# POLITECNICO DI TORINO

**MASTER's Degree in ELECTRONIC ENGINEERING**


**MASTER's Degree Thesis**

# Automotive radar processing optimization by exploiting the hardware accelerator of radar sensor chip

**Supervisors**

**Prof. MAGGIORA RICCARDO**

**Candidate**

**FRANCESCO BILETTA**

**OCTOBER 2021**

# Summary

In recent years automotive safety has become more and more important due to the high numbers of fatal accidents.

The trend is to automate and enhance the safety technologies in order to assist the driver and avoid dangerous situations. Advanced Driver-Assistance Systems (ADAS) can implement various features such as automatic emergency brake, blind-spot detection, automated highway driving and more. ADAS are based on radar technology since it can effectively detect and locate objects without being affected by low visibility or by bad weather conditions.

The most common type of radar for automotive applications adopts linear frequency-modulated continuous waveform and it provides range, velocity and direction of arrival of the detected targets. Several integrated single-chip radar sensors are available in the market, capable of operating in the 76- to 81-GHz band with unprecedented levels of integration in an extremely small form factor.

The TI AWR1843 device is a self-contained single-chip radar sensor that provides a 3TX, 4RX system with built-in PLL and A2D converters. It integrates the Digital Signal Processing (DSP) subsystem, a processor subsystem responsible for radio configuration, control, and calibration, an ARM R4F for automotive interfacing, and an hardware accelerator block to perform radar processing saving MIPS on the DSP for higher level algorithms.

The scope of this thesis is to optimize the performances of automotive radar application based on the TI AWR1843 by implementing the most demanding processing tasks on the hardware accelerator block instead of the general purpose DSP. The performances are evaluated in terms of the time required to execute the different tasks.

# Acknowledgements

This thesis would not have been possible without the support of many people.

Many thanks to my supervisor, Riccardo Maggiora, who, first of all, got me interested so much in the subject by making the classes absorbing and fun.
Secondly because he evaluated my exam based on my knowledge of the subject and not caring the missing weekly homeworks.
Lastly I wanted to thank him for accepting my thesis proposal and for helping me during the related work and writing phase.

I also wanted to thank all the radar team of the Politecnico's Electronic Department. In particular Mattia for guiding me in the approach to the radar world, Sara for letting me steal her desk and teaching me Matlab tricks, Stefano and Maurice for always being ready to help and have a friendly chat.

A huge thank you to my parents, who educated me with love and respect and who always supported me in every situation.
Thank you to my brother and my sister as well, who have always accompanied me in the journey of life.

Thank you to my lifetime friends Stefano, Giorgio, Fabrizio and Alessio, with whom I have spent countless memorable adventures from the top of the Monte Rosa to the most remote place of Europe.
Also thank you to my friend Hussein for making the experience in Madrid so fun and culturally rich.

Last but not least, a gigantic thank you to my lovely Matilda, who was close to me in difficult moments like the lockdown and with whom I happily shared the last 2 and a half years of my life.

*"Fortune favours the brave"*
*Cic.*

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**AoA**

Angle of Arrival

**ADAS**

Advanced Driver Assistance Systems

**ADC**

Analog to Digital Converter

**API**

Application Programming Interface

**ARM**

Advanced RISC Machines

**BIST**

Built-In Self-Test

**CAN**

Controller Area Network

**CFAR**

Constant False Alarm Rate

**CMOS**

Complementary Metal–Oxide–Semiconductor

**CRC**

Cyclic Redundancy Check

**DC**

Direct Current

**DMA**

Direct Memory Access

**DSP**

Digital Signal Processor

**DSS**

DSP Subsystem

**EDMA**

Enhanced Direct Memory Access

**FFT**

Fast Fourier Transform

**FMCW**

Frequency-Modulated Continuous Wave

**GPIO**

General-Purpose Input/Output

**HWA**

Hardware Accelerator

**I/O**

Input/Output

**I/Q**

In-phase and Quadrature

**I2C**

Inter-Integrated Circuit

**IC**

Integrated Circuit

**IF**

Intermediate Frequency

**IP**

Intellectual Property

**LFSR**

Linear-Feedback Shift Register

**LNA**

Low-Noise Amplifier

**LSB**

Least Significant Bit

**LVDS**

Low-voltage differential signaling

**MCU**

Microcontroller Unit

**MIMO**

Multiple-Input and Multiple-Output

**MSB**

Most Significant Bit

**MSS**

Master Subsystem

**PA**

Power Amplifier

**PC**

Personal Computer

**PLL**

Phase-Locked Loop

**RAM**

Random-Access Memory

**RISC**

Reduced Instruction Set Computer

**RF**

Radio Frequency

**ROM**

Read-Only Memory

**RX**

Receiver

**SDK**

Software Development Kit

**SFDR**

Spurious-Free Dynamic Range

**SPI**

Serial Peripheral Interface

**TDM**

Time Division Multiplexing

**TI**

Texas Instruments

**TX**

Transmitter

**UART**

Universal asynchronous receiver-transmitter

# Chapter 1

# Introduction to Automotive Radar

In recent years automotive safety has become more and more important due to the high numbers of fatal accidents. Since human error is one of the main causes, the trend is to automate and enhance the safety technologies in order to assist the driver and avoid dangerous situation.

This group of electronic technologies is called Advanced Driver-Assistance Systems (or ADAS) and it can implement various features such as automatic emergency brake, blind-spot detection, automated highway driving and more. Many of these safety features are based on radar technology since it can effectively detect objects without being affected by low visibility (for example at night) or by bad weather conditions such as rain and fog.

The most common type of radar for automotive applications is called Frequency-Modulated Continuous Wave (or FMCW) [1] and it includes the processing of the received signals to obtain range, velocity and angle of the detected targets.
As the name implies it transmits signals that are modulated in the frequency domain and the most used signal is called "Chirp", which has a linear modulation as function of time. The magnitude of the resulting wave is shown in Figure 1.1 while its frequency is represented in Figure 1.2.

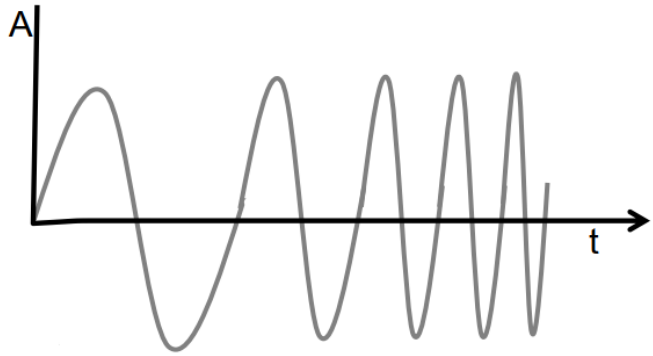


**Figure 1.1:** Chirp magnitude as a function of time.

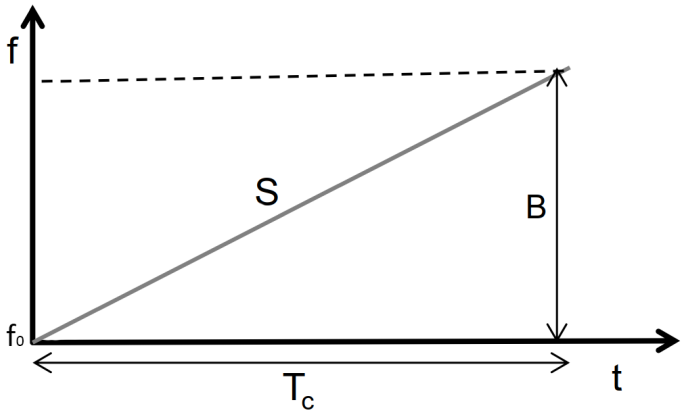


**Figure 1.2:** Chirp frequency as a function of time.

The basic work flow of this type of Radar starts with a synthesizer that generates the chirp with the parameters of choice, that includes: starting frequency ($f_0$), sweeping bandwidth ($B$) and time duration ($T_c$). The signal is transmitted through an antenna after being amplified. Another antenna is then needed to receive the reflected signal so that finally the TX and RX chirps can be mixed and properly filtered to obtain the difference of the frequencies of the two input sinusoids (called intermediate frequency or $f_{IF}$), as shown in Figure 1.3.

The purpose of computing the difference between the two signals is to determine the range of the detected object. In fact, in optimal conditions, the received signal is just the replica of the transmitted one shifted in time by $t_f = 2d/c$, where $d$ is the distance of the object (or range) and $c$ is the speed of light. This time is also called "round-trip delay". Since the TX and RX chirp have the same

linear modulation, the difference of the two will be a constant IF tone equal to the round-trip delay multiplied by the slope of the chirp ($S = B/T_c$). In this case, knowing the resulting frequency, the range of the object can be easily and accurately computed by inverting the formula and obtaining:

$$d = \frac{f_{IF} \cdot c}{2S}$$

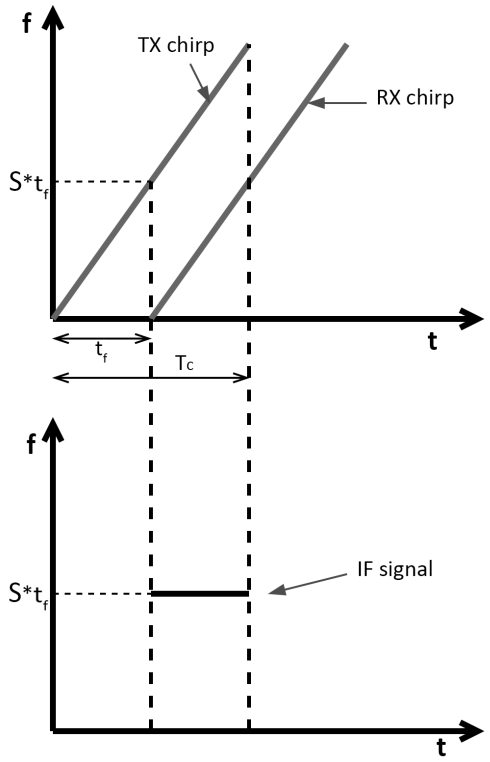


**Figure 1.3:** Result of TX-RX chirp mixing.

An important figure of merit in the design of a FMCW Radar is the range resolution. This is the metric that describes the ability to detect targets near to each other as distinct objects and it depends on the bandwidth of the chirp.

The Fourier transform theory tells that in order to resolve two different tones in the frequency spectrum, they have to be spaced more than $\frac{1}{T}$, being $T$ the window duration and in this case equal to $T_c$.

From this consideration the minimum resolution is obtained:

$$\Delta f > \frac{1}{T_c} \Longrightarrow \frac{2S\Delta d}{c} > \frac{1}{T_c} \Longrightarrow \Delta d > \frac{c}{2ST_c}$$

Since the slope of the chirp is equal to $S = B/T_c$ the range resolution is:

$$d_{res} = \frac{c}{2 \cdot B}$$

From this equation it can be seen that the higher the bandwidth the better the resolution. Infinitely small resolutions can not be achieved though due to the analog to digital conversion process that has finite sampling frequency ($F_s$). For this reason there is also a maximum achievable range that is computed as follows (the factor 2 is due to the Nyquist sampling theorem):

$$F_s \geq 2 \cdot \frac{2Sd_{max}}{c} \Longrightarrow d_{max} = \frac{F_s c}{4S}$$

After the sampling of the IF signal a Fast Fourier Transform (or FFT) is performed to obtain the frequency spectrum where the frequency of the peaks is proportional to the range of the detected objects.

In order to separate two different targets at the same distance from the FMCW Radar, their different relative velocities can be exploited. Since a transform in the frequency domain is performed, the signal is complex and that means that every value is a phasor with an amplitude and a phase.

Figure 1.4 shows that if the round-trip delay changes by a small amount $\Delta\tau$ there is a change in the phase of the computed FFT peak (in the example is the difference between points $C$ and $F$). Comparing the phases of the peak in two consecutive chirps, the relative velocity of one single object can be estimated.

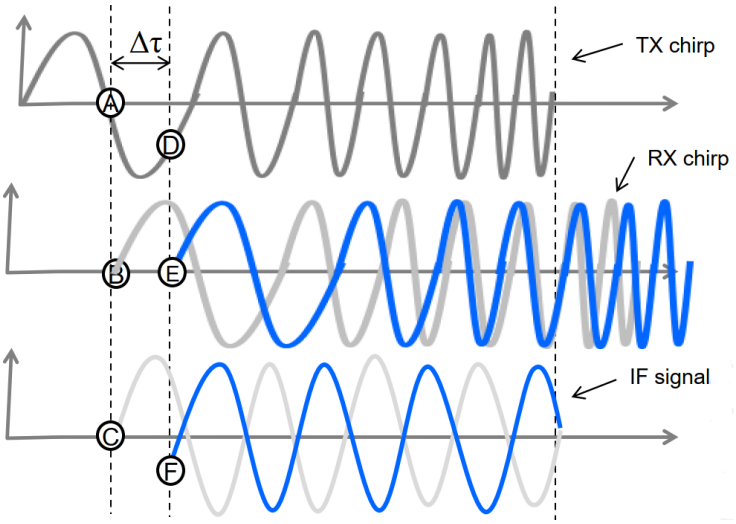


**Figure 1.4:** Phase difference between two slightly delayed chirps.

In order to estimate the velocities of different objects, more than two chirps are needed so generally $N$ equi spaced chirps are transmitted (this set is typically called a "frame"). Once the range FFT is done for every chirp of the frame, a second Fourier Transfor is done for every range "cell" to obtain the Doppler frequency spectrum where the peaks correspond to the velocity of the detected objects. This is called Doppler FFT.

An important figure of merit of the FMCW radar is the velocity resolution. This is the minimum separation between two velocities to be detected as two different peaks and it can be computed knowing that the phase difference $\Delta\Phi$ must be greater than $2\pi/N$, being $N$ the number of chirps in the frame.

$$\Delta\Phi = \frac{4\pi\Delta v T_{chirp}}{\lambda} > \frac{2\pi}{N} \Longrightarrow \Delta v > \frac{\lambda}{2T_{frame}}$$

Once the Doppler FFT is computed, a matrix with two dimensions, respectively range and velocity, is obtained.

The goal is now to detect two different objets at the same range and with the same relative velocity. For this purpose, it is necessary to have at least two receiving antennas. The so called "Angle of Arrival" (or AoA) estimation is based on the received phase shift between antennas due to different path length. Figure 1.5 shows how the path difference depends on the distance between the receiving components and the sine of the angle $\theta$. Since the former is known, the direction of arrival of the signal can be computed using the following formula:

$$\Delta\Phi = \frac{2\pi d sin(\theta)}{\lambda} \Longrightarrow \theta = sin^{-1}(\frac{\lambda\Delta\Phi}{2\pi d})$$

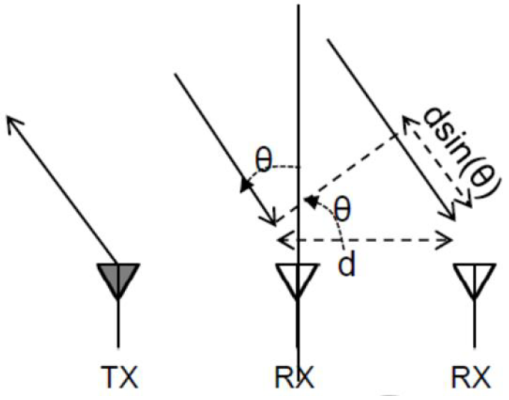


**Figure 1.5:** Path difference between two receiving antennas.

The peaks of the range-Doppler FFT will have different phases for different RX antennas so, in order to compute this phase shift and estimate the angle of the object, a third Fast Fourier Transform is needed. This one is called angle FFT and it is computed on the data of $N$ receiving antennas, so the bigger this number, the better the resolution.

The problem is that increasing the number of RX antennas has an high cost since each one has its own processing chain with related components such as low-noise amplifier, mixer, low-pass filter and analog to digital converter.

The solution to this problem is called MIMO, acronym that stands for Multiple Input Multiple Output, that is a technology that uses more than one antenna to transmit the signal and, by using orthogonal waveforms, gives an higher angle resolution using less antennas. In fact, taking as example a MIMO configuration with two TX and four RX, a resolution equivalent to 8 standard receiving antennas can be achieved.

To obtain this result though it is necessary to be able to distinguish between the different transmitted signals. There are various methods to accomplish this task and the one used in the implementation is now presented.
It is called Time Division Multiplexing (TDM) and it is based on transmitting chirps in different contiguous time slots through the different TX antennas as shown in Figure 1.6. This method is the easiest to separate the TX signals and therefore it is widely used.

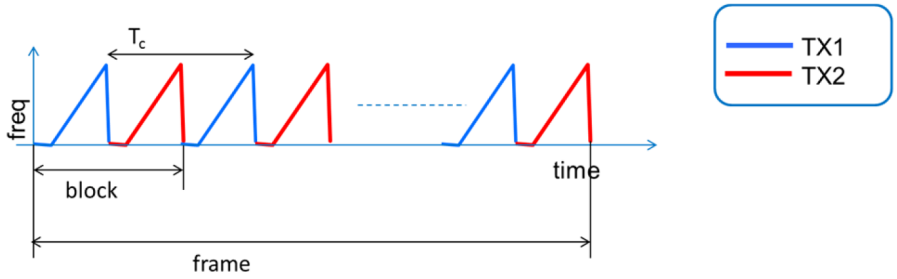


**Figure 1.6:** Time Division Multiplexing MIMO.

The concept is that for every virtual antenna (for example the combination of TX1 and RX1, the one of TX1 and RX2 and so on) a range-Doppler FFT is computed. A thresholding process is then executed to identify peaks corresponding to detected objects and Doppler compensation is applied since the time between chirps of different TX antenna introduces a phase shift.
The Angle FFT can now be performed across the values of all the virtual antennas

giving a higher angle resolution with limited costs.

# Chapter 2

# TI AWR1843 Automotive Radar System

## 2.1 Radar Sensor Chip

The AWR1843 belongs to the AWR1x family of single-chip mmWave radar sensors for Advanced Driver Assistance Systems, or ADAS applications. These devices are built with TI's low-power 45 nanometer RF CMOS-technology and offer high levels of integration in a compact dimension form.
These radar sensors provide a solution for low-power and self-monitored ultra accurate radar systems for advanced automotive applications, such as adaptive cruise control, automatic emergency brake, blind-spot detection, automated highway driving and more.

The AWR1843, in particular, is a complete single-chip radar that integrates analog and digital components, including multiple transmit and receive chains, PLL, ADCs, ARM Cortex-R4F MCU, C674x DSP, FFT accelerator, memories, and various I/O interfaces. This device also features continuous self monitoring and calibration of the RF and analog functionality to a dedicated built-in ARM R4F-based radio subsystem which is responsible for front-end configuration, control, and calibration. In Figure 2.1 is represented the functional block diagram of the chip.

The sensor is based on FMCW radar technology with capability to operate in the 76 to 81 GHz frequency range and supports chirp bandwidths of up to 4 GHz. Three transmit and four receive chains are provided for MIMO radar operation. Programmable and flexible chirp profiles are available to support multiple sensing profiles in the same radar frame.

