TECHNISCHE UNIVERSITÄT BERLIN

Reduction of Radar Processing Latency

vorgelegt von

Tamilselvan Shanmugam

Matriculation Number: 367975

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin

Masterarbeit

Betreuer: Mr. Jürgen Wittig, Airbus Defence and Space,
Prof. Dr. Ben Juurlink, TU-Berlin,
Prof. Dr.-Ing. Olaf Hellwich, TU-Berlin.

Processing period: 01-Jun-2015 to 16-Dec-2015

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Regensburg, 05-Mar-2016

. . . . . .

Tamilselvan Shanmugam

Acknowledgments

AIRBUS Defence and Space GmbH
Wörthstraße 85
Ulm 89077

This thesis originated in cooperation with the Airbus Defence and Space, Airborne Radar Processing and Safety Critical Software.

First of all I would like to thank Mr. Jürgen Wittig for giving me the opportunity to carry out state of the art research in this field. Special thanks to Mr. Hans-Peter Keller, Mr. Joachim Enzinger and Mr. Benjamin Baumgaertner for their guidance.

A wholehearted thanks to AES department, TU-Berlin for allowing me to pursue this thesis, rendering extended support and helping me throughout thesis processing.

I would like to thank my relatives for their support at the needful time. The journey of graduate program would not have been enjoyable without friends. I thank each and every one of them for their assistance and making my abroad stay pleasurable.

Dedicated to

Mr. Shanmugam Maarappan and Mrs. Boopathy Shanmugam

Abstract

Technology transforms everything simpler, faster and smarter. In an attempt to evaluate future Radar processor concepts, an analysis is carried out based on a mock-up of a Radar processing algorithm. An ARM platform is chosen keeping size, weight and power in mind. Several ARM Cortex A9 processors are involved to execute the Radar processing algorithm. An analysis done by Airbus DS investigates various means of scheduling Radar application to the ARM processors. Being safety critical software, determinism in terms of worst case execution time is a crucial factor. However, the analysis concluded that the worst case execution time in ARM processors are not in the acceptable range.

This thesis studies the bottlenecks imposed by the application, data dependencies between the processing chains and possible improvements. The implementation focuses on the parallelism in the Radar application and schedules them in optimal way to achieve best results. As a part of the investigation, peak processor utilization, peak memory utilization, peak bandwidth utilization, worst case execution time, bottlenecks and future scope are discussed in detail. An analysis is presented based on the new implementation schemes.

Keywords: *ARM Cortex A9, Latency, Multicore, Parallelism, Threading.*

Prerequisite: *Basics of multicore processing.*

Zusammenfassung

Technologie macht alles einfacher, schneller und intelligenter. In einem Versuch, zukünftige Radar-Prozessor Konzepte zu bewerten, wird eine Analyse auf der Grundlage eines Mock-ups eines Radarverarbeitungsalgorithmus durchgeführt. Eine ARM-Plattform wurde aufgrund von Größe, Gewicht und Verbrauch gewählt. Mehrere ARM Cortex A9-Prozessoren sind beteiligt, um die Radarverarbeitungsalgorithmen auszuführen. Eine Analyse von Airbus DS untersucht verschiedene Arten von Ablaufplänen bei der Radar-Anwendung auf den ARM-Prozessoren. Als sicherheitskritischer Software ist Determinismus in Bezug auf Worst-Case-Ausführungszeiten ein entscheidender Faktor. Allerdings zeigen die Ergebnisse der Analyse, dass die Worst-Case-Ausführungszeit in den ARM-Prozessoren nicht im akzeptablen Bereich liegen.

Diese Dissertation untersucht die Engpässe, die durch die Anwendung auferlegt werden, Datenabhängigkeiten zwischen den Verarbeitungsketten und mögliche Verbesserungen. Die Umsetzung konzentriert sich auf die Parallelität in der Radar-Anwendung und einen Ablaufplan, der für optimale Ergebnisse sorgt. Als Teil der Untersuchung werden spitzen Prozessor-Auslastung, Speicherauslastung Peak, Spitzenbandbreitennutzung, Worst-Case-Ausführungszeit, Engpässe und künftige Umfänge ausführlich diskutiert. Eine Analyse, basierend auf den neuen Implementierungsschemata, wird vorgstellt.

Stichwort: *ARM Cortex A9, Latenz, Multicore, Parallelität, Threading.*

Voraussetzung: *Grundlagen der Multicore-Verarbeitung.*

Acronyms and Abbreviations

A/A Mode:	Air to Air Mode
AESA:	Active Electronic Scanned Array
A/G Mode:	Air to Ground Mode
ARINC:	Avionics Application Standard Software Interface
BIV:	Binary Integration of Velocity
CFAR:	Constant False Alarm Rate
CMV:	Correlation Martix Velocity
CPU:	Central Processing Unit
D-Cache:	Data Cache
DDR:	Double Data Rate
DGPM:	Data Graphics Processing Module
DMA:	Direct Memory Access
FDP:	Frequency Domain Processing
FFT:	Fast Fourier Transform
GHz:	Giga Hertz. 1GHz = 10^9 Hz
GiB:	GibiByte. 1GiB = 2^{30} Bytes
GPU:	Graphic Processing Unit
GUI:	Graphical User Interface
iCON:	interface CONcentrator Module
I-Cache:	Instruction Cache
IMA:	Integrated Modular Avionics
IFFT:	Inverse Fast Fourier Transform
KiB:	KibiByte. 1KiB = 2^{10} Bytes
MHz:	Mega Hertz. 1MHz = 10^6 Hz
MiB:	MibiByte. 1MiB = 2^{20} Bytes
nm:	nautical mile. 1nm = 1.852km
OS:	Operating System

PCI:	Peripheral Component Interconnect
PRF:	Pulse Repetition Frequency
PRT:	Pulse Repetition Time
PSM:	Platform Support Module
RADAR:	RAdio Detection And Ranging
RCI:	Range Correlation Information
RG:	Range Gates
SATA:	Serial AT Attachment
SDRAM:	Synchronous Dynamic Random Access Memory
SPM:	Signal Processing Module
SIMD:	Single Instruction Multiple Data
SSD:	Solid State Drive
SWaP:	Size Weight and Power
TFLOP:	Tera FLoting point OPerations per second
UAV:	Unmanned Airborne Vehicles
VM:	Virtual Machine
WCET:	Worst Case Execution Time

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Radar Introduction

RADAR stands for RAdio Detection And Ranging. It is intended to detect and locate objects such as aircraft, motor bike, missiles, etc. Radar works the same way as how Bats navigate in the dark. But, instead of ultrasonic sound waves, Radar uses electromagnetic waves, that can travel long distance. The Radar sends an electromagnetic wave to a target, then analyses the echo from the target to determine target's information like position, velocity. Applications of Radar are spanned in many areas of engineering. It includes ultra-wide Band radar for human body monitoring and imaging [7], early warning system in military applications, measuring sea level, wave direction in remote sensing and lot more. Weather applications, precipitation animation in smart phones and weather forecast are some of the use cases of weather Radar. Pre-Collision System and Advanced Driver Assistance System in automobile are using Radars to detect imminent collision as well as takes mitigation plan [8].

1.1.1 Principle of Radar

Radar can be classified as Pulsed Radar or Continuous Wave Radar based on the operating principle[9]. Pulsed Radar, also called Pulse Doppler Radar, sends a short pulse then waits for the echo. The received echo is processed alongside. Pulse Doppler Radar is widely used in military applications. It has an antenna that acts as a transmitter and receiver. A short duration high energy pulse is generated and transmitted through the antenna. Up to this point, the antenna acts as a transmitter. As soon as the pulse is released from the transmitter, it is switched to receiver mode and is listening for echo from the target. Figures 1.1 and 1.2 illustrate the block diagrams of the pulse doppler Radar and target detection respectively. The Continuous Wave Radar, as the name suggests, transmits electromagnetic signal continuously and the received echoes are processed.

The time delay between sending the pulse and receiving the echo reveals the distance to

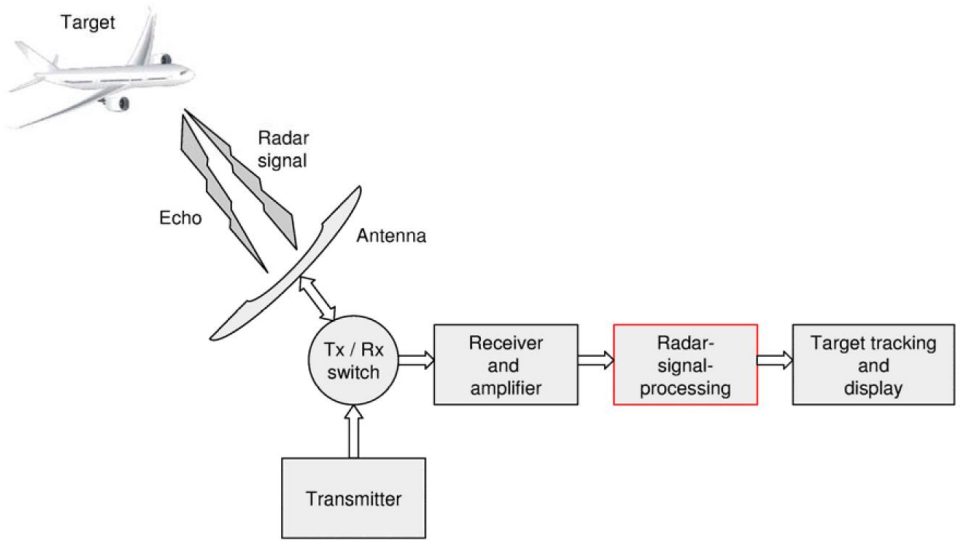

Figure 1.1: Block Diagram of Pulse Doppler Radar [1]

the target. The frequency shift in the echo tells the radial velocity of the target.

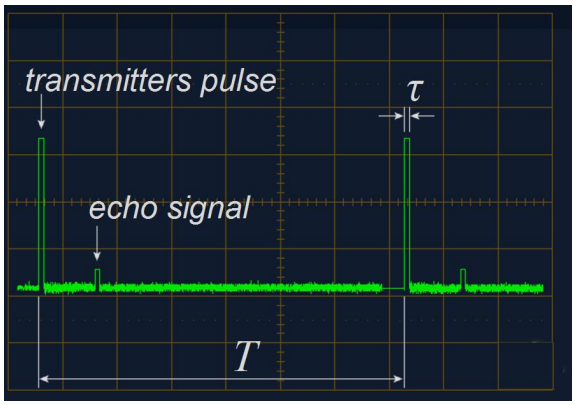

Figure 1.2: Transmitter's Pulse and Echo Signal from Target [2]

Conventional waveband format is followed by manufacturers to address the radar operating frequency. Nowadays, radars are operated from 300MHz to 100GHz in different wavebands. The purpose and installation location of the radar determines the waveband. The accuracy of the radar is proportional to the operating frequency. Also, high frequency signals are more attenuated to water droplets and water vapours in the atmosphere. On the other hand, attenuation of the lower frequency signals is lower than high frequency signals. The typical use case of low frequency radar is in Early Warning Systems whereas high frequency radars are deployed in missile guidance systems [10].

1.1.2 Terminologies

Radar systems use the spherical coordinate system to localize an object in the sky. The three following information are required to pin point an object relative to the Radar's position.

Azimuth angle ***(θ_{az}):*** It is the angle between north and the target in horizontal plane. Azimuth angle can say whether the target is in the left side or right side to the north.

Elevation angle ***(φ_{el}):*** It is the angle between the target and Radar's local plane. Elevation says altitude of the target relative to the Radar. The Azimuth angle and Elevation angle of the Sun are explained in Figure 1.3.

Radial distance (r): Distance between the target and Radar.

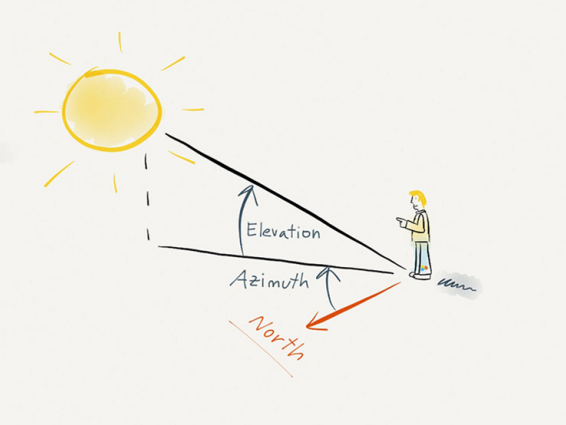

Figure 1.3: Azimuth and Elevation of the Sun [3]

Other terminologies related to this thesis are explained here.

Pulse: The Radar transmits an electromagnetic signal for a short duration (T_{on}) called pulse, then a break (T_{off}) follows to receive the echo of the pulse. This T_{on} and T_{off} together is called Pulse Repetition Time (PRT). Actual frequency of the electromagnetic wave transmitted during T_{on} period is called carrier frequency.

Burst: The Radar sends n number of pulses and listens to the echo signal. The combination of the selected PRT and the number of pulses are called Burst. Figure 1.4 shows two different bursts having 3 pulse count each.

PRF: Inverse of Pulse Repetition Time is called Pulse Repetition Frequency.
Dwell: Group of 8 different bursts form a Dwell.

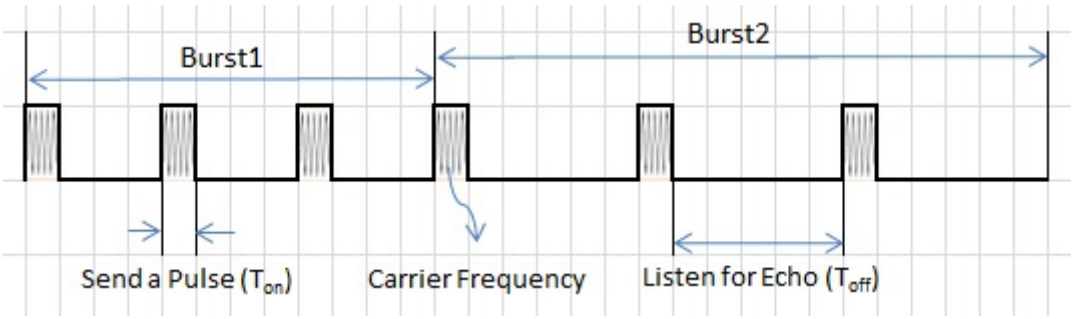

Figure 1.4: Example of Burst

Range Gate: The received echo signal is sampled at given time intervals called as range gates, shown in Figure 1.5. They represent distance from the Radar transmitter.

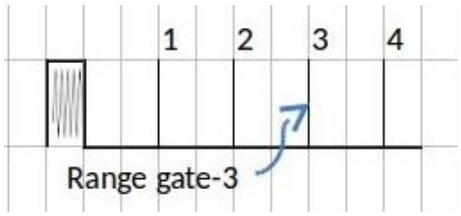

Figure 1.5: Example of Range Gate

Air to Air Mode (A/A Mode): The Radar mounted on an aircraft, scanning the sky.
Air to Ground Mode (A/G Mode): The Radar mounted on an aircraft, scanning the Ground.

1.2 Motivation

Radar is one of the core components of a combat air system. Range and accuracy of the Radar are important for missile system, guidance system, early warning system, etc. Massively parallel processor arrays, DSP processors, FPGA processors and GPU processors[11] are some examples of the Radar processors in the industry. They demand high computation capabilities, more power and so big in size.

Avionics and military applications always thrives to minimize the Size, Weight and Power (SwaP) while demanding high performance. It remains the backbone of the research. This thesis explores the possibilities of bringing a new dimension to the low cost and real-time Radar processor that can be employed in safety critical systems. ARM processor, which is running in millions of smart devices[12] is chosen, keeping compactness, portability and power requirements in mind. As a first step, medium range (40nm) Radars are considered as a suitable candidates to adopt low cost and compact ARM processors, since they do not need several TFLOP of processing power to adhere the real-time requirements.

A baseline analysis is carried out by Airbus DS concludes that the ARM processors cannot meet the real-time requirements, if the scheduling of the Radar processing algorithm not optimized. This thesis extends the baseline analysis and implements a couple of optimized scheduling schemes to meet the real-time requirements. The results of the optimized schemes and possible improvements are discussed in detail.

1.3 Requirements of the Radar Processor

A Radar mounted on an aircraft or satellite is called Airborne Radar. The Airborne Radar processor should be able to process Air to Air (A/A) Mode data and Air to Ground Mode (A/G) data. As the Radar processor is a part of a safety critical system, it should stick to the industrial standards DO-178B/C, ARINC-653.

Since this research is in the early stage of development, and to allow room for further requirements, the CPU utilization is restricted to 50% of the available CPU time, memory utilization should be less than 50% of the available memory size and the peak data transfer rate of the Radar application should be less than 50% of the maximum achievable data transfer bandwidth. The real-time requirements of the Radar processor shall be summarized as follows

- *Resulting processing latency:* The time span between reception of an echo and the processed information is sent out, should be less then 2x Dwell time.
- *Memory transfer bandwidth:* <50%.
- *Memory utilization:* <50%.
- *CPU utilization:* <50%.

1.4 Problem Statement

The Airborne Radar is subjected to operate on different Pulse Repetition Frequencies (PRF) to resolve ambiguity in distance and velocity. Consecutive eight different bursts form a Dwell. That is, on every time the Radar probes the sky, it sends one Dwell and the echoes are received and analysed. The resulting processing latency of the baseline analysis is theoretically calculated as 15x Dwell time (see Table 4.3). It means, to produce the result of the first Dwell, it needs as much time as the 15 Dwell transmission require. A typical Dwell time is 54ms. An Euro-fighter Typhoon moving at a speed of Mach 2 would move a distance of 560 meters during this processing time. Half a kilometre difference between detection and display is very high for a typical Airborne Radar processor.

The real-time requirement confines the processing latency to 2x Dwell time. The investigation of this thesis is to find an optimal scheduling scheme, number of processors required, data distribution, resulting processing latency and exploit the parallelism in the Radar processing algorithm to stick to the real-time requirements.

1.5 Contributions

One of the major works of this thesis is, binding the performance critical functional blocks of the Radar processing algorithm, to replicate the Radar processing chain, targeting multi-core architecture and multi-threaded application. In summary, the main contributions of this thesis are:

- **Multi-core Performance**
  - Scalability and performance of the application on multi-core environment are estimated. Shared resources, resource contention and race condition are considered.
  - All the threads are set to execute the functional blocks simultaneously to ensure the maximum memory transfer bandwidth.
- **Scheduling Scheme**
  - Data dependencies are identified and evaluated by executing them in parallel using POSIX Threads.
  - Constraints on non-thread-safe functional blocks are identified and scheduled them without violation.
  - Optimal scheduling schemes are proposed and their results are assessed.
- **Measurement Tools**
  - Developed automated scripts to measure peak memory utilization, CPU utilization, memory transfer bandwidth and processing latency.

1.6 Thesis Outline

The rest of this thesis is organized as follows:

- **Chapter 2:** Explains the Integrated Modular Avionics (IMA) architecture, experimental set-up and related work concerning ARM processors in Radar application.

- **Chapter 3:** Discusses the results of the baseline analysis done by Airbus Defence and Space GmbH.
- **Chapter 4** Explains the pros and cons of the baseline analysis, test bed information and important design choices.
- **Chapter 5:** Proposes optimized scheduling schemes and verifies their correctness via implementation. Results of the new schemes are discussed.
- **Chapter 6:** Summarizes this thesis with a conclusion and proposes future work regarding the optimized schemes.

Chapter 2

Background and Related Work

Minimizing size, weight and power consumption of the Radar processor without compromising performance is the topic under research. Low cost and portable Radar processor is intended for weight and power limited airborne platform such as Unmanned Airborne Vehicles. On satisfactory performance, this may open doors for new business opportunities in Radar processing and safety critical applications. ARM is the favourable architecture in this regard, and has been running in millions of smart phones, tablets and portable gadgets. Instead of one high capacity, power hungry, bulk processor, several small, energy efficient, compact processors of ARM type can be deployed. The way of grouping the processors, experiment setup [1] and state of the art related work are elaborated in this chapter.

2.1 IMA Processor Architecture

Integrated Modular Avionics is the cluster of real-time computing elements capable of executing multiple tasks having different safety critical levels. Instead of several dedicated computers for different purposes, a common hardware platform for many systems is used. This improves fault isolation, increased airborne functionality and ability to move applications between standardized computers [13]. To comply with the environmental requirements, industrial temperature standard, MCIMX6Q7CVT08AC processor type with 800MHz clock speed is used.

IMA architecture comprises of the following modules [2]

- up to 2 Platform Support Modules (PSM)
- up to 6 Data Graphics Processing Module (DGPM) or Signal Processing Module (SPM) or any combination of these two.

[1] These information are based on the technical documentation provided by Airbus Defence and Space GmbH, titled *Future Combat Air System - Analysis of IMA Modules,* version B.

[2] SPM Module and iCON Module are out of this thesis scope.

- up to 3 Interface Concentrator Module (iCON).
- a rack backplane.

2.1.1 Platform Support Modules (PSM)

The PSM has the following components:

- one iMX6Q CPU, clocked 800MHz.
- one 1GBit/s-Ethernet switch with 10 bi-directional communication ports.
- CPU has it's own private 4GiB SDRAM and non-volatile NOR/NAND Flash memory.
- 16GiB Solid State Disk, connected to the CPU via SATA interface.

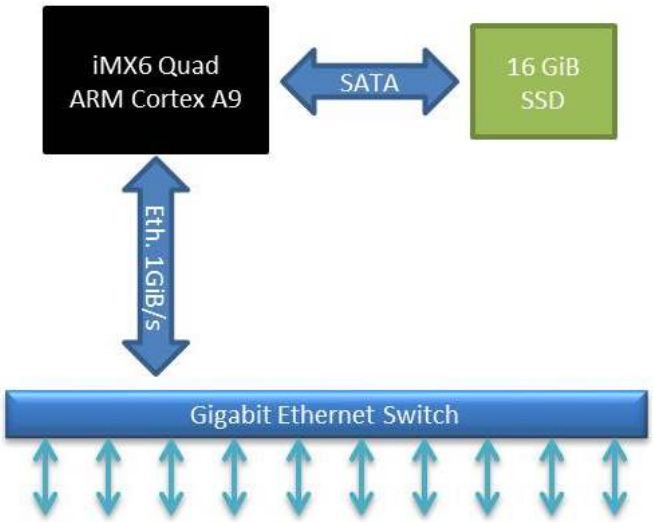

Figure 2.1: Platform Support Module [4]

The PSM Module acts as a router between different modules of the Radar processor. It directs the Radar raw data to an apt processing module (DGPM or SPM) and send the processing result to a target display module.

