPWM=8,
    nbitDDPM=8, f_{sw} \sim 500 kHz)')
thd(file_PS_ol.PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1),...
    file_PS_ol.freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1), rbw_ol+.1,
    20, 'power')
figure('Name', 'THD system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =8, nbitDDPM=8, f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1), rbw, 20, '
    power')
figure('Name', 'THD system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =7, nbitDDPM=7, f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1), rbw, 20, '
    power')
figure('Name', 'THD system open-loop DDPM-DPWM combination (nbitADC=16, nbitPWM=6,
    nbitDDPM=6, f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1), rbw_ol+.1,
    20, 'power')
figure('Name', 'THD system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =5, nbitDDPM=5, f_{sw} \sim 500 kHz)')
thd(file_PS_cl.PODIFF_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1), rbw, 20, '
    power')
%fsw=1MHz
figure('Name', 'THD system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =8, nbitDDPM=8, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM8_nbitDDPM8(1:20005, 1), rbw, 20, 'power
    ')
figure('Name', 'THD system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =7, nbitDDPM=7, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM7_nbitDDPM7(1:20005, 1), rbw, 20, 'power
    ')
figure('Name', 'THD system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =6, nbitDDPM=6, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM6_nbitDDPM6(1:20005, 1), rbw, 20, 'power
    ')
figure('Name', 'THD system closed-loop DDPM-DPWM combination (nbitADC=16, nbitPWM
    =5, nbitDDPM=5, f_{sw} \sim 1 MHz)')
thd(file_PS_cl.PODIFF_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM5_nbitDDPM5(1:20005, 1), rbw, 20, 'power
    ')
%%
%Low frequency flck_SYS ~ 4MHz
load('DatastoreFiles/SimParameters_nbitADC16_TrailingEdge_fclkSYS_LOW.mat', '
    TE_PWM8_fsw1M');
%it is necessary import only one t_int_dec because it is constant for all
%simulations
t_int_dec=TE_PWM8_fsw1M.t_int_dec;
t_sim=1.5; %simulation time used
t_win=1; %analysis window without transient for fft
t_jump=t_sim-t_win; %window time with transient
```

```

sample_start=uint64(t_jump/t_int_dec)+1; %position of first sample in t_win
sample_tot=uint64(t_sim/t_int_dec)+1; %number of total sample in vector

window=rectwin(sample_tot-sample_start); %define the window analysis
rbw=round(enbw(window));

figure('Name', 'THD system with low system frequency (nbitPWM=8, f_{sw} \sim 500
    kHz)')
thd(file_PS_cl.PO_DIFF_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1), rbw, 20, '
    power')

figure('Name', 'THD system with low system frequency (nbitPWM=8, f_{sw} \sim 1 MHz)
    ')
thd(file_PS_cl.PO_DIFF_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_nbitPWM8_fclkSYS_LOW(1:20005, 1), rbw, 20, '
    power')

figure('Name', 'THD system with low system frequency (nbitPWM=8, nbitDDPM=8 f_{sw}
    \sim 1 MHz)')
thd(file_PS_cl.PO_DIFF_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw1M_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1), rbw, 20, '
    power')