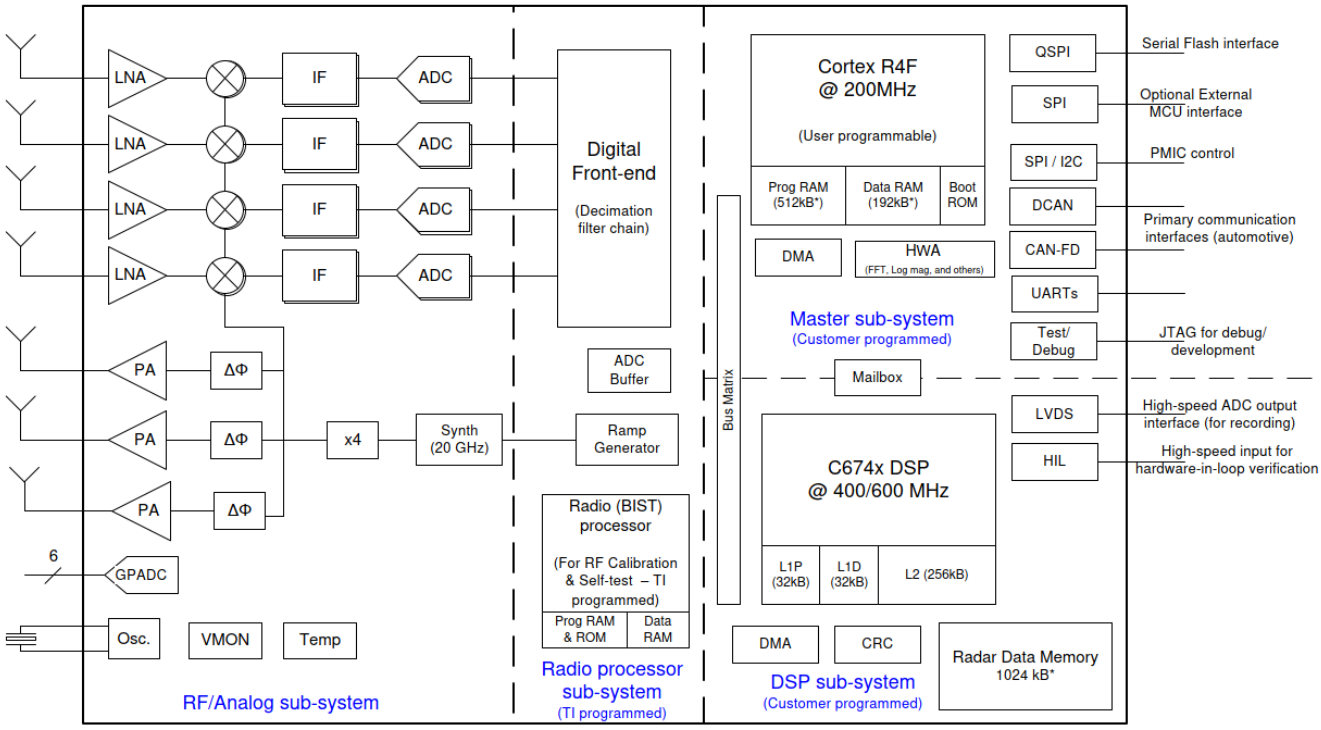


**Figure 2.1:** AWR1843 functional block diagram [2]

The chip provides an onboard hardware accelerator for FFT operations and Constant False Alarm Rate (or CFAR-based thresholding algorithms) [**CFAR**] and a full-featured C674x DSP code for FMCW signal processing and advanced algorithms, such as clustering, tracking, and object classification.

There are two ARM Cortex R4F MCUs running at 200 MHz. One of these is locked and used by the radio subsystem for calibration and monitoring. This R4F is programmed through firmware provided by TI and is not available for user code. However, the second R4F is available for high-level application processing. The device supports various industry standard input-output interfaces, such as CAN, SPI, I2C, UART, and also high-speed raw ADC data output using LVDS.

The typical FMCW processing chain is represented in Figure 2.2.
The receiver chain starts with the RF front end receiving the reflected radar signal, which is mixed with the transmitter signal to generate a intermediate frequency signal that is delivered to the ADC. The ADC converts the analog signal to digital samples which are pre-processed for digital processing.
Successive FFTs are computed on the digitized samples for range, velocity, and

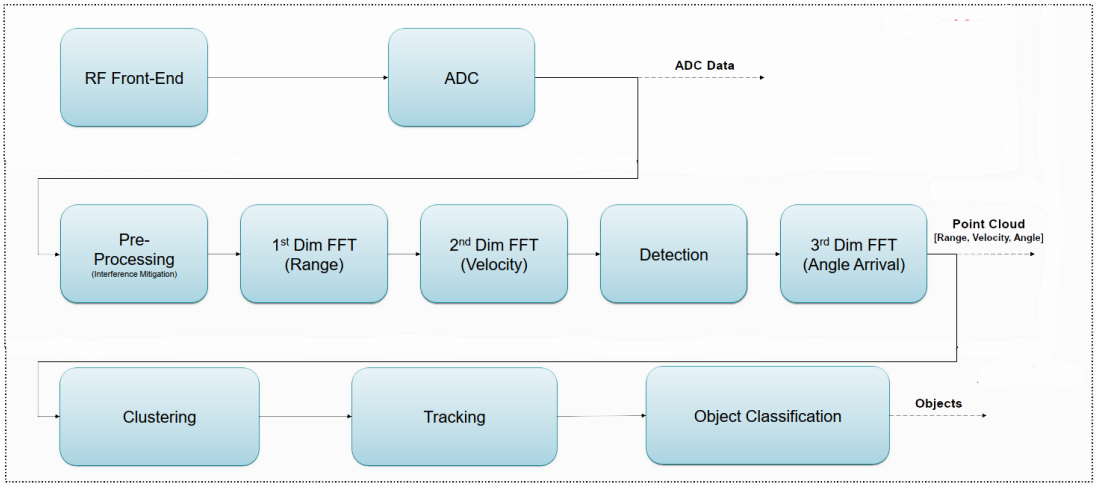


**Figure 2.2:** Signal processing chain.

angle of arrival calculation. The radar hardware accelerator can be used to offload these FFTs and perform thresholding processes to get a point cloud output.

More complex algorithms can be then performed to obtain other informations such as the path of a detected target, the type of the object and so on.

The device architecture can be divided into the following main blocks: the RF (or analog) subsystem, the radar subsystem, the master subsystem and the DSP subsystem.

### Analog subsystem

The analog subsystem includes the RF and analog circuitry used to generate, transmit and receive the signals. It consists of various components such as the synthesizer, the PA, LNA, mixer, IF, and ADC. This subsystem also includes the crystal oscillator and temperature sensors.
The RF and analog subsystem can be divided into three subcomponents, namely, the clock subsystem, the transmit subsystem, and the receive subsystem.

The clock subsystem generates 76 to 81 GHz frequency from an input reference of 40 MHz crystal. It has an inbuilt oscillator circuit, followed by a clean-up PLL and an RF synthesizer circuit.
The output of the RF synthesizer is then processed by a 4x multiplier to create the required frequency in the 76 to 81 GHz spectrum. The output is modulated by the timing engine block in order to create the required waveforms for effective

sensor operation. The timing engine is highly flexible and is programmed via the R4F-based radio controller subsystem.

The clean-up PLL also provides a reference clock for the host processor after system wake-up. The clock subsystem also has built-in mechanisms for detecting the presence of a crystal and monitoring the quality of the generated clock.

The transmit subsystem consists of three parallel transmit chains. Each transmit chain has independent phase and amplitude control.
A maximum of two transmit chains can be operational at the same time. However, all three chains can be operated together in a time-multiplexed fashion. The device also supports binary phase modulation for MIMO radar and interference mitigation.

The receive subsystem, instead, consists of four parallel channels where each receive channel consists of an LNA, a mixer, IF filtering, analog to digital conversion and decimation. All four receive channels can be operational at the same time. Individual power-on option is also available for system optimization.

Unlike conventional real-only receivers, AWR1843 radar sensor supports a complex baseband architecture which uses quadrature mixer and dual IF and ADC chains to provide complex I/Q output for each receiver channel. The bandpass IF chain has configurable lower cut-off frequencies, about 350 KHz and the continuous time sigma delta ADC supports bandwidths of up to 15 MHz.

**Radar subsystem**

The radar processor is actually a second dedicated ARM Cortex-R4F microcontroller running at 200 MHz. It includes the Digital Front-End and the ramp generator, but it is not available for customer application. In fact, this process is programmed by TI and takes care of out-of-calibration self-test and monitoring functions.

User applications running on the master subsystem do not have direct access to the radar system: the master system accesses the radar through well-defined API messages which are sent over hardware mailboxes. This interface is also known as the mmWaveLink and the functions to use it are included in the mmWave SDK.

**Master subsystem (or MSS)**

The master subsystem includes an ARM Cortex-R4F processor clocked at 200 MHz for running user application code. User applications executing on this processor control the overall operation of the device, including radar control via API messages,

radar signal processing, which is assisted by the radar hardware accelerator, or DSP, and peripherals for external interface.

A Quad Serial Peripheral Interface (QSPI) is available and it can be used to download customer code directly from a serial flash. A CAN interface is included that can be used to communicate directly from the device to a CAN bus, very common in automotive applications. An SPI/I2C interface is available for power management IC control. For more complex applications, the device can operate under the control of an external MCU which can communicate with the device over SPI interface.

**DSP subsystem (or DSS)**

The DSP subsystem contains TI's high-performance C674x DSP for FMCW signal processing, including FFT, CFAR thresholding and also advanced radar signal processing. This allows the AWR1843 to serve as a complete single-chip radar with advanced capabilities for clustering, tracking, and object classification.

The DSP Subsystem also contains a high-bandwidth interconnect for high performance (128-bit, 200MHz) and associated peripherals, LVDS interface for measurement data output, L3 Radar data cube memory, ADC buffers, CRC engine and data handshake memory.

Both the master and the DSP subsystems can communicate with the radar hardware accelerator, which is used to offload certain frequently used computations and FMCW radar signal processing from the processors. It will be discussed in detail in Section 4.

## 2.2 Development board

To exploit the radar sensor chip functionalities Texas Instruments provides an Evaluation Board called AWR1843BOOST.

The AWR1843 BoosterPack from Texas Instruments is an easy-to-use evaluation board for the AWR1843 mmWave sensing device, with direct connectivity to the microcontroller (MCU) LaunchPad Development Kit. The BoosterPack contains everything required to start developing software for on-chip C67x DSP core and low-power ARM R4F controllers, including onboard emulation for programming and debugging as well as onboard buttons and LEDs for quick integration of a simple user interface [3].

In Figure 2.3 is represented the front view of the Evaluation Board, where the main components, the switches and the buttons are placed.

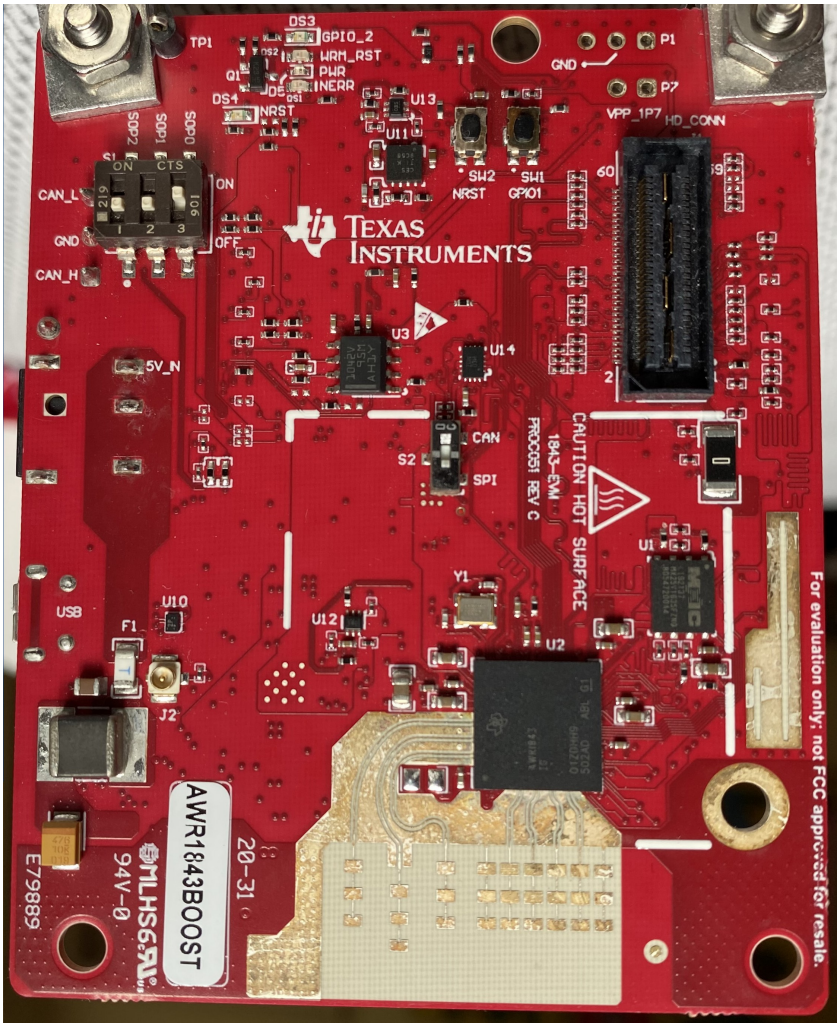


**Figure 2.3:** Evaluation board front view.

AWR1843 sensor chip is placed in the lower part of the board, directly connected to the antennas. The chip and the onboard-etched antennas are represented in detail in Figure 2.4, which also shows the layout parameters.
The Evaluation Board provides antennas for the four receivers and three transmitters that enable tracking multiple objects with their distance and angle information. This antenna design enables estimation of the angle of arrival for both azimuth and elevation, detecting objects in a three-dimensional plane.

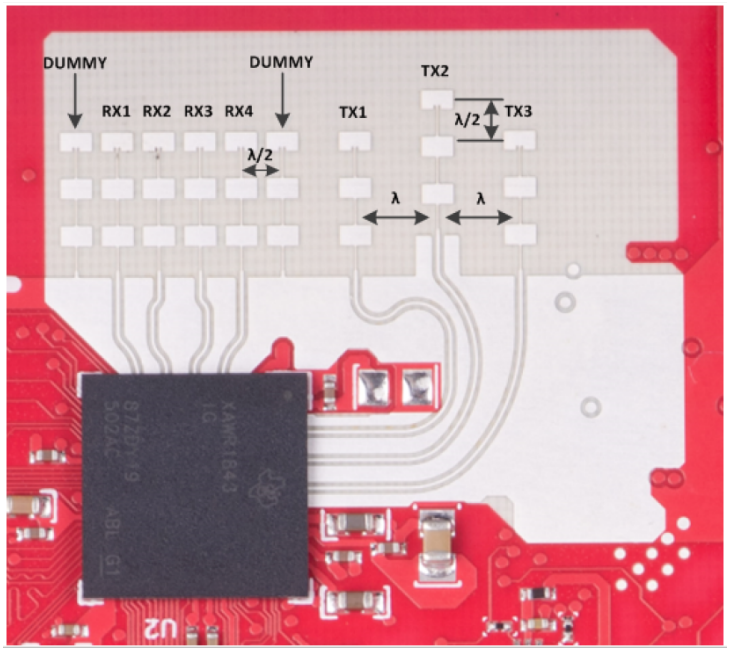


**Figure 2.4:** Chip and antennas detail. [3]

With regard to the switches the most important are related to the outside communication. One is used to select the interface protocol between SPI and CAN-bus, since they are muxed on the same lines. This switch can be seen in the central part of the board (Figure 2.3).
A set of switches is responsible for setting the operation mode of the chip. The possible choices are: debug mode, flash programming and functional mode.
This set is placed in the upper-left part of the Evaluation Board.

There are two buttons to interact with the AWR1843BOOST. One is responsible for resetting the system while the other is used as a general purpose input (GPIO).

On the rear part of the board the most important components are the two 20-pin LaundPad connectors that leverages the ecosystem of the TI LaunchPad, the 5-V power jack to power the board and the micro-USB port used to both program the chip and communicate via UART.

# Chapter 3

# Software starting point

In this chapter the software from which my implementation work started is presented.

As explained in the previous Section, the chip has two cores to be programmed (the ARM Cortex R4F and the C674x DSP) and, in this application, they are managed without operative system. The communication between the two is accomplished using an internal Mailbox and semaphores system that uses just 2 KB of memory.

The ARM is in charge of the external communication and it uses the SPI interface in order to have a good bandwidth for data and commands. For the PC interface a KSZ8851SNL chip is used to convert SPI to Ethernet protocol. The DSP core, instead, is used to perform all the calculations needed for radar processing such as FFT and CFAR thresholding algorithms.

With regard to the analog part of the AWR1843, 4 RX and 3 TX are used and, being the layout of the transmitting antennas not aligned (as shown in Figure 2.4, azimuth and elevation angles can be estimated. To better the angle resolution a time multiplexing MIMO configuration is exploited, providing a total of 12 virtual antennas.

## 3.1 DSP operations

To get an overview of the software the two main processing functions of the DSP are reported:

- *dssInitTask*:
    1. Initializes Mailbox and semaphores.

2. Initializes the MMWaveLink which is a set of functions to generate the messages needed to configure the BIST via Mailbox.
3. Launch *nonOsLoop*

- *nonOsLoop*: during the loop the following functions are called:
  1. *mboxReadTask*: handles mailbox messages received from MSS such as the values for the configuration of the peripherals or sensor start and stop commands.
  2. *Event_checkEvents*: checks if any of the events listed in Table 3.1 has been received from MSS. If so the function *dssDataPathTask* handles it performing the correct operations.
  3. The loop also checks for the presence of radar_start and radar_stop commands.

| EVENT NAME | POSTED BY | FUNCTION |
|---|---|---|
| **bss_stop_complete_evt** | DSS after receiving radar_stop command. | It stops the sensor with the *StopRadar* function. |
| **datapath_stop_evt** | DSS when BIST stop is completed. | It moves DSS into the stop state and communicates it to the MSS. |
| **framestart_evt** | BIST firmware. | It indicates the beginning of the radar frame. |
| **chirp_evt** | BIST firmware. | It indicates that the ADCbuffer is filled with samples. |
| **config_evt** | DSS after receiving mss2dsscontrol_cfg message. | It configures the DSS datapath and start the operations. |

**Table 3.1:** Events handled by DSS

The *dssDataPathTask* operations which need to be highlighted are:

- If **config_evt** is detected *dssDataPathConfig* is called. This function allocates the memory space needed for processing, configures EDMA transfers, configures ADC buffers and updates datapath variables based on the current configuration.

- If **chirp_evt** is detected *dssDataPathProcessEvents* is called. This function performs all the processing such as range, Doppler and angle FFTs, CFAR detection and XY coordinates computation. When interframe processing is completed it transfers the output data to MSS which will send them to the PC interface.

The states in which the DSS can be are the following:

1. **DSS_STATE_INIT**: starting idle state.
2. **DSS_STATE_STOP_PENDING**: preparing for stop state.
3. **DSS_STATE_STOPPED**: stop state.
4. **DSS_STATE_STARTED**: operating state.

The DSS state diagram is shown in Figure 3.1.

## 3.2 DSP datapath processing

The general datapath processing chain is shown in Figure 3.2.

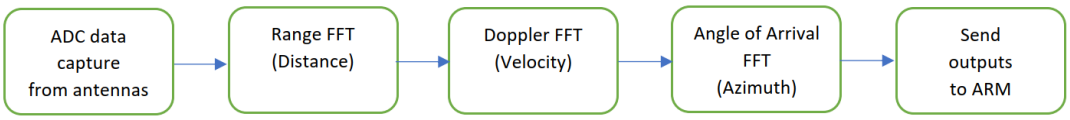


**Figure 3.2:** Datapath processing chain.

The DSP has two ADC buffer of 32 KB each that can be used in ping-pong mode so that the processing is done in parallel with the reception of data. In these buffers the samples of each RX antenna are stored consequently. When one of the two is full a **chirp_evt** is sent and the computation begins while the other one get filled with new data. The ADC buffers become full after a number of chirps that depends on the size of the single chirp.

The first function called after the **chirp_evt** is *chirpProcess* in the dss_data_path.c file and the following operations are executed:

- Data are EDMA transferred from ADCbuffer, which is stored in **L3** memory, to the ADCdataIn buffer in **L1** memory of the DSP (which is much faster than **L3**) alternating even and odd RX antennas in ping-pong manner.
- A Blackman window is applied to the samples for the range FFT.