2.1.2 Data Graphics Processing Module (DGPM)

The DGPM has the following characteristics:

- 4 iMX6Q CPUs, clocked 800MHz.
- one 1GBit/s-Ethernet switch with 6 bi-directional communication ports.
- each of the 4 CPUs has it's own private 4GiB SDRAM and non-volatile NOR/NAND Flash. memory

The iMX6Q6 CPU has four identical ARM Cortex A9 cores with Advanced SIMD unit (NEON), 32KiB I and D-cache, one unified 1MiB L2-cache shared by 4 cores, smart DMA, 3D and 2D

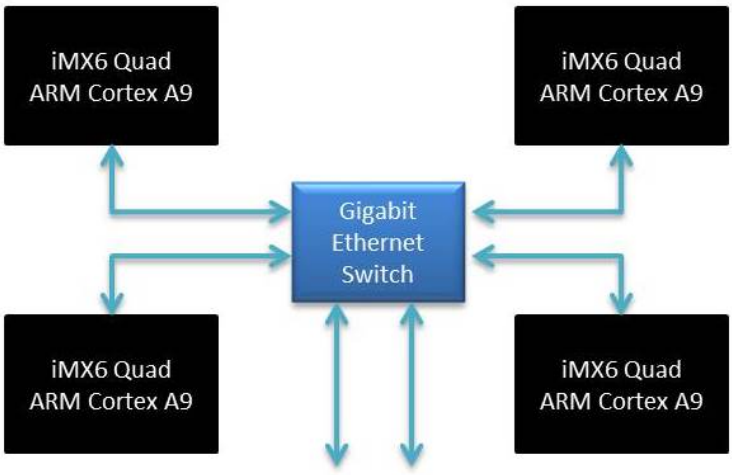

Figure 2.2: Data Graphics Processing Module [4]

Graphics Processing Unit, one common 533MHz 64-bit DDR3 memory interface, one common 1GBit/s Ethernet interface and several other standard interfaces [6], (Appendix A.2). The SPM Module has not been considered due to the decision of Airbus DS.

2.1.3 Radar Processor Design

As shown in Figure 2.3, the received raw data will undergo the following stages to be transformed as desired information.

1. Radar receiver front-end sends the received data to the iCON1 Module along with supplementary information like antenna azimuth, antenna elevation, etc. Data distribution to the PSM modules is controlled by the iCON1.

2. PSM1 and PSM2 routes the received data to the pre-defined DGPM modules. Routing mechanism can be reconfigured.

3. As soon as the DGPM has completed processing, data will be sent back to the respective PSM module.

4. PSM routes the processed data to iCON2.

5. iCON2 routes the processed A/A Mode data, SAR Mode data to the tracking processor, display processor respectively.

Nominal data transfer rate of the communication links are 1GBit/s. Booting the software can be done from SSD or remotely via an appropriate system interface.

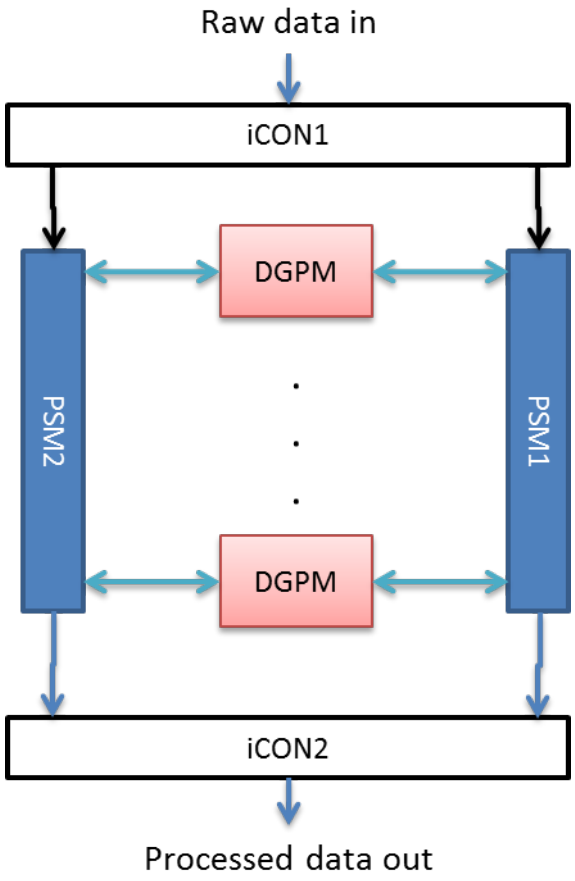

Figure 2.3: Block Diagram of Radar Processor [4]

2.2 Air to Air Mode Processing Chain

Previous work has been carried out by Airbus DS in Future Combat Air System (FCAS) project, see [4]. This section is an excerpt of the FCAS project, describes the Radar processing chain to understand the rest of the thesis.

At first, the Radar system in the flight scans the sky $\pm 60^\circ$ from bore-sight. The azimuth scan of the antenna is sub-divided into 2 x 5 angular segments, called look directions (5 segments in the positive half and 5 segments in the negative half of the scan width). Given the beam width of 2.5°, 48 beams are required to cover the 120° space. Antenna's gain is high along the bore-sight direction [14][15]. So, the look directions adjacent to the bore-sights have 4 beams each and the remaining 8 look directions have 5 beams each, counting 48 beams totally. Figure 2.4 explains the 5 look direction segments in positive half of the azimuth scan. The Air to Air Mode has different PRF set for different look directions. The PRF and range gate configuration determines the amount of input data to the Radar processor.

Table 2.1 shows the maximum execution time of the Radar functions running on a single core, 1GHz iMX6Quad processor. The baseline analysis scales down the results to 800MHz to match the IMA processor architecture.

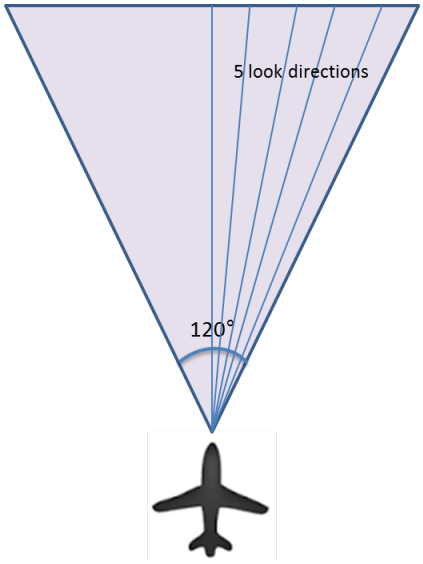

Figure 2.4: Details of the A/A Mode Scan

The Radar antenna has 8 transmit and receive channels and 1 guard channel. The guard channel works as a detector to distinguish main-lobe echoes and side-lobe echoes. Every scan consists several Dwells. All the Dwells are independent. The A/A Mode processing is done independently for each Dwell. It comprises of the following stages.[3]

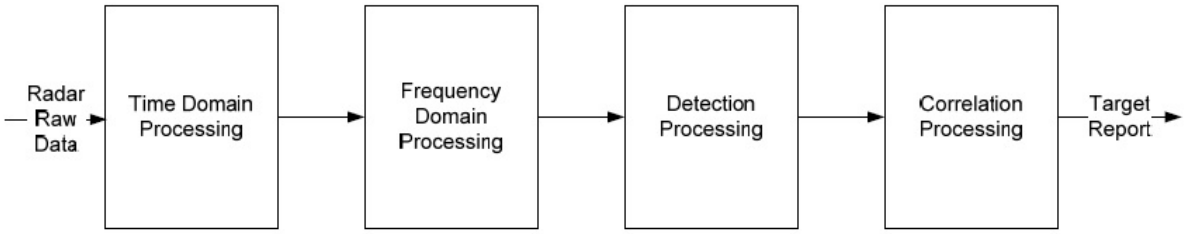

Figure 2.5: Block Diagram of Air to Air Mode Processing

2.2.1 Time Domain Processing

Beam-forming is a Digital Signal Processing technique that allows the Radar antenna to focus on a particular direction. Beam-forming has to be performed for Sum channel(S_S), Azimuth

[3]This section only describes an overview of the algorithm, since the algorithm is classified as Airbus DS Confidential.

channel(S_{AZ}) and Elevation channel(S_{EL}). During Sum channel beam-forming, the input vectors of 8 antenna receive channels are multiplied with pre-calculated Sum channel beam-forming vectors. The resulting vectors are added to produce a Sum channel beam. This step is repeated for Azimuth channel and Elevation channel. Guard (9^{th}) channel doesn't require beam-forming, only requires float conversion. Each of the processed four channel data are multiplied by pre-calculated weighting vector.

Digital Pulse Compression, also called Convolution, is computation expensive in time domain, so it is performed in frequency domain. Convolution in time domain is multiplication in frequency domain, it is performed channel by channel by computing FFT, Multiplication followed by IFFT.

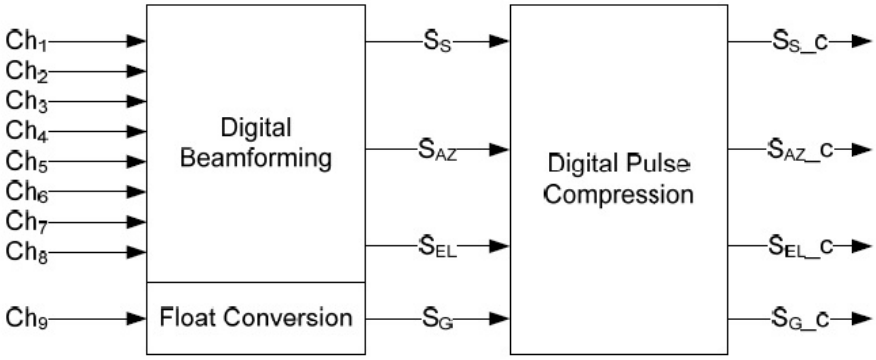

Figure 2.6: A/A Mode Time Domain Processing

2.2.2 Frequency Domain Processing (FDP)

Each channel pulse compressed data is in the form of [pulse x range] matrix. Frequency domain transformation has to be applied on the data corresponding to the same range gates of different pulse. So the pulse compressed data shall be corner turned to [range x pulse] matrix as shown in Figure 2.7.

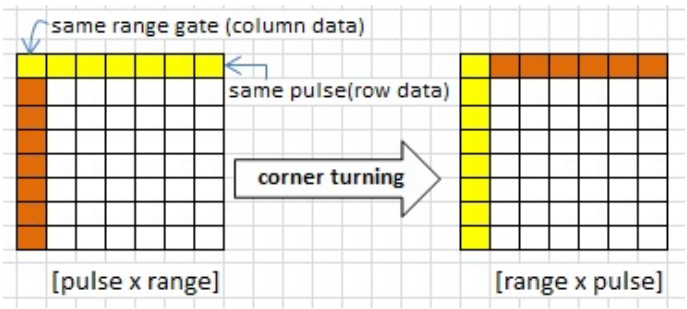

Figure 2.7: Corner Turning

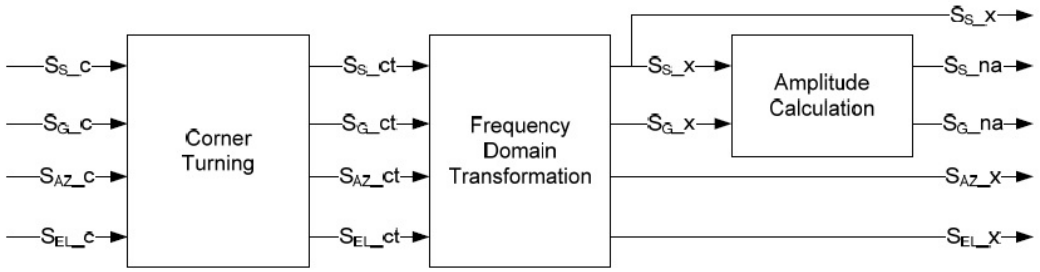

Figure 2.8: A/A Mode Frequency Domain Processing

FFT is computed for the Sum, Guard, Azimuth and Elevation channel followed by magnitude of the Sum channel and Guard channel are computed on the corner turned data..

2.2.3 Detection Processing (DET)

Area threshold of the individual elements in the Sum channel and the Guard channel are determined by computing average of the surrounding elements values. An additional pre-alarm matrix is computed by comparing the amplitude matrix and the area threshold matrix. The pre-alarm says the detection may be a potential target. If the element in the amplitude matrix is greater than the corresponding element in the threshold matrix, pre-alarm is set TRUE in the corresponding pre-alarm matrix, otherwise pre-alarm element is set to FALSE. To prevent the Sum channel alarms caused by side-lobe entries, it is compared with the Guard channel alarms.
The number of alarms are limited by the N_{max_alarms} with the shortest range from the list called Constant False Alarm Rate(CFAR). Alarm list is generated by referring the pre-alarm matrix and corresponding Sum, Azimuth, Elevation channel values.

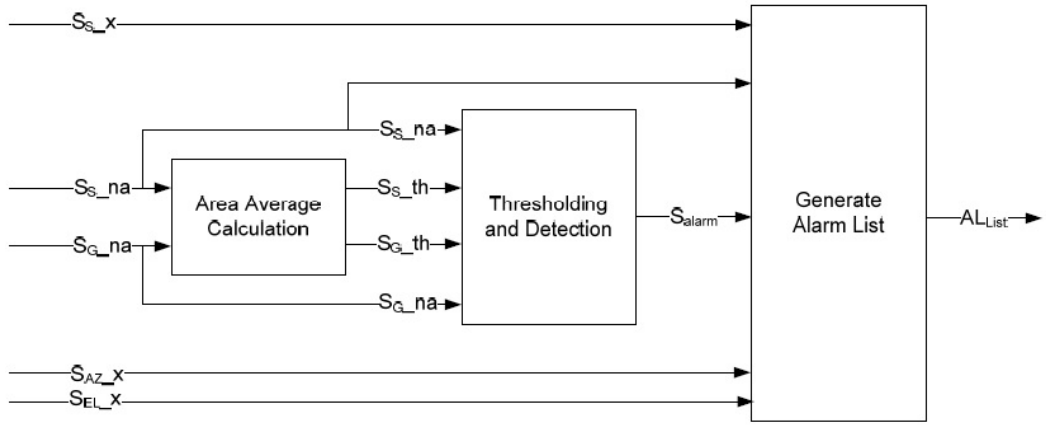

Figure 2.9: A/A Mode Detection Processing

2.2.4 Correlation Processing

This stage resolves the ambiguity in distance and velocity by comparing the current burst data with 7 previous bursts. Correlation processing is not considered as a performance critical module and hence this stage is not benchmarked.

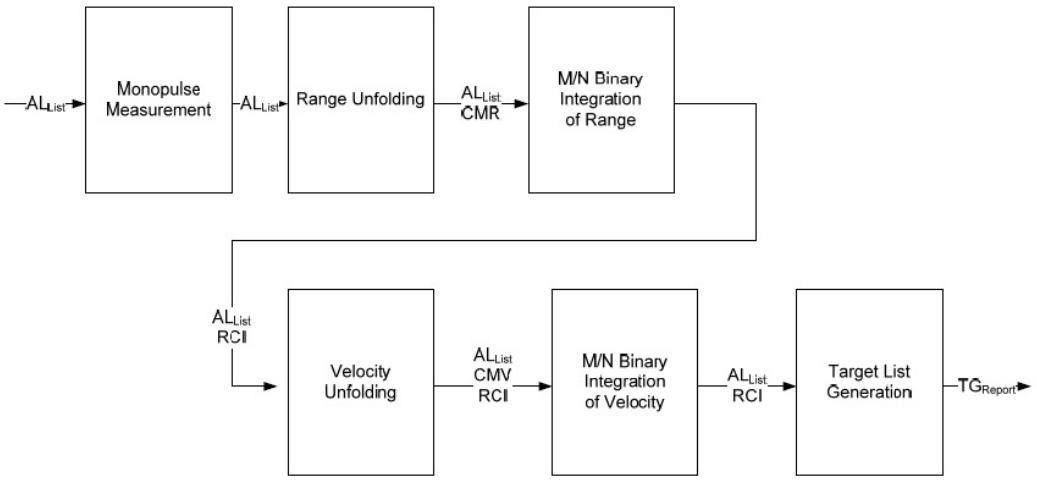

Figure 2.10: A/A Mode Correlation Processing

2.3 A/A Mode Sequence of Execution

Performance critical functions related to the Radar processing algorithm are identified by Airbus DS. The functional blocks are executed in the iMX6Quad processor, clocked 1GHz and their execution times are listed in Table 2.1. The functional block 100CMYACC8, multiplies each element of a [100 x 8] element complex matrix with an 8-element complex vector and accumulate the 8 complex multiplication results to a single complex result, resulting in a [100 x 1] element complex matrix. Explanation about the other functional blocks can be found in Appendix B.

Functional Block	**#cycles in iMX6Quad processor**	**Unit**
100CMYACC8	79	per 8 element
100RMY50	15	per complex element
100CONV128	24100	per 128-pt vector
150COT50	12	per complex element
100FFT64	2,550	per 64-pt vector
50MAG256	20	per complex element

64AVG100	20	per element
64CMPR100	7	per element
64DET100	10	per element

Table 2.1: Measured Execution Cycles of the Functional Blocks

The functional blocks shall be executed as shown in Figure 2.11 to mimic the Radar processing chain. Functions of the Correlation processing are not included here. In the following figure, *3x CMYACC* means, computing CMYACC for three channels (S_S, S_{AZ}, S_{EL}).

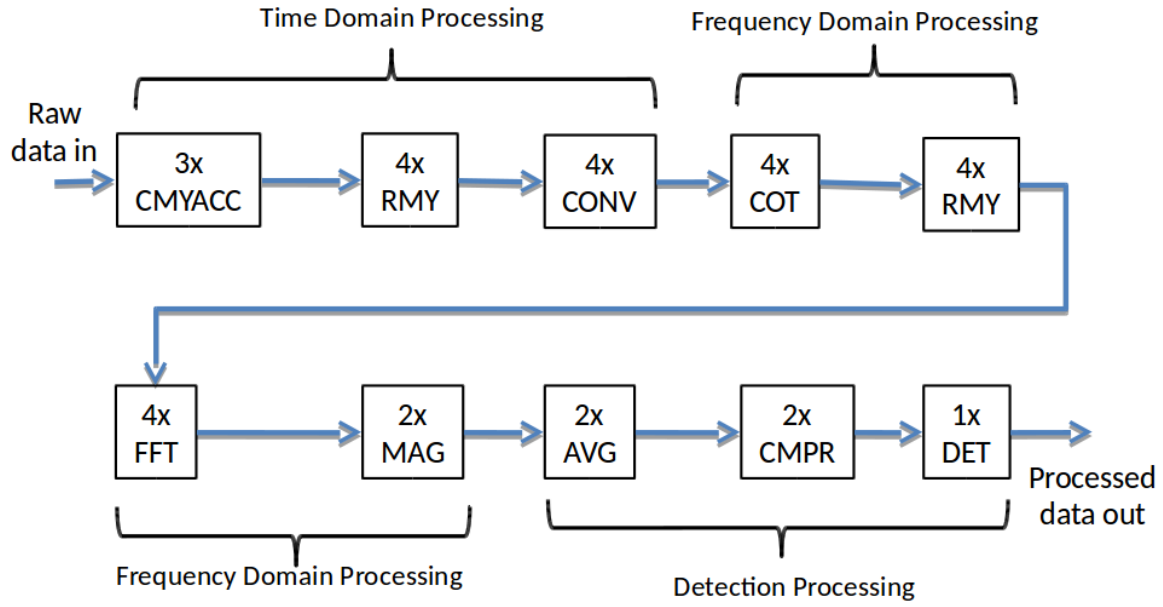

Figure 2.11: A/A Mode, Sequence of Functional Blocks Execution

STREAM benchmark[16] is used to measure memory bandwidth of a machine. The STREAM benchmark is a simple, synthetic benchmark designed to measure sustainable memory bandwidth (in MiB/s) and a corresponding computation rate for four simple vector kernels namely Copy, Scale, Add and Triad [17]. According to the STREAM benchmark, 400 MiB/s data transfer rate is measured while copying data from one memory location of the SDRAM to another memory location, when a single core is active (2cycle/byte). L2 cache to SDRAM data transfer rate of 350MiB/s (2.29cycle/byte) is measured when all the four cores are active. Operating System overhead is assumed as the factor of 1.3 to carry out the execution. Cycle time of the core is 1.25ns, as it is running at 800MHz frequency. Execution cycle for the Correlation

processing is heuristically assumed as follows.

Process	**#cycles**	**Unit**	**#cycles**	**Unit**
Monopulse Measurement	500	per alarm	-	-
Range Unfolding	30	per range gate	200	per alarm
M/N Binary Integration of Range	40	per range gate	500	per alarm
Velocity Unfolding	30	per alarm	-	-
M/N Binary Integration of Velocity	80	per alarm	-	-
Target List Generation	100	per target	-	-

Table 2.2: Execution Cycle of Correlation Processing [4]

The execution cycle results listed in the Tables 2.2 and 2.1 are the fundamentals for the baseline analysis, explained in the next chapter.

2.4 Air to Air Mode Radar Characteristics

Air to Air Mode Radar has the following characteristics. The transmitted carrier frequency is 9GHz and it can detect targets in 40nm range, moving at velocity ±3Mach. A burst shall have a maximum of 32 alarms and a maximum of 16 targets, formed out of the 32 alarms. That is, a burst shall have maximum of 32 detections, of which 16 can be real targets.

Table 2.3 shows the detailed information of the Radar system. The first burst of the Look direction-1 consists of 64 pulses transmitted at 19.5 KHz frequency and every pulse shall have 103 range gates. Range unfolding and Velocity unfolding are performed to compute the unambiguous range and unambiguous velocity of the target respectively. They can be mathematically calculated as follows.

$$
\begin{aligned}
Dwell\ \ time &= \sum_{n=1}^{8} Burst\ \ time_n \\
Unambiguous\ Range &= \frac{1}{2}\,\frac{velocity\ of\ the\ radar\ signal}{PRF\ \ frequency} \\
Range\ Unfoldings &= \frac{max.range}{unambiguous\ range} \\
Unambiguous\ Velocity &= \frac{1}{2}\frac{velocity\ of\ the\ radar\ signal}{carrier\ \ frequency} * PRF\ \ frequency \\
Velocity\ unfolding &= 2 * ROUNDUP\left(\frac{max.velocity}{unambiguous\ velocity}\right)
\end{aligned}
\tag{2.1}
$$

Burst	1	2	3	4	5	6	7	8	
PRF[kHz]	19.50	20.30	22.10	24.10	25.90	29.40	33.80	38.20	
Pulse count for look directions 1...5	64	71	76	85	94	105	117	131	
	76	84	90	101	112	125	139	156	
	91	100	107	119	132	148	164	184	
	107	117	126	141	156	175	194	217	
	126	138	148	165	183	205	228	255	dwell
Range gates	103	99	90	83	77	68	59	52	time[ms]
Burst time[ms] = PRF/pulse count	3.28	3.50	3.44	3.53	3.63	3.57	3.46	3.43	27.84
	3.90	4.14	4.07	4.19	4.32	4.25	4.11	4.08	33.07
	4.67	4.93	4.84	4.94	5.10	5.03	4.85	4.82	39.17
	5.49	5.76	5.70	5.85	6.02	5.95	5.74	5.68	46.20
	6.46	6.80	6.70	6.85	7.07	6.97	6.75	6.68	54.26
#range unfoldings	10	11	11	12	13	15	17	19	
#velocity unfoldings	8	8	6	6	6	6	4	4	

Table 2.3: A/A Mode Radar Characteristics

Active Electronically Scanned Array (AESA) Radar is an airborne Radar, capable of steering beams rapidly [18]. In case of AESA Radar, processing latency influences agility of the system. Beam steering is performed electronically in AESA Radar compared to the mechanical steering in conventional Radars. Dynamic carrier frequency and PRF characteristics of the AESA Radar makes it hard to be intercepted by Radar Warning Receiver [19].

