High Performance Library for a Magnetic Position Sensor

Supervisor :
El Hage Ali Raed
rel@melexis.com

Student :
Audoin Nicolas
Nicolas.Audoin@grenoble-inp.org

Tutor :
Prejbeanu Liliana
Liliana.buda@cea.fr

Company :
Melexis Technologies SA
Chem. de Buchaux 38, Bevaix, Switzerland

Melexis
INSPIRED ENGINEERING

Master Thesis
February 13th - August 4 2023

Acknowledgments

I am profoundly grateful to the entire team at the Melexis Bevaix site for giving me the opportunity to undertake this internship and for their warm atmosphere.

Special thanks go to my supervisor Raed El Hage Ali, who has been a very good guide throughout this journey, granting me his invaluable advice and directions. I am also grateful to Pierre-Marie Signe and Sébastien Deuve for their readiness to assist whenever needed. The guidance and support from all three have been instrumental in my professional growth and learning during this internship.

I extend my gratitude to the rest of the MSDE team, including Bilel Belhadj, Nuno Ferreira, Gilles Curchod, and Dejan Novakovic, for their willingness to take time out of their busy schedules to answer my questions.

I would also like to extend my appreciation to Bertrand Coulon from the analog team, whose assistance was crucial in deepening my understanding of analog design.

Lastly, I would like to express my gratitude to my fellow interns - Arthur Nicola Jeremie Perque, and Cazenave Elodie. Their companionship made these six months an unforgettable experience.

Glossary

AAF : Anti Aliasing Filter

ADC : Analog Digital Converter

AFE : Analog Front End

AMS : Analog Mixed Signal

CM : Connect Module

EENets : Electrical Equivalent Nets

MPS : Magnetic Position Sensor

MSDE : Mixed Signal Design Engineer

MUX : Multiplexer

PHY : Physical layer

RNM : Real Number Modeling

SV : SystemVerilog

SUP : Supply

VAMS : Verilog AMS

Contents

1 Introduction

This internship has been carried out in **Melexis Technology SA**, a global supplier of microelectronic semiconductor, on the site of Bevaix (Switzerland). I joined for 6 months the Mixed Signal Design Engineer (MSDE) team, in the objective of developing a high performance library for a magnetic position sensor.

1.1 About Melexis

Melexis is a semiconductor company based in Belgium that was established in 1988 and specializes in the design and production of integrated circuits (ICs), sensors, and other semiconductor solutions. Their products are essentially used in the **automotive industry**, but they cater on other markets such as medical, or robotics industries. Many automotive applications, including advanced driver assistance systems (ADAS), electric power steering, engine management, and lighting control, use their solutions.

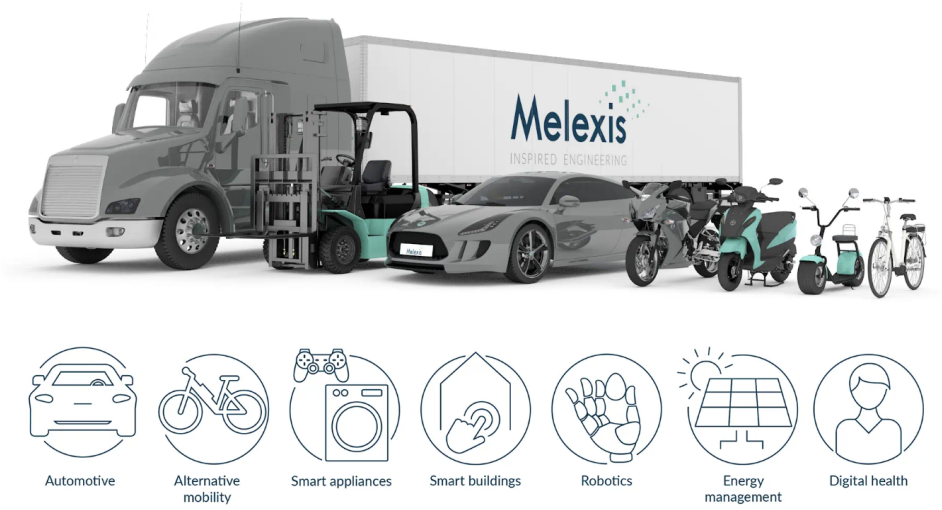


Figure 1: Melexis business sectors

Melexis is known for its expertise in the development of advanced **mixed-signal semiconductor technology**, which combines analog and digital signal processing. The company offers a wide range of semiconductor integrated circuits covering various sensor technologies, drivers and transceivers. Magnetic, temperature, pressure, and optical sensors, are just a few of the many products that Melexis has to offer. The company has operations across a **number of nations**, and its manufacturing facilities, sales offices, and R&D centers are dispersed throughout Europe, Asia, and the Americas.

1.2 About the internship

Melexis' **MPS** (Magnetic Position Sensor) works in a mixed signal-environment, which means that it includes both digital, and analog components. Digital blocks have discrete signals (0s/1s),

whereas analog uses continuous signals. Having a continuous spectrum of values induces very **high-complexity** simulations. It means that, using classical analog electronic circuit simulators, such as Spice, a single simulation of the **transistor-level models** can take up to days.

In order to achieve **high-speed** simulations for mixed signal verification, Melexis currently replaces the design sub-blocks transistor-level models (generated by netlister) with Verilog-AMS **behavioural models** that are developed by the MSDE. It means that they capture only the **key functional characteristics** of the blocks, instead of modelling every detail. Despite the huge advantage that Verilog-AMS models provide compared to transistor-level, they come with their own set of challenges:

- Electrical signals activate the analog kernel and slow down the simulations
- Limited **code coverage** (analog statements excluded)
- Can have a relatively bad accuracy/speed ratio compared to other methods

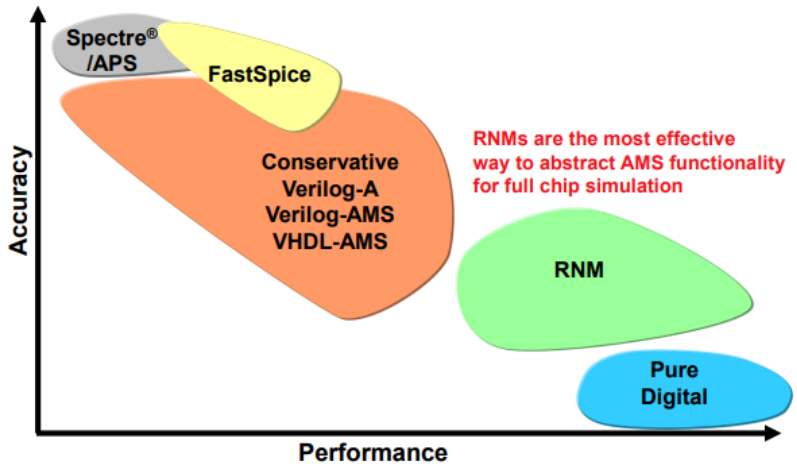


Figure 2: Accuracy Versus Performance of different methods

In fact, in recent years, several approaches have been presented to model analog behavior using digital description languages. By using a **digital solver** and **event-based** simulations, we can easily outperform the V-AMS. The objective of the internship is to try this approach using SystemVerilog as a digital description language. To be precise, we will focus on generating models using real-types signals, a technique known as Real Number Modeling.

To sum up, the main goal of the internship is to create **SystemVerilog Real Number models** based on existing design schematics, **Verilog-AMS models**, requirements, and design specifications for the **MPS MLX90396** (new version of the MPS in development). Finally, a comparative analysis of **key performance metrics** will be conducted between the previous models and the developed ones.

The **Gantt Chart** and the **Archive Card** of the master thesis are given in annex.

2 Theoretical background

The first part of the internship consisted in learning the essential theoretical background for this project. It went through doing two Cadence training on **SystemVerilog** and **Real Number Modeling**.

2.1 Verilog

Verilog is a **hardware description language** (HDL) used to design and model digital circuits, like integrated circuits and FPGAs. Introduced in 1984, it has since become an IEEE standard (IEEE 1364). Verilog allows designers to describe the **circuit behavior** and structure using code, which can be **simulated**, synthesized, and verified. It supports different levels of abstraction, including the gate-level and register-transfer level (RTL), and enables **hierarchical** and modular designs.

```
always
  begin // Always begins executing at time 0 and NEVER stops
    clk = 0; // Set clk to 0
    #1; // Wait for 1 time unit
    clk = 1; // Set clk to 1
    #1; // Wait 1 time unit
  end // Keeps executing - so continue back at the top of the begin
```

Figure 3: Example of a clock in Verilog

2.1.1 SystemVerilog

SystemVerilog is an extension of Verilog that add advanced features for system-level modeling, verification, and high-level synthesis. It was developed in response to the increasing complexity of digital designs and the need for more efficient designs verification. In 2005, SystemVerilog became an **IEEE standard**, known as IEEE 1800. The current version is IEEE standard 1800-2017.

Some SystemVerilog enhancements/features interesting in the context of my work :

SystemVerilog Assertions (SVA): SystemVerilog Assertions (SVA) is a powerful feature of SystemVerilog, it provides a formal way to define and specify the expected behavior and properties of a digital design. Using assertions allow designers to efficiently validate their designs and identify potential issues during simulation, formal verification, and emulation. SystemVerilog assertions are built from sequences and properties, an assertion fails if the property fails.

```
property req_gnt;
    @(posedge clk) req |=> gnt;
endproperty

assert_req_gnt: assert property (req_gnt) else $error("req not followed by gnt.");
```

Figure 4: Example of an assertion using property

Real Number Modeling : SystemVerilog supports **real** and **shortreal** data types for representing and manipulating floating-point numbers. This is useful when dealing with analog/mixed-signal systems. This last, very important in the context of the internship, will be developed in the next subsection.
Constrained random generation : Integer quantities, whether specified in a **class** definition or as independent variables within a particular lexical scope, can have random values assigned to them based on specific constraints. This functionality is useful for generating **randomized verification scenarios**.

```
class eth_frame;
    rand bit [47:0] dest;
    rand bit [47:0] src;
    rand bit [15:0] f_type;
    rand byte       payload[];
    bit [31:0]      fcs;
    rand bit [31:0] fcs_corrupt;

    constraint basic {
        payload.size inside {[46:1500]};
    }

    constraint good_fr {
        fcs_corrupt == 0;
    }
endclass
```

Figure 5: Example of a constrained random generation

Some other notable enhancement are :

- **Object-oriented programming (OOP)**: SystemVerilog introduces classes, inheritance, and polymorphism, enabling more modular and **reusable code** for complex designs and verification environments.
- **New data types and constructs**: SystemVerilog introduces new data types such as dynamic arrays, associative arrays, and queues, offering greater flexibility for data handling and manipulation.
- **Procedural blocks:** SystemVerilog introduces new procedural blocks to describe hardware: **always_comb** (for combinational logic), always_ff (for flip-flops), and always_latch (for latches).
- **Interface:** SystemVerilog introduces interfaces to encapsulate communication between design blocks, reducing the complexity of managing connections.

2.1.2 Verilog AMS

Another extension of Verilog is **Verilog-AMS**. It incorporates **analog** and **mixed-signal (AMS)** capabilities, by combining Verilog's digital design features with a continuous-time language for analog modeling. This enables designers to create and verify mixed-signal systems within a single environment. Key features include compact analog behavioral modeling, modular designs, and seamless integration with digital circuits. V-AMS is widely used for IC design and verification at Melexis.

```
Electrical res;
real r;

analog begin
    analog V(res) <+ r*I(res); //the statements is executed continuously over time
end
```

Figure 6: Example of an analog procedural block in VAMS

2.2 Real Number Modeling

2.2.1 What is (RNM) ?

Real Number Modeling **(RNM)** is the process of modeling an analog circuit's behavior as a **signal flow** model. This approach entails discretely sampling every output of an analog component based on its inputs and internal state. The model is **event-driven** and uses only the digital solver, inducing high-speed simulations. Note that there is also no convergence issue since no analog solver is used. Five different language standards support RNM: Verilog-AMS, **SystemVerilog**, VHDL, Verilog, and e. In the context of the internship, Real Number Modeling was done using SystemVerilog. This language is a popular choice since it presents **enhanced language features** compared to Verilog/VAMS/VHDL. One can, for instance, mention SystemVerilog's robust verification capabilities, including constrained random stimulus generation, functional coverage, and assertions. In this language, RNM starts by using the **real** data type. This data type represents real numbers as double-precision floating-point values.