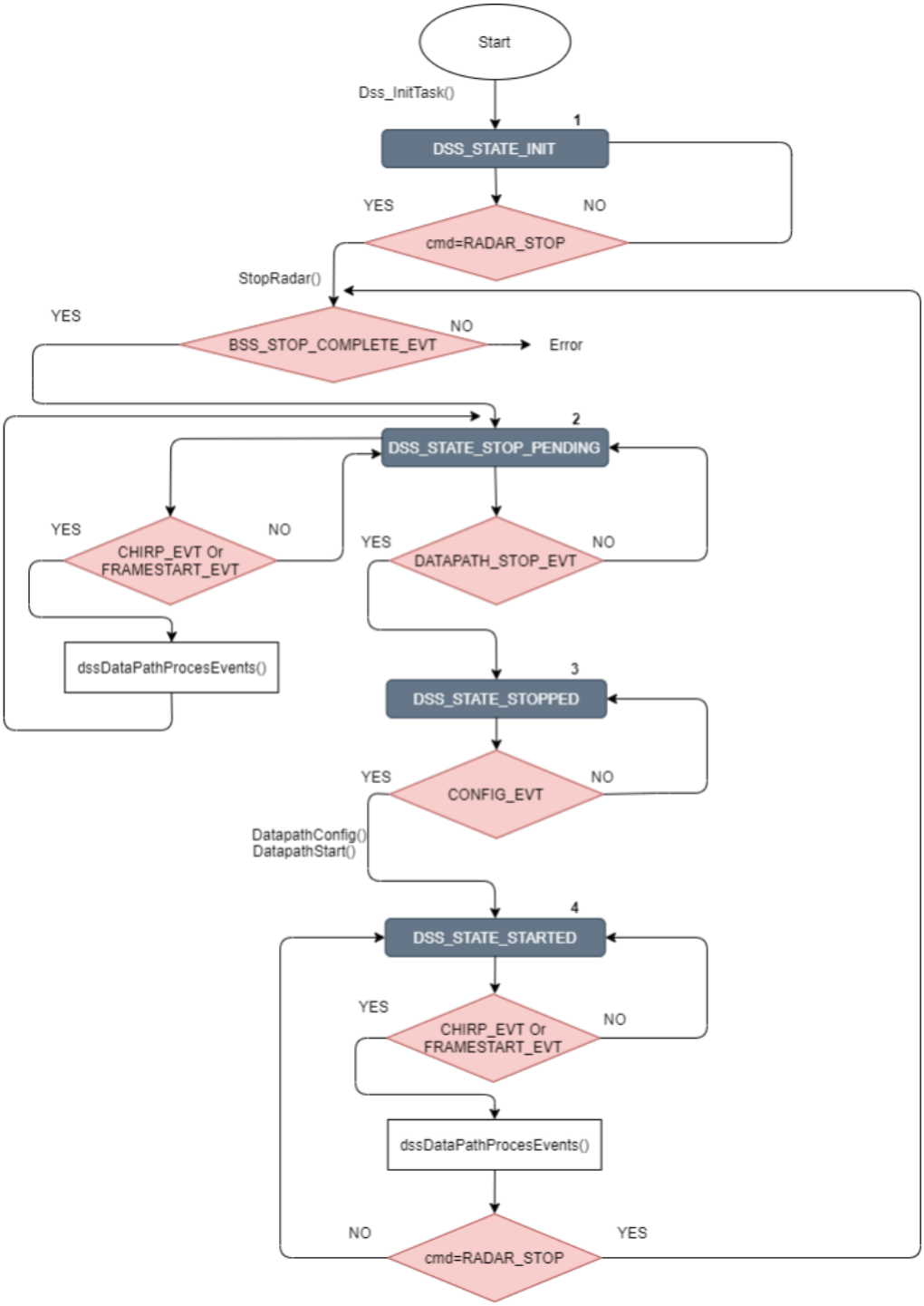


**Figure 3.1:** State diagram

- First dimension (range) FFT is performed on the samples.
- The results of the computation are EDMA transferred in **L3** memory (that has more space than **L1**) in the so called RadarCube matrix, where all the values are stored for each virtual antenna, each chirp and each range bin.

As said these operations are done every time an ADC buffer is full and they continue until every chirp of the frame has been processed. In Figure 3.3 the schematic of the first dimension processing is represented.

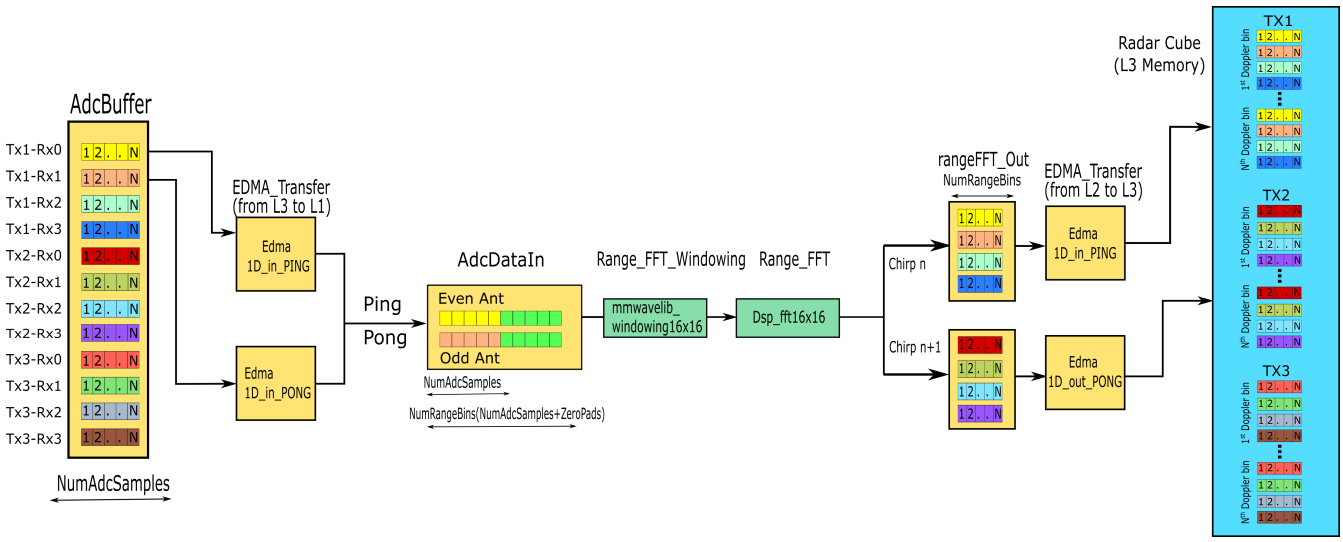


**Figure 3.3:** First dimension datapath schematic.

When all the chirps of the frame have been processed and stored in the RadarCube matrix, all the other computations can start. These operations are referred to as the interframe processing because they are computed in the time between the finish of a frame and the start of the other. The following calculations are executed:

- FFT for velocity detection (second dimension processing).
- CFAR thresholding algorithm in both Doppler and range direction.
- FFT for Angle of Arrival detection.
- XY coordinates of detected objects.

Interframe processing can be divided in two main parts: velocity-targets detection and Azimuth-XY detection.

The former is represented in Figures 3.4-3.5 and it includes the following operations:

- Data are EDMA transferred from RadarCube in **L3** memory to dstPingPong buffer in **L1** memory alternating odd and even chirps in ping-pong manner.
- An Hanning window is applied to the second dimension FFT samples.
- Doppler FFT is performed.
- Magnitudes of the computed values of all the virtual antennas are summed in log2 basis for each Doppler bin.
- Outputs are EDMA transferred in **L3** memory to the DetMatrix, the matrix where each range and Doppler bin values are stored.
- CFAR threshold in Doppler direction is applied for each range bin.
- If an object pass the threshold at a certain Doppler line, a bit corresponding to its range and Doppler is set in the DopplerLineMask matrix in **L2** memory.

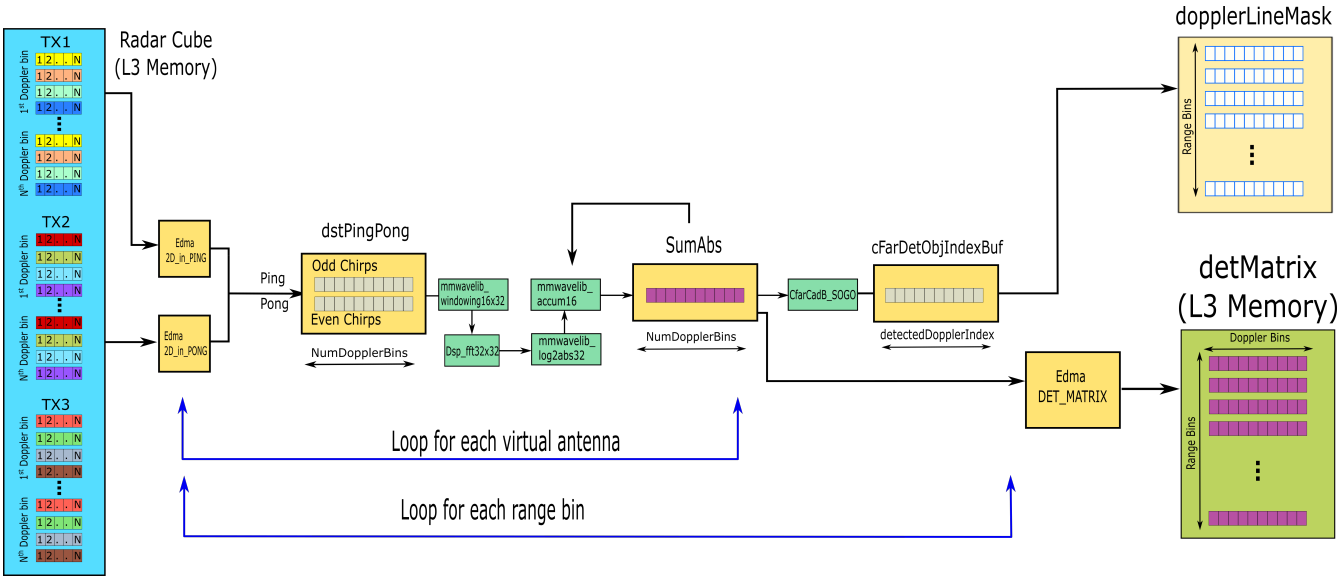


**Figure 3.4:** Second dimension and Doppler CFAR datapath schematic.

- CFAR threshold in range direction is applied for each Doppler line which have passed the previous threshold.
- Peak grouping is performed on groups of near indices both in range and Doppler direction.

- For each detected object range index, Doppler index and magnitude value are saved in a detObj2D structure, that will also be used for storing XY coordinates.

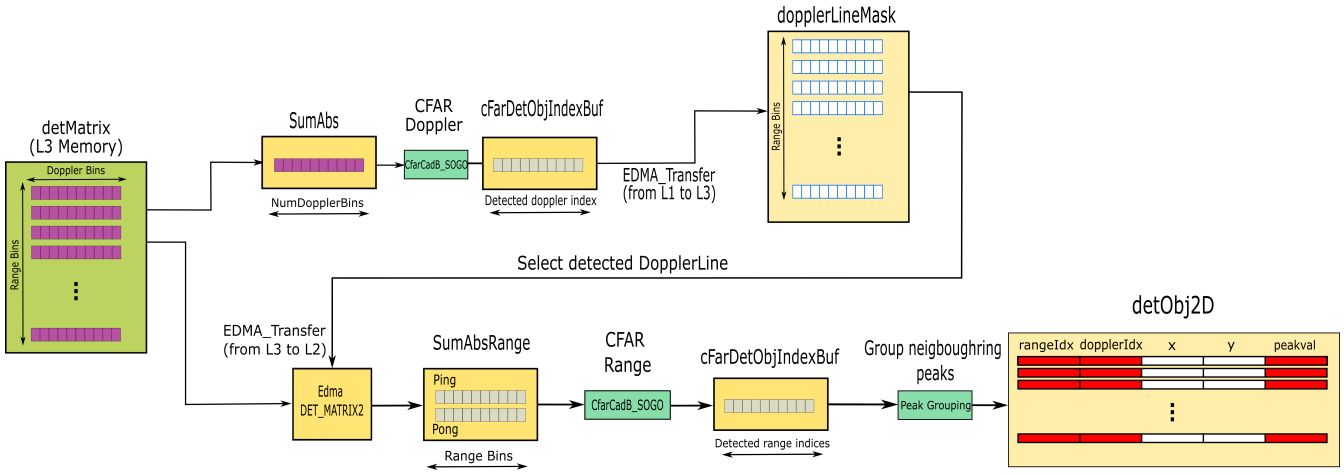


**Figure 3.5:** Range CFAR and peak grouping datapath schematic.

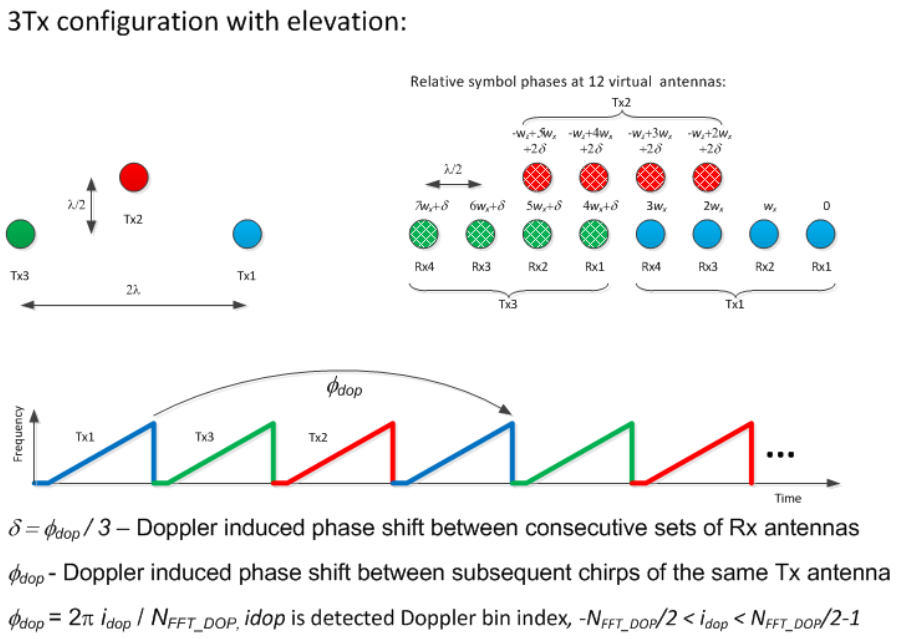


**Figure 3.6:** Doppler compensation for TX2 and TX3 before azimuth FFT.

As mentioned in Section 1, a phase compensation is needed because successive chirps of different TX antenna are separated in time.
This phenomenon is explained in Figure 3.6.

Azimuth and XY detection are represented in Figure 3.7.
The following operations are executed:

- Data of the detected objects are EDMA transferred from RadarCube in **L3** memory to dstPingPong in **L1** memory.
- Single point Discrete Fourier Transform (32 bits precision for both real and imaginary part) with windowing is performed at the detected range indices. This DFT is done because the results of the Doppler FFT are stored in magnitude with precision of 16 bits.
- Doppler compensation is performed before azimuth processing for the second and third TX antennas following the formula reported in Figure 3.6.
- Phase compensation for each virtual antenna:
  The correction coefficients for each antenna are computed with a MATLAB®script (Calibration.m):
  - Input: azimuthFFT_input after Doppler compensation (8 values).
  - Output: real and imaginary part of the correction factors which need to be copied into the compensation table in *AWR1843_MRR_mss/cli.c*.

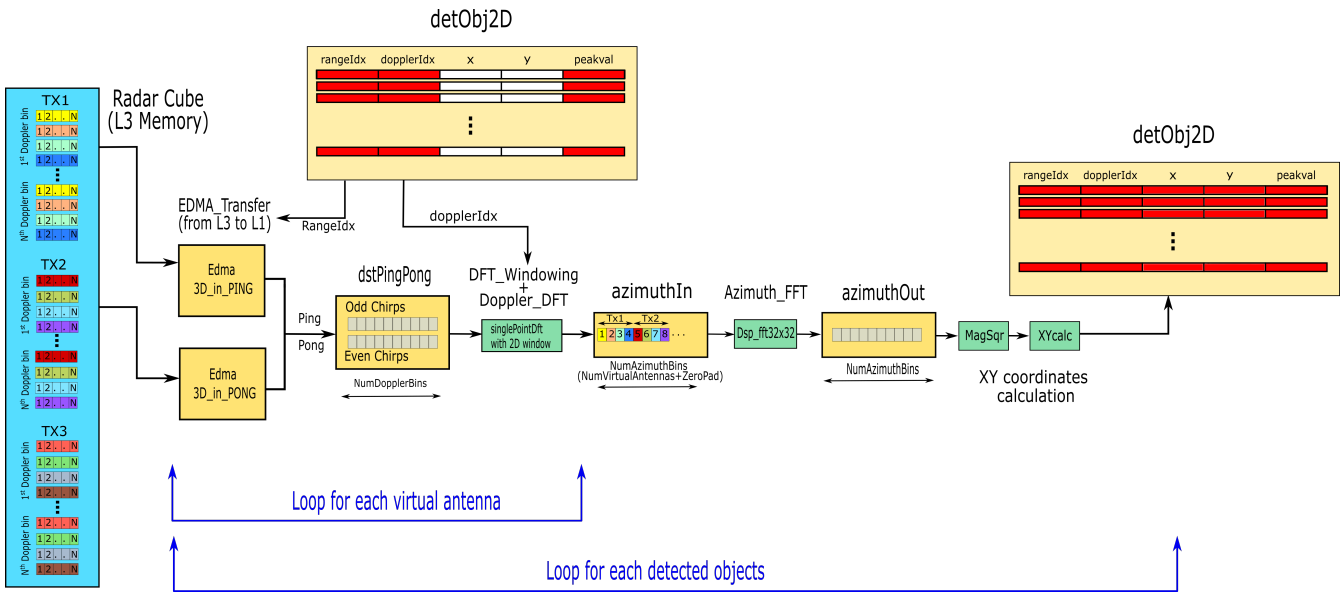


**Figure 3.7:** Azimuth and XY calculation.

- Azimuth FFT is performed on the samples.
- Magnitude square and X, Y coordinates are then calculated and stored in the detObj2D structure.

Once the interframe processing is complete the data are sent to the ARM processor (MSS) via Mailbox. The format of the messages from the DSP to the ARM is as follows:

- Message Type (4 bytes).
- Preamble (6 bytes).
- Doppler zero plot from the DetMatrix(2 bytes*NumRangebins).
- CFAR range threshold at Doppler zero (2 bytes*NumRangebins).
- Doppler FFT (2 bytes*NumDopplerbins).
- Doppler CFAR (2 bytes*NumDopplerbins).
- Detected Objects (Max 15, values preset to 0XFF) with the following structure:
  - Rangeidx (0 to NumRangebins) (2 bytes): to obtain the correct range value a multiplication with the range resolution is needed.
  - Doppleridx (from −NumDopplerbins to NumDopplerbins−1) (2 bytes): to obtain the correct velocity value a multiplication with the velocity resolution is needed.
  - PeakVal (2 bytes): this value is then converted in dB.
  - X position in Q Format $\longrightarrow X_{meter} = X >> Q_{format}$ (2 bytes).
  - Y position in Q Format $\longrightarrow Y_{meter} = Y >> Q_{format}$ (2 bytes).
- Matrix part counter (2 bytes).
- Matrix part address (4 bytes).

## 3.3 ARM operations

The MSS core executes the following tasks:

- *mssInitTask*: This function is called in the *main* and it is a one-time initialization task that executes the following operations:
  1. Initializes GPIO, SPI, DMA, Mailbox and Semaphores.

2. Performs synchronization.
3. Launches *nonOsLoop*.

- *nonOsLoop*: This is the main function and it replace the operative system. Its tasks are:

  1. If the DSP is stopped it checks if commands have been received by the KSZ8851SNL board.
  2. If a command is received and it corresponds to one of the the following, it serves the request:
     - cfg_cmd $\longrightarrow$ *rcvParameterfromcli* and *configureDSS* functions are called.
     - start_cmd $\longrightarrow$ *configureDSS* and *sensorStart* functions are called.
  3. It checks if a message is arrived in the Mailbox from DSS.
  4. If data are ready it sends them through SPI. The message type for this action is: DSS2MSS_DATA_READY_FOR_PLOT.
  5. After data have been sent it checks if a command has been received by the KSZ8851SNL.
  6. If the command stop_cmd is received it calls the function *stopProcedure*.

- *ConfigureDSS*: sets the DSP parameters.
- *rcvParameterformcli*: updates user parameters.
- *sensorStart*: sends configuration to DSP through mailbox.
- *stopProcedure*: stop the DSP operations so that commands can be received.