2.5 Related Work

Deploying ARM processors in fighter aircraft is in the research stage at the moment. Footprints of the low cost Radar processor and optimal scheduling algorithm have been considered as related work of this thesis. Many innovative architectures have been investigated in past to reduce the size, weight and power(SWaP) requirements of the Radar processor. Some of them are discussed below.

In continuation to the bird strike happened in Alaska, a low cost Radar system "eBirdRad" is developed by Tim J. Nohara, et al [20], focused to detect birds activity in the sky. The Radar uses X-band frequency, 4° beam width, 10m resolution, covering 360° azimuth angle and 6nm distance. Proprietary Accipiter Radar processor is used for computations. eBirdRad could detect the birds in real time though there are rooms for improvement. It doesn't aim to reduce weight and power requirements, hence it is not a perfect rival for ARM based Radar processors.

Salama Y, Fitzgerald D et al [21], have used Wafer Scale Signal Processor(WSSP) for power efficient Radar signal processing. WSSP is a general purpose floating point processor with 4FLOPS/cycle peak performance, consuming 2 to 10 GFLOPS/watt roughly. Simulation re-

sults of FFT benchmark shows that up to 3.2FLOPS/cycle can be utilized for 100k FFT size. The WSSP processor has made efforts to reduce power requirements, disregarding size, weight and cost factors.

Research has hit the industry with *Programmable DSP Cores for Radar Processing*. APTCORE [22], uses patented processor architecture to efficiently process the Radar benchmarks. It claims to have 3.3x energy efficiency in FFT processing compared to the standard DSPs. According to the documentation, it is compact, low cost, lowest power, scalable and highly configurable. In addition, range and direction processing of 10 sets of 512 values across 16 antennas take 2.3ms at a clock speed of 200MHz. Although the APTCORE has impressive performance in the product brief document, further investigation is required to compare it against ARM Cortex A9's performance.

ARM type processor has been used by Zeng Y, Xu J and Peng D in Velocity Measuring System [23]. Implementation is done in assembly language to take advantage of the pipeline. It uses 24GHz frequency waves and can track the vehicle velocities from 5km/h to 250km/h in 1.5km range. It is a low cost, low power, compact system meant for traffic management.

Cheng C, Chien-Chung C et al [24] proposed a real-time scheduling algorithm which manages resources of a specially designed parallel system, named programmable radar signal processor (PRSP), a special DSP processor. It is designed to be reconfigurable and applicable to most of the Radar systems. It efficiently allocates the tasks to the processing units, but the SWaP features are not addressed.

ZHAN H, YUAN L, and WANG L [25] have proposed an improved allocation algorithm for Radar task allocation in distributed heterogeneous system using Hungarian algorithm. It uses scheduling queue, workload real-time detection and task accumulation threshold to estimate processor workload and allocate the task accordingly. Again, this paper makes better use of the highly capable resources, disregarding SWaP.

The aforementioned researches either concentrate in the direction of ARM processors to realize low cost and compact Radar processor or efficiently scheduling tasks to the available special purpose processors or using dedicated processors. They succeeded in doing so, but they are not meant for safety critical systems with a reasonably good range and resolution and less SWaP. This thesis connects the both ends, taking advantage of less SWaP from ARM processors and optimal scheduling scheme technique from efficient task allocation to bring less SWaP Radar processor for safety critical system.

Chapter 3

Baseline Analysis

This chapter explains the baseline analysis carried out by Airbus DS[4]. An airborne Radar processor should be capable of processing both the Air to Air Mode data as well as Air to Ground Mode data. The scope of this thesis is limited to A/A Mode processing. Implementation is not done for the baseline analysis. So, it only estimates the approximate processing latency, CPU utilization and memory utilization with reference to the functional block execution cycle listed in the Tables 2.1 and 2.1. The baseline analysis configures scheduling schemes in accordance to Space Partition and Time Partition.

Time Partitioning:
One iMX6Quad CPU is time sliced to run A/A mode and A/G mode alternatively. The timeframe between two subsequent A/A mode execution is called major time frame. Memory and cache are also partitioned for each mode. Intra-partition communication can be done by buffers, semaphores and/or events. When the time slice is completed for Air to Air mode, context switch happens to store the details of A/A Mode and load the details of A/G Mode, afterwards the A/G mode begins execution. Violation of timing behaviour triggers an exception.

Space Partitioning:
A/A mode processing and A/G mode processing are done in separate physical entities. It is the simplest configuration for dedicated data processing. Failure of one of the systems will not affect the other as they are independent. It is assumed that SDRAM and L2 cache are partitioned for 4 cores to improve determinism. Each core has pre-defined accessible address space in Memory and L2 cache.

3.1 Scheme 1 - Space Partitioning

A Radar echo received by the Radar antenna is distinguished as A/A mode or A/G mode by the iCON1 module.

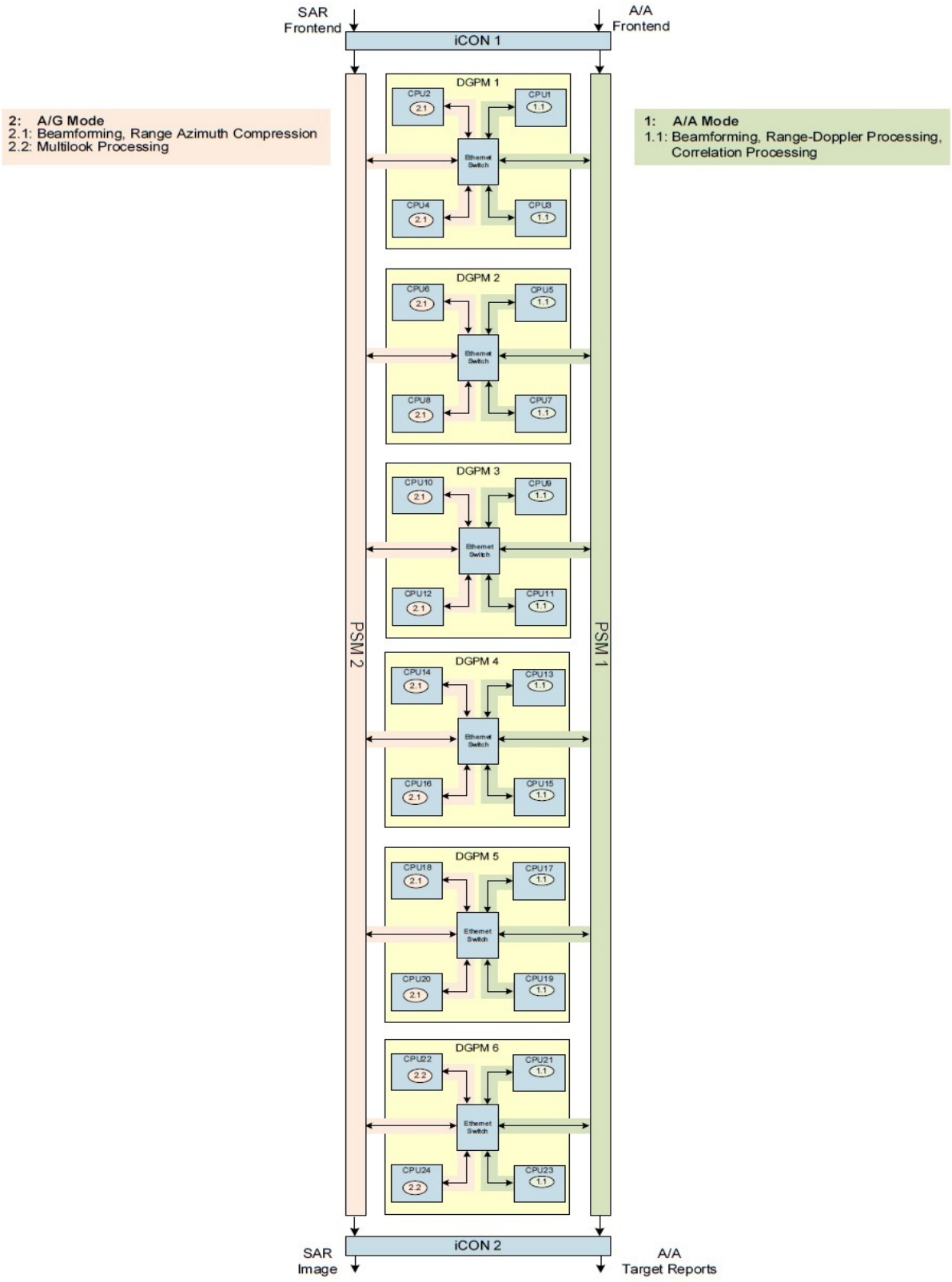

Figure 3.1: Scheduling Scheme

A/A mode data are redirected to PSM1 and A/G mode data are sent to PSM2. Figure 3.1 shows the mapping of A/A Mode and A/G Mode to the IMA processor architecture. The two odd numbered CPUs(CPU1 and CPU3 of DGPM1) of a DGPM are allocated to the processing of the A/A mode. The two even numbered CPUs(CPU2 and CPU4 of DGPM1) of a DPGM are allocated to the processing of the A/G mode. Each CPU with it's 4 cores is completely used for either A/A Mode or A/G mode processing.

Number of DGPMs:	6
Number of CPUs:	6 x 4 = 24
CPUs for A/A Mode:	12
CPUs for A/G Mode:	12

DGPM sends the processed data to iCON2 via respective PSM. iCON2 redirects A/A data to the Tracking processor and A/G data to the Display processor. The SDRAM is assumed to be partitioned for all the four cores, where every core has its buffer memory intended to transfer data between the cores and storage memory to store the data for Radar processing.

3.1.1 A/A Mode Results

Each CPU processes a Dwell of data. The processing results are dispatched to iCON2 by PSM1. As the Dwell data processing is independent to the processing of the predecessor Dwell data and independent to the processing of the successor Dwell data, there is no data dependency between CPUs processing different Dwell data.

Scheduling Scheme

Scheduling functional blocks in the IMA processor architecture is shown in Figure 3.2. As discussed earlier, only odd numbered CPUs (CPU1,3,5...23) take part in processing A/A Mode algorithm.

Core#1 receives incoming data from the Ethernet and stores them into Core#1's buffer memory in SDRAM. Then the data is copied to Core#2's buffer memory.

Core#2 transfers the data from its buffer memory to storage memory to L2 cache and then performs Beamforming. The results of the processing are stored back to the core#3's buffer memory in SDRAM.

Core#3 reads the data from its SDRAM partition and performs Pulse Compression. The results of the processing are stored back to the SDRAM.

Core#4 gets the data from its SDRAM partition and performs FFT, CFAR, Correlation Processing. The results of the processing are stored back to the SDRAM.

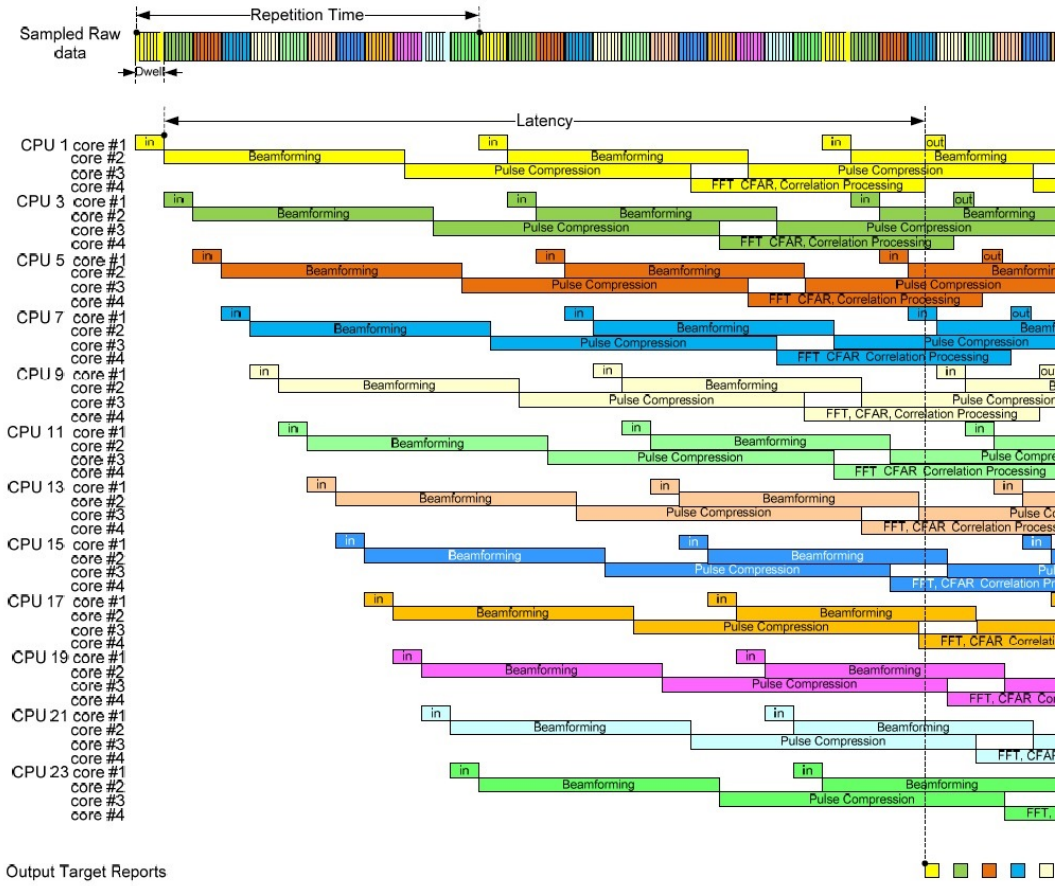

Figure 3.2: A/A Mode Processing

CPU Utilization

CPU utilization is the ratio of processing time to the available time. The worst case available time of a CPU is the time span between receiving two shortest Dwells by the CPU, calculated as 12x shortest Dwell time. The results reported here are rounded to two decimal places. The burst configurations and processor parameters are stated in the Chapter 2.4. Calculations for the first burst of the look direction-1 is explained in Appendix D.

CPU utilization is shown in Figure 3.3 for each Core. On every Core, Look Direction-5 contributes to the highest(worst-case) processing time. It is evident from the figure that the Core#1 and Core#2 are utilized less than 25%, meaning that the I/O processing and Beamforming are not computation intensive. On the other hand, Core#3 and Core#4 are utilized 65% and 75% of the available time, meaning that there is very less room to adapt future growth. Core#3 and Core#4 violates the Radar processor requirements in terms of CPU utilization.

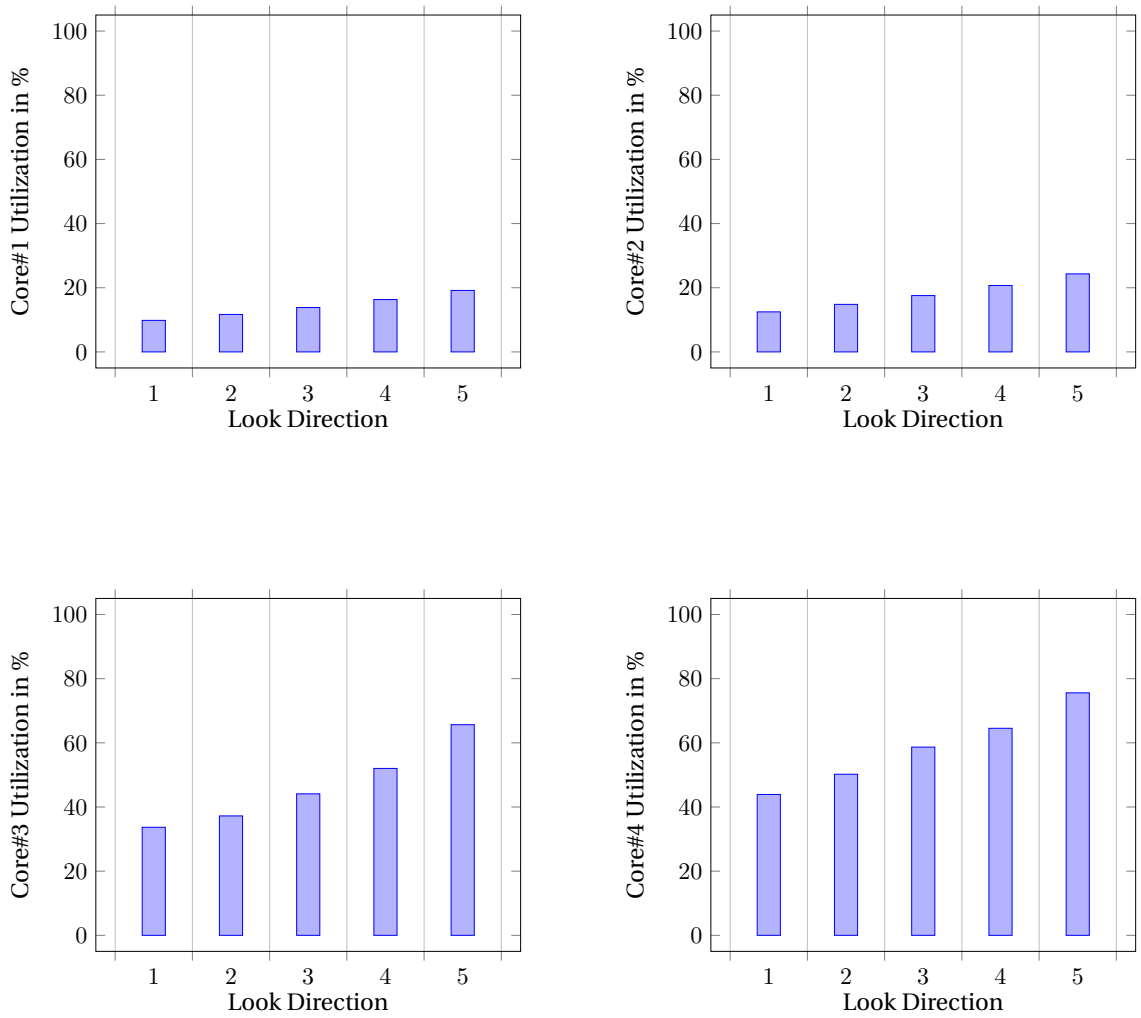

Figure 3.3: CPU Utilization

Processing Latency

Processing Latency is the time between reception of a Dwell and sending of processed result data. As listed in Table 3.1, processing latency between 333ms and 617ms is achieved depending on the Look Direction. Derivation of Look Direction-1's processing latency is explained in Appendix D.1.5. Number of Dwells the Radar can transmit, while the processor is busy in performing computations of a first received Dwell, is given as *#Dwells transmitted*. Higher the *#Dwells transmitted* implies that higher the stagnation of received echoes, which decreases the Radar processors response time. According to the Radar processor requirements, *#Dwells transmitted* should be less than 2.

Time required to process a complete Dwell relate to the look direction-1 is shown in Figure 3.4.

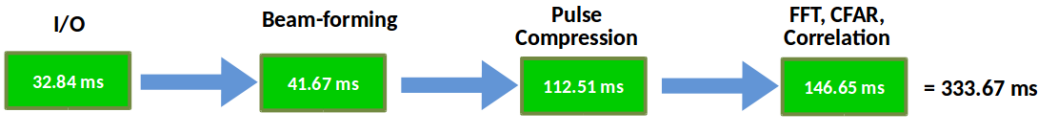

Figure 3.4: Dwell Processing time of Look Direction-1

Look direction	Dwell time[ms]	Latency[ms]	#Dwells transmitted
1	27.84	333.67	11.99
2	33.07	380.61	11.51
3	39.17	448.16	11.44
4	46.20	513.00	11.10
5	54.26	617.05	11.37

Table 3.1: A/A Mode Processing Latency

Memory Utilization

Each CPU has 4GiB of externally connected SDRAM. It is assumed that 3GiB are allocated for OS, storing executable code, etc. 1GiB are available for the actual data processing. Input and output data size for the cores and calculation of the memory requirement are shown Appendix D.1.7. According to the estimation, 7% of the available capacity is sufficient for A/A Mode processing.

Interface Utilization

Interfaces are the data routing paths in the Radar processor. Nominal bandwidth of 100MiB/s data rate is assumed for the interfaces. Peak interface utilization is calculated as 72% of the available 100MiB/s. The calculations are listed in Appendix D.6.

Summary

The CPUs in the DGPM are physically separated for A/A Mode and A/G Mode; accordingly the results will not change if the Radar processor is performing both the modes simultaneously. Space partitioning has 12x Dwell latency, utilizing 7% of the available memory, 72% of the interface and the following CPU utilization factors. Scheme-1 scheduling technique violates the requirements of the Radar processor in terms of processing latency, hence it is not applicable for real-time processing.

	Core#1	**Core#2**	**Core#3**	**Core#4**
Utilization	19.16%	24.32%	65.65%	75.58%

Table 3.2: CPU Utilization

3.2 Scheme 2 - Time Partitioning

Every CPU in the IMA processor architecture runs both the A/A mode and A/G mode application concurrently. CPU time and resources are shared for both the applications. Memory is partitioned for each mode to provide segregation. This analysis assumes

- All the four cores of a CPU can access SDRAM without interfering each other.
- L2 cache is statically partitioned at compile time, partition values are stated in Figure 3.6.
- Partition change time is 0.5ms.

Figure 3.5 shows the time partitioning of a CPU. Each application has to complete processing and suspend itself before the time slice expires. Otherwise a "deadline miss exception" is triggered.

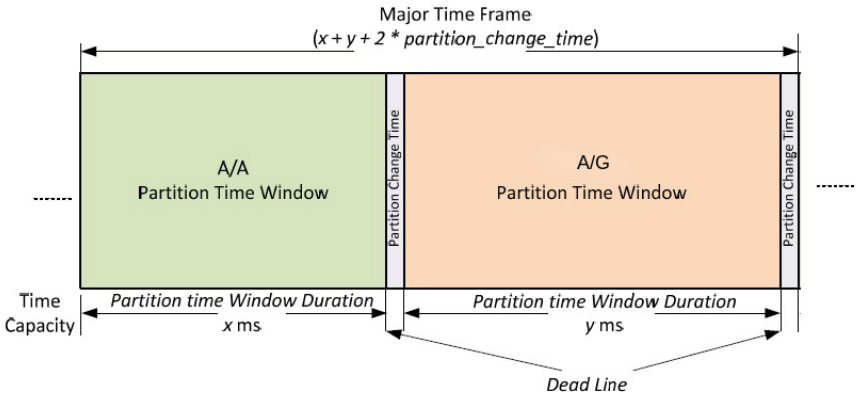

Figure 3.5: Time Partition

A/A data and A/G data are distributed by iCON1 to PSM1 and PSM2 respectively. Processed A/A data are transferred to iCON2 through PSM1. Partially processed A/G results from CPU1...CPU20 are transfered to CPU21...CPU24, where final A/G processing is carried out, followed by the results of the A/G processing are sent via PSM2 and iCON2 to a display.