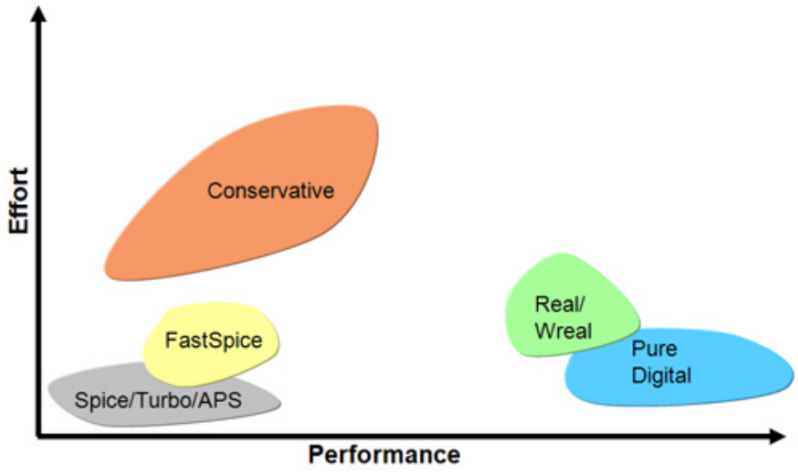


Figure 7: Model Effort spent Versus Performance

This method of modeling has the advantage of having a very good ratio of **effort** spent on modeling over model **performance**. An example of Real Number Modeling is shown in Figure 7. In the analog model, an analog solver is used to determine the values of all node voltages and branch currents, which can be **time-consuming**. In the Real Number model, pins are defined as real variables in

SystemVerilog, and the system recomputes the output only when **p** or **n** changes. In this case, the system uses a single equation, whereas the analog solver uses a **matrix** with coefficients defined at each analog time point, making it less time-consuming.

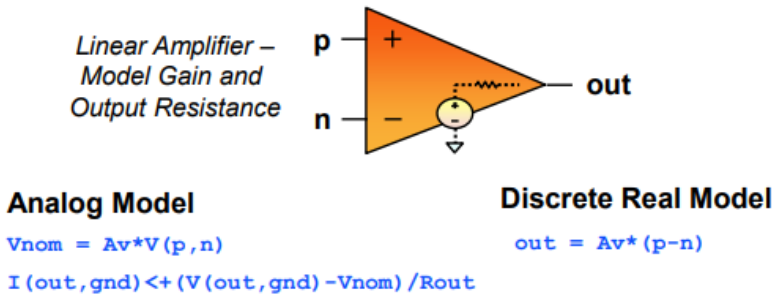


Figure 8: Analog vs Real Modeling

To give an idea of the achievable simulation speed, we can mention [1]. In this article, a SystemVerilog behavioral real number model for a 3-bit flash analog-to-digital converter (ADC) is presented in order to improve simulation **efficiency**. The proposed model is compared to three different models: a transistor-level flash ADC, a **Verilog-A** (V-Analog) ADC model, and a Verilog-AMS with **wreal** ADC model. The obtained simulation speeds are shown in Figure 9. We can see that the SV-RNM option is the fastest. In this case, it is only about a few seconds of gain; however, in our case, we expect to run more complex and lengthy simulations, on the order of days for the SPICE simulation. Therefore, we can expect each of those simulation times to grow **exponentially**, and having the simulation three times faster becomes interesting.

Three-bit Flash ADC models				
	SPICE	**SV-RNM**	**Verilog-A**	**V-AMS wreal**
Run Time	~10min	1.23s	13.178s	4.9s
% of time	100%	0.2%	2.2%	0.7%

Figure 9: Different modeling techniques for a Three-bit Flash ADC

2.3 Magnetic position sensor

2.3.1 Triaxis

Triaxis® is a patented magnetic sensor technology developed by Melexis. It is known for performing precise three-dimensional magnetic field measurements from a single sensor. This innovative technology, which is based on the Hall effect and the use of Integrated Magnetic Concentrators (IMCs), has been a significant success for Melexis, with the **billionth** Triaxis sensor being shipped in 2019. These sensors made a name for themselves across a range of applications. They are popular

in the automotive industry, where they play a key role in creating intricate and accurate control systems. However, they are also hard at work in agricultural machinery, joysticks, wheelchairs, and a **wide** host of other devices.

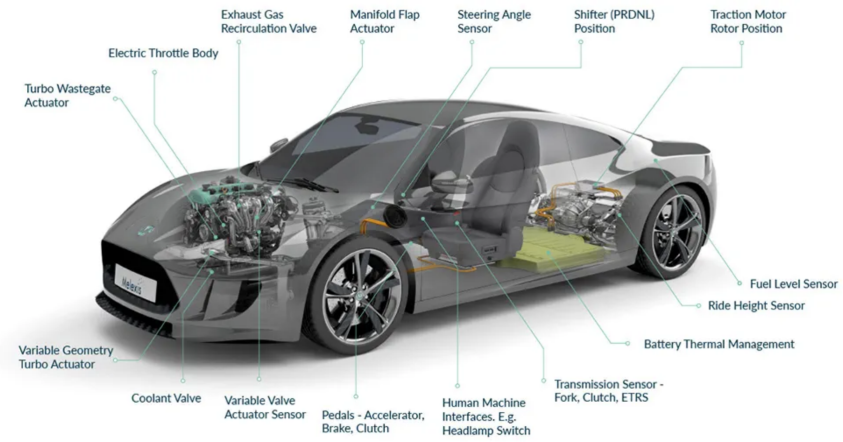


Figure 10: Triaxis sensor application

2.3.2 Working principle

Triaxis technology is based on the **Hall effect** principle, but with a unique twist that allows it to measure magnetic fields in three dimensions.

1. **Hall Effect Sensing:** The Hall effect is a phenomenon in which a voltage difference is created across an electrical conductor that is perpendicular to both the electric current in the conductor and the magnetic field perpendicular to the current. This voltage difference can be measured and is proportional to the strength of the magnetic field.

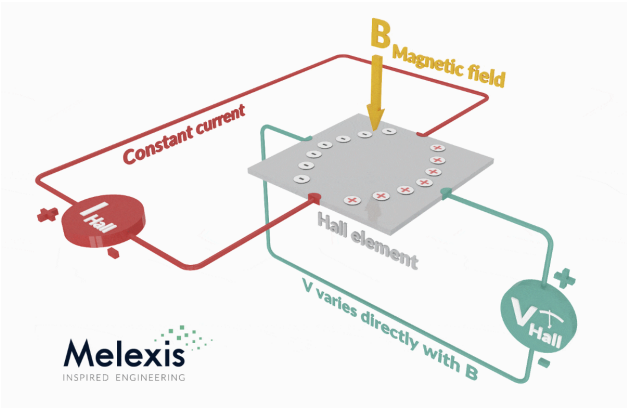


Figure 11: Hall effect

2. **Integrated Magnetic Concentrators (IMCs):** Traditional integrated Hall effect sensors are usually only sensitive to magnetic fields **perpendicular** to the surface of the Hall plate. An IMC is a **ferromagnetic** layer that can locally modify the direction of an externally applied parallel magnetic field. Consequently, we can convert the parallel flux density into an **orthogonal** one, which is suitable for the planar Hall plates. By placing Hall plates near the edge of an IMC, we can measure **both** the perpendicular and parallel components of an applied magnetic field.

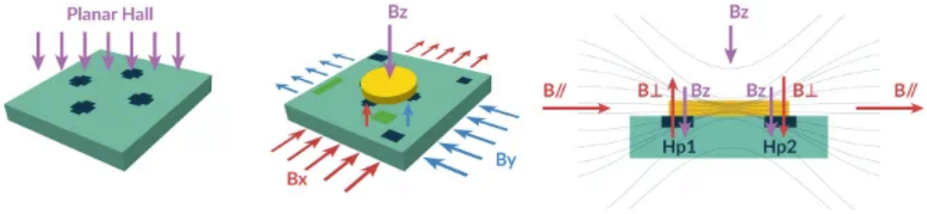


Figure 12: IMC concept

3. **Three-Dimensional Sensing:** In a Triaxis sensor, multiple Hall effect sensors are positioned at different orientations relative to the IMCs. This configuration enables the sensor to measure the strength of the magnetic field in **three dimensions**, namely the X, Y, and Z axes.

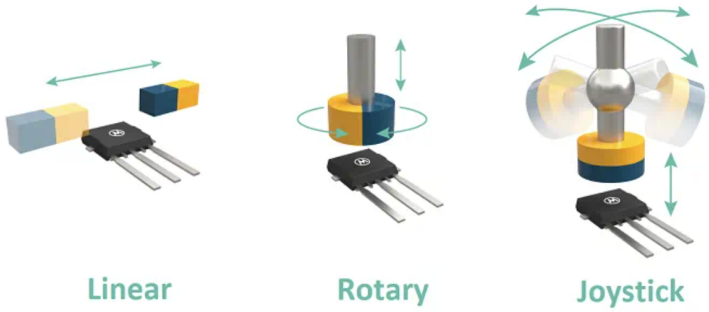


Figure 13: Triaxis 3Ds ensing

4. **Signal Processing:** The signals from the Hall effect sensors are then processed by an on-chip circuit to generate a digital output that accurately represents the **magnitude** and **direction** of the magnetic field in three dimensions.

2.3.3 MLX90396

The **MLX90395** is a tri-axis magnetic sensor capable of measuring 3D magnetic fields, as well as temperature and supply voltage. This feature enables the MLX90395 to accurately detect the absolute position of nearly any nearby magnet. Additionally, it offers **programmable parameters** that can be optimized for power, noise, and speed.
Being one of Melexis' latest products, the MLX90395 stands out by its ability, unlike other magnetometers, to sense magnetic fields generated by a permanent magnet as opposed to the Earth's magnetic field. This makes it suitable for sensing the position of a **static or moving** magnet in

close proximity to the sensor, by measuring the magnetic fields it produces.
On the other hand, the **MLX90396** is an extension of the MLX90395 currently under development. While I worked on this product during my internship, the specific details of the enhancements cannot be disclosed due to **confidentiality**. Nevertheless, it can be safely associated with the functionality of the MLX90395.

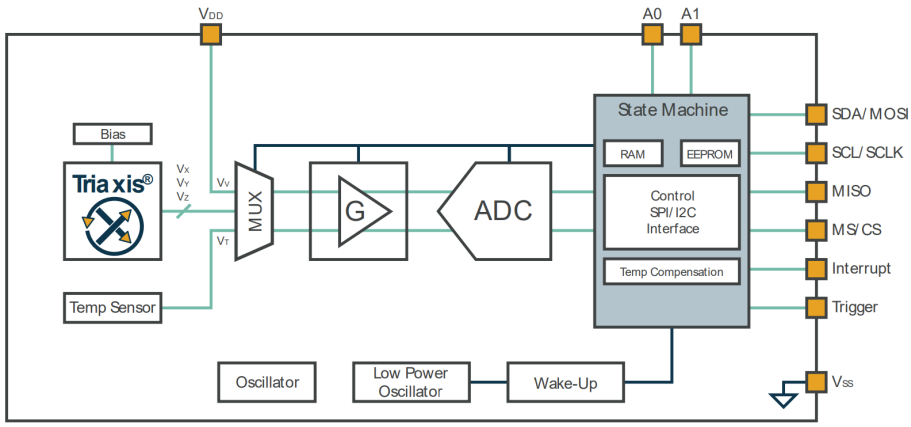


Figure 14: Simplified schematic of the MPS MLX90395

2.4 Review of V-AMS models

As mentioned earlier, some **Verilog-AMS behavioural models** have been developed by the MSDE. These models were, in fact, developed for the MLX90427, which is also a sensor from the **Triaxis product line**. This last one shares a lot of common blocks with the MLX90396, therefore, all those blocks already have working V-AMS models. This is interesting in the context of the internship as it will help us create **SV-RNM models**, plus we will be able to run simulations to compare the performances of V-AMS and SV-RNM models.
We went through some of these models to understand how they were modeled. Each of them is a hybrid electrical/wreal model. They use an electrical datatype on input supply/ground pins to allow current consumption modeling on each block. Alternatively, they use a wreal datatype on other input/output pins to simply reproduce the behavior of the block. Current consumption is modeled on some blocks using an analog block, the method for which can be seen in Figure 15.

```
analog begin
    //=====================================================================
    // Current consumption
    //=====================================================================
    idc_tmp = transition(ENA===1 ? 1.0 : 0.0) * idc_on + idc_off;
    I(VDDA,GNDA) <+ idc_tmp;
end
```

Figure 15: V-AMS current consumption modeling

This **analog** block essentially adds an ideal current source between VDDA and GNDA (supply/ground). The current of the source depends on whether the block is **enabled** or not. When disabled, the current source value is `idc_off`, and when enabled, we add `idc_on` to it.

2.5 SystemVerilog EEnet

EEnets (Electrical Equivalent nets) are a User-Defined Nettype (UDN) defined in Cadence's EEnet package, improving analog behavior modeling in SystemVerilog. They come with a User-Defined Type (UDT) **EEstruct** and a User-Defined Routine (UDR) `res_EE`. The structure `EEstruct` defines three fields: **V**, **I**, and **R**, which provide numerous options for driving the net. Based on the format of EEnet (see Figure 1), we can, for instance, imagine the configuration of Figure 16 (from [2]).