# Chapter 4

# Hardware Accelerator description

The Radar Hardware Accelerator (HWA) is an hardware IP that enables off-loading the burden of certain frequently used computations in FMCW radar signal processing from the main processor. The most common are FFT and log-magnitude, as explained in chapter 1, to obtain a radar image across the range, velocity, and angle dimensions. An advantage of using the Hardware Accelerator is that it doesn't affect the flexibility of implementing other propritary algorithms in the main processor.

## 4.1 High level architecture

An overview of the Hardware Accelerator architecture is represented in Figure 4.1.

As shown in Figure 4.1 the HWA has four local memories of 16KB each called ACCEL_MEM0, ACCEL_MEM1, ACCEL_MEM2, and ACCEL_MEM3 that, for convenience, will be referred to as **M0**, **M1**, **M2** and **M3**. The processing block is called accelerator engine and it includes different sub-modules that will be presented and explained in Section 4.2.

The Radar Hardware Accelerator is connected to the processor via a 128-bits bus that is used to transfer from and to the main memories. At the same bus is also connected the ADC buffer, which receives the samples directly from the Digital Front End.

The Hardware Accelerator has an operating clock frequency of 200 MHz that

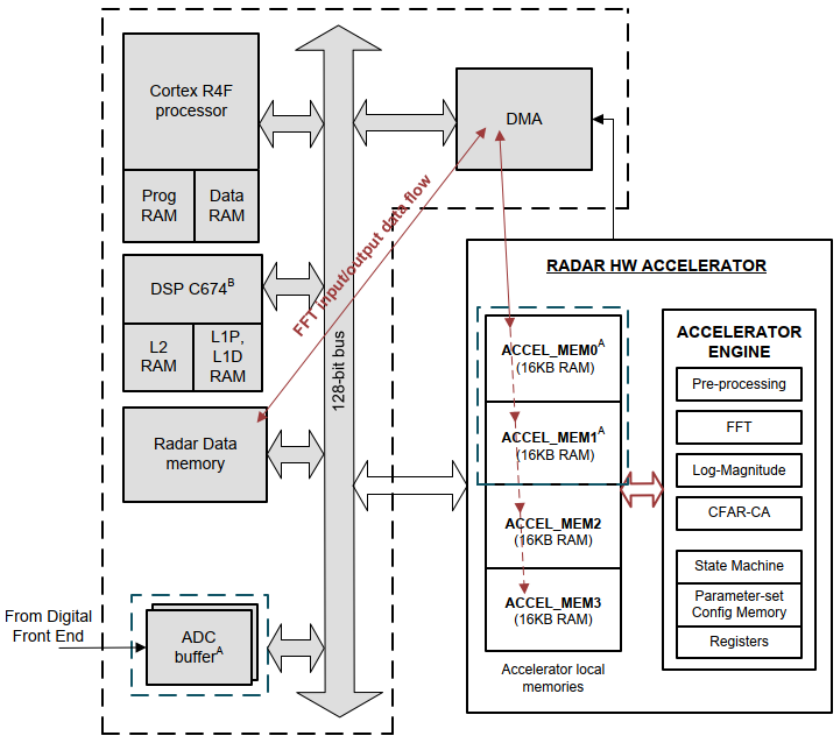


**Figure 4.1:** General HWA architecture. [4]

is a third of the DSP speed but, since the algorithms are more efficient and optimized, the performance is comparable if not better.

The typical data flow is that the samples are transferred from the memory of the main processor to the HWA, the accelerator engine executes the operations and then the results are sent again to the main memories.

A feature of the Hardware Accelerator is that the first two local memories (namely **M0** and **M1**) can be shared directly with the ADC buffers such that the samples for the first dimension FFT are immediately available at the end of each chirp. This property is useful since it saves an heavy DMA transfer (of all the unprocessed samples from main memory to the HWA local memory).

The purpose behind the four separate local memories inside the HWA is to enable the ping-pong mechanism, for both the input and output, such that the DMA read and write operations can happen in parallel to the main computational processing of the accelerator. So, for example, while the accelerator engine performs calculation using **M0** as input memory and **M2** as output, the DMA can write the samples of the following operation in **M1** and read the results of the previous one from **M3**.

The only limitation is that a single local memory cannot be accessed by the DMA and the HWA at at the same time as this would produce an error.

Since the DMA bus is 128-bits wide, the accelerator local memories are implemented to have words of 16 bytes so that the transfers are more efficient (they can reach the maximum throughput of 128 bits per clock cycle).

The use of the local memories is very flexible since any of them can be used as source or destination of the computation, with the only exception that the input and output memory cannot be the same 16KB bank. Also, the ping-pong mechanism is not compulsory so for example 32KB of data can be processed at a time using two banks as source and the other two as destination. This is possible because the address space for the HWA local memories is contiguous.

To configure the operations of the Radar Hardware Accelerator there is a 512-byte RAM that contains the so called *parameter-sets*, which are registers used to pre-program every aspect of the computations so that the HWA can then read one after the other and perform the calculations without the need of DSP intervention. This mechanism will be explained in detail in the following Section.

## 4.2 Accelerator Engine

The detailed block diagram of the Radar Hardware Accelerator is presented in Figure 4.2. As shown, there are two main blocks that constitute the HWA: the four local memories and the accelerator engine.

The Accelerator Engine consists of the following sub-modules:

- State machine: this module controls the operations of the whole HWA. In particular it is responsible for starting, looping and stopping the computations, for the triggering system and for the handshake mechanism with the main processor or the DMA. It is configured by the parameter-set and it executes the programmed tasks like sequencing and chaining various accelerator operations.
- Parameter-set configuration memory: as mentioned it is a RAM that contains registers used to pre-configure the operations of the HWA. The size of this memory is 512 byte and it contains 16 programmable parameter-sets of 32-bytes each. Furthermore there is a group of registers called static (or common) that are used to configure the basic settings of the state machine.
- Input formatter: the purpose of this operational block is to read the samples from the designed source memory and, after a flexible manipulation of the

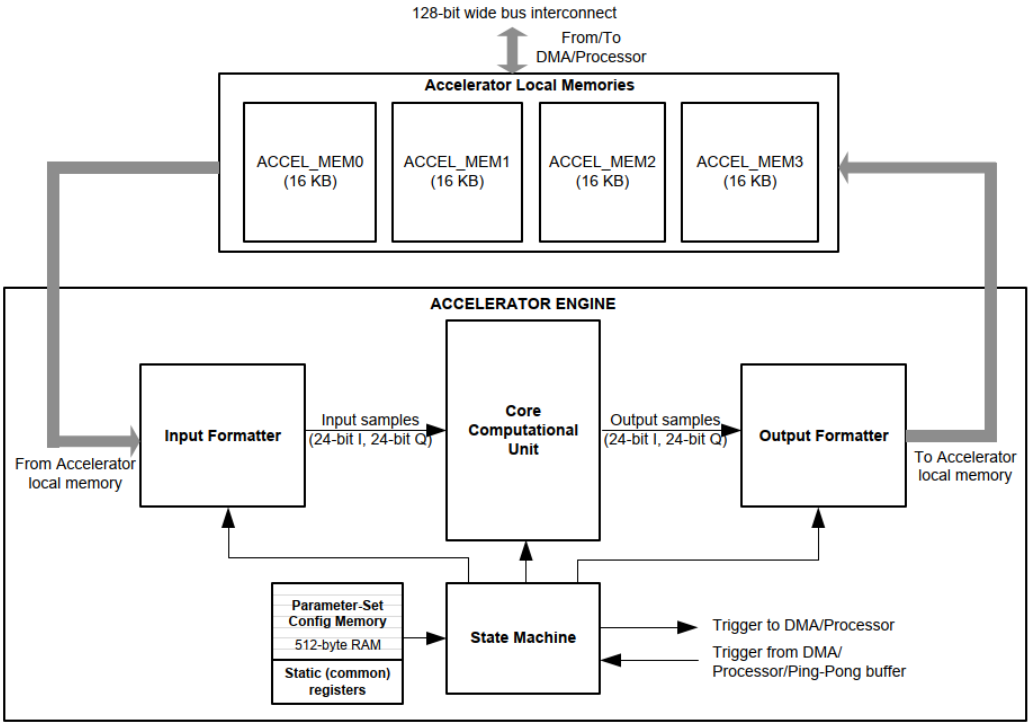


**Figure 4.2:** Detailed HWA architecture. [4]

data (like for example scaling, sign extension, different alignment etc.), to feed them to the core computational unit in a 24-bit complex format.

- Core computational unit: it is the module that contains the logic to perform different common radar-related operations such as CFAR thresholding, windowing, FFT, magnitude and log2. The core computational unit receives the input samples at a rate of one per clock cycle and, after a variable initial latency, produces the results that are then processed by the output formatter.
- Output formatter: the purpose of this block is to receive the output samples of the core computational unit and to write them in the designated destination memory. As the input formatter, it is capable of manipulating the samples in a flexible way in order to reach the user's need in terms of data representation.

The overall accelerator engine operation is the following: the main processor writes in the parameter-set registers to program the functioning and the sequence of operations to be executed. Then, when the accelerator engine is enabled, the state machine is activated, meaning that it starts loading the configuration registers one after the other and running the accelerator as described in the registers.

In the parameter sets are stored all the important operating informations such as the type of calculation, the input and output memory address, the formatting, the trigger mode and so on. The sequence of parameter sets can also be repeated a determined number of times, programmed writing in the NLOOPS state machine register.

### 4.2.1 State machine

As mentioned in the previous section the state machine is the module that controls all the functioning part of the Hardware Accelerator, since it is responsible for: the enabling and the disabling of the whole HWA, the sequencing and looping of the parameter sets and the management of the trigger mechanism.

The settings of the state machine are programmed in the common part of the configuration registers, while the specific informations for each computation are stored in the parameter sets. An example of common settings are the PARAMSTART, PARAMSTOP and NLOOPS registers. When the accelerator is enabled, the state machine start executing the parameter-set defined in PARAMSTART and continues until it reaches the one written in PARAMSTOP. Then, if the register NLOOPS contains a value greater than 0, the sequence is repeated for the programmed number of times.

A setting that is particular for each parameter-set instead is, for example, the TRIGMODE register, that offers different modes of triggering. This is useful to time the operations correctly based on different events and to ensure that the computations start when all the input values are present in the local memory.

The incoming trigger types that can be selected for each parameter-set are:

- Immediate trigger: This mode is not based on a particular event and starts the operations immediately. It is useful when chaining different computations that just need a one-time trigger.
- Software-based trigger: In this case the state machine waits for the main processor to start the sequence of tasks. To do that a self-clearing bit is set in a common register (CR42ACCTRIG).
- ADC buffer switch trigger: Since the first two banks of the local memory can be shared with the ADC buffer, the state machine can be triggered when a switch from ping to pong happens (or vice versa) so that, as soon as the samples are ready, they can be processed and sent to the main memory, reducing the burden and the time needed for the inter-chirp processing.

- DMA-based trigger: This mode triggers the state machine when a DMA transfer finishes. It is useful when the input samples are stored in the main memory. In this way it is ensured that the computation starts when all the data are present in the source memory. Also in this case the trigger is done setting a bit in a common register (DMA2ACCTRIG).

When the state machine is triggered, it loads the registers contained in the configuration memory for the current parameter-set into corresponding internal registers of the accelerator and starts the actual computations for that parameter set. Once these have finished, the accelerator repeats the same steps for the following parameter-set.

When the accelerator engine finish the operations for the current parameter-set it has two possible outgoing trigger mechanism:

- Interrupt to main processor: Once the HWA has reached the end of a parameter-set it can generate an interrupt event in the main processor.
- Trigger to DMA: The state machine can also generate an event to start a determined DMA channel. The configuration is done by writing in a parameter-set register the channel to be triggered.

To modify the configuration memory is necessary to disable the HWA, reset it and then rewrite the new parameters in the RAM.

Table 4.1 shows the main state-machine-related registers, specifying the width, the purpose and whether or not they are proper of each parameter-set (if not it means that they are common).

| Register | Width | Parameter set | Description |
|---|---|---|---|
| ACCENABLE | 3 | No | Enable and Disable Control:<br>This register enables or disables the entire Radar Hardware Accelerator. The reason for a 3-bit register (instead of 1-bit) is to avoid an accidental bit-flip (for example, transient error caused by a neutron strike) from unintentionally turning on the accelerator engine. A value of ACCENABLE = 111b enables the Radar Hardware Accelerator and any other value of the register keeps the accelerator engine in disabled state. |

**Table 4.1:** State machine registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| ACCRESET | 3 | No | Software Reset Control:<br>This register provides software reset control for the Radar Hardware Accelerator. The assertion of these register bits by the main processor will bring the accelerator engine to a known reset state. This is mostly applicable for resetting the accelerator in case of unexpected behavior. Under normal circumstances, it is expected that whenever the accelerator is enabled (from disabled state), it always comes up in a known reset state automatically. The recommended sequence to be followed in case software reset is desired is to write 111b to this register and then a 000b, before the clock is enabled to the accelerator. |
| NLOOPS | 12 | No | Number of loops:<br>This register controls the number of times the state machine will loop through the parameter sets (from a programmed start index till a programmed end index) and run them. The maximum finite number of times the loop can be run is 4094. A value of 4095 (0xFFF) programmed in this register should be considered as a special case and it should be interpreted as an infinite loop mode, for example, keep looping and never stop the accelerator engine unless reset by the main processor. A value of zero programmed in this register means that the looping mechanism is disabled. In this case, the accelerator engine can still be used under direct control of the main processor (without the state machine looping provision coming into the picture). |
| PARAMSTART<br>&<br>PARAMSTOP | 4<br>&<br>4 | No | Parameter-set Start and Stop Index:<br>These registers are used to control the start and stop index of the parameter set through which the state machine loops through. The state machine starts at the parameter set specified by PARAMSTART and loads each parameter set one after another and runs the accelerator as per that configuration. When the state machine reaches the parameter set specified by PARAMSTOP, it loops back to the start index as specified by PARAMSTART. |

**Table 4.1:** State machine registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| FFT1DEN | 1 | No | ADC buffer sharing mode:<br>This register is relevant when the HWA is included in a single device along with the mmWave RF front-end. In such a case, during active chirp transmission and inline first dimension FFT processing, the ACCEL_MEM0 and ACCEL_MEM1 memories of the accelerator are shared as ping-pong ADC buffers. This register bit needs to be set during this time, so that while the digital front end writes ADC samples to the ping buffer, the accelerator automatically accesses the pong buffer, and vice versa. At the end of the active transmission portion of a frame, this bit can be cleared, so that all the four local memories can be accessed independently. |
| TRIGMODE | 3 | Yes | Trigger mode control:<br>This parameter-set register is used to control how the state machine and the operations of the accelerator are triggered for each parameter set.<br>The following modes are supported:<br>• 000b – Immediate trigger<br>• 001b – Software trigger<br>• 010b – Ping-pong switch based trigger (applicable only when FFT1DEN is set)<br>• 011b – DMA-based trigger |
| CR42ACCTRIG | 1 | No | Software trigger bit:<br>This register bit is relevant whenever software triggered mode is used (for example, TRIGMODE = 001b). Whenever software triggered mode is configured for a parameter set, the state machine keeps monitoring this register bit and waits as long as the value is zero. The main processor software can set this register bit, so that the state machine gets triggered and starts the accelerator operations for that parameter set. |

**Table 4.1:** State machine registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| DMA2ACCTRIG | 16 | No | DMA trigger register:<br>This register is relevant whenever DMA triggered mode is used (for example, TRIGMODE = 011b). Whenever a channel has finished copying input samples into the local memory of the accelerator and wants to trigger the accelerator, the procedure to follow is to use a second linked DMA channel to write a 16-bit one-hot signature into this register to trigger the accelerator. In DMA triggered mode, the state machine keeps monitoring this 16-bit register and waits as long as a specific bit (see DMA2ACC_CHANNEL_TRIGSRC) in this register is zero. The second linked DMA channel writes a one-hot signature that sets the specific bit, so that the state machine gets triggered and starts the accelerator operations for that parameter set. |
| DMA2ACC_<br>CHANNEL_<br>TRIGSRC | 4 | Yes | DMA channel select for DMA completion trigger:<br>This parameter-set register is relevant whenever DMA triggered mode is used (for example, TRIGMODE = 011b). This register selects the bit number in DMA2ACCTRIG for the state machine to monitor to trigger the operation for that parameter set. |
| CR4INTREN | 1 | Yes | Completion interrupt to main processor:<br>This parameter-set register is used to enable/disable interrupt to the main processor upon completion of the accelerator operation for that parameter set. If enabled, the main processor receives an interrupt from the Radar Hardware Accelerator at the end of operations for that parameter set, so that the main processor can take any necessary action. |
| PARAMDONESTAT<br>&<br>PARAMDONECLR | 16<br>&<br>16 | No | Parameter-set done status:<br>This read-only status register can be used by the main processor to see which parameter sets are complete that led to the interrupt to the main processor. The individual bits in this 16-bit status register indicate which of the 16 parameter sets have completed. These status bits are not automatically cleared, but they can be individually cleared by writing to another 16-bit register: PARAMDONECLR. |

**Table 4.1:** State machine registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| DMATRIGEN | 1 | Yes | Completion trigger to DMA:<br>This parameter-set register is used to enable DMA channel trigger upon completion of the accelerator operation for that parameter set. This trigger mechanism enables the accelerator to hand-shake with the DMA so that output data samples are copied out of the accelerator local memory. If enabled, the accelerator triggers a specified DMA channel, so that the output samples can be shipped from the local memory to Radar data memory. |
| ACC2DMA_<br>CHANNEL_<br>TRIGDST | 4 | Yes | DMA channel select for accelerator completion trigger:<br>This parameter-set register is used to select which of the 16 DMA channels allocated to the accelerator should be triggered upon completion of the accelerator operation for that parameter set. This register is to be used in conjunction with DMATRIGEN. |
| CR42DMATRIG | 16 | No | Trigger from processor to DMA:<br>This register can be used by the processor to trigger a DMA channel for the first time, so that a full sequence of repeated operations between the DMA and the accelerator gets kick-started. |

**Table 4.1:** State machine registers. [4]

### 4.2.2 Input and output formatters

The input and output formatters are the operational blocks that interface the computational unit with the accelerator local memories. To operate efficiently with the latters, this two modules has a flexible way of accessing to the stored data, very similar to the EDMA multidimensional access patterns. The input and output formatters can also perform different types of data manipulation, since the samples need to be in a 24-bit complex format for the calculations.

The access to the source and the destination memory needs to be programmed in detail in order to make the operation independent of the main processor. The first parameters to be set are the source and destination addresses that, as mentioned, can not be in the same local memory. Then the sample width and type need to be specified between 16 and 32 bits for the former and real or imaginary for the latter.

The flexibility of this operational blocks lies in the programmable access patterns and the possibility to perform various iterations back-to-back. The patterns are programmed using different parameter-set registers that specify for example how many bytes there are between a sample and the following (SRCAINDX and DSTAINDX), how many samples a single operation has to process (SRCACNT and DSTACNT), how many bytes there are between the starting address of an iteration and the following (SRCBINDX and DSTBINDX) and how many back-to-back iterations have to be performed (REG_BCNT).
These registers are explained in detail in Table 4.2.