	Major Time Frame capacity [ms]	X A/A Appl. Partition Time capacity [ms]	A/A L2-cache Partition [Byte]	Partition Change Time [ms] A/A -> A/G	Y A/G Appl. Partition Time capacity [ms]	Partition Change Time [ms] A/G -> A/A	A/G L2-cache Partition [Byte]
Core1	40	19,50	65.536	0,50	19,50	0,50	65.536
Core2	20	5,50	65.536	0,50	13,50	0,50	65.536
Core3	40	25,50	65.536	0,50	13,50	0,50	65.536
Core4	20	14,50	524.288	0,50	4,50	0,50	131.072

Figure 3.6: Values for Time Partition

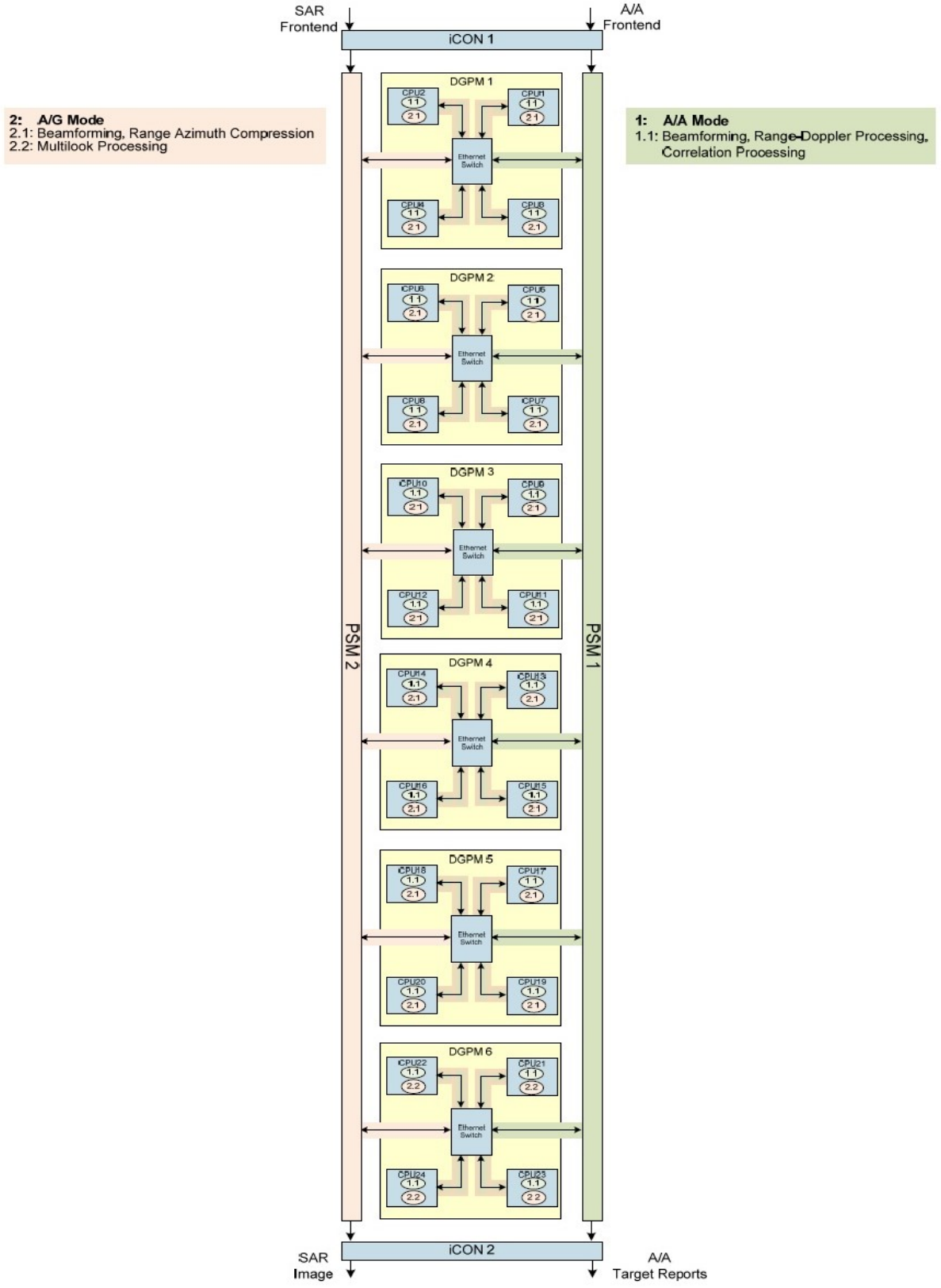

Figure 3.7: Scheduling Scheme

3.2.1 A/A Mode Results

A/A Mode configuration is similar to the configuration discussed in Scheme-1, (see Chapter 3.1.1) except that the data is distributed to 24 CPUs. There may be some additional time slots required to fit the processing into the allotted time slice. For instance, 33 time slots would provide sufficient time to complete a certain processing having 65536 loop counts. Therefore 1985 loops could be processed in each time slot. An additional 34^{th} time slot would be required to execute the remaining 31 loops. This is called Time Slot Adjustment. Calculations are same as Scheme-1 and hence details of the cores processing time are not shown for simplicity.

CPU Utilization

Available time is 24x shortest Dwell time. Since the core is time sliced, effective available time is reduced by the factor of time slice and the processing latency is increased by the same factor. Table 3.3 lists the peak utilization values of each core.

	Core#1	**Core#2**	**Core#3**	**Core#4**
Core Utilization	25%	53%	71%	63%

Table 3.3: CPU Utilization

Processing Latency

Processing latency is higher than Scheme-1, because a core gets approximately half of the available time for A/A Mode processing. Processing latency between 640ms and 1100ms is achieved depending on the look direction, which is also listed in Table 3.4.

Time required to process a complete Dwell relate to the look direction-1 is shown in Figure 3.8.

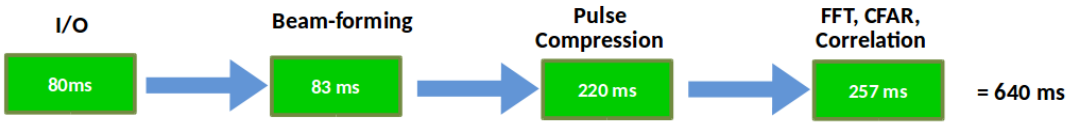

Figure 3.8: Dwell Processing time of Look Direction-1

Look direction	Dwell time[ms]	Latency[ms]	#Dwells transmitted
1	27.84	640	22.99
2	33.07	700	21.17
3	39.17	820	20.93
4	46.20	880.00	19.05
5	54.26	1100	20.27

Table 3.4: A/A Mode Processing Latency

Memory Utilization

Each CPU has 4GiB of externally connected SDRAM. It is assumed that 3GiB are allocated for OS, storing executable code, etc, 1GiB are available for data processing. The peak memory utilization is calculated as 9% of the available memory.

Interface Utilization

Data distribution scheme is the extended version of Scheme-1, hence the peak interface utilization remains same as 72%.

3.2.2 Summary

The Time Partition configuration has 23x Dwell time latency, utilizing 63% of the CPU, 9% of the memory and 72% of the interface capability. Scheme-2 also failed to fulfil the Radar processor requirement. Neither of the baseline analysis scheme satisfies the real-time requirements. A Radar processor, having 23x Dwell time processing latency is not a good choice for airborne Radar processor. Next chapters explain optimal scheduling schemes to bring down the processing latency to an acceptable level.

Chapter 4

Test Bed and Design Decisions

4.1 Pros and Cons of the Baseline Analysis

The contribution of this thesis starts with scrutinizing the Baseline Analysis. Upsides and downsides are investigated and their validity are verified.

4.1.1 Pros

Baseline analysis has considered every possible delicate detail to compute the worst case execution time. Upsides of the analysis are discussed here.

CPU Utilization Balancing

In time partition scheme, CPU utilization factors for A/A mode and A/G mode can be well balanced by configuring time slot period. For instance, in space partition scheme, Core#4 is utilized 75% in A/A mode processing and 25% in A/G mode processing. This information gives a hint for time partitioning that A/A mode needs more time span than A/G mode in Core#4. It is adopted in Scheme-2, allocating 14.5ms for A/A mode and 4.5ms for A/G Mode to balance the CPU utilization. Dynamic partition configuration can be considered in future to improve CPU utilization balancing.

Memory Partitioning

In time partition scheme, shared resources (L2 cache, Memory) are partitioned for A/A Mode and A/G mode to provide segregation. This isolation clearly defines the accessible address space for each core. Even if A/G mode processing of a core consumes huge memory space, A/A mode data of the core remains intact and will be useful during the next time slice. Another advantage of memory partitioning is, it allows utilization balancing among the memory resources.

4.1.2 Cons

Context Switch Time

Context switch time of 0.5ms is assumed for time partitioning. Every core does 2 context switches in 20ms. In time partition analysis (Scheme-2), at the end of 19.50ms all the four cores want to perform context switching. It is illustrated in Figure 4.1. Since the SDRAM has only one port, it is possible for only one core to access the memory. Other cores have to wait till that time.

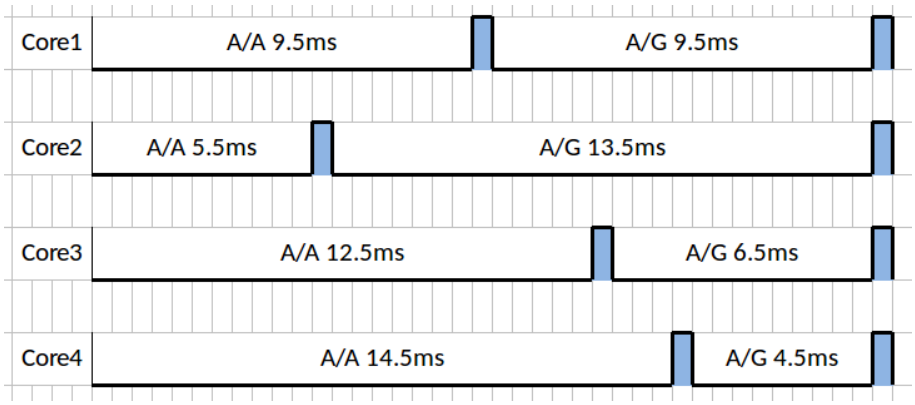

Figure 4.1: Time Partition of Scheme 3

Assuming the order of SDRAM access is Core#1, Core#2, Core#3 and Core#4. Core#2 waits for Core#1 to complete (0.5ms), Core#3 waits for Core#1,2 (0.5ms + 0.5ms), Core#4 waits for Core#1,2,3 (0.5ms + 0.5ms + 0.5ms). This overhead in waiting time sums up to 3ms. Though only one context switch is contending, during the course of run it will be a scenario where two context switches will contend for SDRAM because of its skewed timing behaviour. Worst-case waiting time for four cores is

$$= \frac{4 * T_{cs} \quad + \quad 2 * T_{ow}}{T_a}$$

$$= \frac{4 * 2 * 0.5ms \quad + \quad 2 * 3ms}{4 * 20ms} = 12.5\% \tag{4.1}$$

Legend

T_{cs} : Context-switch time

T_{ow} : Overhead in waiting time

T_a : Available time

In the worst case scenario, 12.5% of the CPU time is spent only for context switching.

This percentage is severe for a Radar processor. It degrades the application performance and much meaningful processing can be done during that time.

Verification: LMbench microbenchmark[26] is used to measure the context switch time of the ARM cores. LMbench is a free software suite of simple, portable benchmarks to compute various system performances including memory copy bandwidth, context switch latency, system call overhead, process creation latency, etc. Different sets of processes and data size are examined for the context switch time measurement. The LMbench suggests that the lowest recorded context switch time measurement is the more realistic one. So, lowest value of the 20 measurements are computed and listed in Table 4.1. 7.43μs in the table belongs to the context switch time measurement of two processes having 8KiB working data set each. The processes are scheduled according to the scheduling policy of the operating system. By monitoring the `top` command, it is evident that the cores execute processes simultaneously, i.e. when four processes are involved in context switch, four cores are performing the execution.

Data Size [KiB]	**Context Switch Time[μs]**			
	2p	**4p**	**8p**	**16p**
8	7.43	9.09	12.76	13.58
16	8.22	16.35	19.31	20.87
32	9.57	18.48	23.68	29.13

Table 4.1: Measured Context Switch Time

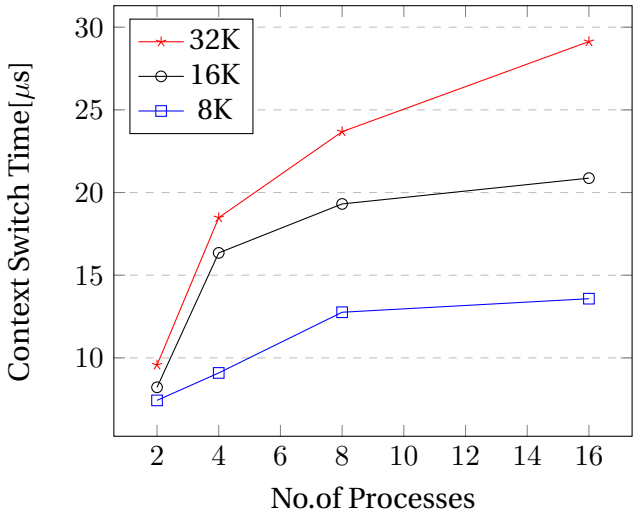

Figure 4.2: Comparison of Context Switch Time

From the above three sets of measurements, worst-case context switch time is 29.13μs

corresponds to the 16 processes operating on 32KiB memory size. Also it reveals that the context switch time is proportional to the number of processes and data set size. On contrary to the assumption of 0.5ms context switch time and 12.5% degradation in CPU utilization, measured worst-case context switch time is much lesser, therefore will not deteriorate the CPU utilization. So, no changes are made in the Baseline Analysis with respect to the context switch time.

Clashing Data Streams

In time partition scheme, assume that the Core#1 of CPU1 is executing A/A Mode time slice. Now, A/G Mode data is routed to the same CPU by PSM2 as shown in Figure 4.3. The core cannot accept the A/G Mode data as it is executing A/A time slice and the PSM have no idea of what the cores are up to or what to do with the data if the core is not accepting. This corner case is not clearly defined in the Baseline Analysis, as a result it leads to data loss if no countermeasure is provided.

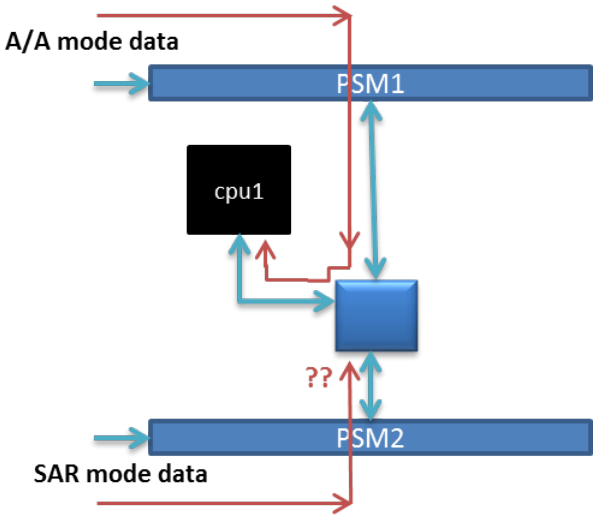

Figure 4.3: Clashing Data Streams

One of the ways to resolve this is to have a buffer in PSM to store the incoming data stream. If the core is not accepting the data, corresponding `core_id` and `cpu_id` shall be stored in the PSM along with the data. When the core is requesting data during next A/G Mode time slice, the PSM can transfer the stored data if the `core_id` and `cpu_id` matches. Since the incoming data stream is stored for one time slice period, it has to be counted for the processing latency calculation. The storing and restoring scheme ensures that the A/A data and A/G data will never be routed to the same CPU at the same time.

Scalability

For time slot adjustment, it is cumbersome task to break the entire application into pieces such that each and every piece will fit in to one time slot. Whenever there is a change in time slot period, the entire application requires rework to suspend itself before the time slot expires. It implies that the time and cost of the application development and maintenance will increase. In addition, amendments are needed if the Radar configuration such as PRF set is changed.

4.1.3 Other Comments

4.1.4 Measured Values of the Baseline Analysis

The execution cycles of the functional blocks gives in the previous chapter (Table 2.3) were not the same when measured in a real hardware. The measured execution cycles are adapted in the Baseline Analysis and the results are presented here.

Scheme-1

	Core#1	**Core#2**	**Core#3**	**Core#4**
Utilization	19.03%	46.20%	65.78%	108.07%

Table 4.2: Measured CPU Utilization of Scheme-1, A/A Mode

Look direction	**Dwell time[ms]**	**Latency[ms]**	**#Dwells transmitted**
1	27.84	416.40	14.96
2	33.07	482.28	14.58
3	39.17	574.52	14.67
4	46.20	662.80	14.35
5	54.26	798.62	14.72

Table 4.3: Measured Processing Latency of Scheme-1, A/A Mode

Core#4 of the A/A Mode processing CPUs in Scheme-1 required more than the available time to process one Burst of the input data. It leads to unfaithful result as it cannot cope up the incoming data stream.

Scheme-2

	Core#1	Core#2	Core#3	Core#4
Utilization	25.00%	103.33%	71.43%	83.33%

Table 4.4: Measured CPU Utilization of Scheme-2, A/A Mode

Look direction	Dwell time[ms]	Latency[ms]	#Dwells transmitted
1	27.84	860.00	30.89
2	33.07	920.00	27.82
3	39.17	1060.00	27.06
4	46.20	1220.00	26.41
5	54.26	1520.00	28.01

Table 4.5: Measured Processing Latency of Scheme-2, A/A Mode

Here, Core#2 is utilized beyond its limits and the Dwell time latency reaches up to 30x Dwell time. Both the Schemes will not adhere to the real-time requirements.

4.2 Test Procedure

A Nitrogen6X development kit is used to match the basic building block of the IMA processor architecture, which is iMX6Quad processor, clocked 1GHz. The communication to the Nitrogen6X board is carried via Ethernet interface. Measurements of the processing latency is performed on the target and exported in `.csv` format. The results are scaled down to 800MHz to match the IMA architecture. Scheduling schemes and processing latency analysis are based on the single iMX6Quad processor result, measured from the Nitrogen6X board.

4.2.1 Target Hardware

The target hardware is a Nirtogen6X development board (Appendix A.1) from Boundary Devices Inc. It is a low cost development kit built with iMX6Quad processor. Some of it's features are:

- Quad-Core ARM Cortex A9 processor clocked 1GHz
- 1GiB of 64-bit wide DDR3 at 532MHz
- Three display ports (RGB, LVDS and HDMI)
- Serial ATA 2.5 (SATA) at 3GBit/s
- Dual SD 3.0/SDXC card slots
- 10/100/1000 Ethernet
- 10-pin JTAG interface
- High speed USB ports (2xHost, 1xOTG)
- Real-Time Clock with battery backup

4.2.2 Software Development Platform

Linaro Linux is installed on the Nitrogen6X board along with Lightweight X11 Desktop Environment. Functional blocks of the Radar processing algorithm are written in `C`. Other details are:

- `Linaro Linux 3.0.35-02828-g5cedf96` is installed on Nirtogen6X board.
- Software Development Environment: Eclipse CDT Version: 3.8.0.
- C compiler: `gcc Debian 4.7.2-5`.
- Cross compiler:
  ARM A9 cross-compiler `gcc-linaro-arm-linux-gnueabihf-4.8-2013.09_linux`.
- Other application library: FFTW 3.3.3 [27].

Measurements of the Radar processor requirement factors such as Processing latency, Memory transfer bandwidth, Peak memory utilization and CPU utilization are performed as follows.

4.2.3 Processing Latency (Worst Case Execution time)

Proper way of measuring the worst case execution time includes the following steps:

- All the cores are set to begin the execution simultaneously to produce maximum communication bandwidth.
- Cache data are invalidated before and after the measurements to ensure that the data is always fetched from memory.

The below mentioned methods are followed to improve the determinism:

- Every core executes a predefined thread.
- Radar application is given higher priority than other user space programs.

In worst case scenario, all the four cores of a CPU will execute the same functional block, generating maximum communication bandwidth, L2 cache contention and memory contention. To ensure that all the cores are executing the same functional block, a synchronization function is provided (Appendix C.2). A thread reaches this synchronization function waits until all the other threads to join, afterwards all the threads begin executing functional block simultaneously.

Invalidating cache data from user space is not allowed in ARM Cortex A9 processor. On the other hand, writing a program that copies huge junk of data from one memory location to other memory location seems to clear the cache contents. But, because of the complex caching strategies, it is not guaranteed that the entire cache data are removed, thus leaving

major portion of the cache intact.[1]

To overcome this, the Radar functional block sequences are executed for 1000 iterations to measure the maximum execution time. A graph is drawn displaying the measured execution times during this 1000 iterations. However, there are some unexpected spikes ranging five times higher than the average execution time, which occurs at sporadic intervals. The Nitrogen6X board is running `ssh, GUI, network-manager`, background measurement tasks and other OS services along with the functional block application. It can be reasoned that any of the other applications interrupts a running thread, causing it to be blocked for a while. So, the measurement is showing unrealistic values. The measured execution time of Comparison (CMPR256) functional block with spike and the resulting execution time after removing the spike are shown in Figure 4.4. Every core runs one instance of the CMPR256 functional block for 1000 iterations. So, four core runs four instances, totally counting 4000 iterations, listed in x-axis of the graph.

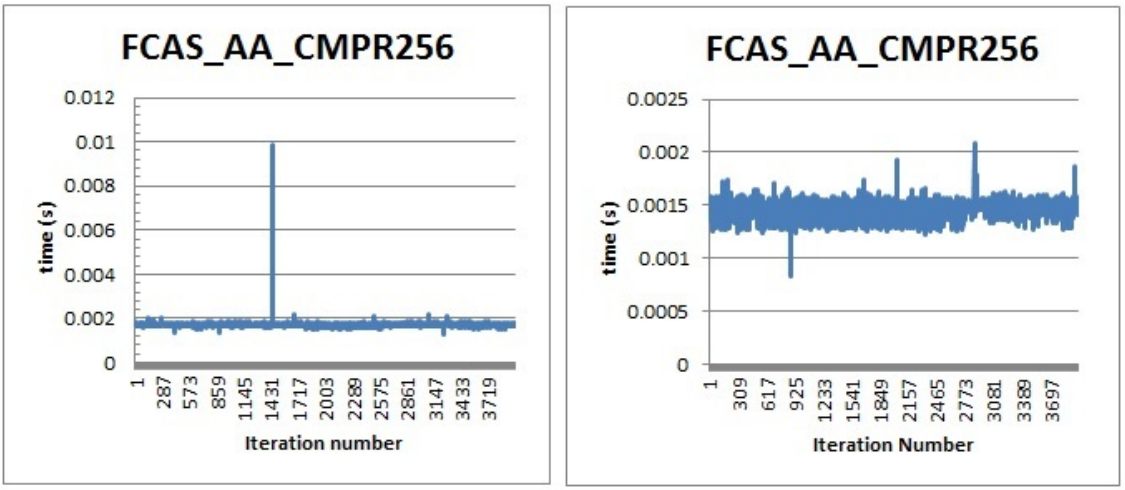

Figure 4.4: Before and After Spike Removal

A test run is made to study the effect of background tasks and services on the spikes. The background measurement tasks and services other than `ssh, network-manager` are stopped. Now, only the minimal OS services are running. The Radar application is allowed to run in such less competitive environment. The result shows that the spikes are still occurring in few functional blocks, but the magnitude has reduced from 5x to 2x. As expected, execution time results of some short lived functional blocks do not have the spikes at all. It is concluded that the OS applications or services might have pre-empted the Radar application. Those spikes

[1] A discussion about this in ARM Connected Community forum can be found at `http://community.arm.com/thread/8799`

are removed from the graph and then the maximum execution time is computed.