figure('Name', 'THD system with low system frequency (nbitPWM=8, f_{sw} \sim 500
    kHz)')
thd(file_PS_cl.PO_DIFF_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1),...
    file_PS_cl.freq_fsw500k_nbitADC16_PWM8_DDPM8_fclkSYS_LOW(1:20005, 1), rbw, 20,
    'power')
```

Bibliography

[1] Wikipedia. Feb. 2021. URL: `https://it.wikipedia.org/wiki/Audio#`.

[2] Daniel Ruiz, Robson Moreno, and Tales Pimenta. “Design of a class D amplifier for hearing aid devices”. In: Sept. 2007, pp. 76–80. DOI: `10.1145/1284480.1284507`.

[3] N. E. Imane Bellili and K. Bekhouche. “Low Power Class D Audio Amplifier with High Performance and High Efficiency”. In: *2019 6th International Conference on Image and Signal Processing and their Applications (ISPA)*. 2019, pp. 1–4.

[4] M. A. Teplechuk, A. Gribben, and C. Amadi. “True Filterless Class-D Audio Amplifier”. In: *IEEE Journal of Solid-State Circuits* 46.12 (2011), pp. 2784–2793. DOI: `10.1109/JSSC.2011.2162913`.

[5] Edgar Sanchez-Sinencio Adrian I Colli-Menchi Miguel A Rojas-Gonzalez. *Design techniques for integrated CMOS class-D audio amplifiers*. Ed. by World Scientific. Vol. 16. Advanced series in Electrical and Computer Engineering. 2017.

[6] Y. Kang et al. “A review of audio Class D amplifiers”. In: *2016 International Symposium on Integrated Circuits (ISIC)*. 2016, pp. 1–4.

[7] T. Karaca and B. Deutschmann. “Electromagnetic evaluation of Class-D switching schemes”. In: *2015 11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*. 2015, pp. 113–116.

[8] M. Auer and T. Karaca. “Digitally assisted EMI-reduction techniques for Class-D amplifiers with digital control”. In: *2017 11th International Workshop on the Electromagnetic Compatibility of Integrated Circuits (EMCCompo)*. 2017, pp. 33–38.

[9] P. Muggler et al. “A filter free class D audio amplifier with 86% power efficiency”. In: *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*. Vol. 1. 2004, pp. I–1036.

[10] M. Kinyua, R. Wang, and E. Soenen. "Integrated 105 dB SNR, 0.0031% THD+N Class-D Audio Amplifier With Global Feedback and Digital Control in 55 nm CMOS". In: *IEEE Journal of Solid-State Circuits* 50.8 (2015), pp. 1764–1771.

[11] International rectifier. *Class D audio amplifier design.* Oct. 2003. URL: `http://www.irf.com/product-info/audio/classdtutorial.pdf`.

[12] R. Bakker and M. Duffy. "Maximising the efficiency of a Class-D audio amplifier output stage". In: *2017 28th Irish Signals and Systems Conference (ISSC).* 2017, pp. 1–5.

[13] EPC. *How to GaN Educational Series.* Mar. 2020. URL: `https://epc-co.com/epc/DesignSupport/TrainingVideos/HowToGaN`.

[14] Toit Mouton and Bruno Putzeys. "Digital Control of a PWM Switching Amplifier with Global Feedback". In: (Aug. 2009).

[15] Joshua Chung, R. McKenzie, and Wai Tung Ng. "A comparison between GaN and silicon based Class D audio power amplifiers with Pulse Density Modulation". In: *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT).* 2016, pp. 90–93. DOI: `10.1109/ICSICT.2016.7998847`.

[16] M. Kinyua, R. Wang, and E. Soenen. "Integrated 105 dB SNR, 0.0031% THD+N Class-D Audio Amplifier With Global Feedback and Digital Control in 55 nm CMOS". In: *IEEE Journal of Solid-State Circuits* 50.8 (2015), pp. 1764–1771. DOI: `10.1109/JSSC.2015.2420314`.

[17] S. Poulsen and M. A. E. Andersen. "Self oscillating PWM modulators, a topological comparision". In: *Conference Record of the Twenty-Sixth International Power Modulator Symposium, 2004 and 2004 High-Voltage Workshop.* 2004, pp. 403–407.

[18] Learn EMC. *EMC Regulations.* 2021. URL: `https://learnemc.com/emc-regulations-and-standards#:~:text=Countries\%20in\%20the\%20European\%20Union\%20\%28EU\%29\%20regulate\%20both,on\%20EMC\%20and\%20be\%20tested\%20and\%20labeled\%20accordingly.`.