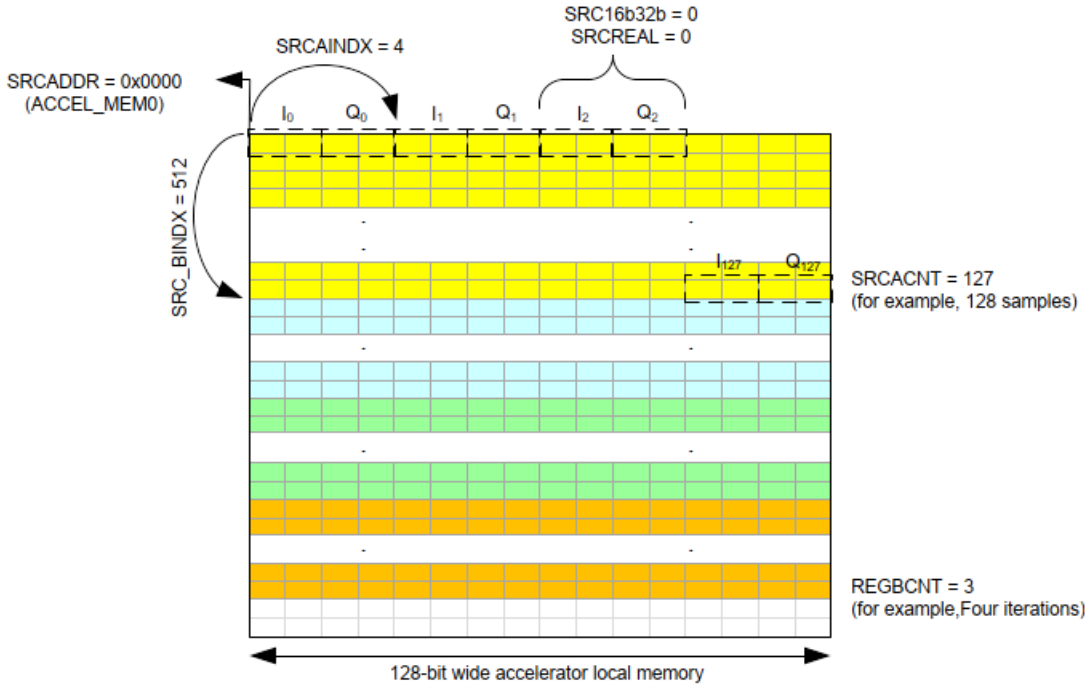


**Figure 4.3:** Example of source memory with the values of the input formatter registers. [4]

To understand better the operations of the input and output formatters an example from HWA User's Guide is reported. In Figure 4.3 the source memory in a first dimension FFT case is analysed.

The colored cells represents the samples received from the four RX antennas of the same transmitted chirp. In this case the data are in a 16-bit real and 16-bit imaginary format and the number of samples per RX is 128. The register

REG_BCNT is set to three $(4-1)$ since four FFT iterations, one for each RX antenna, must be performed. Every clock cycle the input formatter can fetch a complex sample, format it and send it to the core computational unit in order to be processed.

As mentioned, the computational unit works with 24-bit parallelism so each sample has to be scaled or extended to the correct number of bits depending on the initial width. Also this operation can be adjusted by the processor specifying, in the case of 16-bits samples, how many bits to pad at the MSB or, in the case of 32-bits width, how many MSBs to clip. The input formatter will treat the samples according to whether they are signed or not.

These considerations are also valid for the output formatter taking into account that the operation is reversed, that is, when the results need to be stored in 32-bits, a padding is applied while, if the 2 bytes representation is chosen, a clipping is performed.

Another feature of the input formatter is the automatic zero padding when performing an FFT. In fact, if the number of input samples is not a power of two, this modules adds zeros until the programmed FFT size is reached so that the core computational unit can operate correctly.

To continue with the same example, the destination local memory with the output formatter parameters is represented in Figure 4.4. As can be seen the output samples are stored in a transposed manner. The offset between consecutive samples is 16 bytes corresponding to the size of the sample (4 bytes) times the number of RX antennas. The index between two consecutive iterations is 32 bits to effectively interleave the processed samples.

Table 4.2 then explains in detail the registers of the two formatters and their purpose.

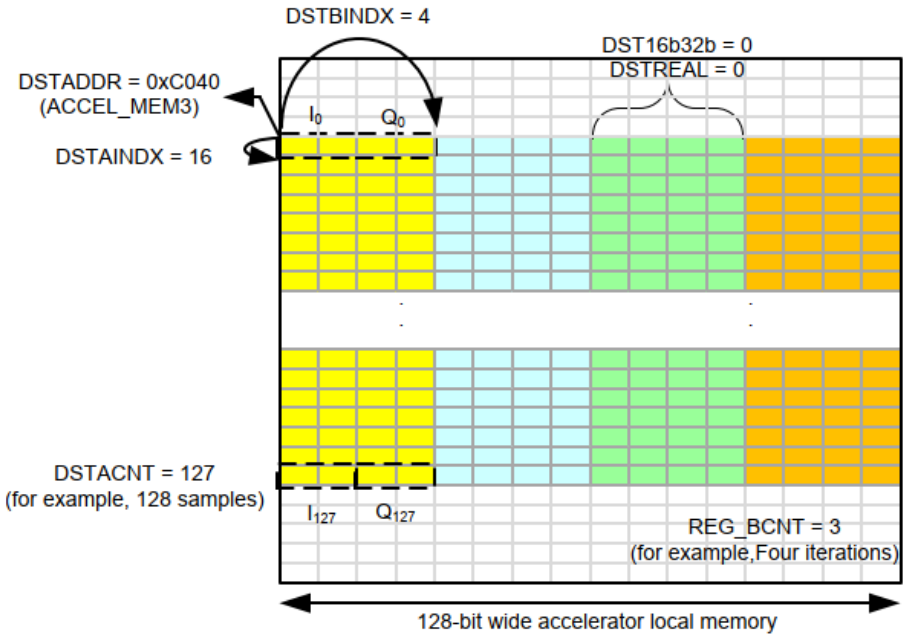


**Figure 4.4:** Example of destination memory with the values of the output formatter registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| SRCADDR & DSTADDR | 16 & 16 | Yes | Source and destination start address: These registers specify the starting address of the input and output samples. These are byte-address and these 16-bit registers cover the entire address space of the four local memories (4 × 16KB = 64 KB). The four accelerator local memories are contiguous in the memory address space and any of them can act as the source or destination memory (as long as the same memory bank is not configured to be used as both at the same time). |

**Table 4.2:** Input and output formatters registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| SRCACNT & DSTACNT | 12 & 12 | Yes | Source and destination sample count: These registers specify the number of samples (minus 1) to be read from or written to the local memory for every iteration. The sample count is in number of samples, not number of bytes. For example, the sample count can be specified as 255 (0x0FF) in a case where 256 samples must be processed. Note however that the sample count register does not always match the FFT size. This can happen when zero-padding of input samples is required or when only a part of the FFT bins must be written to memory. |
| SRCAINDX & DSTAINDX | 16 & 16 | Yes | Source and destination sample index increment: These registers specify the number of bytes separating successive samples in the source memory or to be written to the destination one. For example, a value of DSTAINDX = 16 means that successive samples written to the destination memory should be separated by 16 bytes. The maximum value allowed for these registers is 32767. |
| REG_BCNT | 12 | Yes | Number of iterations: This register specifies the number of times (minus 1) the processing should be repeated. This register can be used to process the four RX chains back-to-back – for example, a value of REG_BCNT = 3 means that the processing (say first dimension FFT processing) is repeated four times. Note the distinction between the NLOOPS register of the state machine block and the REG_BCNT register of the input formatter block. The NLOOPS register specifies how many times the state machine loops through all the configured parameter sets (with each time possibly awaiting a trigger), whereas the register REG_BCNT specifies how many times the input formatter and the computational processing of the accelerator is iterated back-to-back for the current parameter set (without any intermediate triggers). |

**Table 4.2:** Input and output formatters registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| SRCBINDX & DSTBINDX | 16 & 16 | Yes | Source and destination offset per iteration: These registers specify the number of bytes separating the starting address of samples for successive iterations. For example, when using four iterations to process the four RX chains, these registers can be used to specify the offset in the address between the successive RX chains. Note the distinction that SRCAINDX and DSTAINDX specify the number of bytes separating successive samples for a particular iteration, whereas SRCBINDX and DSTBINDX specify the number of bytes separating the starting address of the first sample for successive iterations. The maximum value allowed for these registers is 32767. |
| SRCREAL & DSTREAL | 1 & 1 | Yes | Complex or real input and output: These registers specify whether the input and output samples are real or complex. A value of 0 implies a complex sample and a value of 1 implies real input or output. When real input is selected, the input formatter block automatically feeds zero for the imaginary part, while if real output is selected, the output formatter module automatically stores only the real part into the destination memory. This is useful when the core computational unit is configured to output magnitude or log-magnitude values. |
| SRC16b32b & DST16b32b | 1 & 1 | Yes | Software trigger bit: These registers specify whether the samples are to be read or written as 16-bits or 32-bits wide. A value of 0 implies that the samples are 16-bits wide each (in case of complex data, real and imaginary parts are each 16 bits wide). A value of 1 implies that the samples are 32-bits wide each. |
| SRCSIGNED & DSTSIGNED | 1 & 1 | Yes | Input and output sign-extension mode: These registers, when set, specify that the samples are signed numbers and hence, sign-extension or signed-saturation at the MSB is required when converting 16- bit or 32-bit words to the computational unit's 24-bit wide samples or vice versa. |

**Table 4.2:** Input and output formatters registers. [4]

### 4.2.3 Core computational unit

The core computational unit can perform various mathematical operations that are commonly used in Radar processing such as Fast Fourier Transform, log-magnitude, Constant False Alarm Rate, and so on. As mentioned, it accepts 24-bit complex numbers as input samples at a rate of one per clock cycle and, after a variable latency, produces the results with the same throughput.

There are two main path to process the samples: one for executing windowing, FFT, log2 and magnitude operations and one for the CFAR algorithm. Only the first branch will be presented since the CFAR hardware implementation is beyond the scope of this paper.

The architecture of the computational unit is represented in Figure 4.5 and shows that various operations can be performed in series as well as singularly. This provides to the user the flexibility to choose from different possible configurations for every parameter-set. As for the other modules of the accelerator engine, to program the operational block various configuration registers are used, such as the ones written in Figure 4.5 to control the multiplexers.

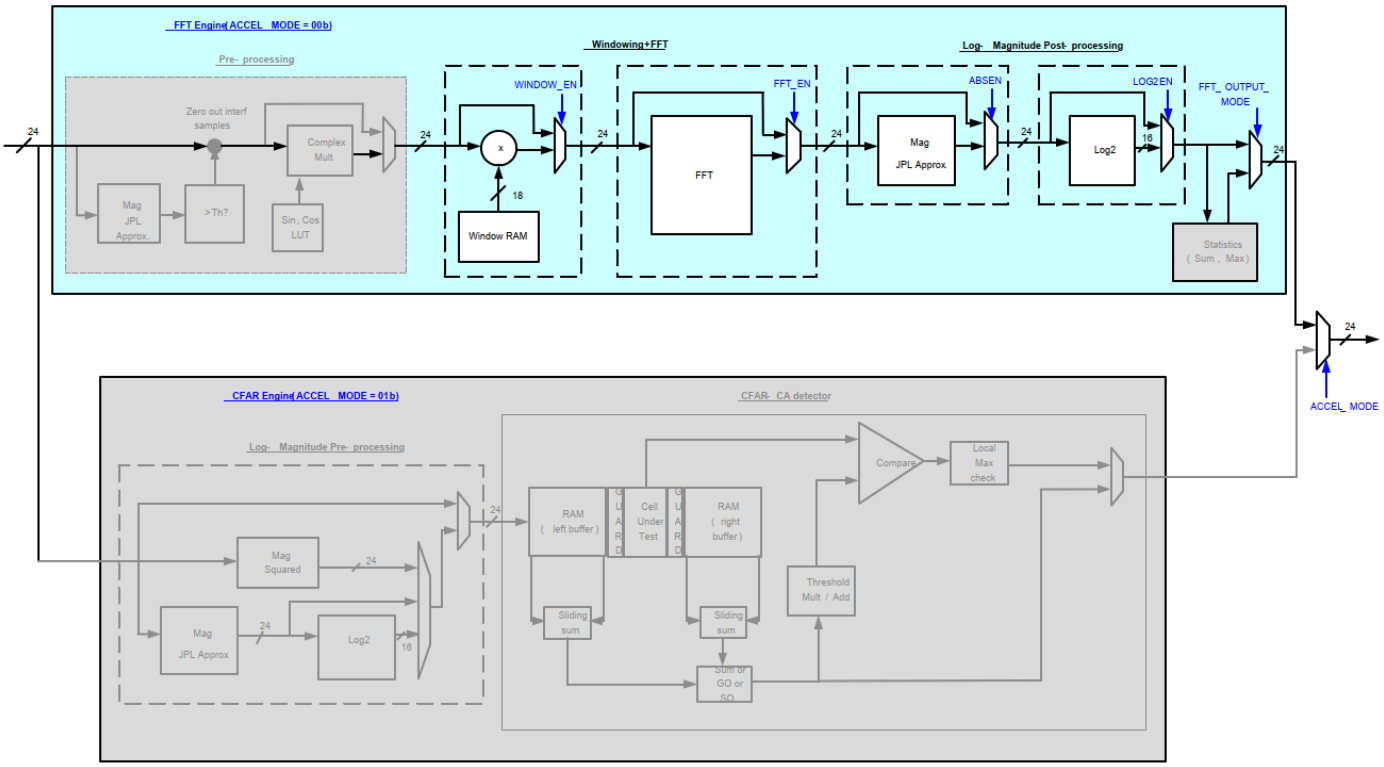


**Figure 4.5:** Core computational unit block diagram. [4]

### Windowing

The sub-block responsible for the windowing operation is the first in the FFT processing chain (excluding the pre-processing module that will not be used). Windowing operation is often required prior to performing FFT, to mitigate the sinc roll-off leakage from one strong FFT bin to the adjacent bins.

To implement this useful operation the input samples are multiplied by coefficients stored in a dedicated programmable RAM. The format of the weights is 18-bit, signed and two's complement. After the multiplication with the input samples the results need to be brought back to 24-bit parallelism so excessing LSBs are dropped.

The coefficients memory has a size of 1024 words so more than one window can be stored and then used when necessary. If the coefficients are symmetric, only half of the window needs to be stored so that the space in the RAM can be optimized.

### FFT

The FFT sub-block, as said, works with 24-bit parallelism and has a throughput of one sample per clock cycle. It supports FFT sizes from 2 to 1024 (the powers of two between these numbers), while for bigger operations a chaining of more computations in needed. Depending on the chosen FFT size, an appropriate number of butterfly stages is used (up to a maximum of ten).

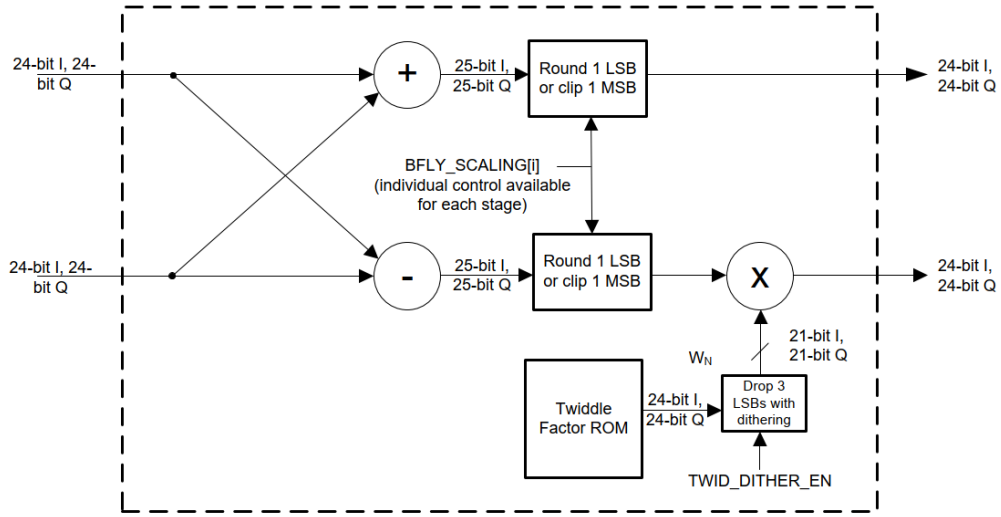


**Figure 4.6:** Single butterfly stage diagram. [4]

The schematic of a single stage in represented in Figure 4.6 and shows that after

the first operation the samples need to loose one extra bit to maintain the same parallelism. To account for that, the user can decide for each stage to divide the result by 2 (rounding the LSB) or to saturate the MSB.

After the rounding there is a multiplication operation with the twiddle factors. These coefficients are stored in a ROM as 24-bit complex data and, before the multiplication, they loose 3 LSBs to avoid overflow. In the process dithering can be optionally applied.
"The purpose of dithering is to eliminate any repetitive quantization noise patterns from degrading the SFDR of the FFT. [...] For dithering, an LFSR is used to generate a random pattern, for which the LFSR seed must be loaded with a non-zero value [...]. The SFDR performance of the FFT, with dithering enabled, is better than –140 dBc" [4], as shown in Figure 4.7.

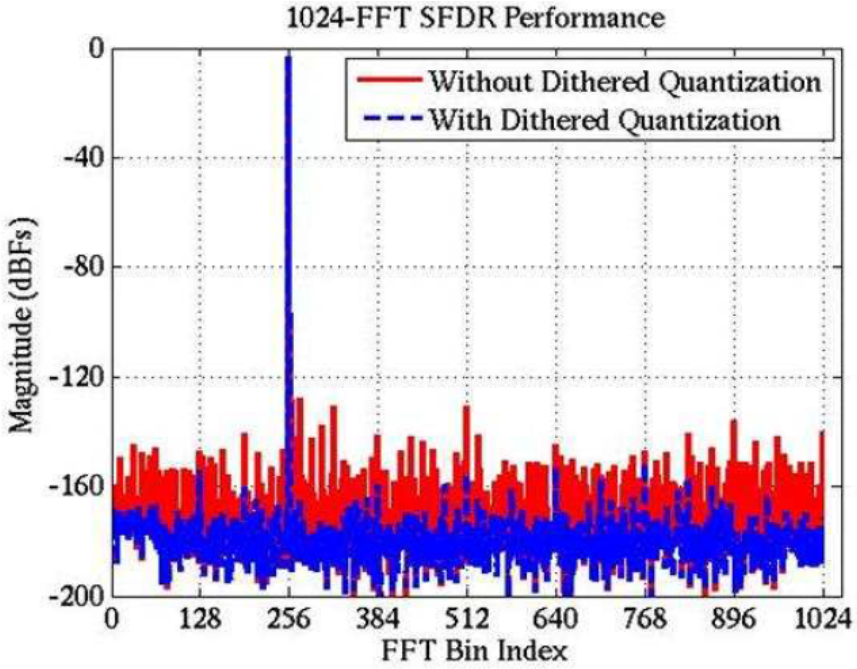


**Figure 4.7:** SFDR performance. [4]

### Magnitude and log-magnitude processing

These sub-blocks perform absolute value and base two logarithm computation. Since they are directly after the FFT module, the output stream of samples can be immediately post-processed.

The Log2 sub-block uses a look-up table to implement the operation while the

magnitude computation is performed using a Levitt and Morris approximation. The latter, given a complex number $I+jQ$, defines the quantities $U = max(|I|, |Q|)$ and $V = min(|I|, |Q|)$.
The following operation is then applied to compute the magnitude:

$$Mag \approx max(U + V/8{,}7U/8 + V/2)$$

With regard to the computation of base two logarithm the following definition of generic unsigned number is used: $N = 2k(1 + f)$.
This means that $log2(N) = k + log2(1 + f)$. Starting from this concept, the look-up table of the module implementation provides the second addendum (i.e. $log2(1 + f)$). The performance of the log2 module is shown in figure 4.8.

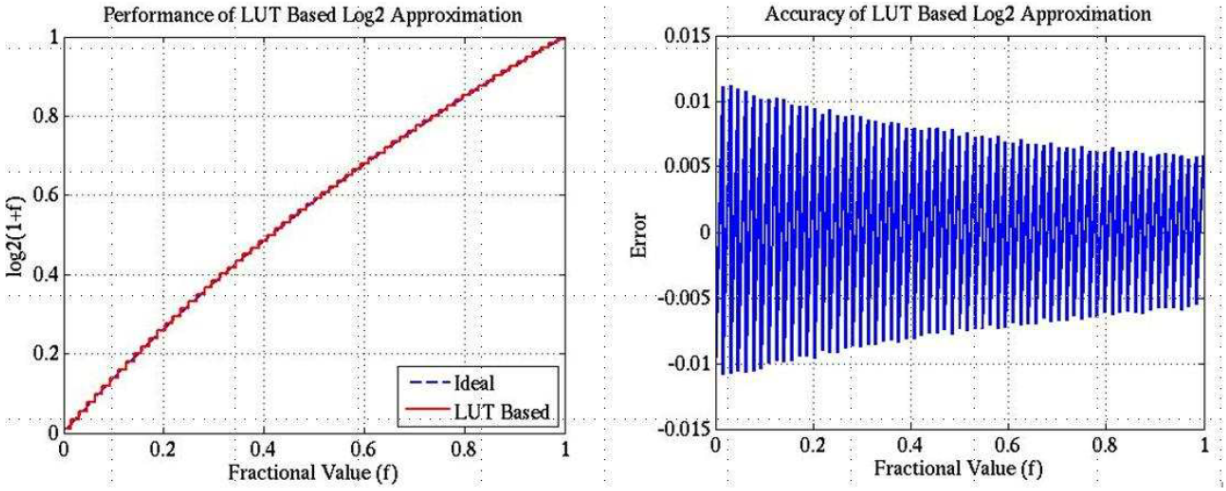


**Figure 4.8:** Log2 performance. [4]

The results of this sub-block are in a 16-bit real fixed point format with 5 bits of integer part. Since the parallelism must get back to 24 bits, if the log2 and magnitude blocks are enabled, 8 zeros are padded to the MSB.