To effectively utilize all the available four cores, four threads are spawned from main thread to carry out the parallel execution. Core affinity of each thread is set such a way that four threads are mapped to four cores (Appendix C.1). For example, thread1 will run on Core#1, thread2 on Core#2, thread3 on Core#3 and thread4 on Core#4. The Radar application is given higher priority than other user space tasks to give precedence.

4.2.4 Memory Transfer Bandwidth

Peak achievable memory bandwidth is measured by running threaded STREAM benchmark on Nitrogen6X board. Only STREAM benchmark is started by the user to guarantee that the measurement is not affected by other user applications.

The measurement result of copying data from one memory location to another memory location is used for the analysis. To measure the peak data transfer rate, the best result from 5 consecutive run is computed. As shown in Table 4.6, the iMX6Quad is capable of transferring **1048 MiB/s** when all the four cores are running in parallel.

Function	Best Rate [MiB/s]
Copy	1048.7

Table 4.6: Idle Memory Transfer Bandwidth

To compute the peak memory bandwidth of the Radar application, STREAM benchmark is allowed to run in background along with the Radar application. STREAM is given low priority to give precedence to the Radar application. When the Radar application is consuming maximum bandwidth, STREAM can only consume the minimum leftover bandwidth. Subtracting minimum recorded bandwidth from the peak memory bandwidth of the iMX6Quad (1048 MiB/s), gives peak memory bandwidth utilized by the Radar application.

4.2.5 Peak Memory Utilization

The percentage of the memory being used by a process can be extracted from `top` command. A bash script(Appendix C.3) is written to read memory utilization of the Radar application every 10 times a second. The recorded peak memory utilization is exported to an output file for further analysis. This script is also run in background at low priority.

4.2.6 CPU Utilization

CPU utilization is the ratio of busy time to the total time. Busy time is the execution time on a particular CPU and total time is the time delay between two consecutive inputs to the same CPU. The CPU utilization says the percentage of utilization as well as the remaining buffer time for future growth. 50% spare time is a healthy value for CPU utilization. This spare time can be used for health monitoring that reports hardware and software failures. By doing so, faults can be isolated and stopped from propagating.

4.3 Performance Comparison of Single Core vs Four Cores

A performance comparison study has been done on the Nitrogen6X board equipped with ARM Cortex A9 quad core processor running functional blocks of the Radar processing algorithm. In Figure 4.5, measurements of 1-thread, CMYACC means running CMYACC functional block on Core#1 while keeping other cores in idle state. Likewise, measurements of 4-threads states that the CMYACC functional block is executed in all four cores simultaneously. The figure implies that the four core performance is not as good as single core performance. Bottleneck is imposed by the shared resources L2 cache and Memory. In-case more than one core wants to access a shared resource simultaneously, only one core is allowed to access them at a given time. Other cores have to wait until the shared resource is freed again. This increases worst-case execution time of the functional blocks when running on four cores. Although running four cores in parallel does increase the processing time, this thesis sticks to four core version to make use of all the execution resources.

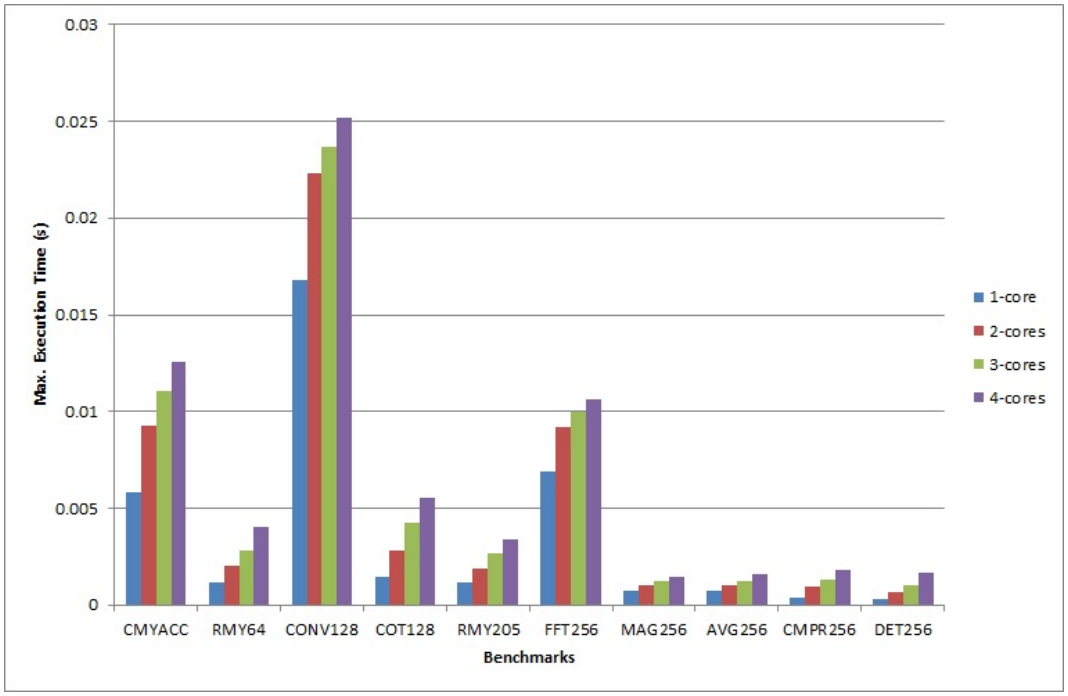

Figure 4.5: Performance of Single Core vs Four Cores

4.4 Design Decisions

4.4.1 Parallel Execution

Figure 4.6 illustrates the data distribution of Scheme-1 per CPU level. The diagram implies that the nature of execution is serial manner, in other words, it is equivalent to running the application on a single core processor. More CPUs and cores are used to improve the utilization factor, disregarding the processing latency. Thumb rule to reduce the processing latency is to process data independent portions of the application in parallel.

Figure 4.6: Serial Execution of A/A Mode Data in Baseline Analysis

4.4.2 Balanced Utilization

In all the schemes of the Baseline Analysis, Core#1 of the A/A Mode CPU does only Input/Output operations, utilizing a smaller amount of the available CPU capability. Due to this, other cores of the CPU have to carry out rest of the processing, leaving them over utilized. This kind of skewed utilization figure pushes up the CPUs worst case utilization factor, so having well balanced utilization is a gesture of a healthy system. Core#1 should also take part in processing the Radar application for utilization balancing as well as latency reduction. Static scheduling scheme shall be employed to balance the CPU utilization.

4.4.3 Space Partition

Evaluating the results of Time Partition scheme against Space Partition scheme, former one is considered to balance the CPU utilization by defining time slice periods. Down sides of the Time Partition scheme are:

- It magnifies latency by 2x compared to the Space Partition scheme.
- As it requires breaking the entire application to fit into time slot, it is not scalable (see Chapter 4.1.2).
- Implementation needs more attention when it comes to memory partitioning.
- Failure of one CPU will affect both the A/A mode and A/G mode processing.

To sum up, Time Partition demands more effort and time to realise but producing no better result than Space Partitioning. So, this thesis decides to choose Space Partition as a base for further scheduling schemes.

4.4.4 Dedicated Communication Channels

Distributing the Radar raw data to different CPUs and gathering processed data increases total amount of communication. As depicted in Figure 4.7, dedicated channels for input, output between the DGPM and PSM shall be established in Space Partition configuration. It facilitates receiving incoming data stream and sending out processed data simultaneously without interference. The DGPM has 6 port Ethernet interface, of which 4 port shall be connected to 4 CPUs and remaining 2 ports shall act as dedicated input, output channels.

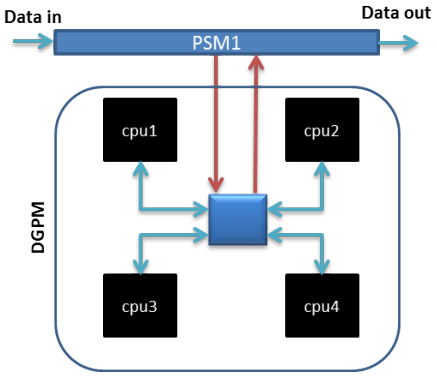

Figure 4.7: Dedicated Communication Channels

4.4.5 Summary

In a nut shell, this chapter has scrutinized the Baseline Analysis, discussed the techniques to measure the Radar processor requirement aspects and design choices that should be considered for optimal scheduling.

Chapter 5

Optimized Scheduling

5.1 Scheme 3

According to the A/A Mode processing chain, only Correlation Processing is dependent on the results of the 8 bursts of a dwell. The burst processing chain except the correlation processing has no dependency and can therefore performed independently. The previous 7 bursts. Rest of the processing steps do not have any dependency, thus can be performed independently. The final Correlation Processing shall be computed serially. Space Partition configuration is adopted for Scheme-3 implementation.

5.1.1 Hypothesis

A Burst shall be processed as soon as a core receives it. This avoids unnecessary waiting time to receive a complete Dwell (8 Bursts) data.

In case of the baseline analysis, as illustrated in Figure 5.2, beam-forming of the first burst will start only after receiving a complete dwell. Though the first burst is ready for Beam-forming, redundant time is spent in waiting for the complete dwell data. This applies to the Pulse Compression, FFT, and CFAR processing also.

Scheme-3 exploits the fact that every burst can be processed independently until the Thresholding and Detection stage. Every burst is sent to an individual core to process them conveniently as soon as they are received. This saves waiting time during Data receive period, Beam-forming, Pulse compression, FFT and CFAR processing. Nothing has changed in terms of Correlation Processing, hence it will not contribute to the latency reduction.

5.1.2 Scheduling Scheme

As shown in Figure 5.1, CPU1...6 are allocated for burst processing and CPU7 executes four instances of Correlation Processing in four cores, performing one instance per core. Burst processing comprises of Beam-forming, Pulse Compression, FFT and CFAR processing. A

burst processing CPU gets four bursts and executes one instance of burst processing per core.

No.of cores for burst processing:	6 x 4 = 24
No.of cores for correlation processing:	1 x 4 = 4
Data distribution-burst processing:	One burst per core
Data distribution-correlation processing:	One dwell per core

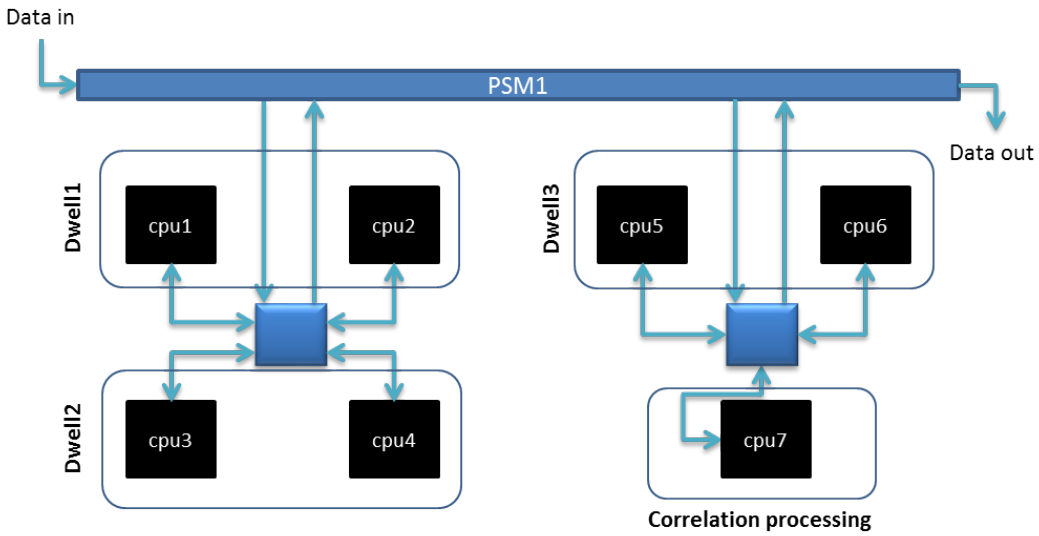

Figure 5.1: Scheduling Scheme

The received A/A mode raw data will be processed as follows

1. PSM1 routes each burst data to each core starting from core1 of CPU1 to core4 of CPU6 in round robin fashion.
2. Each core in the CPU1...6 performs burst processing followed by storing the results in SDRAM.
3. In a burst processing CPU, a core completing the processing steps last will transfer all the cores(Core#1...#4) results to the CPU7 for correlation processing. This avoids frequent communication between burst processing CPUs and correlation processing CPU. The data passed to the CPU7 is alarm list and their related information, which is smaller in size and assumed that it requires only 0.01ms to transfer.
4. Each core of the CPU7 waits for processed 8 burst data, and then performs Correlation Processing followed by sending out the target detections to PSM1.
5. PSM1 directs the results to tracking processor or display processor. Scheduling scheme is shown in Figure 5.2.

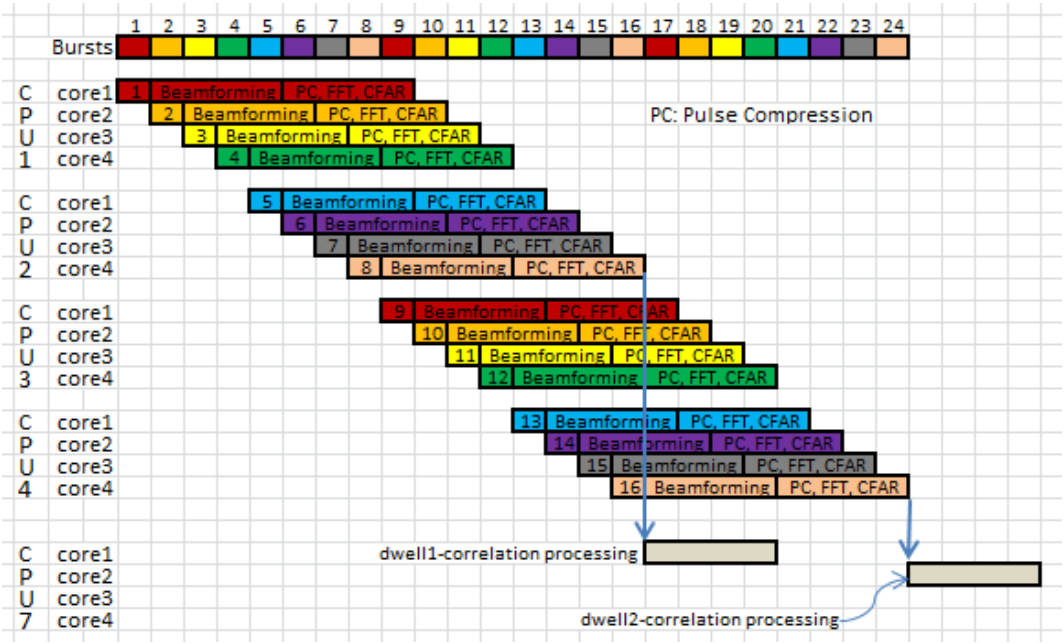

Figure 5.2: Scheduling Scheme

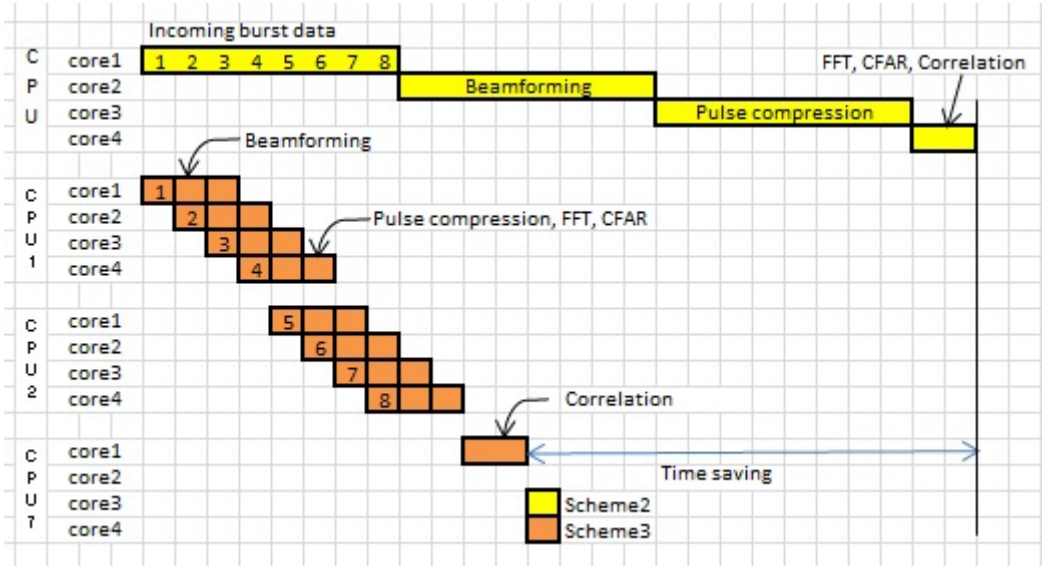

Figure 5.3: Comparison of Scheduling Schemes

Another difference between Scheme-1 and Scheme-3 is the amount of data processed by a CPU. Scheme-1 performs 8 bursts processing whereas Scheme-3 performs 4 bursts processing per CPU. This decreases memory requirement and memory transfer bandwidth compared to the Scheme-1.

5.1.3 Processing Latency

Processing latency is measured on a real hardware clocked 1GHz and the results are scaled down to 800MHz. Data transfer time of 0.02ms is assumed between CPU1...6 and CPU7. Processing latency calculations of every core is shown in Appendix E. Figure 5.4 shows the latency components corresponds to PRF1, Look Direction-1 of one Burst. Table 5.1 summarizes the time required to process one burst until Thresholding and Detection stage.

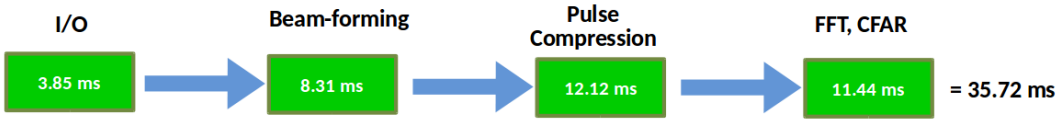

Figure 5.4: Burst Processing time for PRF1, Look Direction-1

The stated values in the table say that the look direction-1, PRF1 needs 35.72ms time to complete the execution from the moment a core starts receiving a burst. Elapsed wall clock time to process burst by burst is listed as *Time spent[ms]*. Example calculations for the look direction-1 are explained here with reference to the Figure 5.5. A core waits for the predefined burst data to be received, i.e. a core processing PRF8 should wait till the PRF1...7 are distributed by the iCON. Burst receive time are derived from the Radar characteristics (see Chapter 2.4).

	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	Look dir
Sum all req. time [ms]	35.72	45.78	44.70	45.69	46.73	46.72	34.23	44.57	1
	49.61	51.77	50.74	52.22	53.74	53.91	51.79	49.65	2
	56.72	59.15	58.08	59.58	77.95	76.68	57.44	55.34	3
	64.31	66.98	66.29	86.28	87.28	86.39	64.23	62.05	4
	73.31	97.78	94.99	96.08	97.79	97.17	71.91	69.77	5
Time spent [ms]	35.72	49.07	51.47	55.90	60.48	64.09	64.09	68.98	1
	49.61	55.67	58.78	64.33	70.04	74.53	76.67	78.64	2
	56.72	63.82	67.68	74.01	97.32	101.15	101.15	101.15	3
	64.31	72.47	77.54	103.23	110.09	115.21	115.21	115.21	4
	73.31	104.24	108.25	116.04	124.59	131.04	131.04	131.04	5

Table 5.1: Processing Time

$$T_{el} = MAX((T_{pb} + T_{ex}), T_{el})$$
$$PRF1 = MAX((0 + 35.72), 0) = 35.72\,ms$$
$$PRF2 = MAX((3.28 + 45.78), 35.72) = 49.07\,ms$$
$$PRF3 = MAX((3.28 + 3.50 + 44.7), 49.07) = 51.47\,ms$$

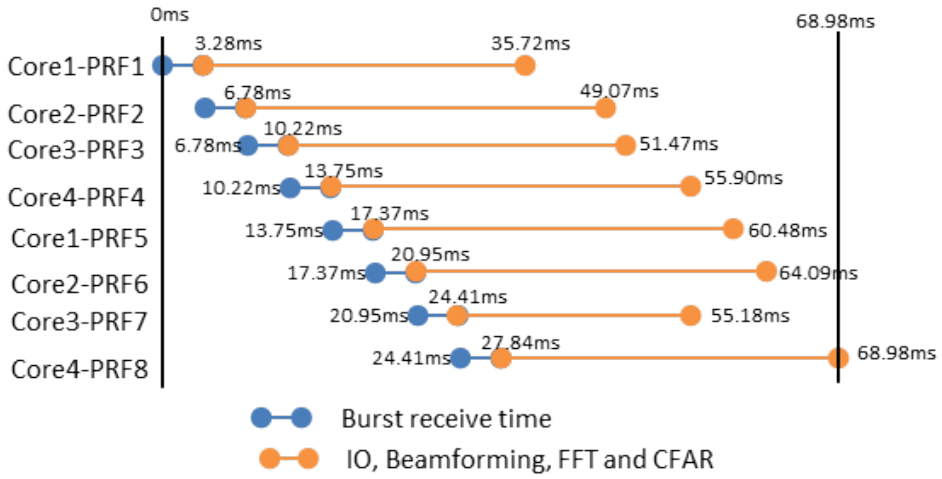

Figure 5.5: Elapsed Time Calculation

.

.

.

$$PRF8 = MAX((3.28 + 3.50 + 3.44 + 3.53 + 3.63 + 3.57 + 3.46 + 44.57), 64.09)$$
$$= 68.98\ ms \tag{5.1}$$

Legend

T_{el} : Elapsed time

T_{ex} : Execution time

T_{pb} : Previous burst receive time

Data transfer time of 0.02ms between burst processing CPUs and correlation processing CPU, and 0.2ms between correlation processing CPU and tracking/display processor is assumed.

$$T_l = T_{bp} + T_{dt} + T_{cp} + T_{tr}$$
$$= 68.98 + 0.02 + 43.34 + 0.2 = 112.5\ ms \tag{5.2}$$

Legend

T_l : Processing latency

T_{bp} : Burst processing time

T_{dt} : Data transfer time from CPU1...6 to CPU7

T_{cp} : Correlation processing time

T_{tr} : Result transfer time from CPU7 to Tracking/Display processor

The processing latency between 112.52ms and 174.58ms are achieved depending on the look direction. Table 5.2 lists the processing latency of every look direction.

Look direction	Dwell time[ms]	Latency[ms]	#Dwells transmitted
1	27.84	112.52	4.04
2	33.07	122.18	3.69
3	39.17	144.69	3.69
4	46.20	158.75	3.44
5	54.26	174.58	3.22

Table 5.2: Processing Latency

5.1.4 CPU Utilization

Burst Processing CPUs

CPU utilization is the ratio of processing time to the available time. 6 CPUs are involved in A/A mode processing; meaning 24 cores are processing 24 burst (3 Dwell) data. Available time is 3x Dwell time. Utilization of each core per look direction is listed in Appendix E.3. Summarized utilization result is presented in Figure 5.6. From the figure, it can be seen that the maximum core utilization reaches up to 66%.