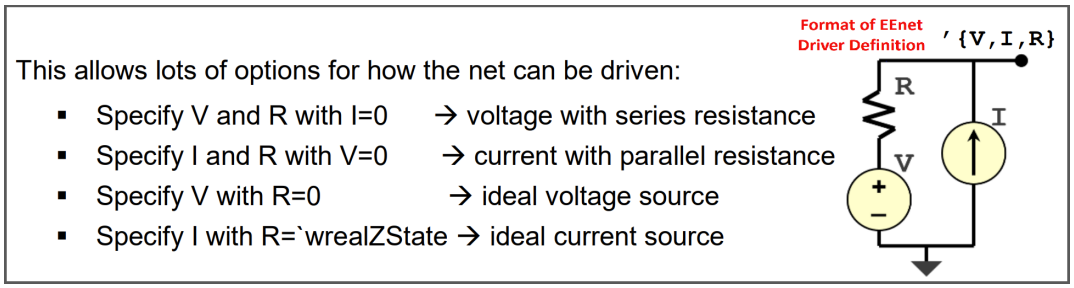


Figure 16: EEnet drivers configuration

When there are multiple drivers, the resolution function `res_EE` computes the **output voltage** using Norton's equivalent. It provides an output `EEnet` with `V` as the resolved node voltage, `I`=0 (except when driven by an **ideal V source**), and `R` as the effective impedance at the node. An example of the evaluation required for two `V+R` driver and one current source driver to an `EEnet` node is shown in Figure 17 (also from [2]).

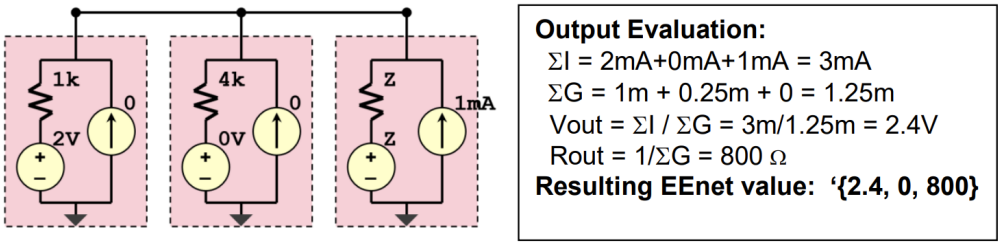


Figure 17: EEnet resolution function

EEnets is a potent tool to replicate **analog behavior** in a digital environment. Cadence provides many components using this package, including differential capacitors, inductors, current sources, and others. This implies that analog circuitry can be modeled effortlessly using the *EEnets* approach. Moreover, this can be done without any need for an **analog solver**, thereby maintaining high-speed simulations.

3 Create/Update RNM Models

The top level schematic of the MLX90396 is composed of 4 blocks, called **AFE**, **PHY**, **SUP** and **TEST**. With the help of the MSDE team, we were able to model most of the components of each block using **RNM** approach.

- **AFE (Analog Front End):** This block is responsible for the initial processing of the analog signal from the sensor. It typically includes components for signal conditioning, such as amplification and filtering. This block was **partially modeled** by me.
- **PHY (Physical Layer):** This block manages the communication between the sensor and the external environment. It converts the processed sensor data into a format that can be transmitted over a communication interface (like SPI, I2C, etc.). This block was **exclusively modeled** by other team members.
- **SUP (Supply):** This block oversees the power supply for the entire chip. It ensures that all other blocks receive the correct voltage and current levels for optimal operation. This block was also **partially modeled** by me.
- **TEST:** This block is utilized for testing and calibration of the sensor during manufacturing. It enables the assurance that the sensor is functioning correctly. This block was **fully modeled** by me.

3.1 AFE : Amplification chain

The amplifiers chain is a block of the **AFE** (Analog Front End). This last consists of two **amplifiers** (IA1 and IA2), an anti-aliasing **filter** (AAF), and a common mode voltage provider (VCM).

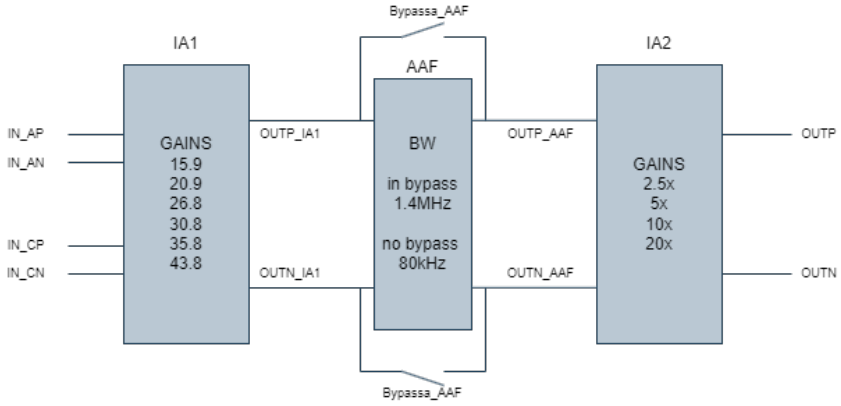


Figure 18: Amplification chain : Simplified schematic

3.1.1 Model : afe-vcm

The **VCM generator**, is the first model we decided to update. It is interesting to begin with, since it is a **very simple** but crucial block. This component, part of the amplification chain (which

is not shown), is responsible for generating and maintaining a stable common-mode voltage. The **common-mode voltage** is the average voltage of two input signals, balanced with respect to a common reference voltage. In our case, the two input signals are; the supply voltage **VDDA**, and the ground voltage **VSSA**. Therefore, the output of the VCM is $VCM = (VDDA - VSSA)/2$.

```
module m90427vAA_afe_vcm (  VCM,  EN,  VDDA, VSSA );

//================================================================================
// Interface Signals
//================================================================================
    input           VSSA;  real VSSA;
    input           VDDA;  real VDDA;
    inout           VCM;   wreal4state VCM;
    input           EN;    logic EN;

//================================================================================
// Internal Signal
//================================================================================
    real vcm_r;

//================================================================================
// Computing
//================================================================================
    always_comb begin
        vcm_r = (VDDA - VSSA) / 2.0;
    end

//================================================================================
// Output assignment
//================================================================================
    assign VCM = (EN === 1) ? vcm_r : vcm_r*2;

endmodule
```

Figure 19: VCM block : SV-RNM code

3.1.2 Model : ia1

The IA1 block is a **low-noise** amplifier positioned at the input of the amplifier chain. It offers an option to adjust its gain among **16 values**, ranging from 16 to 43.7. Additionally, the amplifier provides **offset** trimming, enabling it to compensate for signal path offsets.

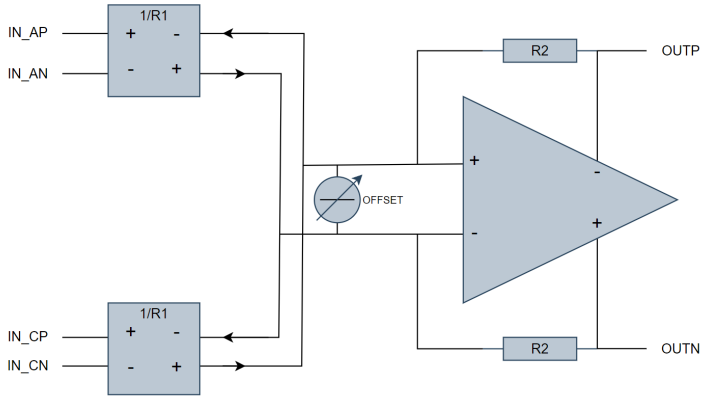


Figure 20: IA1 block : Schematic

The circuit is in fact here very simplified, it contains other inputs such as **GAINSELA** to set the gain, **OFFSETA** to set the offset, **ENAGM** to disable/enable a pair, and **ENA** to enable/disable the amplifier. To model this block, we follow the circuit : the first blocks will give the differential voltage from each input pair, then an offset is added, and lastly the signal is amplified depending on the gain value. The developped Real Number model is shown Figure 21, note that OUT_CLAMP is a function restricting output between **VDDA** (supply) and **VSSA** (ground).

```
//=========================================================================
// Gain selection
//=========================================================================
localparam real amp_gain_typ [0:15]= '{16, 16.9, 17.9, 19.4, 20.9, 21.9, 23.8, 25.8, 26.8, 28.8, 30.8, 33.7, 35.7, 38.7, 40.7, 43.7};
assign gain = amp_gain_typ[(GAINSELA)];//select the needed value for the gain, according to GAINSELA's input value

//=========================================================================
// Offset compensation and noise injection
//=========================================================================
assign VOFFSET =  (iLSB * rtrans) * (OFFSETA-64.0); // binary offset encoded, iLSB = 0.0625uA (see schematic), referred back at one input pair via 3 kOhm trans resistance

//=========================================================================
// IA1 -- transconductance amplifier output computation
//=========================================================================
always @( *) begin
    // differential voltage input - pair A
    VDM_IA1_INA = IN_AP - IN_AN;
    // differential voltage input - pair C
    VDM_IA1_INC = IN_CP - IN_CN;
    // if the pair is enabled, add the voltage
    VDM_IA1_IN = (VDM_IA1_INA * ENA_GM[0] + VDM_IA1_INC * ENA_GM[1] + VOFFSET) / 2;

    if ~(ENA) begin
        // 00 - circuit is disabled and the outputs are pulled down
        OUTP_real = 0;
        OUTN_real = 0;
    end
    else begin
        // 10 - circuit is enabled, normal operation
        OUTP_real = OUT_CLAMP(VCM + gain / 2 * VDM_IA1_IN, GNDA, VDDA);
        OUTN_real = OUT_CLAMP(VCM - gain / 2 * VDM_IA1_IN, GNDA, VDDA);
    end

end

//=========================================================================
// Output pins assignment
//=========================================================================
assign OUTP = OUTP_real;
assign OUTN = OUTN_real;
```

Figure 21: IA1 block : SV-RNM code

3.1.3 Model : aaf

An **Anti-Aliasing Filter** (AAF) is placed between the two amplifiers of the chain. This filter is a simple RC filter used to **condition** the Hall signal for the subsequent Analog/Digital conversion process that follows the amplifier chain. The main purpose of this filter is to eliminate all frequencies that surpass the Nyquist frequency. This action is taken to prevent **aliasing**, a phenomenon where higher frequency signals can be misrepresented as lower frequency signals (aliases) in the digital output, potentially leading to **inaccuracies**.

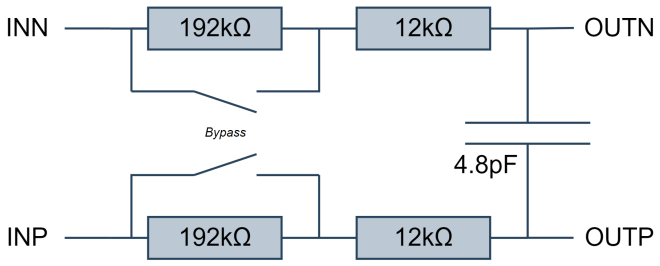


Figure 22: AAF block : Schematic

Since the inputs of the block are centered around VCM, this can be perceived as a simple RC filter with an additional feature - a switch to bypass the filter. The filter possesses a cut-off frequency of **84kHz** in normal mode, and **1.17MHz** when bypassed. To model this block, we discretely evaluate output values every 10ns based on the filter's RC value. This value is calculated using a **combinational block**, and is updated whenever the state of **BYPASSA** changes.

```
//=========================================================================
// AAFILTER
//=========================================================================
always_comb begin
    if (BYPASSA) RC = RC_bypass
    else         RC = RC_normal;
end

always @(RC) begin
    alpha  = dt/(RC+dt);  // Filter coeffecient
end

always #(dt) begin // Evaluate every dt = 10 nanoseconds
    outn_aaf = alpha*IN + (1-alpha)*outn_aaf;
    outp_aaf = alpha*IN + (1-alpha)*outp_aaf;
end

//=========================================================================
// Output pins assignment
//=========================================================================
assign OUTN = outn_aaf;
assign OUTP = outp_aaf;
```

Figure 23: AAF block : SV-RNM code

3.1.4 Model : ia2

The IA2 block is the second amplifier of the chain, positioned at the output. It has a **simpler** design than IA1, it offers an option to adjust its gain among **4 values**, 2.5/5/10/20. The circuit is basically two amplifiers with **negative feedback** and a variable resistor to change the gain value.