### Statistics

Another important operational block is the one responsible for computing statistics such as the sum and the maximum value of the output samples. In Figure 4.9 is reported a detail of the core computational unit's block diagram.
At the end of the FFT computational branch the 24-bit complex samples can either be sent directly to the output formatter or they can be processed to obtain some useful informations such as the sum and the maximum of the processed values.

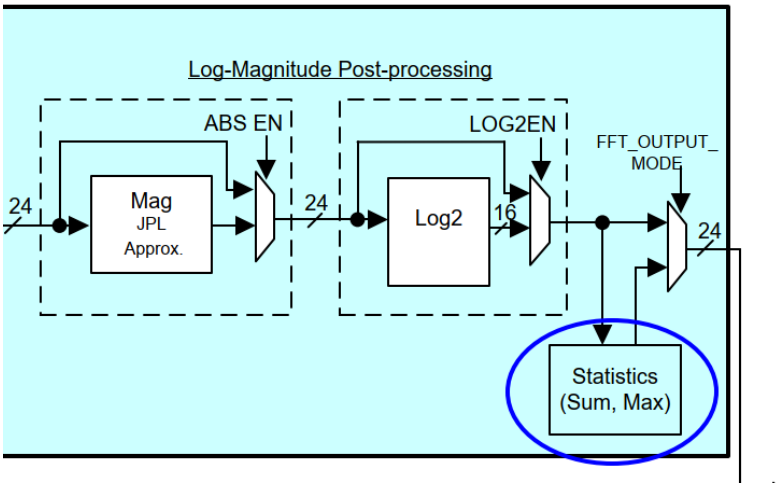


**Figure 4.9:** Statistics block detail. [4]

The statistics are computed for each iteration of the processing and they are logged in dedicated registers that can be read by the main processor. There are four of these register-sets for each statistic so, if the number of iterations is greater than four, these informations need to be stored in the destination local memory of the accelerator.
To do that a parameter-set register (namely FFT_OUTPUT_MODE) needs to be properly configured in order to choose which data should be fed to the output formatter.

The possible configurations are reported in Table 4.3.

| FFT_OUTPUT_MODE Register | I Channel Output | Q Channel Output |
|---|---|---|
| 00b - Default output mode | Main output of core computational unit | |
| 10b – Max statistics output (One output per iteration) | Max Value | Max Index |
| 11b – Sum statistics output (One output per iteration) | Sum of I values | Sum of Q values |

**Table 4.3:** FFT computational branch output modes. [4]

Only the sum operation is now presented since, to compute the maximum values of the FFT peaks, more complex algorithms will be used.

As mentioned, the sum statistic is computed on a "per-iteration" basis and this implies that many values can be accumulated every time. To avoid saturation, the parallelism of the sum output is 36 bits so that 12 bit of MSB growth are allowed from the standard computational unit's data width.

If the number of iteration is lower than 4 the statistics are saved in the registers maintaining the 36-bit format while, if more iterations are needed, the data must be scaled down to 24 bits to match the Output Formatter specifics.
To do that, the user can configure the number of LSBs to drop writing in the common accelerator register called FFTSUMDIV.
To be noted that the Output Formatter in this case is not meant to be used normally since there is only one value per iteration so its configuration is standard and it provides 32-bit complex samples even if magnitude values are being summed.

### Computational Unit's registers

Table 4.4 presents the main computational-related registers and their purpose.

| Register | Width | Parameter set | Description |
|---|---|---|---|
| WINDOW_EN | 1 | Yes | Windowing Enable:<br>This register-bit enables or disables the pre-FFT windowing operation. If this register is set to 1, then the windowing is enabled, otherwise, it is disabled. The exact window function (coefficients) to be applied is specified in a dedicated Window RAM. |
| FFT_EN | 1 | Yes | FFT Enable:<br>This register-bit is used to enable the FFT computation. If FFT_EN = 1, then the FFT computation is enabled. Otherwise, it is disabled (bypassed). |
| ABSEN | 1 | Yes | Magnitude Enable:<br>This register-bit is used to enable the magnitude calculation. If this register bit is set, then the magnitude calculation is enabled, else it is bypassed. When enabled, the magnitude (absolute value) of the input complex samples are calculated using Levitt and Morris approximation and the resulting magnitude value is sent on the I-arm of the output. The Q-arm is made zeros. |

**Table 4.4:** Core computational unit registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| LOG2EN | 1 | Yes | Log2 Enable:<br>This register-bit is used to enable the Log2 computation. If this register bit is set, then the Log2 computation is enabled, else it is bypassed. Note that setting this register bit only makes sense if the inputs to the Log2 computation are unsigned real numbers, such as when the Magnitude Enable bit (ABSEN) is also set. When enabled, the Log2 of the magnitude of the input samples is calculated and sent out on the I-arm of the output. The Q-arm is made zeros. |
| WINDOW_START | 10 | Yes | Windowing coefficients start index:<br>This register specifies the starting index of the window coefficients within the Window RAM. The value of this register ranges from 0 to 1023. The purpose of this register is to allow multiple windows (for example, one window of 512 coefficients and another window of 256 coefficients) to be stored in the window RAM and one of these windows can be used by programming this start index register appropriately in the current parameter set. |
| WINSYMM | 1 | Yes | Window symmetry:<br>This register-bit indicates whether the complete set of window coefficients are stored in the Window RAM or whether one half of the coefficients are stored. If this register bit is set, it means that the window function is symmetric and therefore, only one half of the window function coefficients are stored in the Window RAM. |
| FFTSIZE | 4 | Yes | FFT size:<br>This register specifies the FFT size. The mapping of the FFTSIZE register to the actual FFT size is as follows: Actual FFT size = 2 elevated to FFTSIZE. For example, a register value of 0110b specifies that the FFT size is 64. The maximum FFT size that is supported is 1024. Therefore, this register value is never expected to exceed 1010b. For large-size FFT (> 1024 point) that might be useful for industrial level-sensing applications, an FFT stitching procedure is supported, which is based on performing multiple smaller size FFTs in a first step and then stitching them in a second step (using a subsequent parameter set). |

**Table 4.4:** Core computational unit registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| BFLY_SCALING | 10 | Yes | Butterfly scaling:<br>This register is used to control the butterfly scaling at each stage of the FFT structure. Because the maximum FFT size is 1024, there are up to ten butterfly stages. Each butterfly stage has an add-and-subtract structure, at the output of which the bit-width would temporarily increase by 1 (from 24 to 25 bits wide). If BFLY_SCALING = 0, then the 25-bit output is saturated at the MSB to get back to 24 bits. Otherwise, it is convergent-rounded at the LSB to get back to 24 bits. The user can thus control the scaling at each of the 10 butterfly stages. The LSB of this register corresponds to the last stage and the MSB of this register corresponds to the first stage. For an FFT size of 64, only the LSB 6 bits are relevant. |
| DITHERTWIDEN | 1 | No | Twiddle factor dithering enable:<br>This register-bit is used to enable and disable dithering of twiddle factors in the FFT. The twiddle factors are 24-bits wide (24-bits for each I and Q), but they are quantized to 21-bits before twiddle factor multiplication. This quantization is implemented with dithering on the LSB, to avoid periodic quantization pattern affecting SFDR performance of the FFT. TI recommends keeping this register bit set to 1 (i.e. dithering enabled), with appropriate LSFR seed loaded. |
| LFSRSEED | 29 | No | Seed for LFSR (random pattern):<br>For twiddle factor dithering, there is an LFSR that is used, whose seed value is loaded by writing to this 29-bit LFSRSEED register. The LFSRSEED register should be set to any non-zero value, for example 0x1234567. |
| FFT_OUTPUT_ MODE | 2 | Yes | FFT Path output mode:<br>This register specifies the output mode of the FFT path. Instead of the default mode where the main output of the core computational unit is sent to the destination memory, this register can be configured such that either the max or sum statistics can be sent to the destination memory. |

**Table 4.4:** Core computational unit registers. [4]

| Register | Width | Parameter set | Description |
|---|---|---|---|
| FFTSUMDIV | 5 | No | Right-shifting for Sum statistic:<br>This register specifies the number of bits to right-shift the sum statistic before it is written to destination memory. The internal sum statistic register is 36-bits wide, but this statistics value needs to be scaled down to 24 bits to match the data path width going to the Output Formatter. This register specifies how many LSBs to drop to convert the sum statistics to 24-bit value. |

**Table 4.4:** Core computational unit registers. [4]

# Chapter 5

# Implementation

In this section the implementation of the Hardware Accelerator functions in the radar software will be described. The integrated development environment (IDE) used is the Texas Instrument's Code Composer Studio and the version of the software development kit (SDK) is 3.0.0.8, which includes many useful functions.

## 5.1 Common configurations

The first step of the process is to include the HWA library in the project. To do that the library search path must be specified in the linker options. In this case the files to be included are in *<sdk_install_directory>/packages/ti/drivers/hwa/lib* directory. In the same folder the library file must be also specified and its name is: *libhwa_xwr18xx.ae674*.

Secondly, the HWA local memory must be defined (as HWA_RAM) in the *dss_mmw_linker.cmd* file. The offset of this memory is 0x21030000 and the length is, as said, 16KB $*$ 4 (or 0x00010000).
The .hwaBufs section is also added, which is responsible for loading the accelerator RAM with the **NOINIT** option.
A memory buffer is then defined to produce the **M0**, **M1**, **M2** and **M3** partition addresses and the #**pragma** DATA_SECTION command links them to the memory section just defined.

The HWA memory buffer is used in the definition of the destination addresses of various operations such as the ADC sampling(**M0**), the output of the first dimension FFT ping (**M2**) and pong (**M3**) channels and the input and output of the second dimension FFT (explained in detail later).

An Hardware Accelerator handle is defined in the *MmwDemo_DSS_dataPathContext_t_* structure to manage the HWA driver.
In the same header file (dss_data_path.h) the prototypes of two accelerator-related functions are added: *MmwDemo_hwaInit* and *MmwDemo_hwaOpen*.
Both methods call the homonymous functions defined in the already mentioned SDK library in order to respectively initialize the HWA and open the accelerator instance. These methods will be then added to the *dssDataPathInit* function (in the dss_main.c file), which is responsible for initializing the drivers of the chip such as ADC and EDMA.

## 5.2 First dimension FFT

The first step for implementing in hardware the range FFT is to configure the ADC buffer.
As explained in Section 4.2.1, the ADC buffer can be shared with the Hardware Accelerator local memories to have the samples ready at the end of each chirp. To implement this feature the register FFT1DEN must be set to 1 but first the *dssDataPathConfigAdcBuf* function must be adjusted to the HWA settings.
For example the ADC must have complex output format, real part in the MSB bytes while imaginary part in the LSB bytes, non-interleaved channel mode and chirp threshold equal to 1 (i.e. every chirp the ADC switch from ping to pong or vice versa).

The offset between the addresses of the RX antennas is also changed to $(16 * 1024)/4$ (because 16 KB is the size of the HWA local memory and four is the number of RX antennas). This means that, being the samples of 4 bytes each (two I and two Q), a maximum of 1024 range bins can be set.

The *MmwDemo_dataPathConfig_FFTs_HWA* function is then needed to perform the preliminary FFT configurations and its purpose is:

- To disable and reset the Hardware Accelerator in order to modify the parameter-sets.
- To compute the window coefficients for the range FFT and to save them in the ad hoc RAM explained in Section 4.2.3. As for the DSP version of the software, the window type chose for first dimension FFT is Blackman.
- To enable the HWA interrupt for NUMLOOPS completion, setting a callback function whose only operation is to raise a flag (*hwa_1d_done_flag*).
- To enable twiddle coefficients dithering by writing in TWIDDITHERENABLE and LSFRSEED registers respectively 1 and a non-zero value.

### 5.2.1 HWA parameter-sets configuration

The Hardware Accelerator range FFT can now be configured. The first dimension block diagram is shown in Figure 5.1. Each part of the schematic will be explained in detail starting with the parameter-sets configuration (represented as light blue boxes). EDMA channels and their purposes will then be discussed.

The first step is configuring the registers common for all the parameter-sets:

- The FFT1DEN must be set to 1 to have ADC buffer samples directly available.
- The PARAMSTARTIDX and PARAMSTOPIDX registers are respectively set to 0 and 3 because, as shown in Figure 5.1, four parameter-sets are programmed (two dummy and two operational).
- NUMLOOPS is set to the number of Doppler bins times the number of TX antennas divided by 2 (this factor is present because every loop two chirp are processed).

The configuration of each parameter-set is now explained starting with the ping-related ones. The first parameter-set is called dummy because it is programmed only to trigger the computation of the accelerator.

The trigger mode is set to DMA-based but, as can be read in Figure 5.1, there is also a software trigger and that seems to be an impossible configuration. The explanation of this conflict is that in the SDK there is a function called *HWA_setDMA2ACCManualTrig* that lets the user manually trigger the execution of the state machine waiting on DMA.

The software trigger is used only to start the computations since, after the first loop, the dummy parameter-set will be then effectively triggered by the DMA completion. In particular, the HWA's DMA channel to be triggered is the number 0.

The second parameter-set is programmed to perform the FFT computation and its configuration is now explained:

- The trigger mechanism is based on the Digital Front-End switch from ping to pong (as discussed in Section 4.2.1).
- The source memory is **M0**.
- The number of samples to be processed in a single FFT (SRCACNT) is equal to the number of ADC samples (that can be a lower value than the FFT size).

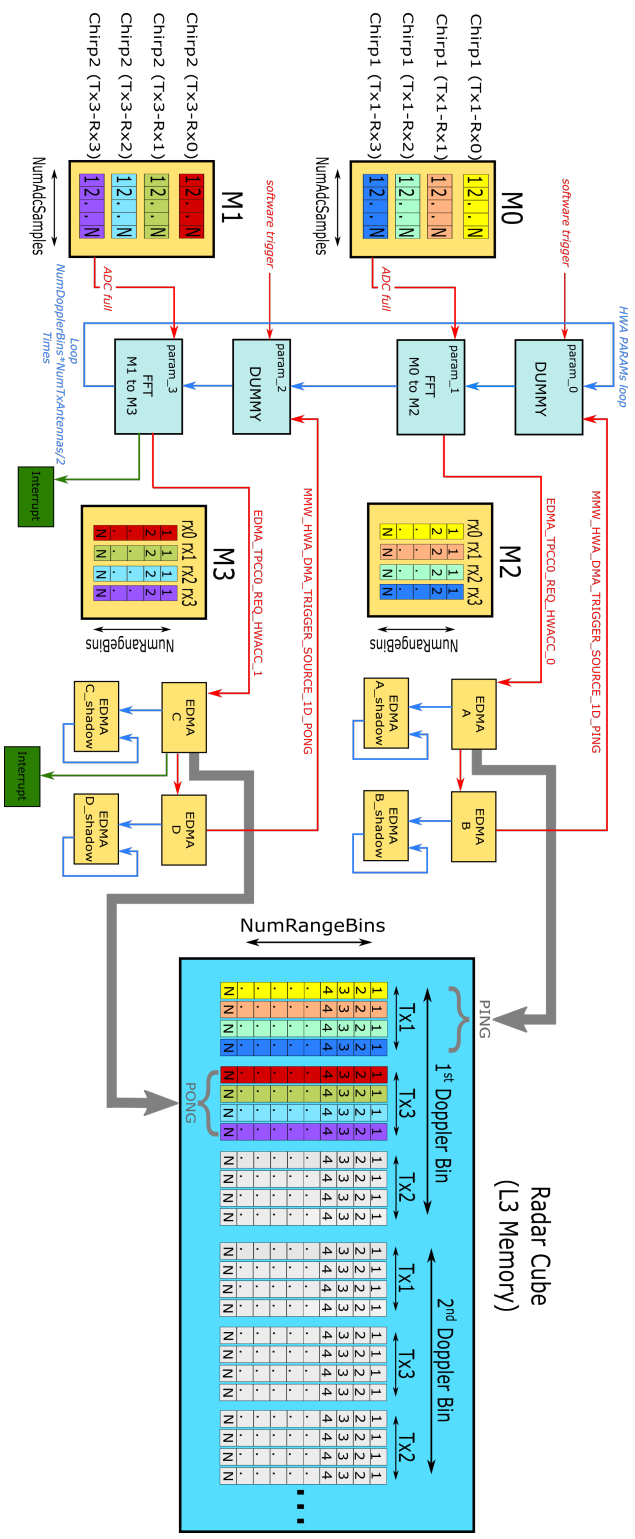


**Figure 5.1:** First dimension FFT block diagram.

- The number of bytes between a sample and the following (SRCAINDX) is equal to four since the data representation is 16-bit I and 16-bit Q.
- The number of back-to-back operations (SRCBCNT) is equal to the number of RX antennas because a single chirp is processed per iteration.
- The offset between the starting address of an iteration and the following (SRCBINDX) is equal to the offset configured in the *dssDataPathConfigAdcBuf* function (i.e. 16 KB/4).

To be noted that the destination memory layout is transposed with respect to the source one (as can be seen in Figure 4.3 and 4.4). This choice is made because it increases DMA efficiency. In fact, since the bus connecting the HWA and the processor is 128 bit wide and the EDMA can perform a transfer per cycle, the bandwidth is exploited if 16 bytes are transferred at a time.
In this case, the same range bin for each RX (up to a maximum of four) is moved and, apart from improving the efficiency of the single transposed transfer, it also makes the access to the range bins contiguous for the second dimension processing.

The configurations of the destination memory are the following:

- The destination memory is **M2**.
- The number of output samples per operation (DSTACNT) is equal to the size of the FFT.
- The number of bytes between an output sample and the following (DSTAINDX) is the size of the single complex sample times the number of RX antennas (transposed write).
- The offset between the start address of an iteration and the following (DSTBINDX) is equal to the size of a single sample (interleaved RX).

The FFT operation settings are now presented:

- The FFT operation is enabled (FFTEN register is set to 1).
- The FFT size is equal to the base-two logarithm of the programmed number of range bins.
- Windowing is enabled (WINDOW_EN set to 1).
- The window-symmetry option is set to 0 and the window address offset is the one defined in the *MmwDemo_dataPathConfig_FFTs_HWA* function (for the range FFT is 0).

- The log-magnitude post processing is disabled since it is not needed.
- The butterfly scaling needs to be revisited for all FFT sizes and data widths. For the configuration of the software (256 range bins and 16-bit complex samples) this register is set to 0x7, meaning that the last three butterfly stages operate a factor 2 scaling to prevent saturation.

The last configuration of the ping range FFT is specifying the type of interrupt for parameter-set completion: in this case a DMA interrupt event is generated in order to trigger the transfer of the computed output samples from **M2** to RadarCube matrix.

The configuration of the two pong parameter-sets is basically the same. For the dummy parameter-set the only different setting is the source trigger channel (DMA channel number 1 instead of 0 of the accelerator). With regard to the pong FFT parameter-set the source and destination memories need to be changed (respectively to **M1** and **M3**) and also the output DMA trigger channel is different.

### 5.2.2 EDMA configuration

Once the Hardware Accelerator is set, the following step is to configure the EDMA transfers. Since the ADC buffer is shared with the HWA memories there is no need for an input transfer. Again, the ping and pong configurations are very similar so the former will be discussed in detail while the differences of the latter will be then reported.