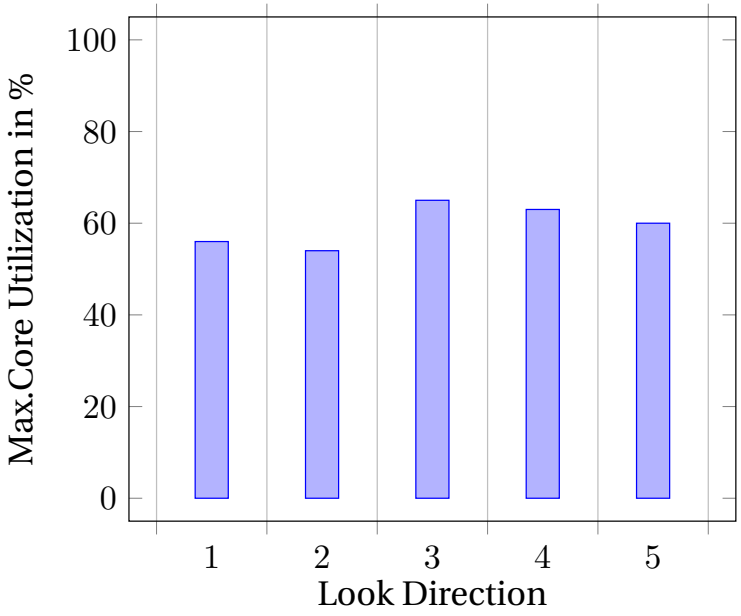

Figure 5.6: CPU Utilization - Burst Processing CPUs

Correlation Processing CPU

Correlation processing time values are listed in Appendix E.4. Processing time of one Dwell on a single core is given in Figure 5.7.

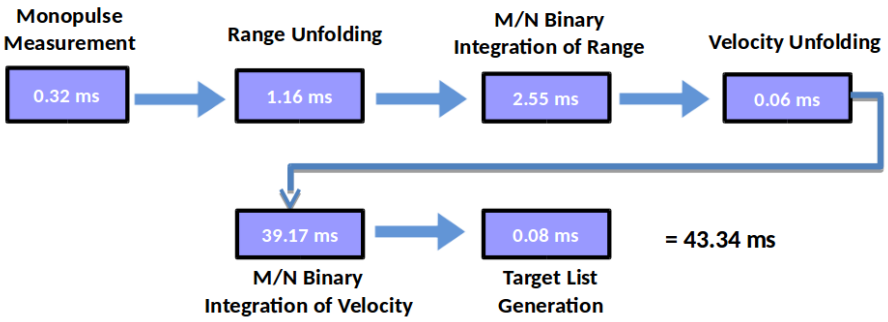

Figure 5.7: Correlation Processing Time on a Single Core

Every core of the correlation processing CPU is in idle state until it receives the results of 8 burst data from CPU1...6. Then it continues processing for 43.34ms before returning to idle state. Dwells are distributed to 4 cores of the CPU7 in round robin manner. Core#1 of the CPU7 can start processing as soon as 8 bursts of a Dwell are received. 1ms delay is assumed between burst processing CPUs sending out the data and CPU7 starts processing. From the Table 5.1, look direction-1 needs 68.98ms to do burst processing. Until this time, Core#1 of the CPU7 is in idle state. Correlation processing takes place for the next 43.34ms in Core#1 followed by waiting for the next set Dwell5 data. Dwell5 will be supplied to the burst processing CPUs at 115.3ms (4 x 27.84ms) by the incoming data stream. Received Dwell5 data will be fed to CPU7 after 68.98ms (processing) + 1ms (transfer). From Figure 5.9, peak utilization of the CPU7 is 39%, belongs to the look direction-1. Utilization calculation for the look direction-1 is depicted in Figure 5.8, and the utilization of Core#1 is derived as follows.

$$U_1 = \frac{T_{cp}}{T_{cp} + T_i} = \frac{43.34}{43.34 + 69.98} = 38\%$$

$$\begin{aligned} T_{ndi} &= I_{d5} + T_{bp} \\ &= 4 * 27.84 + 69.98 = 181.3\ ms \\ T_i &= T_{ndi} - T_{ldo} \\ &= 181.33 - (69.98 + 43.34) = 68.01\ ms \\ U_5 &= \frac{43.34}{43.34 + 68.01} = 39\% \end{aligned} \tag{5.3}$$

Legend

U_1 : Utilization at Dwell1

T_{cp} : Correlation processing time

T_i : Idle time

T_{ndi} : Next Dwell in

T_{ldo} : Last Dwell out

T_{bp} : Burst processing time

I_{d5} : Dwell5 input at burst processing CPU

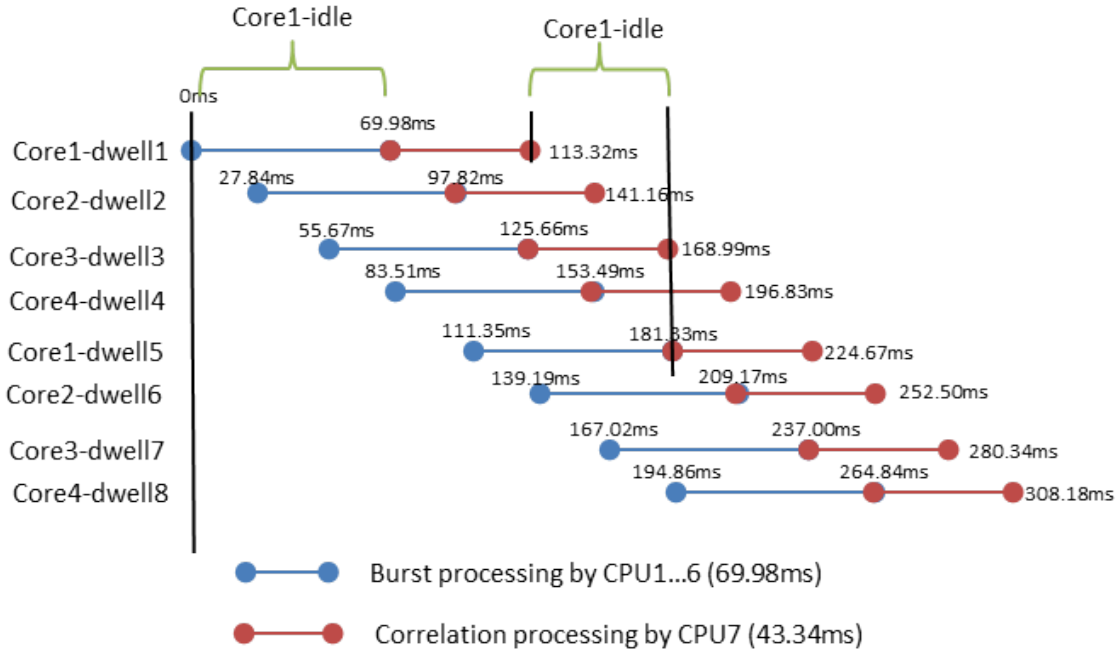

Figure 5.8: Core#1 - Idle Time, While Processing Look Direction-1

Utilization drops for ascending look direction, since look direction 5 has longest data receive time and processing time but maximum target detections are same for all the look directions.

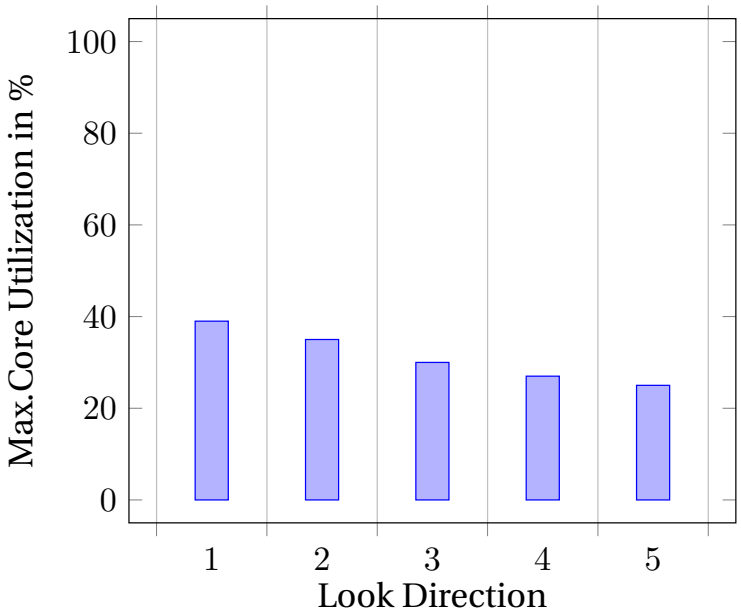

Figure 5.9: CPU Utilization - Burst Processing CPUs

5.1.5 Memory Transfer Bandwidth

Lowest recorded memory transfer bandwidth by running the STREAM benchmark as a background task is listed in Table 5.3 Peak memory transfer bandwidth of the Radar application is measured as 39.4% of the available 1048MiB/s.

Function	Best Rate [MiB/s]
Copy	635.0

Table 5.3: Lowest Recorded Memory Transfer Bandwidth of the STREAM Benchmark

$$\begin{aligned} BW_p &= BW_i - BW_l \\ &= 1048 - 635 = 413\ MiB/s \\ &= \frac{413}{1048} = 39.4\% \end{aligned} \tag{5.4}$$

Legend

BW_p : Peak bandwidth of the optimized scheme

BW_i : Idle bandwidth of the Nitrogen6X board

BW_l : Lowest recorded bandwidth

5.1.6 Memory Utilization

Peak memory utilization of the optimized scheme (Scheme-3) is measured as 0.9% of the available 879MiB memory. Memory utilization is sampled 10 times a second until the A/A Mode sequence is running. Figure 5.10 shows memory utilization footprint in a graphical view.

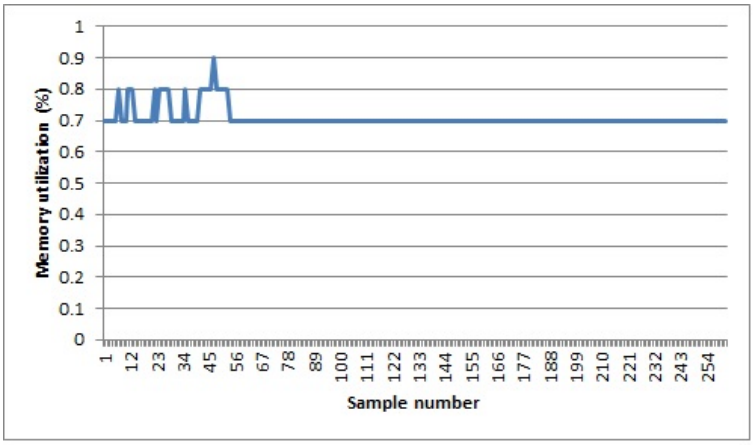

Figure 5.10: Scheme-3, Memory Utilization Footprint

5.1.7 Summary

Scheme-3 has 4x Dwell time latency, 66% CPU utilization, 39.4% memory transfer bandwidth utilization and 1% memory utilization. CPU utilization can be improved by adding more DGPMs. An IMA processor architecture can have upto 6 DGPMs comprising of 24 CPUs. Scheme-3 has utilized 28 cores of 7 CPUs, while rest of the 17 CPUs can be used for other purpose including A/G mode processing. A comparison of Scheme-1, Acceptable values and Scheme-3 is given below.

Parameter	**Scheme-1**	**Acceptable Values**	**Scheme-3**
Dwells transmitted	14.96	2	4.04
CPU utilization	75.5%	<50%	66%
Memory utilization	7%	<50%	1%
Memory transfer bandwidth	NA	<50%	41%

Table 5.4: Comparison of Scheme-1 vs Acceptable Values vs Scheme-3

5.2 Scheme 4

Space Partition configuration is adopted for Scheme-4 implementation.

5.2.1 Hypothesis

Burst processing of each channel*(S_S, S_G, S_{AZ}, S_{EL}) *shall be performed in parallel as long as they are independent. This eliminates waiting time while processing other channel data.

- Taking a closer look into the processing chain in Chapter 2.2, revels that a burst data has 4 channel processing namely Sum, Guard, Azimuth and Elevation. These 4 channels do not have data dependency, allowing for parallel execution. In Scheme-3, Core#1 does Beam-forming of one Burst serially, which comprises of computing Beam-forming for Sum channel, Azimuth channel and Elevation channel one after another. Scheme-4 proposes to perform them in parallel, leading to faster execution. Data dependency diagram for A/A mode processing is shown in Figure 5.11.

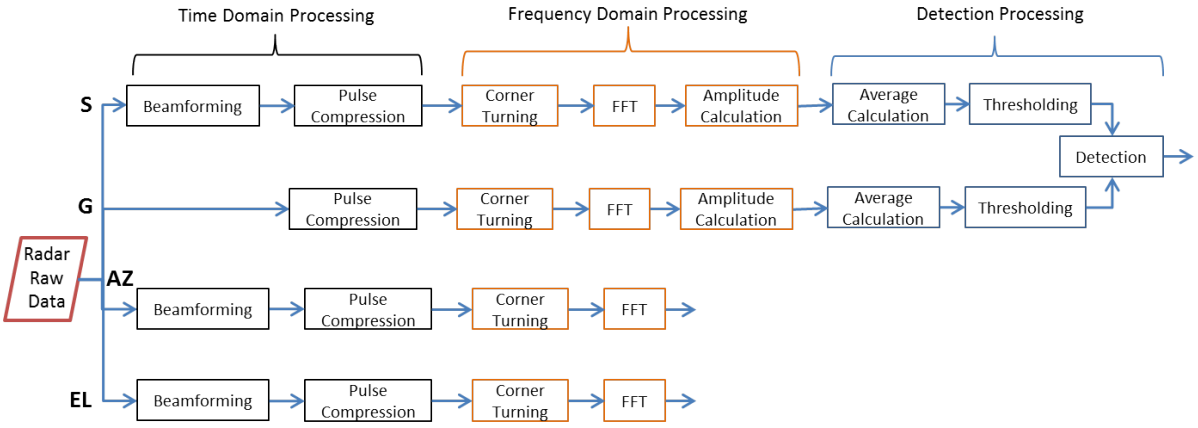

Figure 5.11: A/A Mode - Data Dependency of the Radar Application

Note: The data dependency diagram does not consider constant table values such as Sum channel beam-forming vector, Azimuth channel beam-forming vector, etc, as they are pre-calculated and available at any point of time.

- Correlation processing shall be optimized to bring down the execution time.

5.2.2 Scheduling Scheme

Four channel processing are done by 4 cores of a CPU. CPU1...12 gets one burst each. CPU13 and CPU14 are allocated for correlation processing. PSM1 routs the Radar raw data to the respective DGPMs. Burst results are sent to PSM1 by the CPUs 1..12. PSM1 stores the burst

result until 8 burst results are received. Then 8 burst results are sent to the cores of CPU13 and CPU14 in round robin manner. The DGPMs and PSM are arranged as shown in Figure 5.12 and the data distribution is illustrated in Figure 5.13.

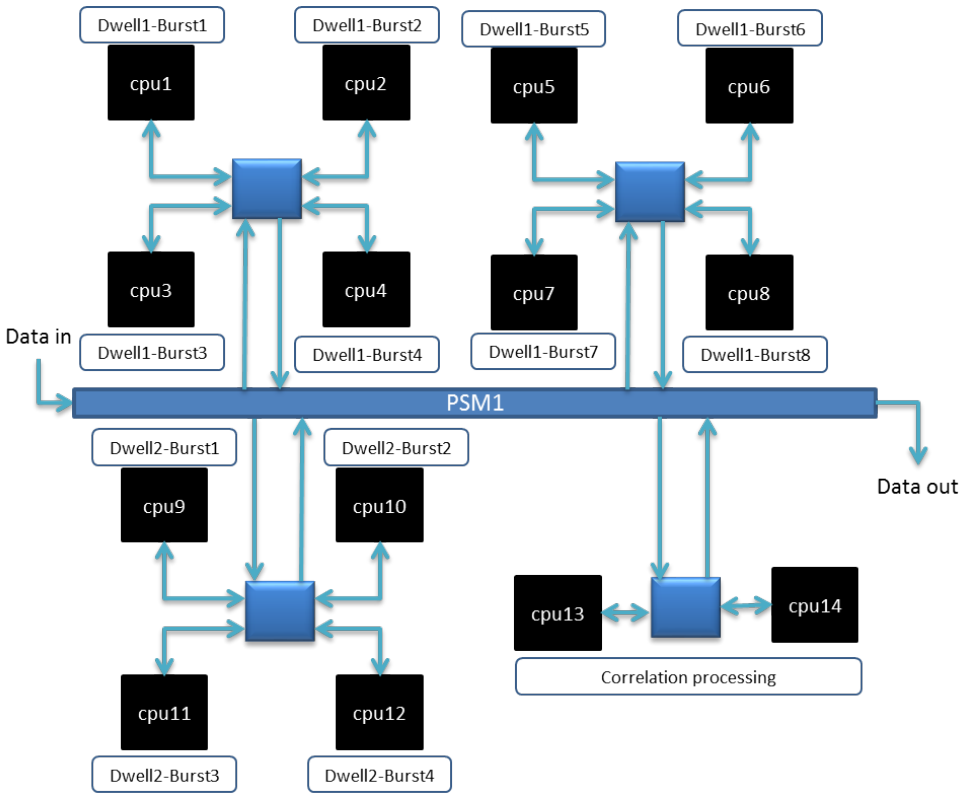

Figure 5.12: Scheduling Scheme

In terms of burst processing, every CPU in Scheme-3 performs computation on 4 burst data, whereas a CPU in Scheme-4 does computations on one burst. Single burst is placed in the shared memory, so that all the four cores can access them. CPU1 gets the first burst. Core1 of the CPU1 performs Beamforming, Pulse Compression, FFT and CFAR processing of Sum channel on the burst data. Rest of the three cores repeat the same for Guard channel, Azimuth and Elevation channel processing simultaneously. It results in approximately 4x reduced memory requirement than Scheme-3. A comparison of Scheme-3 and Scheme-4 is shown in Figure 5.14.

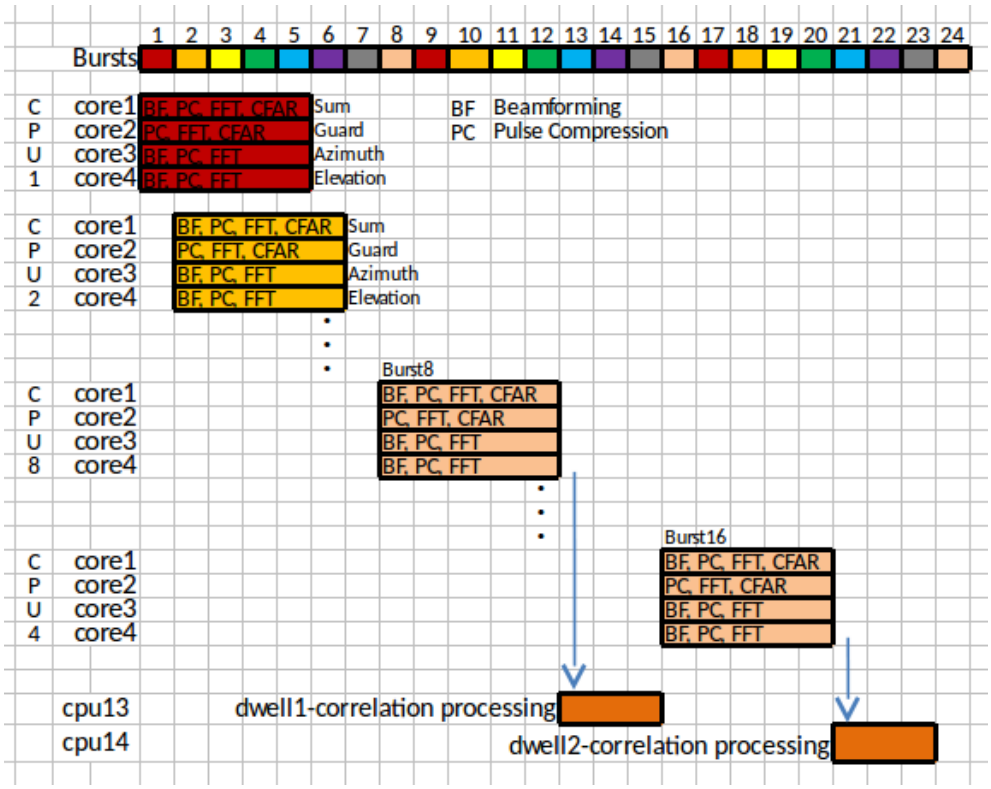

Figure 5.13: Scheduling Scheme

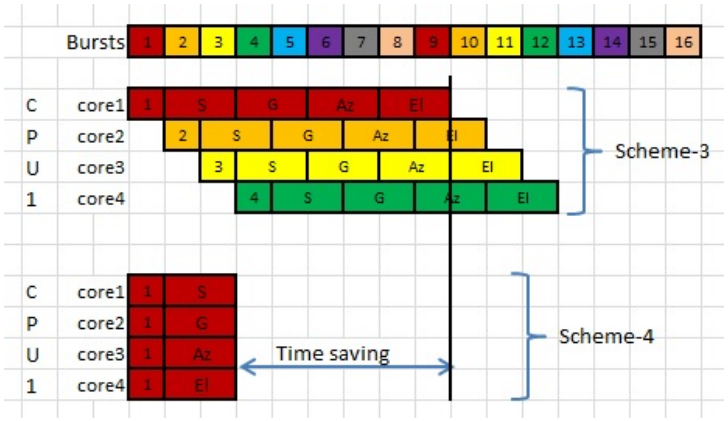

Figure 5.14: Scheme-3 vs Scheme-4

Correlation processing requires 43.34ms (see Chapter 5.1.4) to process one Dwell data. Dwell time of the look direction-1 is 27.84ms. Correlation processing itself consumes 1.5x Dwell transmission time. Best result cannot be achieved without optimizing the correlation processing. Pictorial view of the Correlation processing execution time is shown in Figure 5.15.

According to the derivations of measurement time, M/N Binary Integration of Veloc-

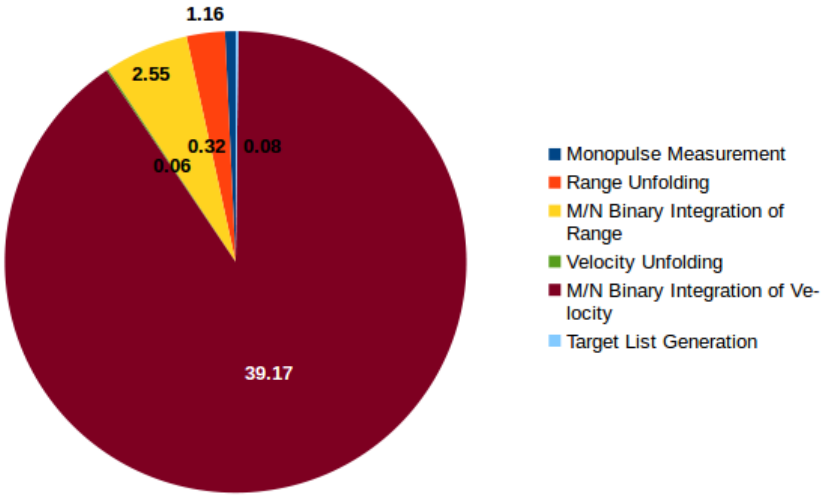

Figure 5.15: Correlation Processing - Processing Time in [ms]

ity(BIV) consumes 39.16ms of total 43.34ms. It is a good candidate for optimization. M/N BIV is implemented in C according to the pseudo-code shown in Appendix F, and the execution time is measured as 31.72ms when four instances are running on four cores. The loop iterations(Algorithm 1, Line 3) of a single instance of M/N BIV are independent of the previous loop processing and next loop processing. They are broken into four sub-loops to run on four cores of a CPU in parallel. 4x speed-up is achieved for M/N Binary Integration of Velocity, reducing execution time from 31.72ms to 7.93ms. Any correlation processing CPU in Scheme-4 processes one dwell data, on contrary to the 4 dwell data in Scheme-3. Scheduling scheme and time saving are shown in Figure 5.16, data dependency is shown in Figure 5.17.