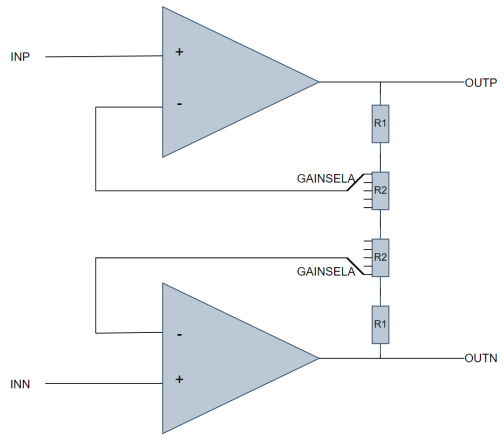


Figure 24: IA2 block : Schematic

From this circuit, we can compute the outputs voltage:

```
OUTP = (INP + INN)/2.0 + gain * (INP - INN)/2.0
OUTN = (INP + INN)/2.0 - gain * (INP - INN)/2.0.
```

Note that IA2 also have an **ENA** pin, the final SV-RNM code is shown Figure 25.

```
//=========================================================================
// Gain selection
//=========================================================================
//select the needed value for the gain, according to GAINSELA's input value
localparam real amp_gain_typ [0;3]= '{2.5, 5, 10, 20};
always @(GAINSELA) begin
    gain = amp_gain_typ[(GAINSELA)];
end

//=========================================================================
// IA2
//=========================================================================
always_comb begin
    if (~(ENA)) begin
        // circuit is disabled and the outputs are pulled down
        OUTP_real = 0;
        OUTN_real = 0;
    end
    else if ((ENA)) begin
        // circuit is enabled, normal operation
        OUTP_real = OUT_CLAMP ((INP+INN)/2.0 + gain * (INP-INN)/2.0, GNDA, VDDA);
        OUTN_real = OUT_CLAMP ((INP+INN)/2.0 - gain * (INP-INN)/2.0, GNDA, VDDA);
    end
end

//=========================================================================
// Output pins assignment
//=========================================================================
assign OUTN = OUTN_real;
assign OUTP = OUTP_real;
```

Figure 25: IA2 block : SV-RNM code

3.2 TEST : Test block

The **TEST** block is responsible for validating and ensuring the proper operation of the MLX90396. It allows checking different elements of the system. It permits ongoing system checks and fault detection, potentially catching and isolating issues before they impact the system's operation or cause failures. The TEST block is a **key block**, maintaining the sensor's operational integrity. It is mainly composed of four blocks; **iatest** that manages the connection of the testbus, pd_test allowing pulling down the testbus, crack_ring_test checking the functionality of the ring, and nv_ram_sw a switch connecting the testbus to a pin of the NV-RAM.

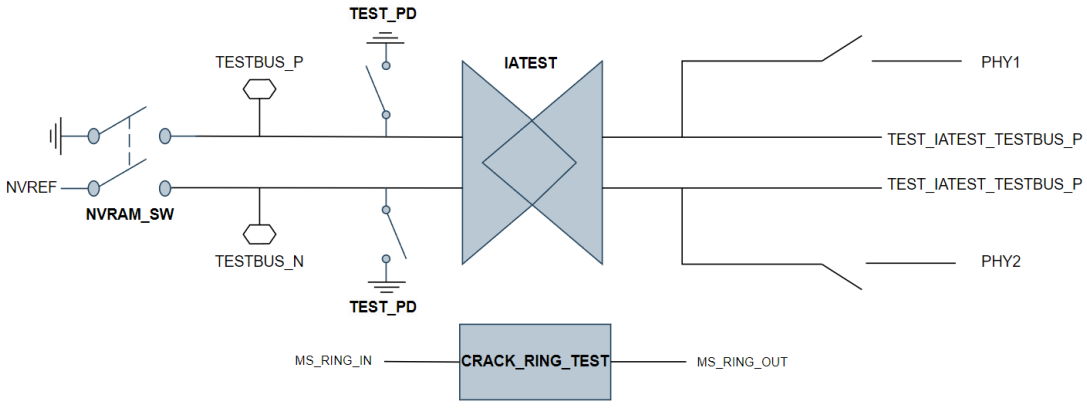


Figure 26: TEST block : Simplified schematic

3.2.1 Model : iatest

IATEST is a complex block managing the **testbus** in the TEST block. It has **2 output pins** MUX (connected to TEST_IATEST) and **4 inout pins** IN/OUT (INs are connected to Testbus, and OUTs to a pin of the PHY block). It also has **9 logic inputs**, that set the driving configuration of IN/MUX/OUT. A simplified schematic of the block (with only the 4 most important logic inputs) is shown in Figure 31. Note that the amplifiers have a gain of **approximately 1**.

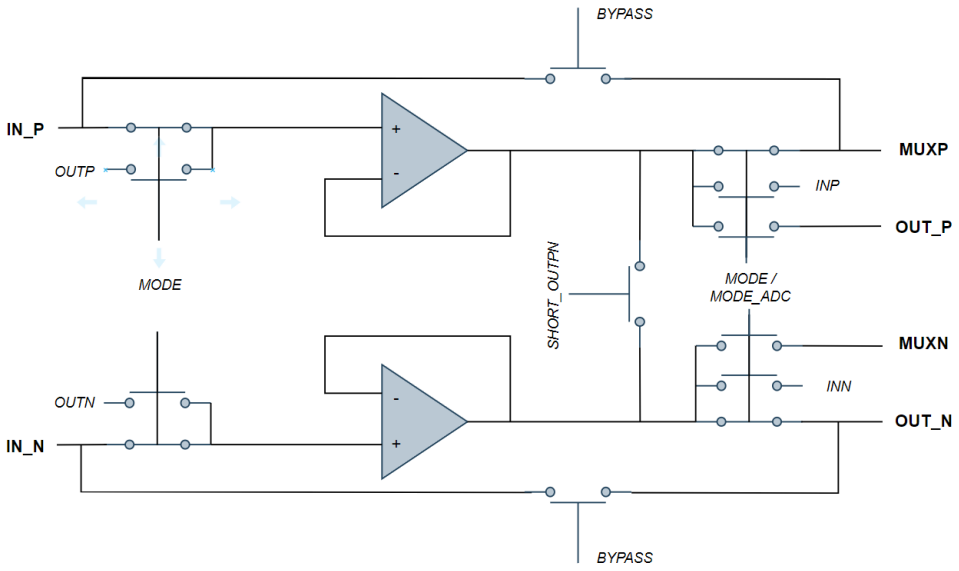


Figure 27: IATEST block : Simplified schematic

The main driving configurations are summed up in the table Figure 28. Note that there are 5 other input logics; **INVERTMUX** that inverts MUX outputs, **SHORT_OUTPN** that creates a short between outputs, **OUTP_SWITCH_OFF and OUTN_SWITCH_OFF** that disable an output, and **SWAP** that swaps the inputs.

EN	MODE	BYPASS	MODE_ADC	FUNCTION
0	X	0	X	Amplfiier Disabled
0	X	1	X	Amp Disabled and Bypassed
1	0	X	0	Driving OUT - > IN
1	1	X	0	Driving IN - > OUT
1	0	X	1	Driving OUT - > MUX
1	1	X	1	Driving IN - > MUX

Figure 28: IATEST block : Driving config

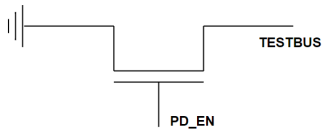
To model this block, we consider each configuration of the block and make a big **case inside** statement for each possible configuration. This statement allow to use question mark "?" as a **wildcard character** that represents any value. A snip of the code is shown Figure 29.

```
//========================================================================
// Behavioral description
//========================================================================

logic  [8:0]  sel;
assign sel = {BYPASS, MODE, EN, MODE_ADC, OUTP_SWITCH_OFF, OUTN_SWITCH_OFF, SWAP, INVERTMUX, SHORT_OUTPN};

always_comb begin
    case (sel) inside
        9'b1_?0?0_00?0 : // Inputs to outputs bypass
                begin
                    in_pi = `wrealZState;
                    in_ni = `wrealZState;
                    out_pi = IN_P;
                    out_ni = IN_N;
                    mux_pi = `wrealZState;
                    mux_ni = `wrealZState;
                end
```

Figure 29: IATEST block : SV-RNM code

3.2.2 Model : i_test_pd

On the opposite, I_TEST_PD is a very simple block, it allows to **pull-down the testbus**. It means connecting them to VSS. The code of this block is straightforward.



(a) I_TEST_PD : Schematic

```
//========================================================================
// Interface Signals
//========================================================================
    output OUTPUT;          real OUTPUT;
    input PULLDOWN;         logic PULLDOWN;
    input VSSA;             real VSSA;
//========================================================================
// Pull-Down
//========================================================================
    assign OUTPUT = (PD_EN === 1'b1) ? VSS : `wrealZState;
```

(b) I_TEST_PD : code

Figure 30: I_TEST_PD block modeling

3.2.3 Model : nv_ram_sw

This block is a **bidirectional** switch connected to a pin of the Non-Volatile Random-Access Memory (**NVRAM**) in the system. It means that it is used in both directions, to assign either to the NV-RAM pin or to the testbus. The switch is controlled by NVRAM_TEST and HPORB.

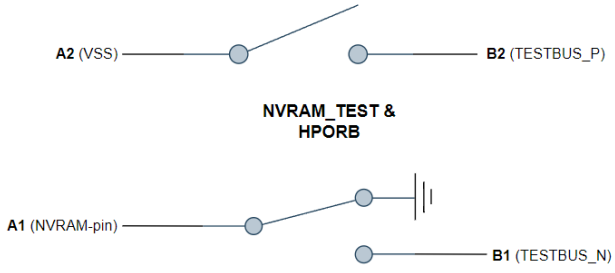


Figure 31: NV_RAM_SW block : Simplified schematic

Modeling a **bidirectional** switch in real-number modeling can be a bit complex because it deals with both inputs and outputs. The real datatype doesn't permit multiple drivers, implying for instance that we can't drive **testbus** if it's already driven by the NV-RAM switch. Consequently, we cannot model a bidirectional switch directly, so we resolved to model an **unidirectional** switch with selectable direction. The direction of the switch will be determined by a bit **direction_AB**; this method enables us to assign in both directions as intended.

```
//================================================================================
// Internal Signals
//================================================================================
    bit direction_AB = 0; // Will give the direction of the switch, need to be defined in top level using hierarchical reference

//================================================================================
// Implementation
//================================================================================

    // Direction B->A
    assign A1 = direction_AB ? `wrealZState : (TEST1 & HPORB_3v3) ? B1 : VSSAP;          // Connected to VSSAP when disabled
    assign A2 = direction_AB ? `wrealZState : (TEST1 & HPORB_3v3) ? B2 : `wrealZState;

    // Direction A->B
    assign B1 = direction_AB ? ((TEST1 & HPORB_3v3) ? A1 : `wrealZState) : `wrealZState;
    assign B2 = direction_AB ? ((TEST1 & HPORB_3v3) ? A2 : `wrealZState) : `wrealZState;
```

Figure 32: NV_RAM_SW block : Code

3.2.4 Model : crack_ring_test

This block allows the performance of a **crack-ring test**. This is a self-test mechanism used to detect physical faults or 'cracks' in the chip. The MPS utilizes a **'ring oscillator'**, a test structure capable of detecting faults such as cracks or thinning in the **silicon**, which affect the timing of the ring oscillator. The schematic of this block is depicted in Figure 33.

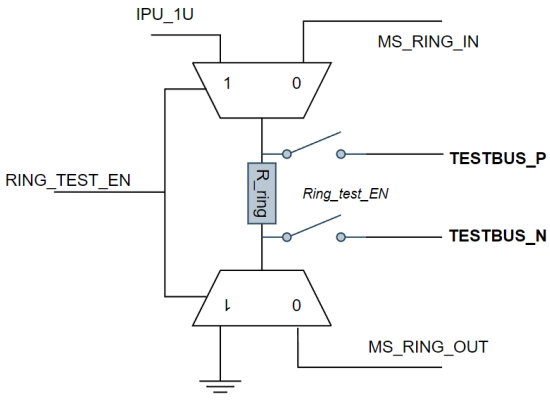


Figure 33: CRAK_RING_TEST block : Simplified schematic

Here, R_{Ring} represents the resistance of the ring. When RING_TEST_EN is active, the **testbus** will be connected to the current source IPU_1U and VSS. When disabled, the testbus is left floating and MS_RING_IN and MS_RING_OUT are connected. The current is expected to be **close to 0**, hence the voltage is the same across the resistor. The simple code of this block can be seen Figure 34.

```
//================================================================================
// Implementation
//================================================================================
    assign TESTBUS_N = (RING_TEST_EN===1) ? VSSA : `wrealZState;
    assign TESTBUS_P = (RING_TEST_EN===1) ? IPU_1U : `wrealZState;
    assign MS_RING_OUT = (RING_TEST_EN===1) ? `wrealZState : MS_RING_IN;
```

Figure 34: CRAK_RING_TEST block : SV-RNM code

3.3 Other modeled block

Several other blocks have been modeled during the internship, which include:

- **AFE: MUX:** This model is a **multiplexer** in the Analog Front-End (AFE) that routes analog signals from multiple inputs to a single output line. Its function is essential for **selecting** specific signals for further processing, based on control inputs.
- **AFE: TEMPSENSOR:** This model is a temperature sensor within the AFE. It is used for temperature measurement and to compensate the **thermal characteristic** of the hall signal.
- **SUP: IBIAS:** This model is responsible for generating the bias currents. It provides a **consistent** and reliable current source, allowing other components to **function correctly**.
- **TOP: VDD/VSS:** These two blocks are responsible for assigning voltage **supply** and **ground** to all other blocks in the design, based on one input source. They ensure the correct **power distribution** across the entire system.