Starting with the EDMA transfer represented in Figure 5.1 as *EDMA A*, it is triggered by the FFT parameter-set completion that generates an event on DMA channel 0 of the HWA (called *EDMA_TPCC0_REQ_HWACC_0*). After being triggered, this transfer starts sending the output samples to the RadarCube matrix (stored in **L3** memory) and it ends when half of the total chirps in a frame are transferred (the other half are managed by the pong EDMA). This is possible thanks to the intermediate chaining option of the EDMA IP.

Every time a ping chirp is sent to the RadarCube matrix, the transfer *A* is stopped and the chained *EDMA B* is automatically started. This EDMA channel does not actually transfer data but it is only used to write a one-hot signature to the *DMA2ACCTRIG* register in order to trigger the ping dummy parameter-set of the Hardware Accelerator.
The SDK function to do that is called *HWA_getDMAconfig* and it returns a structure filled with the correct parameters with which an EDMA transfer has to be programmed to effectively trigger the desired accelerator channel.

As shown in Figure 5.1, both *EDMA A* and *EDMA B* have a linked shadow transfer (linking is represented by light blue arrows while red arrows mean chaining).
The shadow channels are used to reload the parameters of the respective EDMA channels once these have finished. The linking feature makes the reprogramming of the EDMA unnecessary (it becomes a one-time only programming).

The pong EDMA configuration is basically the same except that the *EDMA C* has a destination address offset with respect to the ping equal to the number of receiving antennas times the size of the single complex sample (offset represented in bytes). Another difference of this channel with respect to the *EDMA A* is that, once the transfer is complete (after that the second half of the chirps is sent to **L3** memory), an interrupt event is generated and a corresponding flag (*edma_1d_done_flag*) is raised.

### 5.2.3 Range FFT implementation conclusion

To implement the described functions in the software presented in Section 3 some final modifications are needed.
The first step is to add the HWA configuration functions in the *dssDataPathConfig* method (in dss_main.c file), which is responsible for the one-time programming of the main drivers such as ADC and EDMA.
The methods added are:

- *MmwDemo_dataPathConfig_FFTs_HWA*: the purpose of this function has been already discussed in Section 5.2.
- *MmwDemo_config1D_EDMA*: as the name implies, it is the method responsible for configuring the EDMA channels for first dimension processing.
- *MmwDemo_config1D_HWA*: the parameter-set configurations for range FFT are done in this function.
- *MmwDemo_dataPathTrigger1D*: it enables the HWA instance and manually triggers both the ping and pong dummy parameter-sets. This function is called before the start of Front End chirp generation so that the accelerator is immediately ready for processing.

Also in the *dssDataPathProcessEvents* some little modifications are needed. In the case of a **chirp_evt** message, a counter is needed to know when the total number of chirp per frame is reached. This operation is done in a function called *chirpProcess* which, in the old version of the software, was used to process the

chirps as well. This processing is not needed anymore since it is automatically done by the HWA. Once the end of the frame is reached, a function is added to wait the HWA loop completion and the related EDMA transfers. This is done by checking if the flags in the callback functions are asserted.

After the range FFT is completed, the inter frame processing is done (the details of which will be discussed in Section 5.3). Now, since the RadarCube matrix has a different layout with respect to the previous version of the software, the EDMA transfers that read from that matrix must be modified. In fact, while the size of the samples and the number of bins remains the same, the offset between them is different.
For example the offset in bytes between two Doppler bins, that used to be *numRxAntennas*numRangeBin*sizeof(cmplx16ImRe_t)*, becomes equal to *numVirtualAntennas*sizeof(cmplx16ReIm_t)*.

Another difference of the RadarCube matrix is the real and imaginary part are swapped (i.e. the I is stored in the 16 MSBs while the Q in the LSBs). This change is due to the Hardware Accelerator specifics.
The samples of the matrix are used to compute the Doppler FFT and, to account for the swap, there is a function in the SDK library that performs the windowing operation and then reverse the output samples.

The following modification implies that, just before the start of the successive frame, the two aforementioned functions *MmwDemo_config1D_HWA* and *MmwDemo_dataPathTrigger1D* must be called to reconfigure the HWA parameters and to trigger the operations.

Since the first dimension FFT is now done by the Hardware Accelerator the old DSP processing chain can be dismantled. The first step is removing the ADCdataIn buffer and the related EDMA transfers, that used to bring the data from the output of the ADC to the input of the DSP. The method *interChirpProcess*, where the range FFT was performed, can also be removed.

Other old functions that are not used anymore due to the hardware FFT implementation are: the one responsible for generating the window for DSP computation and the one that calculates the twiddle factors for range FFT.

## 5.3 Second dimension FFT

The second dimension processing is programmed and executed in the inter frame period, that is the time between two successive groups of transmitted chirps. To clarify the timing of the operations, Figure 5.2 is reported.

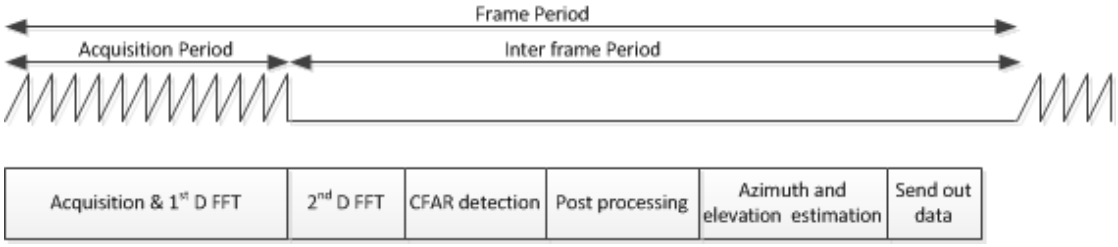


**Figure 5.2:** General timing of radar processing.

The preliminary operation for implementing the Doppler FFT is to compute the windowing coefficients and to save them in the HWA's designated RAM.
As for the DSP version of the software, the window type chose for second dimension FFT is Hanning. This configuration is done, for simplicity, in the same function of the range FFT (*MmwDemo_dataPathConfig_FFTs_HWA* explained in Section 5.2).

### 5.3.1 HWA parameter-sets configuration

The Radar Hardware Accelerator second dimension processing is configured in the function *MmwDemo_config2D_HWA*. The block diagram is represented in Figure 5.3.

As for the first dimension processing, the configuration begins with the registers common for all the parameter-sets:

- The FFT1DEN register must be set to 0 since the input samples come from the main memory and not from the ADC buffer.

- The PARAMSTARTIDX and PARAMSTOPIDX registers are respectively set to 4 and 11 because eight parameter-sets are programmed: four for FFT computation, two for the sum between virtual antennas and two for the formatting of the results.

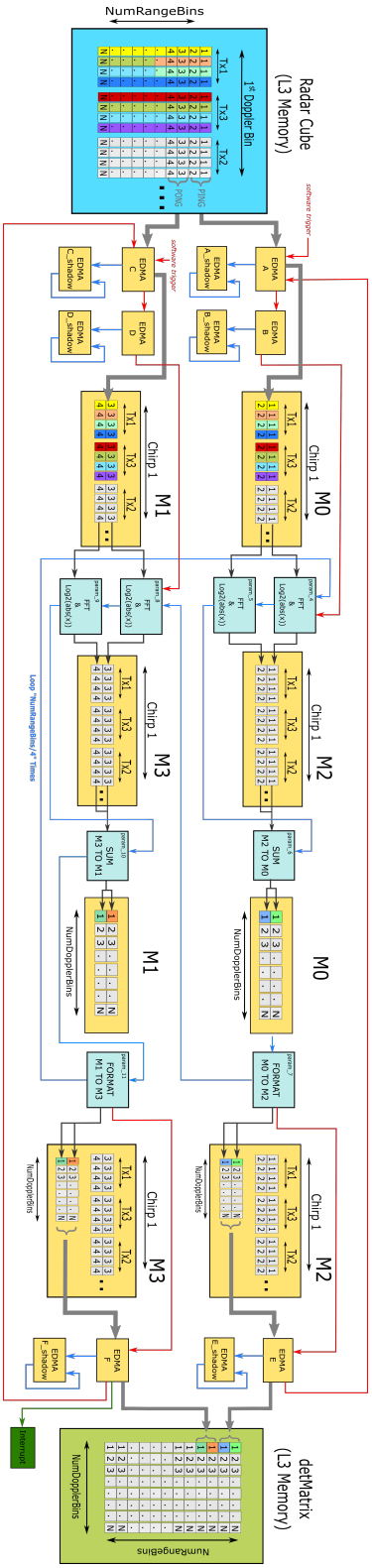


**Figure 5.3:** Second dimension FFT block diagram.

- NUMLOOPS is set to the total number of range bins divided by 4: this factor is obtained by multiplying the number of ping and pong operations every loop (2) and the number of range bins processed by the single operation (in this case is also equal to 2).

The description of the parameter-sets configuration is now presented. As usual, the ping is treated in detail and the differences of the pong are then discussed.

The first parameter-set is in charge of computing the Doppler FFT of one single range bin at a time throughout all the chirps.
The configuration is the following:

- The FFT is triggered by DMA event. In fact, the computation needs to wait the input samples being transferred from the main memory.
- The source memory is **M0** and the destination one is **M2**.
- The FFT operation is enabled (FFT_EN is set to 1) as well as the magnitude and base two logarithm computation (ABSEN and LOG2EN both set to 1).
- The windowing function is also enabled and the offset corresponding to the Hanning window is set.
- The output samples are stored as 16-bits real unsigned numbers since the log2-magnitude is performed.

The second parameter-set is the copy of the first with only two exceptions:

- The trigger mode is set to immediate since there is no need to wait external events.
- An offset equal to the size of the range bin processed by the first parameter-set (*numDopplerBin*numVirtualAntennas*sizeof(cmplx16ReIm_t)*) is added to the source memory starting address. The same is done for the destination memory with a different offset (*numDopplerBin*numVirtualAntennas*sizeof-(uint16_t)*) to avoid overwriting the previous results.

The third ping-related parameter-set performs the sum of the values of the virtual antennas in order to have a better estimation of the average received power in each range and Doppler bin. This operation is performed with the statistics block of the FFT branch described in Section 4.2.3.
The configuration is the following:

- The trigger mode is set to immediate.
- The source memory is **M2** and the destination one is **M0**.

- The computational branch is set to FFT.
- All operational blocks are disabled except for the statistics one.
- The register FFT_OUTPUT_MODE is set to $11b$ to enable the writing of the sum results to the destination memory.
- Every iteration a number of samples equal to the number of virtual antennas is processed by the computational unit.
- The FFTSUMDIV register is set to drop 8 LSBs.
- The output samples are stored as 32-bit I and 32-bit Q signed data since the statistics block has fixed output format.

The last parameter-set of the ping processing is responsible for formatting the samples to the correct width and data representation.
This parameter-set is needed because the statistics block stores the samples in a 32-bit complex format regardless of the type of data treated.
The Hardware Accelerator settings are the following:

- The trigger mode is set to immediate.
- The source address is the start of **M0**.
- The destination address is the starting address of **M2** memory plus an offset equal to the size of the results of the first two parameter-sets (*numDopplerBins* numVirtualAnt*numRangeBinsPerIter*sizeof(uint16_t)*).
- All operational blocks are disabled including the statistics one.
- The accelerator only requires a single iteration in which it processes *numDopplerBins*numRangeBinsPerIter* samples.
- The input samples are, as said, in a 32-bit signed complex format while the output is configured to be 16-bit unsigned real.

After the registers configuration, a DMA interrupt is enabled for this last parameter-set in order to correctly trigger the output EDMA transfer at the end of ping computations.

The pong configuration is literally copied from the ping parameter-set with the only modification being the source and destination memory (respectively **M1** and **M3**) and the different source trigger DMA channel for the first parameter-set.
Of course also the output interrupt must generate an event on a different channel to trigger the pong EDMA.

### 5.3.2 EDMA configuration

The EDMA transfers of the second dimension processing are used to bring the data of the RadarCube matrix to the accelerator memories and to send the results of the computation to the range/Doppler matrix (DetMatrix).
As usual a ping pong mechanism is exploited to transfer all the samples efficiently and the description is mainly focused on the former part.

The configuration of the first EDMA channel (represented as *EDMA A*) is very similar to the range FFT's output transfer but it also has some differences. The similarities are that both transfers have a linked shadow channel to automatically reconfigure their parameters and that the chained channel (namely *EDMA B*) has a one-hot signature function to trigger the accelerator computation.

The differences are that the first ping channel, as can be seen in Figure 5.3, is chained to the output transfer. This implies that to work properly it needs to be started via software.
The *MmwDemo_dataPathTrigger2D* function is responsible precisely of that, enabling the HWA instance and triggering (at the beginning of the loop) simultaneously both the ping and the pong input transfer. The samples are therefore brought to **M0** and **M1** memories and, once the transfers are complete, the respective one-hot signature bits are set in order to trigger the HWA computations.

To be noted is the fact that the input EDMA channels work in parallel for the first iteration of the loop, so that the ping parameter-set (which is the initial one) is triggered, while the pong waits to be executed. Once the ping HWA parameter-sets are completed, the pong is executed and it starts immediately since the register to poll is already set to 1.
After the first iteration the EDMA transfers are alternated normally following the usual ping-pong pattern.

The ping output transfer (*EDMA E*) is responsible for sending the results of the second dimension processing to the range/Doppler matrix in **L3** memory. This transfer is triggered by the accelerator parameter-set number 7 generating a DMA interrupt on the fourth HWA channel (called *EDMA_TPCC0_REQ_HWACC_4*). This channel has a linked shadow transfer, a chained one (namely *EDMA A*) and the option of intermediate chaining enabled.

The pong EDMA configuration is basically the same with respect to the ping except that the *EDMA F* has a destination address offset to interleave the results that is equal to *numDopplerBins*numRangeBinsPerTransfer*sizeof(uint16_t)*.

Another difference of this channel is that, once the transfer is complete, an interrupt event is generated and a corresponding flag (*edma_2d_done_flag*) is raised.

### 5.3.3 Doppler FFT implementation conclusion

In this section the last steps for implementing the described functions in the software are presented. All the second dimension related methods are added at the beginning of the *interFrameProcessing* function, substituting the loop for FFT and magnitude computation.

The methods called are, in order:

- *MmwDemo_config2D_EDMA*: with this function all the previously described EDMA channels are configured. This method could have also been called in the *dssDataPathConfig* since it performs a one-time configuration.
- *MmwDemo_config2D_HWA*: in this method all the accelerator parameter-sets are configured but the HWA remains disabled.
- *MmwDemo_dataPathTrigger2D*: as mentioned, this function enables the HWA and starts the ping and pong input transfers.
- *MmwDemo_waitEndOf2D_HWA*: this method waits for the pong output transfer completion by polling the *edma_2d_done_flag*.

Since the Hardware accelerator can be programmed and, once triggered, it doesn't require additional software control, these functions are all what is needed to perform the second dimension processing and store the results in the range/Doppler matrix (DetMatrix).

The old DSP processing chain can be now dismantled since it is not needed anymore. This implies removing the loop where the Doppler FFT and log2-magnitude operations were computed for each virtual antenna (smaller blue loop in Figure 3.4). The related EDMA transfers can also be removed such as the channels responsible for bringing the samples from the RadarCube matrix to the input of DSP or vice versa from the output to DetMatrix.
The functions used to wait for the end of these EDMA transfers are removed as well.

# Chapter 6

# Results and conclusions

The aim of this work is to optimize the radar processing. That implies improving the time performance while maintaining the correctness of the computations.

## 6.1 Results

The first step is therefore to verify that the Hardware Accelerator provides the same numerical results of the software-computed version.
To do that two successive acquisitions are taken in the same environment, the former using the old version of the application while the latter using the developed software. The data of the computations are saved through the PC interface and analysed with MATLAB®software.

The test is performed placing a corner reflector at a distance of 10 meters and comparing the values of the range/Doppler matrix (detMatrix) since the latter is where the results of the second dimension processing are stored. Being the environment constant, the zero velocity Doppler bin is analyzed.

In Figure 6.1 the content of the matrices throughout an acquisition period of 100 frames is represented. The curves shown in the picture represent range FFT of the static objects (0th bin).
The main peak is positioned between range bin number 30 and 31 meaning that the corner reflector is detected between 9.837 and 10.165 meters (since the range accuracy is $0.3279\,m$).
Other peaks are present between range bin number 75 and 125. These are due to environment objects (in particular a long metal fence).

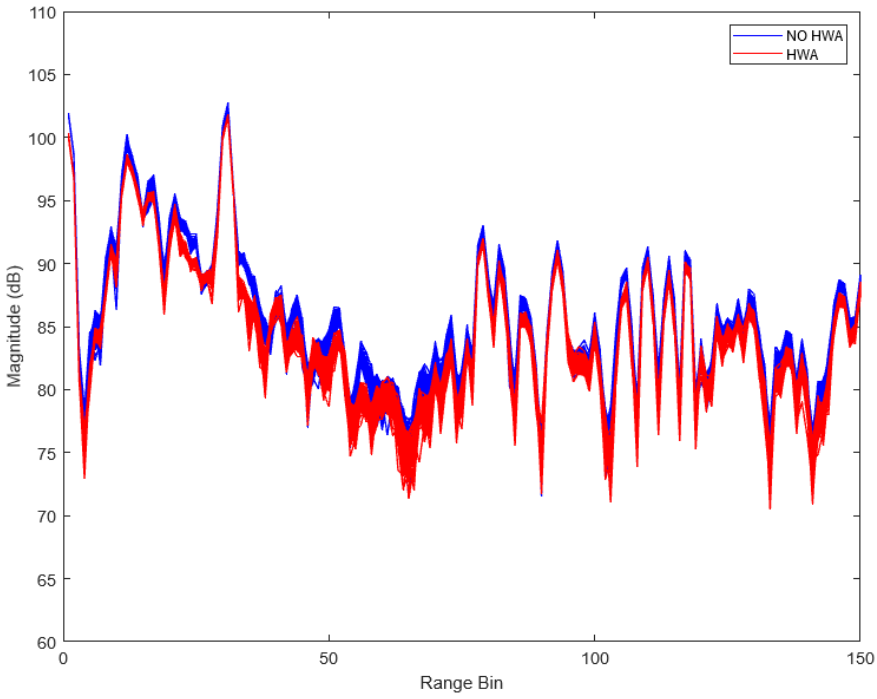


**Figure 6.1:** Zero velocity bins from the detection matrix of the two softwares.

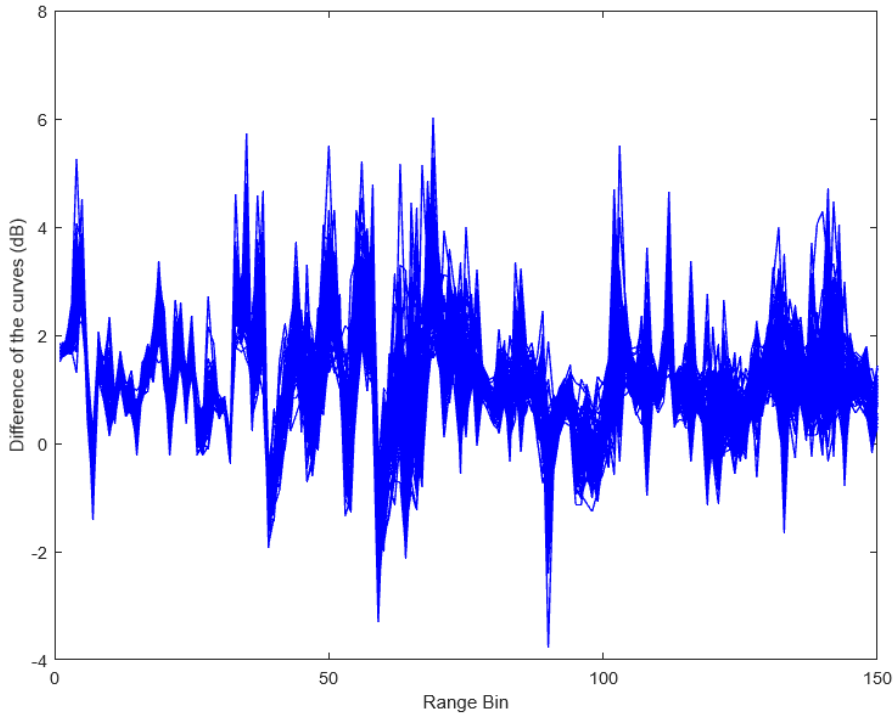


**Figure 6.2:** Difference of the two curves.

The aim of this test was to prove the correctness of the numerical operations so in Figure 6.2 is shown the difference of the two curves in dB. In the peak-related range bins the value is less than 1 dB (approximately 0.7).
Also the other peaks have comparable values being around one dB of difference.