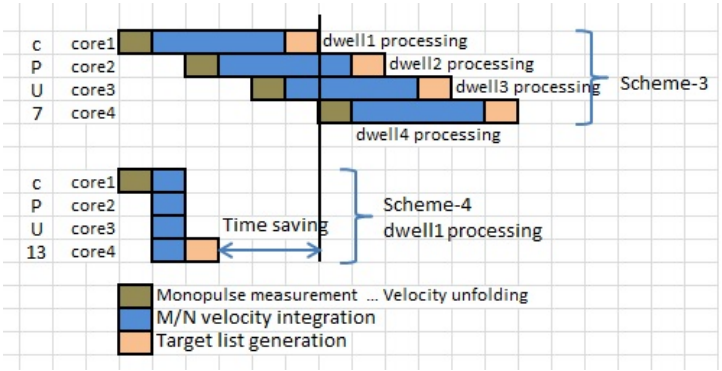

Figure 5.16: Comparison of Correlation Processing Schemes

At first, Core#1 receives the dwell data for correlation processing, it does Monopulse measurement, Range unfolding, M/N range integration and Velocity unfolding serially. Afterwards, four threads are spawned to perform M/N BIV in parallel; each tread is allocated to

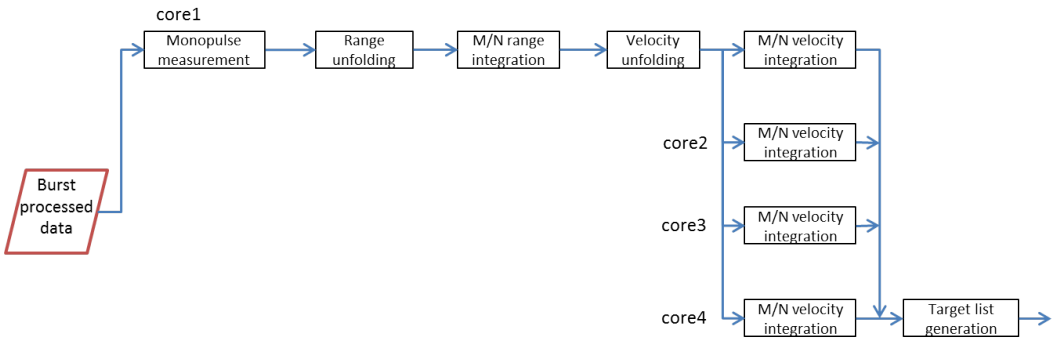

Figure 5.17: Correlation Processing - Data Dependency and Scheduling Scheme

a core. The thread completing M/N BIV last, will perform Target list generation followed by sending out the results to tracking/display processor via PSM1. Time delay of 0.02ms is assumed to spawn the threads and collect the results back. Resulting correlation processing time is calculated as follows. (Note: Execution time of the correlation processing steps can be seen in Figure E.4)

$$\begin{aligned} T_{cp} &= T_{cp1} + T_{biv} + T_{tlg} + T_{ov} \\ &= 4.09 + 7.93 + 0.02 + 0.02 = 12.06\ ms \end{aligned} \tag{5.5}$$

Legend

T_{cp1} : Monopulse measurement to Velocity unfolding processing time
T_{biv} : Time to process M/N BIV
T_{tlg} : Time to process Target list generation
T_{ov} : Overhead

5.2.3 CPU Utilization

Burst Processing CPUs

Twelve CPUs are processing one burst each. Available time of a CPU is the time span between reception of two bursts to the CPU, which is 12x average burst time. Figure 5.19 summarizes the CPU utilization for every look direction. Total time required to process one channel of a burst by a core is calculated as the sum of IO processing, Beam-forming, Pulse compression, FFT, CFAR, Thresholding, Detection and data copy time between SDRAM to the core. Detailed calculations can be found in appendix F.2. As shown in the figure, it is implied that

the peak CPU utilization reaches up to 50.8%. Figure 5.18 shows the processing time of the Radar processing chain for Look direction-1, PRF1.

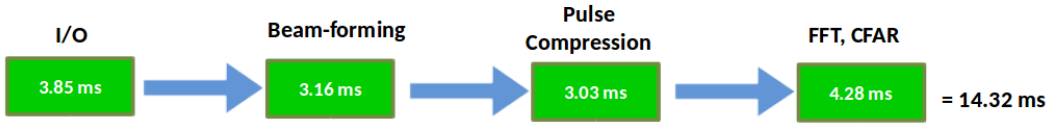

Figure 5.18: Burst Processing time for PRF1, Look Direction-1

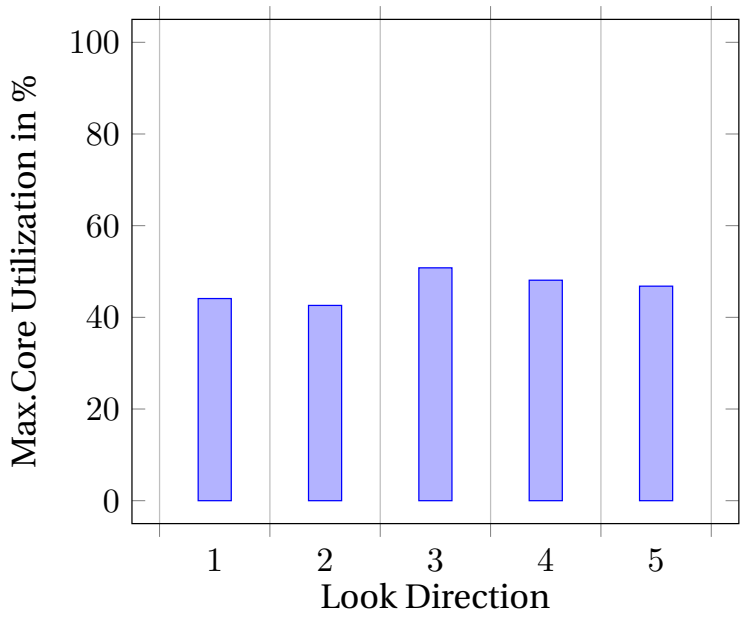

Figure 5.19: CPU Utilization - Burst Processing CPUs

Correlation Processing CPUs

The data processed by the CPU1...12 are grouped as Dwells and sent to CPU13 and CPU14 alternatively. Correlation processing CPUs are in idle state until they receive a dwell data then performs computation for 12.06ms followed by waiting for next set of burst results from the next Dwell. Time delay of 0.02ms is assumed to transfer results from burst processing CPUs to correlation processing CPUs. Timeline diagram is illustrated in Figure 5.20. Utilization calculations for the look direction-1 are listed below.

$$U_1 = \frac{T_{cp}}{T_{cp} + T_i} = \frac{12.06}{12.06 + 42.54} = 22\%$$

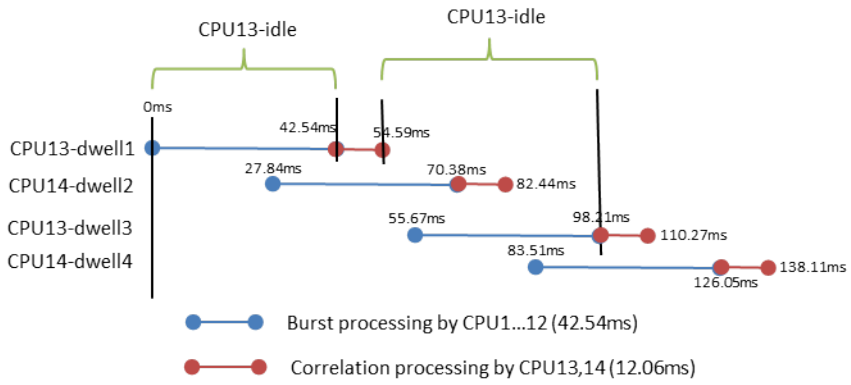

Figure 5.20: Look Direction-1, Utilization of the Correlation Processing CPU

$$
\begin{aligned}
T_{ndi} &= I_{d3} + T_{bp} \\
&= 2 * 27.84 + 42.54 = 98.2\ ms \\
T_i &= T_{ndi} - T_{ldo} \\
&= 98.2 - (12.06 + 42.54) = 43.6\ ms \\
U_5 &= \frac{12.06}{12.06 + 43.6} = 22\% \qquad (5.6)
\end{aligned}
$$

From the above calculation, it is seen that the peak utilization of the correlation processing CPU reaches upto 22% when using two CPUs. Utilization of the Correlation processing CPUs for every look direction is given in Figure 5.21.

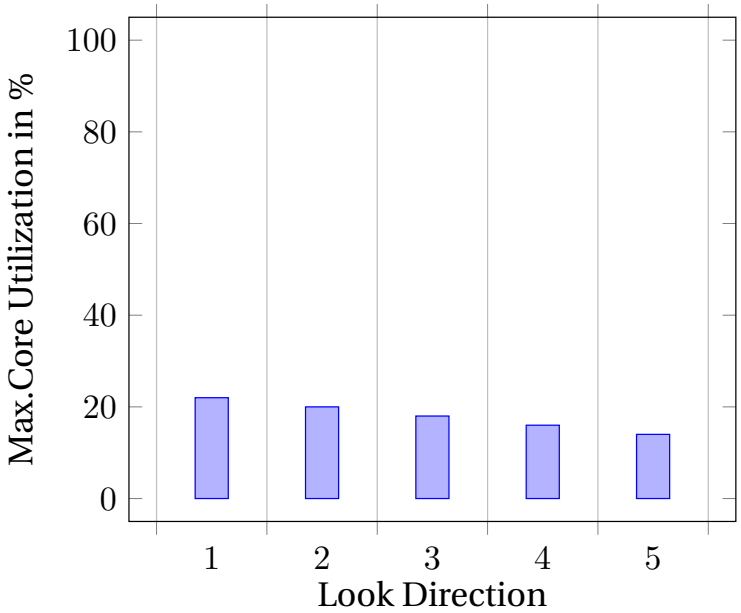

Figure 5.21: CPU Utilization - Correlation Processing CPUs

5.2.4 Processing Latency

Processing latency between 54.79ms and 88.17ms is achieved depending on the look direction, which is the time required for burst processing and correlation processing. Maximum Dwell time latency(*#Dwells transmitted*) is 1.97x Dwell time contributed by the look direction-1. The processing latencies of the look direction-1...5 are listed below.

Look direction	Dwell time[ms]	Latency[ms]	#Dwells transmitted
1	27.84	54.79	1.97
2	33.07	61.43	1.86
3	39.17	69.10	1.76
4	46.20	77.98	1.69
5	54.26	88.17	1.62

Table 5.5: Processing Latency

5.2.5 Memory Utilization

Memory utilization of Scheme-4 is measured as 0.5% of the available 879MiB memory. Memory requirement is reduced since, one CPU is working on one burst data, whereas in Scheme-3, one CPU gets 4 burst data. Memory utilization footprint of the Scheme-4 implementation is shown in Figure 5.10. Though the memory requirement is reduced by a factor of 4, memory utilization of the Scheme-4 is not reduced by the same figure. Since, the Scheme has FFTW library, spawning threads and other processing steps, that are required to run the application. They consume certain amount of memory regardless of the input data set.

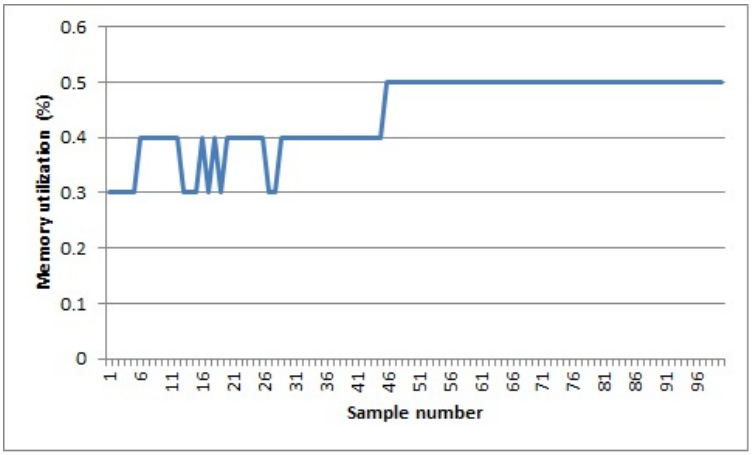

Figure 5.22: Memory Utilization Footprint

5.2.6 Memory Transfer Bandwidth

Memory footprint is reduced compared to the Scheme-3, leading to a smaller amount of operating data set per core. Private L1 cache and shared L2 cache can hold large portion of the operating data set, reducing memory transfer between SDRAM to core and so reducing memory transfer bandwidth. The lowest recorded memory transfer bandwidth by running the STREAM benchmark as a background task is listed in Table 5.6. Peak memory transfer bandwidth of the Radar application is measured as 30.3% of the available 1048MiB/s.

Function	Best Rate [MiB/s]
Copy	729.9

Table 5.6: Lowest Recorded Memory Transfer Bandwidth of the STREAM Benchmark

$$\begin{aligned} BW_p &= BW_i - BW_l \\ &= 1048 - 729.9 = 318.1\ MiB/s \\ &= \frac{318.1}{1048} = 30.3\% \end{aligned} \tag{5.7}$$

5.2.7 Summary

Processing latency of 1.92x Dwell time and peak CPU utilization of 50.8% are healthy values for Air to Air Mode Radar processor. It leaves remaining 49% of the CPU utilization to accommodate future development. In fact, 8 CPUs are sufficient to give same processing latency. More CPUs are used to improve the utilization. Figure 5.23 shows the relationship between number of CPUs used and their utilization without affecting the latency values. Any combination of the CPU sets can be chosen to achieve desired CPU utilization. Selecting 10 CPUs for burst processing and 1 CPU for correlation processing will have peak CPU utilization of 61% and 1.92x dwell latency, leaving rest of the 13 CPUs for A/G mode processing in an IMA processor architecture. A comparison of Scheme-1, Acceptable values and Scheme-4 is listed in Table 5.7.

Parameter	Scheme-1	Acceptable Values	Scheme-4
Dwell latency	14.96	2	1.97
CPU utilization	75.5%	<50%	50.8%
Memory Utilization	7%	<50%	0.5%
Memory transfer bandwidth	NA	<50%	30%

Table 5.7: Comparison of Scheme-1 vs Acceptable Values vs Scheme-4

#CPUs	Max Utilization
7	87.1%
8	76.2%
9	67.8%
10	61.0%
11	55.4%
12	50.8%

Correlation Processing	
#CPUs	Max Utilization
1	44%
2	22%

Figure 5.23: Relationship Between Number of CPUs and Utilization

5.3 Overview

The Baseline Analysis has been observed to spot the latency contributors, bottlenecks and data dependencies. Upsides and downsides are evaluated to bring up the best scheduling scheme. Examined information have been supported in the design decisions of the scheduling scheme. Two new schemes are proposed to reduce the processing latency and they are validated via implementation.

Scheme-3 makes use of the fact that, the bursts in a Dwell can be processed independently until the Thresholding and Detection stage. This phenomenon skips waiting time in Beam-forming, Pulse compression, FFT, CFAR, Thresholding and Detection stage. This can be called as coarse grained parallelism. A dedicated processor is assigned for Correlation processing, without changing the execution method. Final outcome of the Scheme-3 is 4x Dwell time latency with 66% CPU utilization.

Every burst data has large amount of fine grained parallelism. Four channel processing of a burst is done independently in Scheme-4, saving the waiting time for the other channel processing. Correlation processing poses a bottleneck as the serial execution requires 1.5x Dwell time. Further, it is also parallelised and executed in dedicated CPUs. Deemed 2x Dwell time processing latency is achieved along with 50.8% CPU utilization, provided 13 CPUs are performing computation. From the Table 5.7 data, speed-up of the Scheme-4 compared to the best scheme(Scheme-1) in the Baseline Analysis is drawn as follows.

The best results of the Baseline Analysis are 75.5% peak CPU utilization and 15x Dwell time latency, provided 12 CPUs are executing the Radar processing algorithm. For the same utilization value, Scheme-4 needs only 9 CPU resources and delivers acceptable latency of 2x Dwell time.

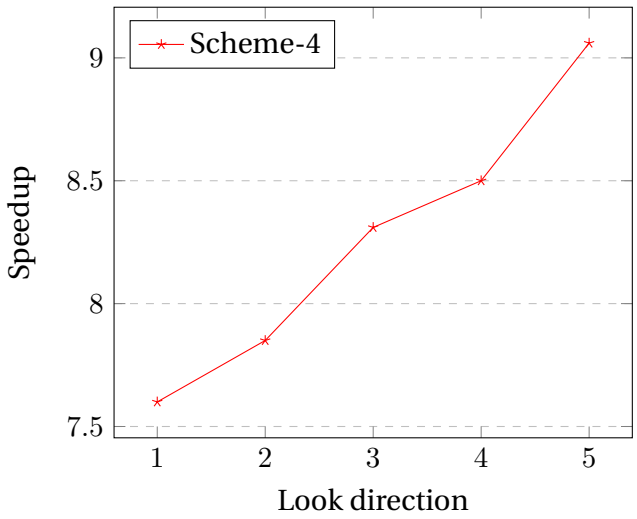

Figure 5.24: Speed-up, Scheme-4 vs Scheme-1

Chapter 6

Future Scope and Conclusion

6.1 Future Scope

There are lot of space for fine tuning the application and extending this research work. Some of them are discussed below.

6.1.1 Parallel Executable Modules

Latency reduction is greatly influenced by the amount of parallelism available in the application. Among the A/A mode benchmarks, Convolution and FFT are the major latency contributors. Scheme-3 executes FFT for every channel (Sum, Guard, Azimuth and Elevation) in sequential order, whilst Scheme-4 performs channel computations in parallel. In any case, single instance of FFT library is executed in serial. Breaking the FFT and Convolution functional block into parallel executable modules will allow greater improvement of execution time. This enables more than one core performing FFT computation of one channel, resulting in speed-up factor of broken parallel modules.

Functional blocks like Complex Multiply Accumulate (CMYACC), Range Multiply (RMY), Magnitude (MAG), Area Average Calculation (AVG) and Comparison (CMPR) have high level of parallelism. Loop iterations in the above functional blocks can be split to run in parallel. Effective speed-up will be equal to number of parallel executable modules. Although breaking the application for parallel execution increases speed-up, it needs more computing entities to carry out the task and new scheduling schemes. This is a starting point of the optimization and requires further feasibility study.

6.1.2 Architectural Features

The `FFTW` library might not have used special architecture features, since regardless of turning on the `arm` specific compiler switches, it complained

```
not vectorized: not enough data-refs in basic block.
```

```
not vectorized: failed to find SLP opportunities in basic block.
```

So it is not optimized for ARM Cortex A9 platform. 128-bit advanced SIMD unit of the ARM Cortex A9 processor can perform four math operations of 32-bit data at the cost of one math operation. Exploiting such feature will reduce the latency of the computation heavy modules in single threaded as well as multi-threaded environment.

6.1.3 Cache Performance

The more the cache hit rate, the less the memory transfer between SDRAM and core. Fetching data from memory is relatively slow and expensive. Corner turning is implemented as generic row-column transpose form. The data access pattern row-column wise, makes frequent cache misses. Corner turning can be implemented by Cache-Oblivious Algorithm as explained in *Cache Oblivious Matrix Transposition:Simulation and Experiment* [28], to improve data locality. This will reduce memory transfer bandwidth as well as execution time.

In Scheme-5, the burst data size after beam-forming is 52KiB. A core having 32KiB private L1 cache can hold up to 61% of the beamformed data. As an over approximation approach, 0% cache hit rate is assumed for the analysis. This is not the case in real world. Cache hit rate prediction mechanism shall be employed to have more realistic results.

Effect of various L2-Cache size and Memory size can be performed to determine the optimal size for working set data. Another interesting cache profiling is *cache data utilization ratio*, it is the ratio of the amount of data fetched from memory to the amount of data used by the CPU before the data is evicted. Such measurements in L1 cache and L2 cache indicate whether changes in the algorithm will yield significant reduction in execution time.

6.1.4 A/G Mode

This thesis has only considered A/A Mode processing of the Radar processor. The techniques shall be extended to adopt A/G Mode processing. A Radar processor should be capable of performing A/A Mode and A/G Mode processing in real time.

6.2 Conclusion

This thesis has presented the Airborne Radar processing chain on IMA processor architecture for safety critical systems and baseline analysis. Inference from the baseline analysis is that the latency is far higher than the acceptable values. Data dependency is scrutinized to leverage parallelism in the application. Pseudo algorithm of the Radar processing chain is implemented and the latency is measured on a real hardware. Measurement tools for mem-

ory utilization and bandwidth utilization have been made.

Couple of optimization schemes have been implemented for Air to Air Mode processing. Based on the new scheduling schemes, the optimized scheme(Scheme-4) guarantees 7.5x speed-up than the baseline analysis with lesser number of resources. The achieved 2x Dwell time latency is healthy enough for the A/A Mode Radar processor.

This thesis concludes that the simplified A/A mode Radar processing can be done on the IMA processor architecture in real time. It saves space, weight and power requirements. To compare this in day to day life, eight Samsung Galaxy S-III mobiles are sufficient to deliver 2x Dwell time latency with 87% CPU utilization!

Bibliography

[1] Ludloff and K. Albrecht, *Praxiswissen Radar und Radarsignalverarbeitung*. Springer, 1998.

[2] Radar Tutorial, [Online]. Available: http://www.radartutorial.eu/index.en.html (visited on Sep. 21, 2015).

[3] Understanding Azimuth and Elevation, [Online]. Available: http://www.photopills.com/articles/understanding-azimuth-and-elevation (visited on Sep. 22, 2015).

[4] H. P. Keller, "Future Combat Air System - Analysis of IMA Modules," Airbus Defence and Space GmbH, Tech. Rep., Dec. 2014.

[5] Boundary Devices, [Online]. Available: http://boundarydevices.com/wp-content/uploads/2014/06/Nitrogen6x1.jpg (visited on Sep. 23, 2015).

[6] Freescale-Semiconductor, "Datasheet - i.MX 6Dual/6Quad Applications Processors for Industrial Products," ARM, Tech. Rep., Mar. 2014.

[7] Medical Applications of Ultra-WideBand (UWB), [Online]. Available: http://www.cse.wustl.edu/~jain/cse574-08/ftp/uwb/ (visited on Sep. 21, 2015).

[8] Collision avoidance system, [Online]. Available: https://en.wikipedia.org/wiki/Collision_avoidance_system (visited on Sep. 21, 2015).

[9] Classification of Radar systems, [Online]. Available: http://www.radartutorial.eu/02.basics/Classification%20of%20Radar%20systems%20(1).en.html (visited on Mar. 05, 2016).