3.4 Modeling current consumption

As noticed in the review of the **V-AMS** model (subsection 2.4), some of the models consider **current consumption** using analog procedural blocks. For low-power designs, system current consumption is a crucial specification. We aim to ensure the **lowest** possible current consumption while maintaining system performance. The goal in modeling the current consumption is to allow potential issues to be identified early in the **design cycle**. Instead of waiting for the complete **schematic AMS simulation**, we can quickly ensure that the design meets the low-power requirements using the models. SystemVerilog, a digital description language, does not allow analog procedural blocks. Therefore, to model current consumption, we will use **Cadence's EEnet**. As mentioned in subsection 2.5, this package provides a structured **UDN** consisting of three fields: Voltage, Current, and Resistance, allowing the creation of digital equivalents of analog circuits. It implies that we can use EEnets to model supply nodes as **constant current sinks** to simulate current consumption (V=0, I=-block_current, R=Z). The current consumption modeling method is shown in Figure 35. For further details, an example of this method has been applied in [3].

```
import EE_pkg::*;
EEnet VDD, VSS, Ib2u;
//================================================================================
// Current consumption
//================================================================================
always_comb idc_tmp = (BLOCK_EN)? idc_on + idc_off : idc_off; //calculating current consumption
assign VDD = '{0.0, idc_tmp,`wrealZState};
assign VSS = '{0.0,-idc_tmp,`wrealZState};
```

Figure 35: Current consumption modeling using EEnet

We extended each block of the AFE using the `EEnet` package. The objective is that, when running top-level simulations, we assign an **ideal voltage driver** to `VDD` and `VSS` {Vdrive,0,0}. This way, the drive voltage would directly define the output voltage, and the resolution function would compute the **current** flowing into that source, providing a resolved value in the form {Vdrive, I, 0}. In our case, 'I' would represent the **resolved current** considering the current consumption of every block. If we are only assigning ideal voltage sources, it corresponds to the **sum** of all currents.

4 Testing Functionality

Now that we have developed models, we need to ensure their functionality, The idea is to create a **verification environment** to compare each SV-RNM model with each actual transistor-level simulation.

4.1 Methodology

To compare models, both the SV-RNM and schematic block are instantiated in a **testbench**, and their respective inputs/outputs are connected to a **checker** module. This module is responsible for reporting an error in the log when the **discrepancy** between the output values of the schematic simulation and the SV-RNM model exceeds a certain value. Two stimuli modules inside a **wrapper** are employed to assign varying input values to the models. This setup is referred to as a **testcase**, a fairly straightforward verification environment previously utilized in the MLX90427 project.

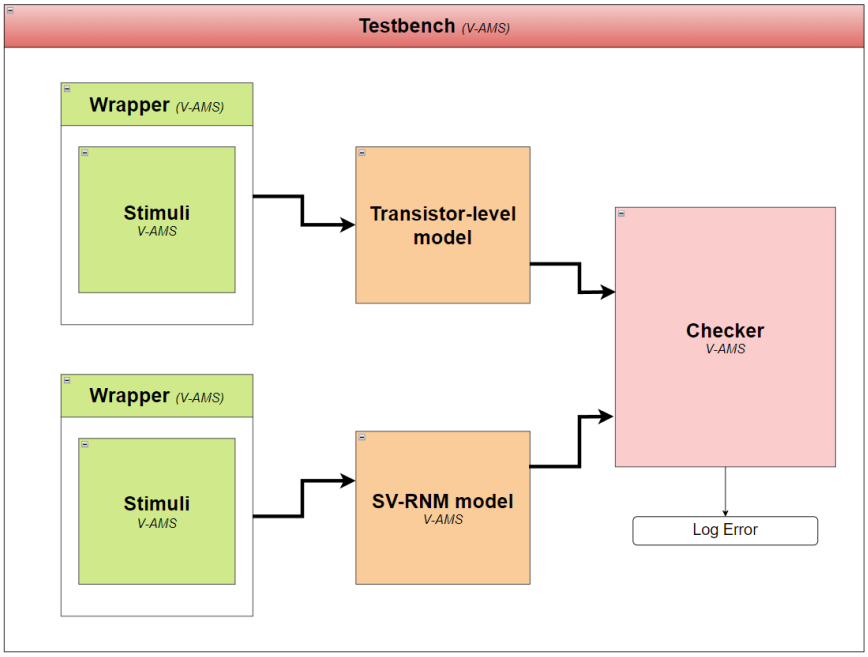


Figure 36: Verification environment

4.1.1 Testbench/Stimuli

The testbench orchestrates the actual **testing**. In this setting, the transistor-level model, the SV-RNM model, the stimuli/wrapper, and checker are instantiated and wired **together**.
The stimuli block in the testbench is tasked with generating **diverse inputs** for the models. It functionally tests the system under various scenarios, introducing **edge cases** that push the design to its limits. The stimuli block, by providing a comprehensive set of testing conditions, plays a critical role in verifying that the SV-RNM model operates correctly under all anticipated conditions.

4.1.2 Wrapper

The wrapper (V-AMS) essentially serves as an **interface** between the stimuli and the models. For instance, this wrapper file defines **supply/current sources** and grounds based on their value generated in the stimuli. Furthermore, it is used to define the supply sensitive **connect modules (CM)**. These modules bridge the analog and digital domains in mixed-signal designs, with the analog domain using continuous 'electrical' signals, and the digital using discrete 'logic' or 'real'. In the **continuous** domain, high (1) logic is often considered as VDD, and low (0) logic as the ground. The wrapper imparts power supply information to enable the CM to discern which **analog voltages** correspond to the 1/0 logic values. These definitions are **imperative** for the proper operation of the transistor-level model.

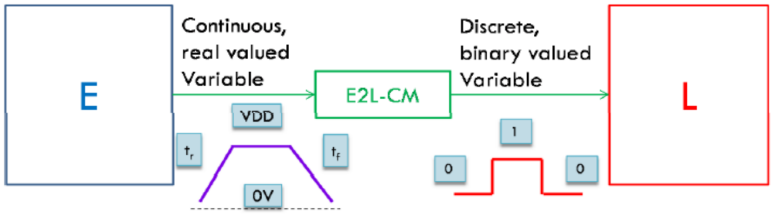


Figure 37: Electrical to Logic connect module

4.1.3 Checker

The `checker` is tasked with comparing the `SUT` (System Under Test) outputs. If a **discrepancy** too high is found, the `checker` will **raise an error** in the log file. The log-error will report the **time** at which the error was discovered. Additionally, we can also compare the transistor-level model outputs with the **specifications** of the block, to ensure correct functionality. This auto-checker allows for easy detection of any issue with the model, regardless of the stimuli used.

```
    parameter real max_error = 10e-3;

// ===== OUTP/OUTN test =========
    always @(DoCheck_outputs) begin

        // Testing schematic vs SV-RNM
        OUTN_equal = (abs(V(OUTN_sch) - OUTN_mod) < (max_error)) ? 1'b1 : 1'b0;
        OUTP_equal = (abs(V(OUTP_sch) - OUTP_mod) < (max_error)) ? 1'b1 : 1'b0;

        // Generate an error message if signals don't match
        if(!OUTN_equal)
            `log_Error("Mismatch between model and schematic on OUTN !");
        if(!OUTP_equal)
            `log_Error("Mismatch between model and schematic on OUTP !");
    end
```

Figure 38: Checker code

4.2 Amplification chain

Each model's functionality is tested using our **verification environment**. The simulations are plotted using Cadence's `SimVision` tool. The functional validation of a model is affirmed when it successfully passes the **visual inspection** on `SimVision` and **no log-errors** are reported by the auto-checker. In this section, we will validate the functionality of our models for the entire **amplification chain**.

4.2.1 Stimuli

The stimuli for the amplifier chain block are implemented in a **multi-step** process. In the `AFE`, the amplification chain inputs are directly connected to the **Hall Plates**. Therefore, sinusoids are continuously generated on `IN_Ax` and `IN_Cx`, with an amplitude of 15mV, which corresponds to the typical output of the Hall-Plates. Each of these are **discretely updated** every 20us. We then conduct three steps of simulation:

- **Step 1:** Generates 25 random low-value gains for IA1, and high-value for IA2, with offset=0.
- **Step 2:** Generates 25 random high-value gains for IA1, low-value for IA2, and a random offset.
- **Step 3:** Enables the filter bypass, and generates 10 random low-value gains for IA1, and a high-value gain for IA2, with offset=0.

The first step checks the functionality of the chain **without offset**. Low/high value gains are generated to avoid **saturation**; a high value in both would cause the block to saturate to VDD. The second step is used to verify the functionality of the **offset** feature. Lastly, the third step is for verifying the **bypass** feature. Note that here the stimuli is simplified for explanation; in actuality, we include other steps such as power-up, **fault-injection**, and verification of test-buses routing (each block is connected to test-buses).

```
`log_Note("Step 1- Generate 25 Random IA1 low-value gain, IA2 gain with 0 offset");
for (i=0; i < 25; i=i+1) begin
    offseta = 0;
    gainsela_ia1 = $urandom(seed+i*1) % 4;              // ranging on only 4 IA1 gains out of 16
    gainsela_ia2 = $urandom(seed+i*2) % 4; #5_000; end  // ranging on the 4 IA2 gains

`log_Note("Step 2- Generate 25 Random IA1 gain, IA2 low-value gain and offsets");
for (i=0; i < 25; i=i+1) begin
    gainsela_ia1 = $urandom(seed+i*1) % 16;               // ranging on the 16 IA1 gains
    gainsela_ia2 = $urandom(seed+i*2) % 2;                // ranging on only 2 IA2 gains out of 4
    offseta      = $urandom(seed+i*3) % 128; #5_000; end  // ranging on all possible offset value (7bits)

`log_Note("Step 3- Generate 10 Random IA1 gain, IA2 low-value gain, with bypass and 0 offset");
bypassa_aaf = 1;  offseta = 0;
for (i=0; i < 10; i=i+1) begin
    gainsela_ia1 = $urandom(seed+i*1) % 16;             // ranging on the 16 IA1 gains
    gainsela_ia2 = $urandom(seed+i*2) % 2; #5_000; end  // ranging on only 2 IA2 gains out of 4
```

Figure 39: Amplification chain stimuli : Simplified code

The **random** generation is performed using $urandom(seed + i * 1)\%16$. This line generates a random unsigned integer, using a variable **seed**, and obtains the **remainder** of the random number divided by 16. This method enables the generation of a random value between 0 and 15. The seed allows the random number to be **reproducible**. The waveform of these inputs is shown Figure 40.

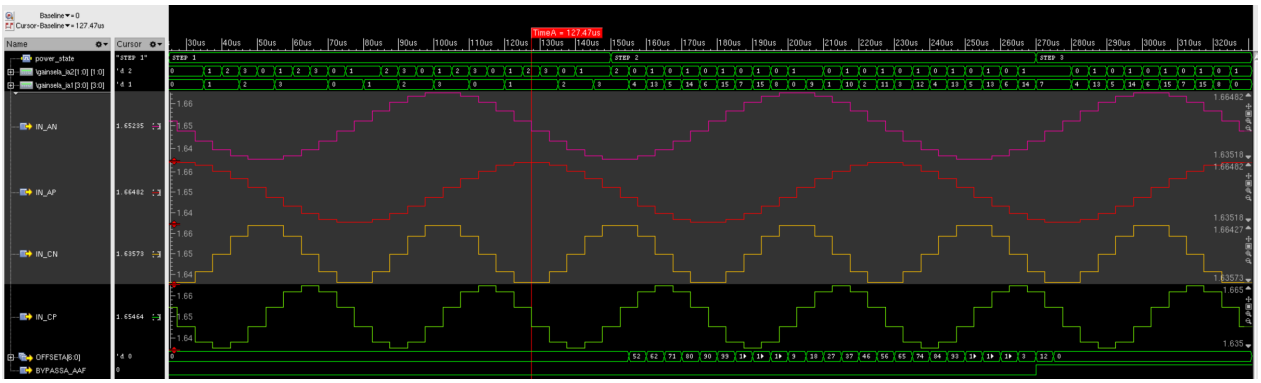


Figure 40: Stimuli : SimVision waveform

4.2.2 Simulations results

The simulation is executed with the previously generated **stimuli**. We then compare the **differential** output of the chain (OUTP-OUTN) between the SV-RNM model and the schematic simulation. Given the length of the simulation, a complete screenshot would appear too small in the report. Therefore, we will only present **extracts** from each step. An extract from **step 1** is shown Figure 41.