On average the values of the range/Doppler matrix of the developed software are lower by 1.087 dB. Figure 6.3 shows that also the output of the CFAR thresholding algorithm is slightly lower with respect to the DSP version of the software (on average by 1.093 dB). This means that the signal to threshold ratio remains constant in both softwares.

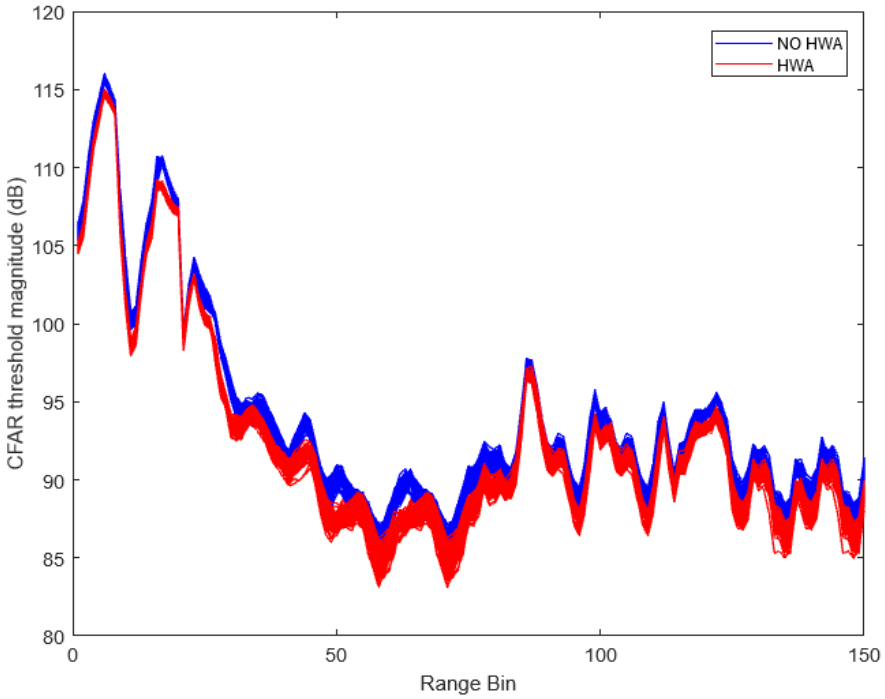


**Figure 6.3:** CFAR threshold comparison.

The difference in the computed range/Doppler matrix values is due to the different algorithm of magnitude estimation used. In fact, the DSP has a math library in which the following operation is performed in order to compute the magnitude of the Doppler FFT output samples:

$$Mag \approx (max(|a|, |b|) + min(|a|, |b|) * 3/8)$$

The Hardware Accelerator, as mentioned in Section 4.2.3, uses the Levitt and Morris approximation instead.

To test the performance of both algorithms a comparison is done using a single set of Doppler FFT output samples. The results are shown in Figure 6.4 and, as can be seen, the approximation performed by the HWA algorithm produces a peak that is around 0.6 dB lower.

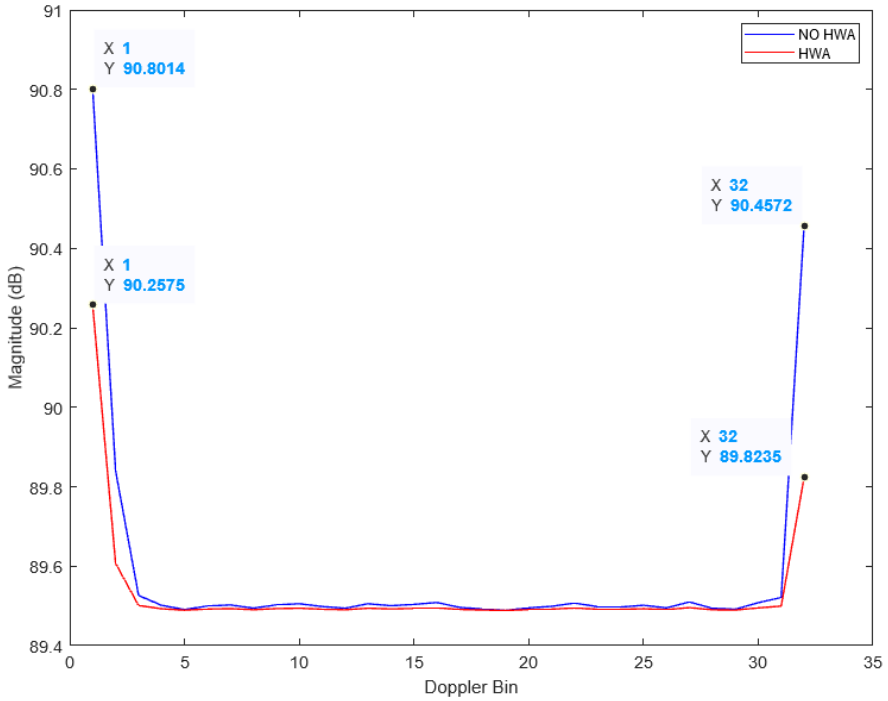


**Figure 6.4:** Comparison of the magnitude estimation algorithms.

The second test is performed to verify the correct detection of targets and the Angle of Arrival computation. As explained in Section 5 the layout of the RadarCube matrix changed and so did the related EDMA transfers.
The purpose is to control that the third dimension processing is not affected by the implementation of the accelerator FFT computations.

The test consists in performing a constant path with a 10 dB corner reflector and compare the radar detections through the PC interface. The path is reported in Figure 6.5.

The two softwares are run and the data of the radar are saved to file through the PC interface. The acquisitions are then analysed with MATLAB®. Figure 6.6 and 6.7 show the results of the acquired data.


**Figure 6.5:** Test path. [source: Google Earth]

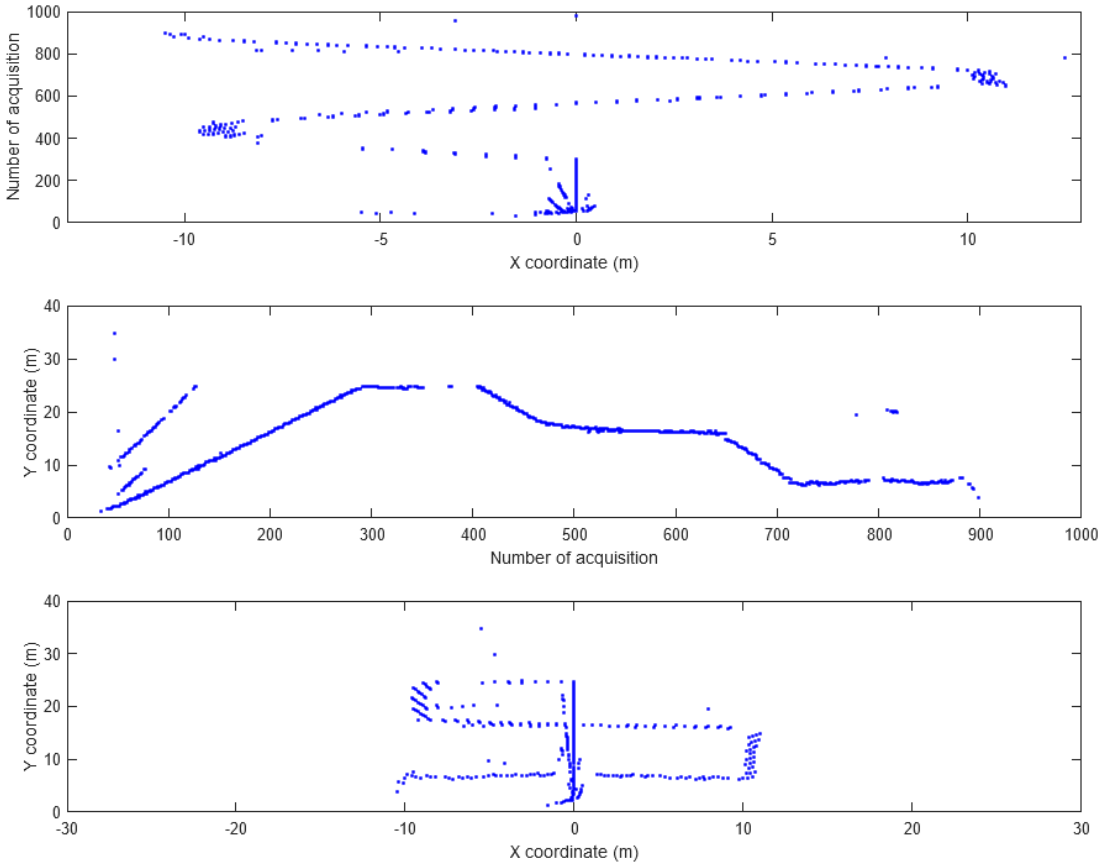


**Figure 6.6:** Detection results of the software without the HWA.

Both Figure 6.6 and 6.7 show the X coordinate of the detected targets versus time (number of acquisitions) in the first subplot.
In the second subplot is shown the Y coordinate (or distance from the radar in the longitudinal direction). To be noted that in the left part of the plot there is a reflection due to the wall behind the radar.
In the third subplot the X-Y coordinates of the detected targets are represented.

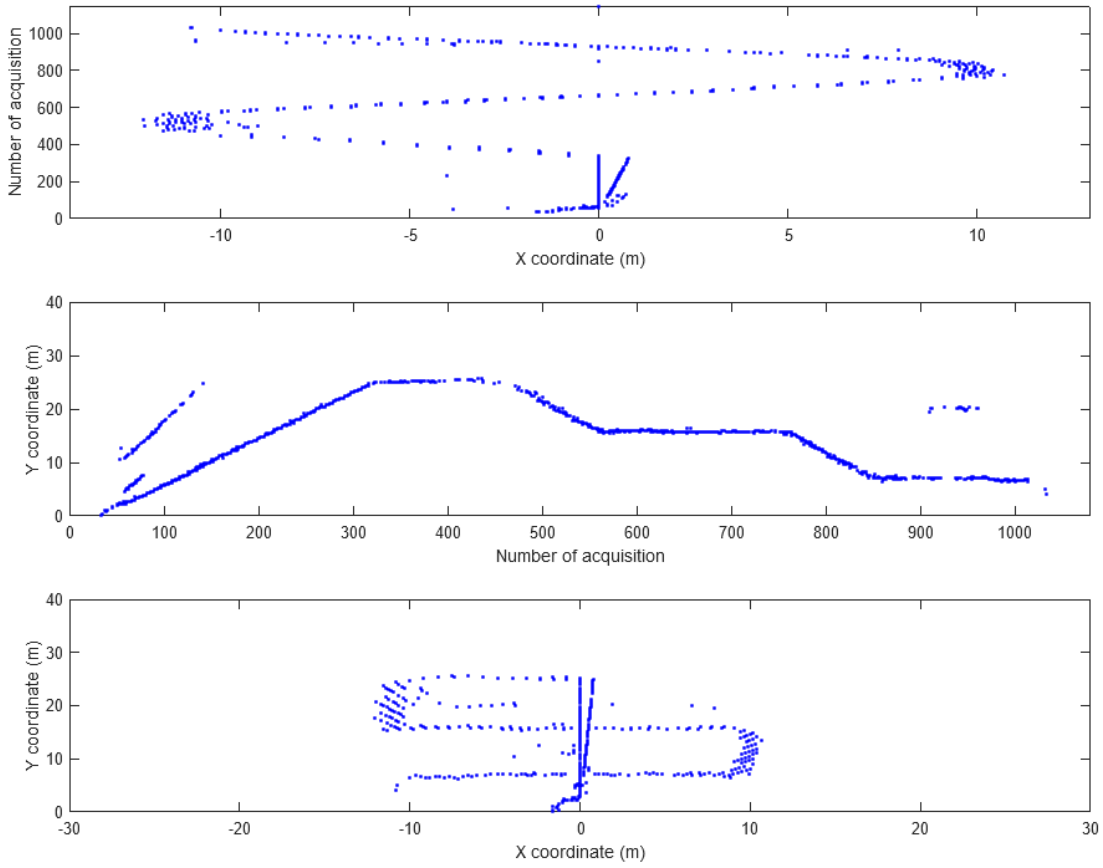


**Figure 6.7:** Detection results of the software with the HWA.

Once the correctness of the results has been proved the following step is to verify the computational time performance of the software.
As far as the first dimension processing is concerned the main goal is to reach the real time since, as shown in Figure 5.2, the operations are performed during the frame time between a chirp and the following.
Apart from that, further reductions of the processing time is not fundamental since the range FFT is the only computation performed in the inter-chirp period.

The most important time reduction is needed in the inter-frame period instead.

In fact, during this lapse all the complex algorithms such as tracking and elevation processing are performed so, since these computations require more time to execute, the faster the second dimension processing the better.

To verify the second dimension processing performance the two softwares are run again in a constant environment and, using the DSP cycle profiler, the number of clock cycles are stored and analysed in MATLAB®.
Two operations are compared between the softwares: the single second dimension processing and the whole inter-frame function (which contains the former).

The timing of the Doppler FFT and log2-magnitude computations are reported in Figure 6.8.

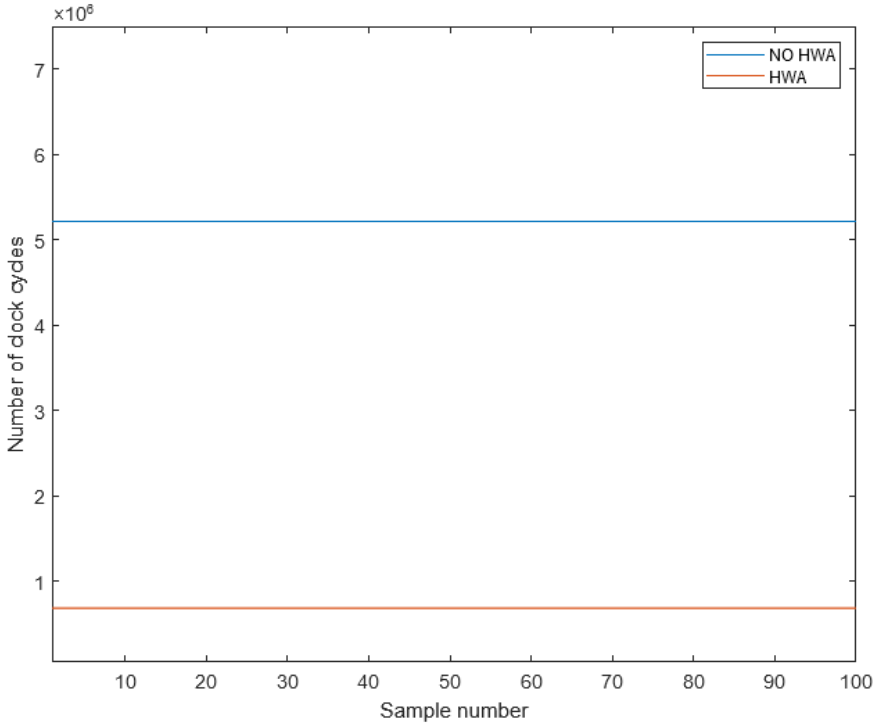


**Figure 6.8:** Time profiling of the second dimension processing.

As can be seen in the graph, the performance of the single computation is really stable (the variance is close to zero for both curves) and this is due to the fact that the timing of the FFT, the log2-magnitude and sum operations is not data dependent.

To compare the results the mean value of the samples is used. To be noted that the DSP runs at 600 MHz so for the conversion between clock cycles and time

that value is used.

The results are:

- DSP software version: $5.210 \cdot 10^6$ clock cycles or $4.795\,ms$.
- HWA software version: $6.827 \cdot 10^5$ clock cycles or $1.138\,ms$.
- The ratio of the mean values is: 0.131. This means that the Hardware Accelerator implementation saves more than 85% of the second dimension processing time.

The same is done for comparing the performance of the inter-frame processing which, as shown in Figure 5.2, consists of second and third dimension, CFAR, tracking and elevation operations.

The results of the profiling are reported in Figure 6.9. The curves have more variation and that happens because, unlike the second dimension operations, it is data dependant. In fact, for every detected target some algorithms must be performed and that requires time.

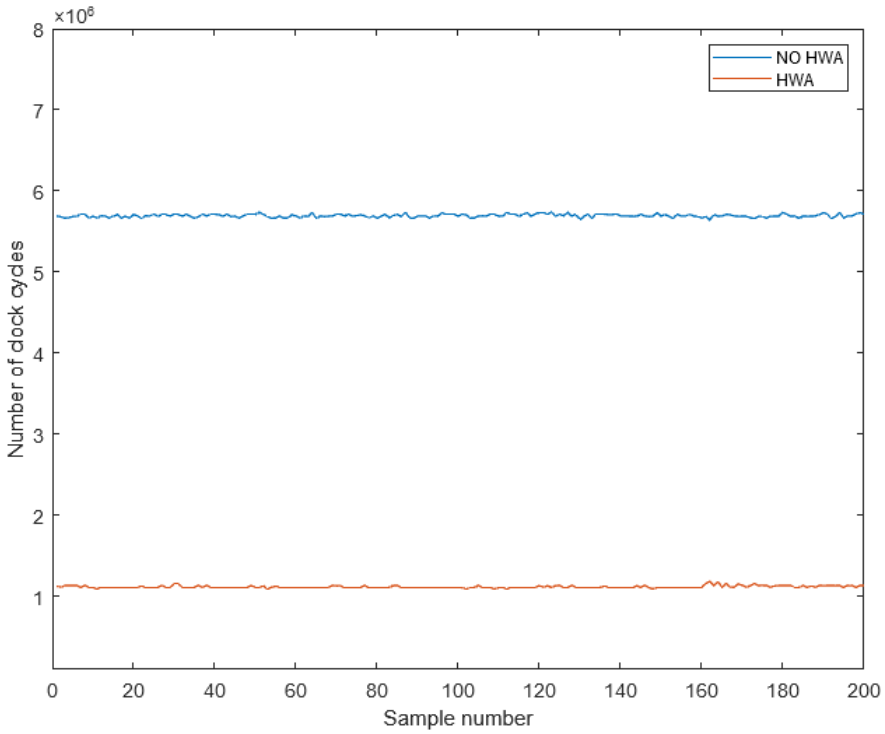


**Figure 6.9:** Time profiling of the inter-frame processing.

The results are:

- DSP software version: $5.695 \cdot 10^6$ clock cycles or $9.513\,ms$.
- HWA software version: $1.119 \cdot 10^6$ clock cycles or $1.848\,ms$.
- The ratio of the mean values is: 0.196. The reduction of clock cycles is therefore more than 80%.

## 6.2 Conclusions

The Hardware Accelerator strong advantage is to drastically reduce the time needed for inter-frame processing that reflects in many useful possibilities.
For example the Frame Repetition Interval can be lowered thus increasing the number of radar observation per second. Otherwise the number of chirp per frame can be increased providing more Doppler bins and therefore a better velocity resolution.

Another benefit of using the Accelerator is that a lot of temporary buffers and also some mathematical functions are not needed anymore so a good portion of fast memory can be saved leading to a further software optimization.

The great results in terms of time performance without affecting the computations' accuracy will therefore lead to an implementation of the technology in the softwares of the Politecnico's radar team.

# Bibliography

[1] Merrill I. Skolnik. *Introduction to radar systems*. New York (State): McGraw-Hill, 1962 (cit. on p. 1).

[2] *Datasheet AWR1843 Single-Chip 77- to 79-GHz FMCW Radar Sensor*. Texas Instruments. 2018 (cit. on p. 9).

[3] *User's Guide xWR1843 Evaluation Module (xWR1843BOOST) Single-Chip mmWave Sensing Solution*. Texas Instruments. 2018 (cit. on pp. 12, 14).

[4] *User's Guide Radar Hardware Accelerator*. Texas Instruments. 2018 (cit. on pp. 26, 28, 30–35, 37–48).