[19] T. Kim et al. "Low Power Digital PWM Buck Converter With a Clock-Gating Shift-Register". In: *2019 International Conference on Electronics, Information, and Communication (ICEIC).* 2019, pp. 1–3.

[20] P. S. Crovetti. "All-Digital High Resolution D/A Conversion by Dyadic Digital Pulse Modulation". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 64.3 (2017), pp. 573–584.

[21] M. Usmonov et al. “Suppression of Quantization-Induced Limit Cycles in Digitally Controlled DC-DC Converters by Dyadic Digital Pulse Width Modulation”. In: *2019 IEEE Energy Conversion Congress and Exposition (ECCE)*. 2019, pp. 2224–2231.

[22] S. Jana and S. Srinivas. “Design of a digital PWM module with independent carrier amplitude modulation control”. In: *2019 2nd International Conference on Smart Grid and Renewable Energy (SGRE)*. 2019, pp. 1–6.

[23] P. S. Crovetti et al. “Limit-Cycle-Free Digitally Controlled DCDC Converters Based on Dyadic Digital PWM”. In: *IEEE Transactions on Power Electronics* 35.10 (2020), pp. 11155–11166.

[24] A. V. Peterchev and S. R. Sanders. “Quantization resolution and limit cycling in digitally controlled PWM converters”. In: *2001 IEEE 32nd Annual Power Electronics Specialists Conference (IEEE Cat. No.01CH37230)*. Vol. 2. 2001, 465–471 vol.2. DOI: `10.1109/PESC.2001.954158`.

[25] R. D. d’Aparo et al. “DC-AC power converter using sigma-delta modulation”. In: *2010 8th Workshop on Intelligent Solutions in Embedded Systems*. 2010, pp. 79–84.

[26] DigiKey. *Microcontroller*. Apr. 2021. URL: `https://www.digikey.it/products/it/integrated-circuits-ics/embedded-microcontrollers/685?k=microcontroller`.

[27] Sam Ben-Yaakov. *Diode Reverse Recovery*. June 2017. URL: `https://www.youtube.com/watch?v=DT8kCmXbSDg`.

[28] Sam Ben-Yaakov. *So what is GaN MOSFETs’ reverse conduction all about?* June 2020. URL: `https://www.youtube.com/watch?v=QNHzaiWCUeE&t=774s`.

[29] Giuseppe Greco. “AlGaNGaN heterostructures for enhancement mode transistors”. thesis. Università dgli studi di Catania, 2012.

[30] S. Piotrowicz et al. “Overview of AlGaN/GaN HEMT technology for L- to Ku-band applications”. In: *International Journal of Microwave and Wireless Technologies* 2 (Feb. 2010), pp. 105 –114. DOI: `10.1017/S1759078710000061`.

[31] JBL. *JBL GO2 Specifications*. 2021. URL: `https://www.jbl.com/on/demandware.static/-/Sites-masterCatalog_Harman/default/dw28f379bc/pdfs/JBL_GO2_Spec_Sheet_English.pdf`.

[32] Mathworks. *Relay*. 2021. URL: `https://www.mathworks.com/help/simulink/slref/relay.html?s_tid=doc_ta`.

[33] Shih-Hsiung Chien, Yi-Wen Chen, and Tai-Haur Kuo. "A Low Quiescent Current, Low THD+N Class-D Audio Amplifier With Area-Efficient PWM-Residual-Aliasing Reduction". In: *IEEE Journal of Solid-State Circuits* 53.12 (2018), pp. 3377–3385. DOI: 10.1109/JSSC.2018.2873613.

[34] Mathworks. *Continuous-Discrete Conversion Methods*. 2021. URL: https://www.mathworks.com/help/control/ug/continuous-discrete-conversion-methods.html.

[35] Brian Douglas. Oct. 2020. URL: https://www.youtube.com/watch?v=rL_1oWrOplk&t=1052s.

[36] Salim Ahmed. July 2020. URL: https://www.youtube.com/watch?v=vai5BggQiK8.

[37] Thomas A. Lipo D. Grahame Holmes. *Pulse Width Modulation for Power Converters-Principles and Practice*. Ed. by IEEE PRESS. 1st. 2003.

[38] Gabor C. Temes Shanthi Pavan Richard Schreier. *Undestanding sigma delta converter*. Ed. by IEEE Press editorial board. 2nd. 2017.

[39] Mathworks. *matlab.io.datastore.SimulationDatastore class*. 2021. URL: https://www.mathworks.com/help/simulink/slref/matlab.io.datastore.simulationdatastore-class.html.