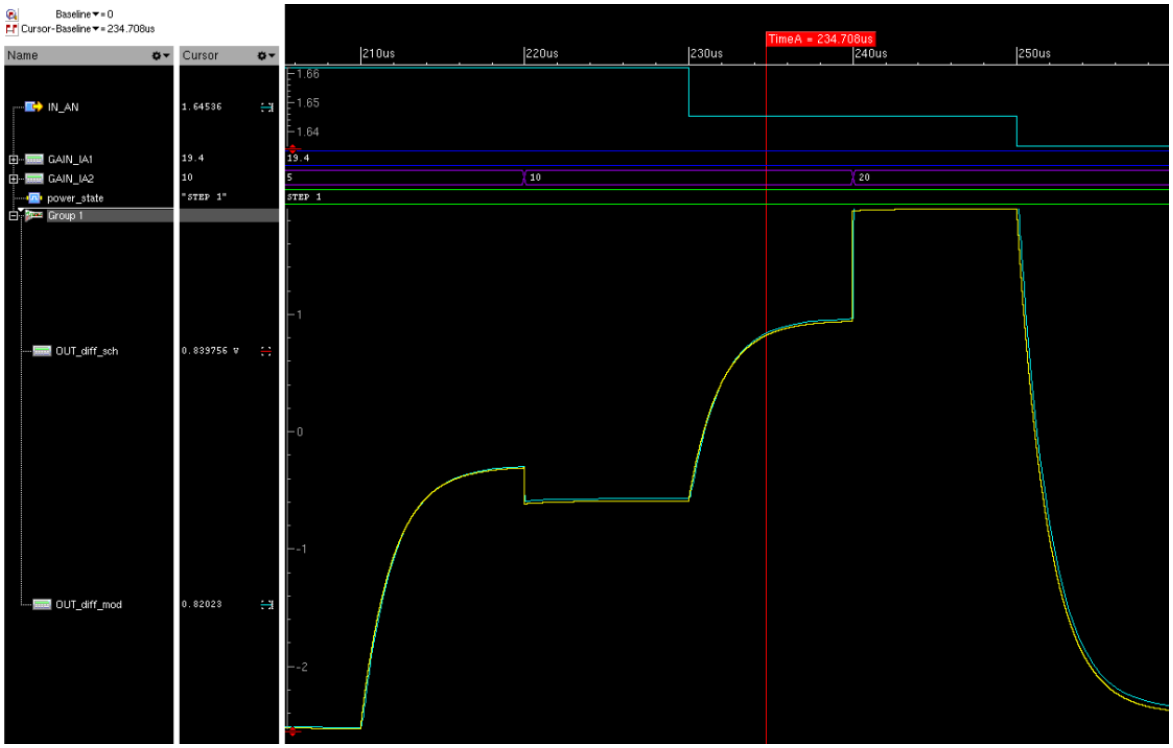


Figure 41: Step 1 : SimVision waveform

Here, an input (IN_AN) is depicted at the top, accompanied by the IA1/IA2 gains. In this section, the tested gains **configurations** include 19.4/2.5, 19.4/5, and 19.4/10. Below, an **overlay** of the differential outputs is shown. The blue curve represents the schematic simulation, and the yellow represents our model. Notably, the two are barely **distinguishable**, indicating that the behavior is accurately modeled. It is important to note that the transitions clearly exhibit the behavior of the **RC filter** (AAF).

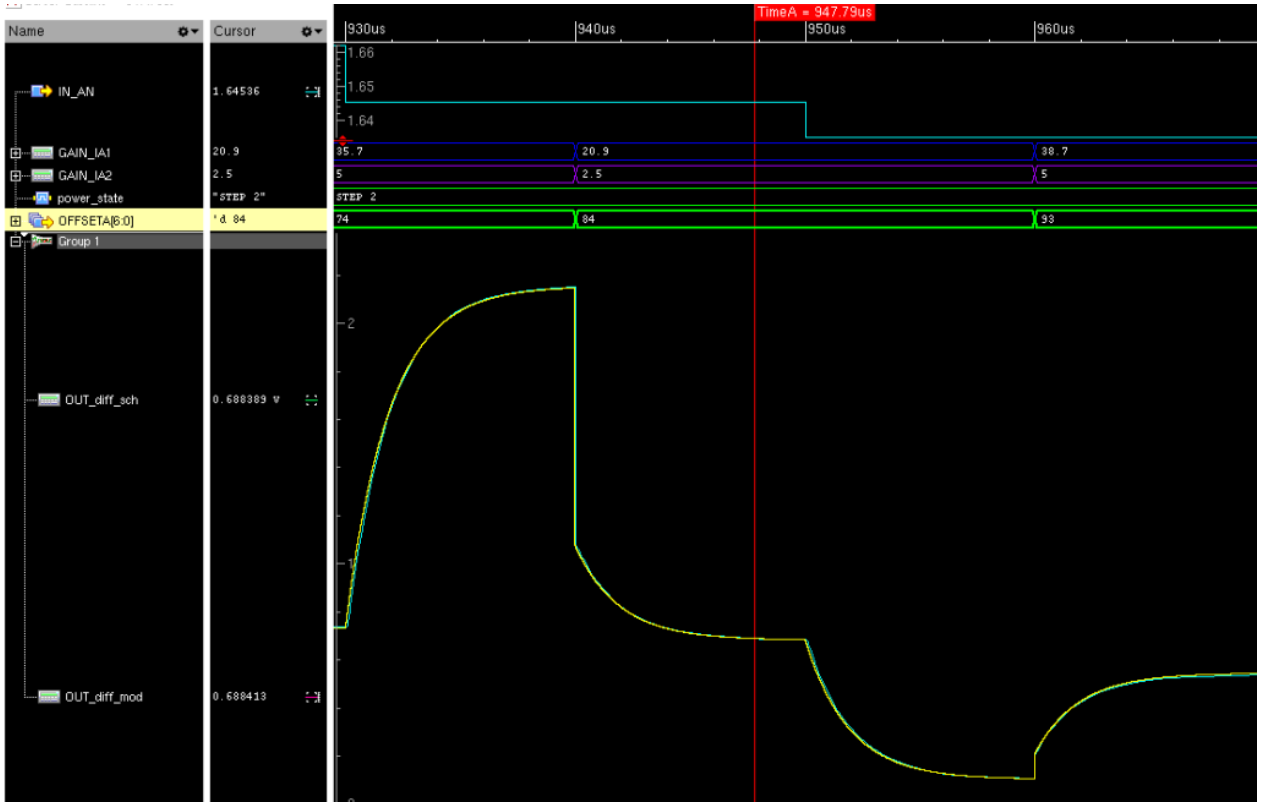


Figure 42: Step 2 : SimVision waveform

Moving on to **Step 2**, we incorporate the offset into the configuration. The tested IA1/IA2/Offset **configurations** are 35.7/5/74, 20.9/2.5/84, and 38.7/5/93. Once again, it is nearly impossible to differentiate between the two curves, confirming that the behavior is being correctly modeled.

Finally, for the third step (Figure 43), we **activate** the filter. The IA1/IA2 gain configurations tested here are 25.8/5, 30.9/2.5, and 38.7/5. Once again, we barely **differentiate** the two curves, indicating that the behavior is **accurately** modeled. Importantly, we also confirm that the transitions are significantly faster, which affirms the **bypassing** of the filter.

In conclusion, since the **auto-checker** did not return any log-errors, we confirm that the model is accurately modeling the behavior of the **amplification chain**.

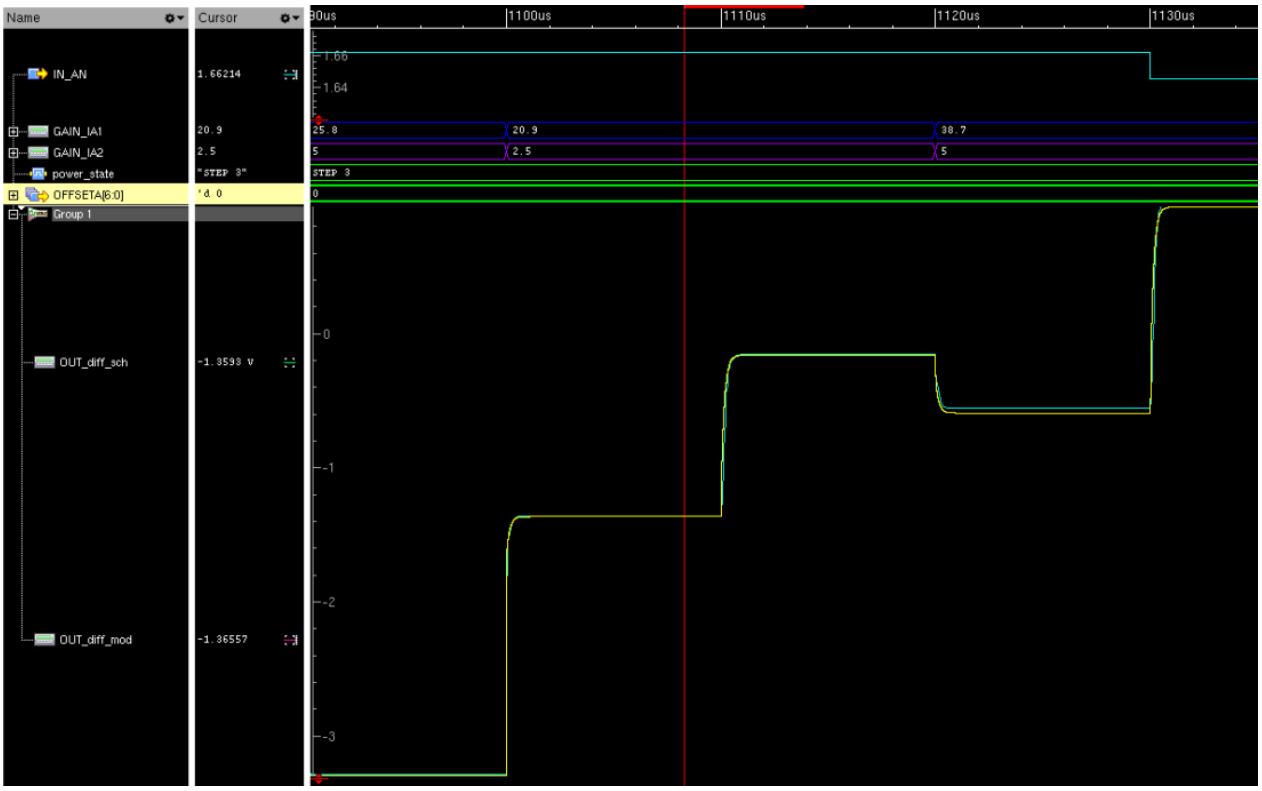


Figure 43: Step 3 : SimVision waveform

4.3 Test block

4.3.1 Stimuli

The same approach is used to check the functionality of the **Test block**. The stimuli is also implemented in a **multi-step** process. The test block comprises many components, each of which needs to be tested for functionality. This is achieved in four simulation steps:

- **Step 1:** Testing iatest; injecting 16 different **configurations** into the 9 logic inputs. Examples of these configurations can be seen in Figure 44.
- **Step 2:** Testing NV-RAM **switch**; enabling the switch and writing to the NV-RAM **ref pin**.
- **Step 3:** Testing Pulldowns; this step enables the **pulldowns** to connect test-buses to VSSA.
- **Step 4:** Testing the Crack Ring; this step involves activating the ring test.

```
assign {BYPASS, MODE, EN, MODE_ADC, OUTP_SWITCH_OFF, OUTN_SWITCH_OFF, SWAP, INVERTMUX, SHORT_OUTPN} = iatest_config[i];
iatest_config[0] = 9'h100;  // EN = 0, MODE = 0, BYPASS = 1 => IN -> OUT (short IN&OUT)
iatest_config[1] = 9'h101;  // EN = 1, SHORT_OUTPN = 1 => OUTP -> OUTN (short OUTP&OUTN)
iatest_config[2] = 9'h104;  // EN = 0, MODE = 0, BYPASS = 1, MODE_ADC = 0 => IN -> OUT
iatest_config[3] = 9'h140;  // EN = 1, MODE = 0, BYPASS = 1, MODE_ADC = 0 => OUT -> IN
```

Figure 44: Stimuli : Example of configurations in step 1

Every step involves driving the *test-buses* and other *inout* pins only once, requiring **careful management** of stimuli assignment. This implies that if a pin is a driver in the block, it is assigned a **random** value between 0 and VDD; otherwise, a **high-impedance** state (indicating disconnection) is assigned. In *SimVision*, high-impedance states are represented by **orange** rectangles. It is important to note that each *inout* pin in the transistor-level model is connected to a **pull-down resistor**, ensuring a low state value read, as the electrical datatype does not support high-impedance states.

4.3.2 Simulations results

In Figure 45, we show the waveform of **step 1** based on the three first configurations values. As we can see, the behavior between the SV-RNM model and the schematic is the **same**. We also confirm that, when the model is in a high-impedance state, the schematic is **pulled-down**.

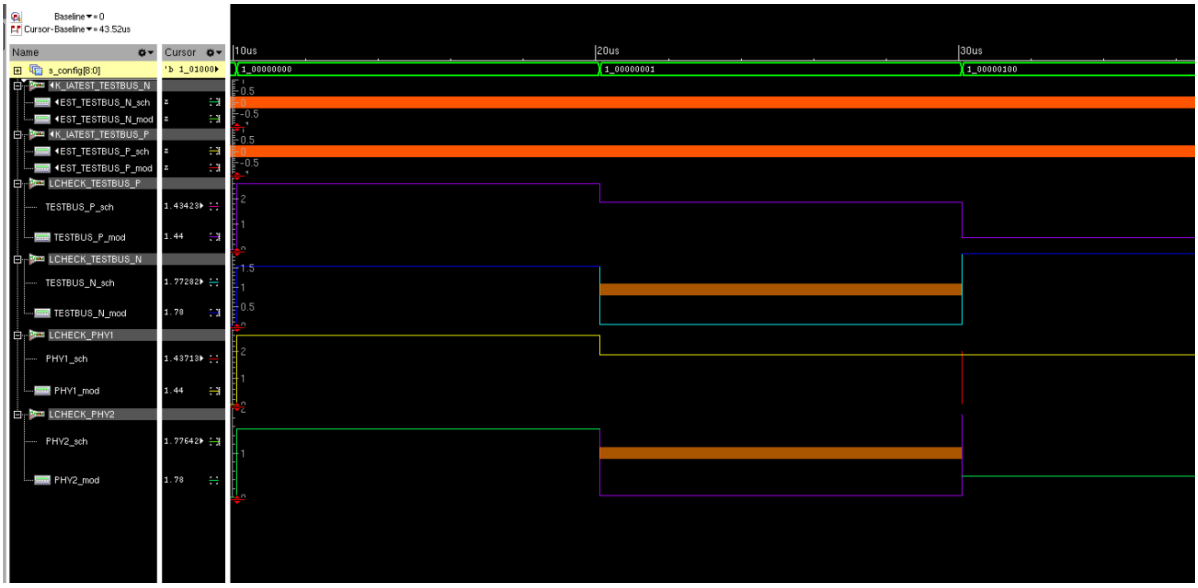


Figure 45: Test : exract of Step 1

In Figure 46, we show the waveform of step 2 and 3. The behavior between SV-RNM model and the schematic is also the same; test-bus N drives NVRAM pin in step 2, then pulled-down in step 3.

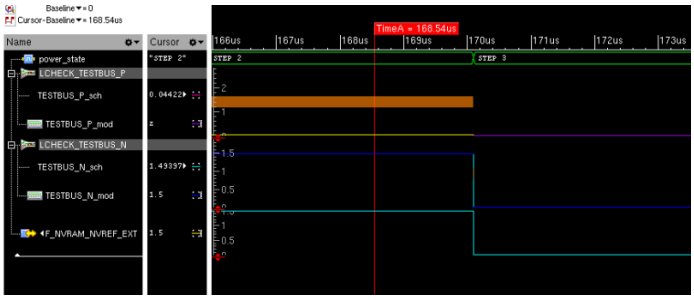


Figure 46: Test : Step 2 and 3

Lastly, we also confirm the functionality of **step 4**, and that the auto-checker is returning 0 log-error. Hence **validating** the SV-RNM model of the test block.

5 Comparative Study : RNM vs VAMS

In this part, we wanted to propose and apply a **state-of-the-art** approach to measure **performances** of SV-RNM vs VAMS models.

5.1 Methodology

5.1.1 What to compare ?

In order to compare the models, we first need to **identify** what could be interesting to **compare**. Among the factors we can evaluate, there are four main points that arise:

- **Numerical accuracy:** Compare the numerical accuracy of both models by evaluating the maximum **discrepancy** between the simulated results and the expected values. Lower deviations indicate higher numerical accuracy and **better performance**.
- **Overall "stopwatch" time (real):** Corresponds to the time it takes to run a simulation. This is one of the most **critical** aspects of performance.
- **CPU time in user space (user):** Corresponds to the CPU time used to perform the **code** in itself. It helps gauge simulation efficiency and pinpoint **optimization** opportunities within code or the simulator.
- **CPU time in system space (sys):** Corresponds to the CPU time used by the operating system to perform tasks for the program. It typically includes, **system calls** like file I/O and **memory management**

5.1.2 Numerical accuracy

The first thing we decided to evaluate was the **numerical accuracy**. Discrepancies were observed between the *SV-RNM/V-AMS* models and the transistor-level models in the *AFE* (**analog** components). To measure these discrepancies, we decided to use the **auto-checker**. We simply added variables '`diff`' to measure the difference, and '`max_diff`' to capture the maximum difference.

```
//Calculate diffs
diff_N = abs(V(OUTN_sch) - OUTN_mod);
diff_P = abs(V(OUTP_sch)  - OUTP_mod);

// Calculate max diff
if (diff_N > max_diff_N) max_diff_N = diff_N;
if (diff_P > max_diff_P) max_diff_P = diff_P;
```

Figure 47: Added lines in checker

Note that it is important to **define** the outputs from the schematic and the real as they are respectively defined in each model: **electrical** and **real**. If we use a different datatype, a **connect module** will be inserted to translate the datatype. However, connect modules are often **event-based**, meaning that they only update when the value changes by more than a predefined **delta**. This can introduce unnecessary inaccuracies, thus **distorting** our measurement of numerical accuracy. Ultimately, the maximum measured error displayed in the log.

```
`log_Note("NOTE: End of b2b check for model m90396vAA_amplifiers");

$display("Max observed error between SCh and MOD on OUTN: %f",i_checker.max_diff_N);
$display("Max observed error between SCh and MOD on OUTP: %f",i_checker.max_diff_P);
```

Figure 48: Added lines in the stimuli

5.1.3 Time comparison

Regarding the time comparison, we observed that the run log displays a information about the **simulation time** for each simulation. Among these, we noticed the **CPU system/user** time, as well as the overall simulation time. An extract of the log is shown in Figure Figure 49.

```
Simulation started at: 3:21:56 PM, Thur Jul 13, 2023, ended at: 3:27:35 PM, Thur Jul 13, 2023, with elapsed time (wall clock): 339 s (5m 39.0s).
spectre completes with 0 errors, 16 warnings, and 31 notices.
Total License Acquisition Time : 4.43 s.
  xmsim  : License Acquisition Time -- 4 s.
  spectre: License Acquisition Time -- 0.433 s.
xmsim: Memory Usage - Final: 88.0M, Peak: 111.9M, Peak virtual: 481.3M
xmsim: CPU Usage - 29.5s system + 320.3s user = 349.7s total (346.0s, 100.0% cpu)
TOOL:   xrun(64)    23.03-s001: Exiting on Jul 13, 2023 at 15:27:35 UTC  (total: 00:06:00)
```

Figure 49: Information diplayed by the run log

Therefore, for each simulation, we extract the desired information. To ensure a meaningful **comparison** between different models, we conducted 10 **consecutive** simulations using **command line** entry exclusively, all executed on the **same node** (Melexis provides multiple nodes, and we utilized a high-clock machine with 12 cores). From each simulation, we obtained the corresponding execution times, and make the average. We decided to develop a **command shell script** to automate these tasks. The code for this script is provided in annex Figure 54.

5.2 Amplification chain

Once we know what to measure and how to do it, the next step is to **perform** the measurements. The **analog** blocks from the AFE are crucial for conducting these measurements. They may exhibit unexpected **discrepancies** and require **long** simulation times. Consequently, we opted to focus our comparison on the **amplification chain**.

5.2.1 Accuracy

The simulation with the updated **checker/stimuli** is run for the SV-RNM model using the transistor-level model. It is important to note that we conducted the **same** simulation as the one mentioned earlier to verify the functionality in subsubsection 4.2.2. The maximum error values obtained are **displayed** in the log, as shown in Figure 50.

```
csp_logger: *N,MLXTC (time 1285000 NS) NOTE: End of b2b check for model m90396vAA_amplifiers
Max observed error between SCh and MOD on OUTN: 0.018504
Max observed error between SCh and MOD on OUTP: 0.024865
```

Figure 50: Displayed max error : SV-RNM vs Schematic

The measured maximum error shows a **precision** of around 20 to 25 mV. This error corresponds to a precision of **1.5%** for the SV-RNM model, comparing to a VCM of 1.65V. The correctness of the displayed error is confirmed with the **SimVision** waveform. Consequently, the next step involves running simulations on the **V-AMS** model compared to the **Schematic**. The displayed errors are shown in Figure 51.

```
csp_logger: *N,MLXTC (time 1285000 NS) NOTE: End of b2b check for model m90396vAA_amplifiers
Max observed error between SCh and MOD on OUTN: 0.026820
Max observed error between SCh and MOD on OUTP: 0.021550
```

Figure 51: Displayed max error : V-AMS vs Schematic

The measured **maximum error** now exhibits a precision of approximately 20 to 25 mV, which is **comparable** to SV-RNM. It is worth noting that the measured values are slightly higher than those of the SV-RNM model. Thus, we can **confidently** assert that we obtained a **similar** level of accuracy as the V-AMS models.

5.2.2 Performances

The next step is to compare the **performance** of each model. By performance, we refer to the **simulation speed**. We use the same simulation as depicted in subsubsection 4.2.2 and execute the shell script for each model: **V-AMS**, **SV-RNM**, and **Transistor-level**. Additionally, we will run simulations for the SV-RNM **EEnets** model, taking into account the current consumption. The initial focus is on the simulation time itself. The time for each simulation is shown in Figure 52.

amp-chain simulation time				
	Schematic	**SV-RNM**	**EENet**	**V-AMS**
Run Time	~12min	2.2s	2.5s	17.4s
% of time	100%	0.31%	0.35%	2.4%

Figure 52: Simulation time comparison

The obtained results show that for a simulation of **12 minutes**, we are able to achieve it in only **2.2 seconds** using our SV-RNM model. In contrast, the V-AMS model took **17.4 seconds**. It's important to note that both models have the **same level of accuracy**. However, the V-AMS model specifically models current consumption. To compare simulation speeds at the same level of precision, we can compare it to the **EEnet** model, which completes the simulation in **2.5 seconds**, compared to the 17.4 seconds taken by V-AMS. Now, let's examine the **CPU times**. The corresponding values are plotted in Figure 53.

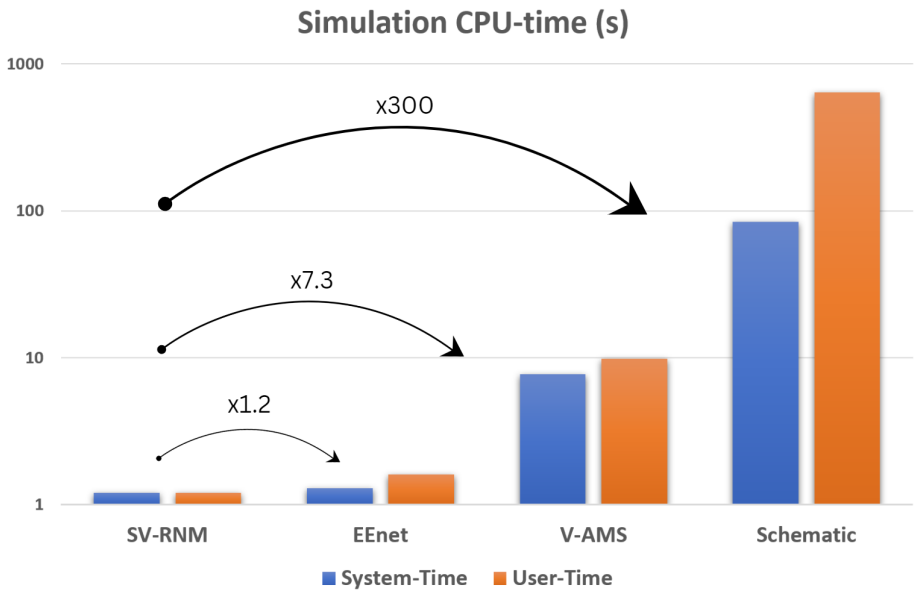


Figure 53: CPU time comparison

Here we highlight the fact that the simulation is **300 times** faster than the Schematic, and **7.3 times** faster than the V-AMS. Note that the sum of CPU system + user time is often close to overall simulation time (wall clock), but does not equals it. It can be explained by **multitasking** or **system overhead**. What is shown in this plot, is that the more complex is the model, the **more CPU** user time is expected. Which was expected since it represent the time running the **code in itself**. Note that the CPU system-time is not always close to the user-time. In fact, the system-time will less be affected by simulation complexity/duration. This means that for very long simulations, it is expected to have **way less impact** than user-time, whereas for short simulation it will have a **big impact**.

5.2.3 Conclusion

As a conclusion to this **comparative study**, we can affirm that our SV-RNM models demonstrate **promising** results. They exhibit simulation speeds that are more than 7 times faster, while maintaining the **same level of accuracy** as the previously developed V-AMS models. It is worth noting that we successfully modeled current consumption using **EEnets**, without compromising simulation performance.

6 Bonus : Internship Cost Analysis

As a bonus, we found interesting to calculate the **overall cost** incurred to host and support an intern effectively for the company. It is interesting for the three sides of this internship, giving insights for budgeting allocation to the **company**, allows **students** to appreciate the value of their learning experience, and helps **Phelma** understanding industry needs. This analysis will provide an overview of the various elements contributing to the internship cost for the company.

1. **Remuneration:** The company provided a total remuneration of 2500 CHF per month. With an internship duration of 5.75 months, the total remuneration cost amounts is **14375 CHF**.

2. **Trainings:** I participated in two Cadence trainings, each lasting 40 hours. These training's are part of Cadence's subscription paid by Melexis. Since this one would be the same if I was not there, we considered this **free of charge**.

3. **Support and Review:** On average, I received one hour per day of support and review from a member of the MSDE team. At an hourly rate of 45 CHF, the total cost of this support over 108 working days amounts to **4860 CHF**.

4. **Software:** The software used during the internship, including Polarion, Gitlab, Google Suite, and Jira, are considered as **negligible** costs. Even if they have a cost, they are part of big bundles paid by the company that are hard to estimate. The overall cost/employee would not be very high, and since Melexis is a big company, this bundle doesn't change for 1 intern.

5. **Equipment:** I was provided with necessary equipment, including laptop, two screens, and diverse equipments. The cost of this equipment was estimated at 1500 CHF. Considering an equipment duration of 5 years and its shared nature, the portion of equipment cost attributed to the interns during their 5.75 months of the internship is calculated to be **125 CHF.**

6. **Accident Insurance:** The company provided accident insurance for the interns. These kind of insurance typcally have a cost of 100 CHF/months. Therefore, we can estimate that **600 CHF** were spent on insurance by the company.

7. **Working Permit:** The working permit required for the interns incurred a cost of **65 CHF** for the company

Total Estimated Cost: 20025CHF It is important to note that while the internship program incurs costs for the company, it also brings valuable contributions from the interns, such as fresh perspectives, new ideas, and potential future talent.

7 Conclusion

To conclude this project, it can be stated that the internship's objectives were **fully achieved**. New **SystemVerilog models** were successfully developed, demonstrating simulation speeds over **7 times faster** than previous **V-AMS models**, while maintaining the **same level of accuracy**. Notably, current consumption was accurately modeled using **EEnets**, without compromising simulation performance.

On a personal note, I found this experience **highly instructive**. It gave me a comprehensive understanding of an engineer's role. Throughout the project, I encountered numerous unexpected issues and had to find innovative solutions with the assistance of fellow team members. This internship allowed me to expand both my **skills** and **knowledge**, including learning to use **Gitlab**, **SystemVerilog**, or **Verilog-AMS** for instance.

Lastly, I enjoyed being an integral part of the company. This goes through **collaborative work**, engaging with colleagues, and learning from others. I realized the significance of the work environment in a job, and it will certainly be a crucial factor in my future job pursuits.

8 Annexes

8.1 Annex A - Script Shell

```
# Number of simulations to run, default value is 10
num_simulations=${1:-10}

# Output file to store simulations result
output_file="${parent_directory}/results.txt"

# Initialize variables to store the sums
sum_system_time=0
sum_user_time=0
sum_overall_time=0

# Run the simulations and extract CPU usage
for i in $(seq 1 $num_simulations); do
    echo "Running simulation $i ..."

    # Run the simulation and create a log file for each run
    runsim.sh --bench b2b --tc tc_m90396vAA_amplifiers_perfmetrics -grid_args "-m lsfnode04" > "${log_directory}/${log_file_pattern}${i}.txt"

    # Extract the CPU usage line from the log file
    cpu_usage_line=$(grep -E 'xmsim: CPU Usage' "${log_directory}/${log_file_pattern}${i}.txt")

    # Extract the clock time line from the log file
    wall_clock_line=$(grep -E 'elapsed time' "${log_directory}/${log_file_pattern}${i}.txt")

    # Extract from the lines
    system_time=$(echo "$cpu_usage_line" | awk '{gsub("s", "", $5); print $5}')
    user_time=$(echo "$cpu_usage_line" | awk '{gsub("s", "", $8); print $8}')
    overall_time=$(echo "$wall_clock_line" | awk '{print $(NF-1)}')

    # Add the times to their respective sums
    sum_system_time=$(echo "$sum_system_time + $system_time" | bc)
    sum_user_time=$(echo "$sum_user_time + $user_time" | bc)
    sum_overall_time=$(echo "$sum_overall_time + $overall_time" | bc)

    echo "Simulation $i completed. System time: $system_time s, User time: $user_time s, Overall time: $overall_time s">> "$output_file"
done

# Calculate the average system and user times
avg_system_time=$(echo "scale=1; $sum_system_time / $num_simulations" | bc)
avg_user_time=$(echo "scale=1; $sum_user_time / $num_simulations" | bc)
avg_overall_time=$(echo "scale=1; $sum_overall_time / $num_simulations" | bc)

# Write the summed times to the output file
echo "" >> "$output_file"  # Add a newline
echo "Total sum of system time: $sum_system_time s" >> "$output_file"
echo "Total sum of user time: $sum_user_time s" >> "$output_file"
echo "Total sum of overall time: $sum_overall_time s" >> "$output_file"

# Write the average times to the output file
echo "" >> "$output_file"  # Add a newline
echo "Average system time: $avg_system_time s" >> "$output_file"
echo "Average user time: $avg_user_time s" >> "$output_file"
echo "Average overall time: $avg_overall_time s" >> "$output_file"

echo "All simulations completed. Averages saved to $output_file."
```

Figure 54: Time measurement : Script Shell

8.2 Annex B - Gantt Chart

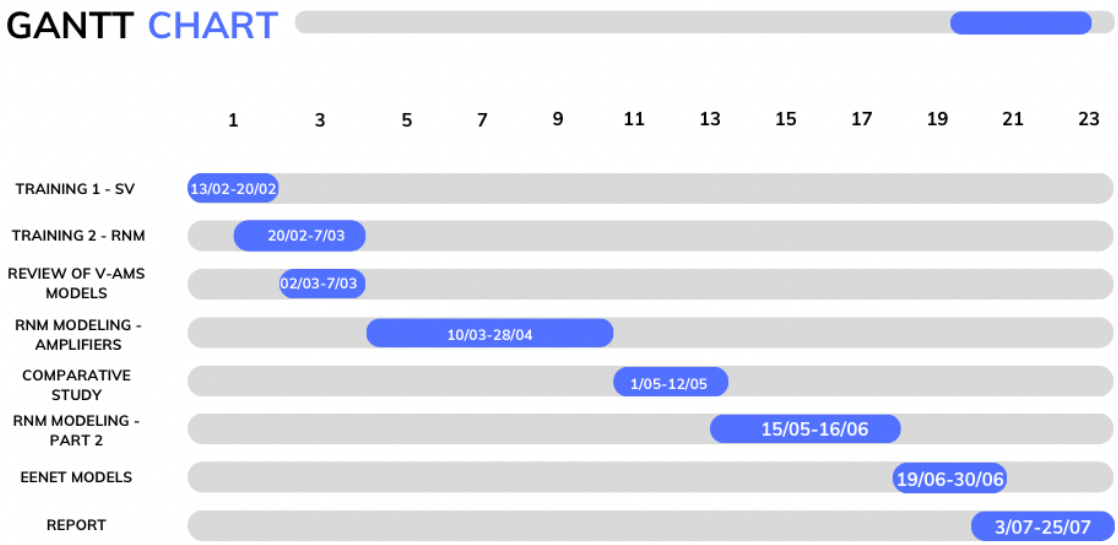


Figure 55: Gantt Chart

8.3 Annex C - Archive Card

Nano- Microtechnologies for Integrated Systems
2022-2023
Internship done February 13th - August 4th 2023

Melexis Supervisor :
El Hage Ali Raed
rel@melexis.com

Supervisor : Prejbeanu Liliana
Liliana.Buda@cea.fr
Student : Audoin Nicolas
Nicolas.Audoin@grenoble-inp.org

Project Description

The core objective of this internship is to develop high performance models for mixed signal simulations on a new Melexis' Magnetic Position Sensor. For this end, a review of the state-of-the-art is to be carried out, with focus on SystemVerilog Real Number Models (SV-RNM).

Responsibilities

- Learn a new hardware description Language System Verilog (training).
- Learn modelling techniques with System Verilog (training).
- Analyze the existing set of models (Verilog-AMS).
- Develop equivalent and new models in System Verilog Real-Number Models (SVRNM).
- Develop equivalent and new models in System Verilog Real-Number Models (SVRNM).
- Provide a comparative study (time and accuracy) between models in Verilog-AMS, SVRNM, and SV EEnets.
- Develop shell scripts to automate the simulations and extraction of performance metrics.

A complete access to the company's tool was granted : various software tools such as Gitlab, Cadence Virtuoso/SimVision/Xcelium, Polarion, textbook resources and help provided by every other MSDE team member when needed. Work space and equipments were also provided as for every other employee.

Melexis
INSPIRED ENGINEERING

Melexis Technologies SA
Chem. de Buchaux 38
2022 Bevaix SUISSE
https://www.melexis.com

List of Figures

Bibliography

[1] Nikolaos Georgoulopoulos and Alkiviadis Hatzopoulos. "Real Number Modeling of a Flash ADC Using SystemVerilog". In: (2017) (cit. on p. 6).

[2] *Using EEnet to perform Electrical Equivalent Modeling in SystemVerilog Rapid Adoption Kit (RAK)*. Learn more at Cadence Support Portal - https://support.cadence.com. Cadence Design Systems, Inc., 2022 (cit. on p. 10).

[3] Radhika Anand. *System Verilog EEnet (SV-EEnet) application: Modeling block currents in Mixed Signal Verification*. Feb. 2021 (cit. on p. 19).

[4] Wolfgang Scherr and Karsten Einwich. "Beyond real number modeling: Comparison of analog modeling approaches". In: (2020).

[5] Melexis. *Triaxis®: Unique sensing solution*. 2023. URL: https://www.melexis.com/en/tech-talks/triaxis-position-sensing-solution.

[6] Cadence Design Systems. *SystemVerilog Real Number Modeling (SV-RNM) Advanced Verification*. Cadence Training Course. URL: https://support.cadence.com/.

[7] Cadence Design Systems. *SystemVerilog for Design and Verification*. Cadence Training Course. URL: https://support.cadence.com/.

[8] Cadence Design Systems. *Mixed Signal Verification – Connectivity Overview*. Cadence Training Course. 2019.

[9] IEEE Computer Society. *IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*. IEEE Std 1800-2017. IEEE, 2017. URL: https://ieeexplore.ieee.org/document/8299595.

[10] Accellera Systems Initiative. *Verilog-AMS Language Reference Manual*. Accellera Systems Initiative, 2021.

[11] Wolfgang Scherr and Karsten Einwich. "Beyond Real Number Modeling: Comparison of Analog Modeling Approaches". In: ().

Abstracts

English

During my internship, I developed a high-performance library for the MPS MLX90396 using SystemVerilog and Real Number Modeling (SV-RNM) in a mixed-signal environment. The aim was to reduce simulation complexity and time for analog blocks. To do this, we replaced transistor-level models with behavioral models, capturing only the key functional characteristics of the blocks. For previous MPS, the MSDE team had developed Verilog-AMS behavioral models. However, these came with their own set of challenges, leading the team to switch to SV-RNM. The RNM method, using SystemVerilog's real datatype, allows for high-speed, event-driven simulations without needing an analog solver. With the team's assistance, we developed and tested models for most of the MLX90396 blocks. Lastly, I conducted a comparative study on execution and CPU times. The SV-RNM method proved to be 300 times faster than transistor-level simulation and 8 times faster than the previous V-AMS model, enhancing performance while maintaining accuracy.

Français

Au cours de ce stage, j'ai développé une bibliothèque haute performance pour le MPS MLX90396 en utilisant SystemVerilog et la modélisation en nombres réels (SV-RNM). L'objectif était de réduire la complexité et le temps de simulation pour les blocs analogiques. Pour ce faire, nous avons remplacé les modèles au niveau transistor par des modèles comportementaux, capturant uniquement les caractéristiques fonctionnelles clés des blocs. Pour les MPS précédents, l'équipe MSDE avait développé de tels modèles en Verilog-AMS. Cependant, certains points-noirs sur ceux-ci ont conduit l'équipe à passer au SV-RNM. La méthode RNM, utilisant les variables real de SystemVerilog, permet des simulations à haute vitesse et evenementielles sans avoir besoin d'un solveur analogique. Avec l'aide de l'équipe, nous avons développé et testé des modèles pour la plupart des blocs MLX90396. J'ai également mené une étude comparative sur les temps d'exécution et de CPU. La méthode SV-RNM s'est avérée être 300 fois plus rapide que la simulation au niveau transistor et 8 fois plus rapide que le modèle V-AMS précédent, améliorant les performances tout en maintenant la précision.

Italiano

Durante il mio tirocinio, ho sviluppato una libreria ad alte prestazioni per l'MPS MLX90396 utilizzando SystemVerilog e la modellazione con numeri reali (SV-RNM) in un ambiente di segnali misti. L'obiettivo era ridurre la complessità e il tempo di simulazione per i blocchi analogici. Per fare ciò, abbiamo sostituito i modelli a livello di transistor con modelli comportamentali, catturando solo le caratteristiche funzionali chiave dei blocchi. Per i precedenti MPS, il team MSDE aveva sviluppato modelli comportamentali Verilog-AMS. Tuttavia, questi presentavano una serie di sfide proprie, che hanno portato il team a passare a SV-RNM. Il metodo RNM, utilizzando il tipo di dati reali di SystemVerilog, consente simulazioni ad alta velocità e basate su eventi senza la necessità di un risolutore analogico. Con l'aiuto del team, abbiamo sviluppato e testato modelli per la maggior parte dei blocchi MLX90396. Infine, ho condotto uno studio comparativo sui tempi di esecuzione e di CPU. Il metodo SV-RNM si è rivelato essere 300 volte più veloce della simulazione a livello di transistor e 8 volte più veloce del precedente modello V-AMS, migliorando le prestazioni pur mantenendo la precisione.