[10] Radar Basics, [Online]. Available: http://www.radartutorial.eu/07.waves/Waves%20and%20Frequency%20Ranges.en.html (visited on Oct. 07, 2015).

[11] Z. ul Abdin., A. Anders, and S. Bertil, "Real-time radar signal processing on massively parallel processor arrays," in *IEEE*, 2013.

[12] Mobile-ARM, [Online]. Available: http://www.arm.com/markets/mobile/ (visited on Feb. 27, 2016).

[13] E. Cary and R. Spitzer, *The Avionics Handbook*. CRC Press LLC, 2001.

[14] Antennas: The Interface with Space, [Online]. Available: http://www.aticourses.com/antennas_tutorial.htm (visited on Mar. 05, 2016).

[15] C. Alabaster and E. Hughes, "An examination of the effect of array weighting function on radar target detectability," Cranfield University, Tech. Rep.

[16] J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," University of Virginia, Charlottesville, Virginia, Tech. Rep., 1991-2007, a continually updated technical report. http://www.cs.virginia.edu/stream/.

[17] Optimizing Memory Bandwidth on Triad, [Online]. Available: https://software.intel.com/en-us/articles/optimizing-memory-bandwidth-on-stream-triad (visited on Oct. 22, 2015).

[18] B. Hans van, F. Heinz-Peter, and H. Wolfgang, "Status and Trends in AESA-based Radar," in *IEEE*, 2010.

[19] THE GREAT RADAR RACE: AESA DEVELOPMENT IN HIGH GEAR, [Online]. Available: http://aviationintel.com/the-great-radar-race-aesa-development-in-high-gear/ (visited on Mar. 05, 2016).

[20] T. Nohara, P. Weber, A. Premji *et al.*, "Affordable Avian Radar Surveillance Systems For Natural Resource Management And BASH Applications," in *IEEE*, 2005.

[21] Y. Salama, D. Fitzgerald, G. Bright, and J. Rooks, "Power Efficient Radar Signal Processor," in *Radar Conference, 2005 IEEE International* , 2005.

[22] APTCORE - Product Brief, [Online]. Available: http://community.arm.com/servlet/JiveServlet/previewBody/10144-102-2-19761/AptCore_ACR10X_Product_Brief.pdf (visited on Oct. 13, 2015).

[23] Y. Zeng, J. Xu, and D. Peng, "Radar Velocity-measuring System Design and Computation Algorithm Based on ARM Processor," in 8^{th} *World Congress on Intelligent Control and Automation*, 2010.

[24] C. Cheng, C. Chien-Chung, C. Yao-Liang, and H. Fu-Shin, "Real-time Scheduling in a Programmable Radar Signal Processor," in *IEEE*, 1997.

[25] H.-y. ZHAN, L.-f. YUAN, and L. WANG, “Improved Allocation Algorithm Schedules Radar Tasks Parallel ,” in *IEEE, International Workshop on Microwave and Millimeter Wave Circuits and System Technology*, 2013.

[26] LMbench - Tools for Performance Analysis, [Online]. Available: http://lmbench.sourceforge.net/ (visited on Oct. 27, 2015).

[27] FFTW Application Library, [Online]. Available: http://www.fftw.org/ (visited on Sep. 23, 2015).

[28] D. Tsifakis, A. P.Rendell, and P. E.Strazdins, “Cache Oblivious Matrix Transposition:Simulation and Experiment,” in *Computational Science - ICCS 2004*. Springer Berlin Heidelberg, 2004.

Appendix A

Block Diagrams

A.1 Nitrogen6X Development Kit

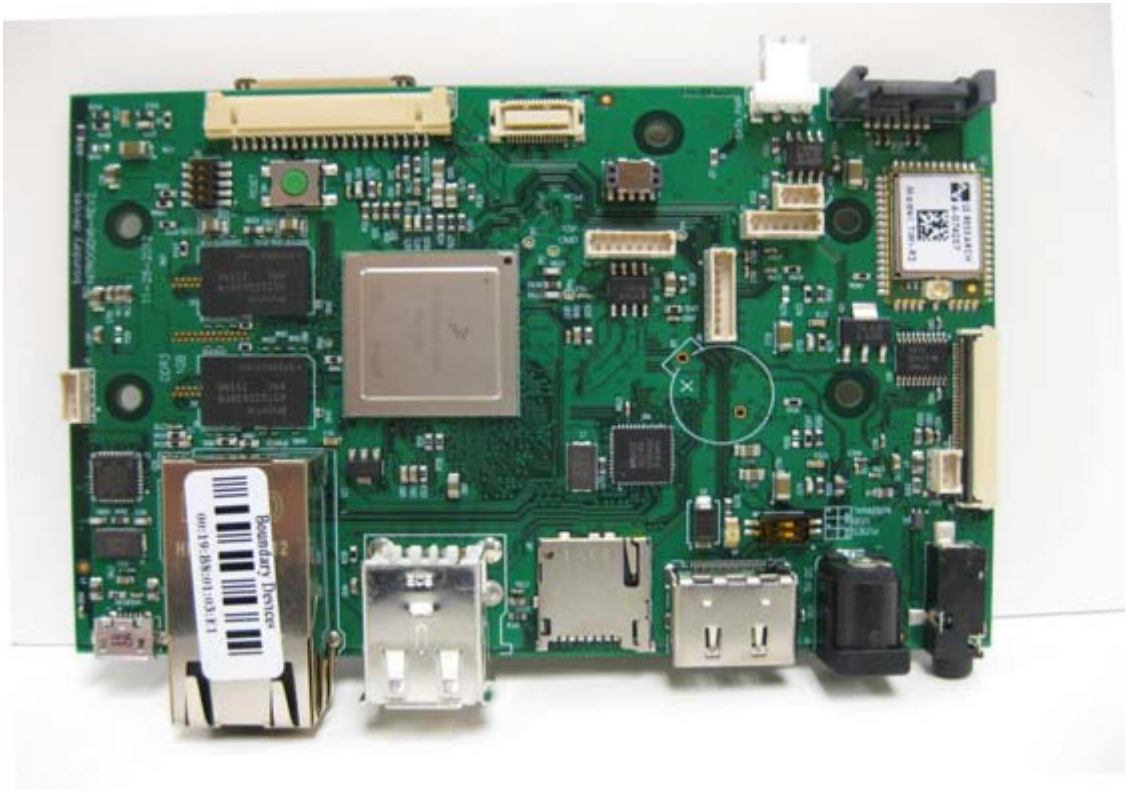

Figure A.1: Nitrogen6X Development Kit [5]

A.2 iMX6Quad CPU

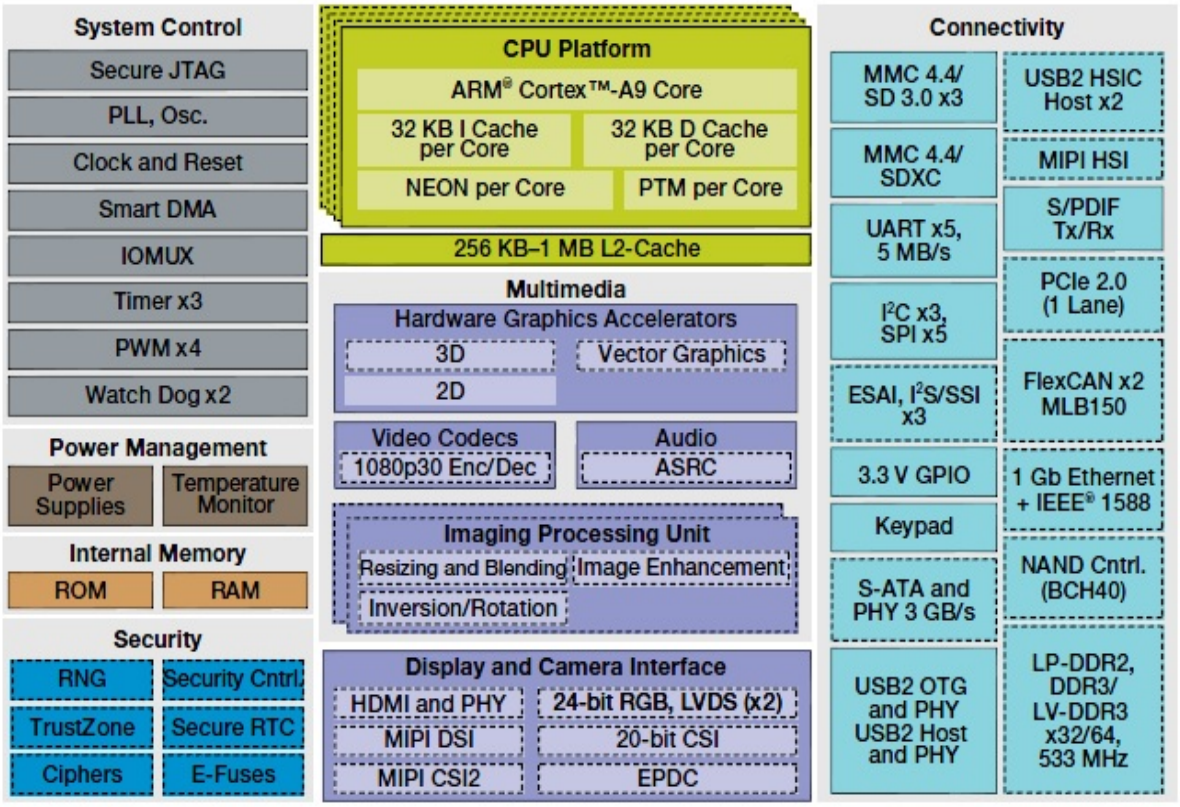

Figure A.2: Internals of iMX6Quad CPU [6]

Appendix B

Description of the Benchmark Functions

100CMYACC8: Multiply each element of a 100 x 8-element complex matrix with an 8-element complex vector and accumulate the 8 complex multiplication results to a single complex result, resulting in a 100 x 1-element complex matrix.

100RMY50: Multiply each vector in a 100 x 50-element complex matrix with a 50-element float vector, resulting in a 100 x 50-element complex matrix.

100CONV64: Perform fast convolution of a 100 x 64-element complex matrix: complex 64-pt FFT, followed by a multiplication with a 64-element float vector, followed by a 64-pt complex inverse FFT, resulting in a 100 x 64-element complex matrix.

150COT50: Perform a corner turning of a 150 x 50 complex matrix into a 50 x 150 complex matrix incl. zero-padding of the elements from 150 to 256

50MAG256: Perform with a 50 x 256-element complex matrix a magnitude square-root calculation, resulting in a 50 x 256 float matrix.

64AVG100: Calculate from a 64 x 100-element float matrix the average within a 5x5 area around the cell. Resulting in a 64 x 100-element float matrix.

64CMPR100: Compare a 64 x 100-element float matrix with a float threshold. If the threshold is exceeded, the corresponding element in a 64 x 100 boolean matrix shall be set to TRUE, otherwise set to FALSE.

Appendix C

Code Snippets

C.1 set_core_affinity()

```
1  int set_core_affinity(int core_id)
2  {
3    int num_cores = sysconf(_SC_NPROCESSORS_ONLN);
4    if (core_id < 0 || core_id >= num_cores)
5      return CPU_AFFINITY_ERROR;
6
7    cpu_set_t cpuset;
8    CPU_ZERO(&cpuset);
9    CPU_SET(core_id, &cpuset);
10
11   pthread_t current_thread = pthread_self();
12   if(pthread_setaffinity_np(current_thread, sizeof(cpu_set_t), &cpuset)
       != 0)
13     return CPU_AFFINITY_ERROR;
14
15   return CPU_AFFINITY_OK;
16 }
```

Listing C.1: Set core affinity of calling thread

C.2 wait_for_all_threads()

```
void wait_for_all_threads()
{
  while(awakenedThreads%NUM_THREADS != 0)
    sleep(0.001);

  pthread_mutex_lock(&mutex);
  readyThreads += 1;

  if(readyThreads == NUM_THREADS) {
    pthread_cond_broadcast(&cv_count);
  }

  while(readyThreads != NUM_THREADS)
    pthread_cond_wait(&cv_count, &mutex);

  ++awakenedThreads;
  if(awakenedThreads == NUM_THREADS){
    awakenedThreads = 0;
    readyThreads = 0;
  }

  pthread_mutex_unlock(&mutex);
}
```

Listing C.2: Wait for other threads to join

C.3 mem_util.sh

```
#!/bin/bash
ITER=1
PEAK_MEM=0
MEM_SIZE=‘free -m | sed -n ’2p’| awk ’{print $2}’‘
DATE_BEGIN=$(date +%Y-%m-%d" "%H:%M:%S)
OUTPUT_FILE="logs/peak_mem_util.csv"

echo "DATE BEGIN,${DATE_BEGIN}" > ${OUTPUT_FILE}

# Check for user input
if [ "$#" = "0" ]
then
# Read from file
PID=‘cat logs/pid‘
else
PID=$1
fi
echo "PID = ${PID}"

NAME=‘cat /proc/${PID}/cmdline‘
echo "Application name, ${NAME}" >> ${OUTPUT_FILE}
RES=‘eval top -b -n ${ITER} | grep ${PID}‘

while [ ! -z "$RES"  ]
do
#echo "res = ${RES}"
array=($RES)
PEAK_MEM=${array[9]}
echo "${PEAK_MEM}" >> ${OUTPUT_FILE}

#echo "Peak mem = ${PEAK_MEM}"
RES=‘eval top -b -n ${ITER} | grep ${PID}‘
sleep 0.1
done
```

Listing C.3: Peak Memory Utilization

Appendix D

Baseline Analysis Example Calculations

D.1 Scheme-1

D.1.1 Utilization of Core#1

I/O by Core 1										
	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	SUM	Look Dir.
Burst data input [byte]	237.312	253.044	246.240	253.980	260.568	257.040	248.508	245.232	2.001.924	1
	281.808	299.376	291.600	301.788	310.464	306.000	295.236	292.032	2.378.304	2
	337.428	356.400	346.680	355.572	365.904	362.304	348.336	344.448	2.817.072	3
	396.756	416.988	408.240	421.308	432.432	428.400	412.056	406.224	3.322.404	4
	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328	5
data receive time [ms]	3,28	3,50	3,44	3,53	3,63	3,57	3,46	3,43	27,84	1
	3,90	4,14	4,07	4,19	4,32	4,25	4,11	4,08	33,07	2
	4,67	4,93	4,84	4,94	5,10	5,03	4,85	4,82	39,17	3
	5,49	5,76	5,70	5,85	6,02	5,95	5,74	5,68	46,20	4
	6,46	6,80	6,70	6,85	7,07	6,97	6,75	6,68	54,26	5
data copy time [ms]	0,59	0,63	0,62	0,63	0,65	0,64	0,62	0,61	5,00	1
	0,70	0,75	0,73	0,75	0,78	0,77	0,74	0,73	5,95	2
	0,84	0,89	0,87	0,89	0,91	0,91	0,87	0,86	7,04	3
	0,99	1,04	1,02	1,05	1,08	1,07	1,03	1,02	8,31	4
	1,17	1,23	1,20	1,23	1,27	1,25	1,21	1,19	9,76	5
required time [ms]	3,88	4,13	4,05	4,16	4,28	4,21	4,08	4,04	32,84	1
	4,60	4,89	4,80	4,95	5,10	5,02	4,85	4,81	39,02	2
	5,51	5,82	5,71	5,83	6,01	5,94	5,72	5,68	46,21	3
	6,48	6,81	6,72	6,90	7,10	7,02	6,77	6,70	54,50	4
	7,63	8,03	7,90	8,08	8,33	8,23	7,96	7,87	64,02	5
							available time [ms]		334,05	
							utilization [%]		9,83%	1
									11,68%	2
									13,83%	3
									16,32%	4
									19,16%	5

Table D.1: Scheme-1, Core#1 Utilization

$$Burst\ \ data\ \ input = 64 pulses\ \ *\ \ 103 \frac{rangegates}{pulse}\ \ *\ \ 9 channel * 4 \frac{byte}{sample}$$

$$= 237,312\ \ byte$$

$$Data\ \ receive\ \ time = \frac{1}{19.5\ \ *\ \ 10^3\ \ Hz}\ \ *\ \ 64 = 3.28\ \ ms$$

$$Data\ \ copy\ \ time = 237,312byte\ \ *\ \ 2\frac{cycle}{byte}\ \ *\ \ 1.25\frac{ns}{cycle} = 0.59\ \ ms$$

$$Required\ \ time = 3.28ms + 0.59ms = 3.88ms$$

$$Available\ \ time = 12 * min.dwelltime = 12 * 27.84ms = 334.05ms \tag{D.1}$$

D.1.2 Utilization of Core#2

Beamforming by Core 2

	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	SUM	Look Dir.
Burst data input [byte]	237.312	253.044	246.240	253.980	260.568	257.040	248.508	245.232	2.001.924	1
	281.808	299.376	291.600	301.788	310.464	306.000	295.236	292.032	2.378.304	2
	337.428	356.400	346.680	355.572	365.904	362.304	348.336	344.448	2.817.072	3
	396.756	416.988	408.240	421.308	432.432	428.400	412.056	406.224	3.322.404	4
	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328	5
data copy time [ms]	0,59	0,63	0,62	0,63	0,65	0,64	0,62	0,61	5,00	1
	0,70	0,75	0,73	0,75	0,78	0,77	0,74	0,73	5,95	2
	0,84	0,89	0,87	0,89	0,91	0,91	0,87	0,86	7,04	3
	0,99	1,04	1,02	1,05	1,08	1,07	1,03	1,02	8,31	4
	1,17	1,23	1,20	1,23	1,27	1,25	1,21	1,19	9,76	5
SDRAM to L2 cache time [ms]	0,68	0,72	0,70	0,73	0,74	0,73	0,71	0,70	5,72	1
	0,81	0,86	0,83	0,86	0,89	0,87	0,84	0,83	6,80	2
	0,96	1,02	0,99	1,02	1,05	1,04	1,00	0,98	8,05	3
	1,13	1,19	1,17	1,20	1,24	1,22	1,18	1,16	9,49	4
	1,33	1,41	1,37	1,41	1,45	1,43	1,38	1,36	11,15	5
Beamforming processing [ms]	2,54	2,71	2,63	2,72	2,79	2,75	2,66	2,62	21,42	1
	3,01	3,20	3,12	3,23	3,32	3,27	3,16	3,12	25,44	2
	3,61	3,81	3,71	3,80	3,91	3,88	3,73	3,68	30,14	3
	4,24	4,46	4,37	4,51	4,63	4,58	4,41	4,35	35,54	4
	5,00	5,26	5,13	5,27	5,43	5,37	5,18	5,11	41,75	5
Burst data after Beamforming [byte]	210.944	224.928	218.880	225.760	231.616	228.480	220.896	217.984	1.779.488	1
	250.496	266.112	259.200	268.256	275.968	272.000	262.432	259.584	2.114.048	2
	299.936	316.800	308.160	316.064	325.248	322.048	309.632	306.176	2.504.064	3
	352.672	370.656	362.880	374.496	384.384	380.800	366.272	361.088	2.953.248	4
	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736	5
L2 to SDRAM cache time [ms]	0,60	0,64	0,63	0,65	0,66	0,65	0,63	0,62	5,08	1
	0,72	0,76	0,74	0,77	0,79	0,78	0,75	0,74	6,04	2
	0,86	0,91	0,88	0,90	0,93	0,92	0,88	0,87	7,15	3
	1,01	1,06	1,04	1,07	1,10	1,09	1,05	1,03	8,44	4
	1,19	1,25	1,22	1,25	1,29	1,27	1,23	1,21	9,91	5
data copy time [ms]	0,53	0,56	0,55	0,56	0,58	0,57	0,55	0,54	4,45	1
	0,63	0,67	0,65	0,67	0,69	0,68	0,66	0,65	5,29	2
	0,75	0,79	0,77	0,79	0,81	0,81	0,77	0,77	6,26	3
	0,88	0,93	0,91	0,94	0,96	0,95	0,92	0,90	7,38	4
	1,04	1,09	1,07	1,10	1,13	1,12	1,08	1,06	8,67	5
required time [ms]	4,94	5,27	5,13	5,29	5,42	5,35	5,17	5,10	41,67	1
	5,87	6,23	6,07	6,28	6,46	6,37	6,15	6,08	49,51	2
	7,02	7,42	7,22	7,40	7,62	7,54	7,25	7,17	58,64	3
	8,26	8,68	8,50	8,77	9,00	8,92	8,58	8,46	69,16	4
	9,73	10,24	9,98	10,26	10,56	10,45	10,08	9,94	81,23	5
								available time [ms]	334,05	
								utilization [%]	12,48%	1
									14,82%	2
									17,56%	3
									20,70%	4
									24,32%	5

Table D.2: Scheme-1, Core#2 Utilization

$$SDRAM\ to\ L2cache\ time = 237,312byte\ *\ 2.29\frac{cycle}{byte}\ *\ 1.25\frac{ns}{cycle} = 0.68\,ms$$

$$Beamforming\ processing = 3channel * 64pulses * 103\frac{rangegates}{pulse} * 79\frac{cycle}{8element}$$
$$* 1.25\frac{ns}{cycle} * 1.3(OSoverhead) = 2.54ms$$

$$Beamformed\ data = 4channel * 64pulses * 103\frac{rangegates}{pulse} * 8\frac{byte}{sample} = 210,944byte$$

$$L2cache\ to\ SDRAM\ time = 210,944byte\ *\ 2.29\frac{cycle}{byte}\ *\ 1.25\frac{ns}{cycle} = 0.60\,ms$$

$$Data\ copy\ time = 210,944byte\ *\ 2.2\frac{cycle}{byte}\ *\ 1.25\frac{ns}{cycle} = 0.53\,ms$$

$$Required\ time = 0.59ms + 0.68ms + 2.54ms + 0.60ms + 0.53ms = 4.94ms \quad \text{(D.2)}$$

D.1.3 Utilization of Core#3

$$Pulse\ compression = \left(\#channel * \#pulses(CONV128 + \#\frac{rangegates}{pulse} * RMY50)\right)$$
$$* \#\ cycle\ time * OSoverhead$$
$$= (4 * 64(24100 + 103 * 15)) * 1.25 * 1.3 = 10.67ms \quad \text{(D.3)}$$

Pulse Compression by Core 3

	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	SUM	Look Dir.
data copy time [ms]	0,53	0,56	0,55	0,56	0,58	0,57	0,55	0,54	4,45	1
	0,63	0,67	0,65	0,67	0,69	0,68	0,66	0,65	5,29	2
	0,75	0,79	0,77	0,79	0,81	0,81	0,77	0,77	6,26	3
	0,88	0,93	0,91	0,94	0,96	0,95	0,92	0,90	7,38	4
	1,04	1,09	1,07	1,10	1,13	1,12	1,08	1,06	8,67	5
SDRAM to L2 cache time [ms]	0,60	0,64	0,63	0,65	0,66	0,65	0,63	0,62	5,08	1
	0,72	0,76	0,74	0,77	0,79	0,78	0,75	0,74	6,04	2
	0,86	0,91	0,88	0,90	0,93	0,92	0,88	0,87	7,15	3
	1,01	1,06	1,04	1,07	1,10	1,09	1,05	1,03	8,44	4
	1,19	1,25	1,22	1,25	1,29	1,27	1,23	1,21	9,91	5
Pulse Compression [ms]	10,67	11,81	12,57	14,00	15,43	17,14	5,62	6,20	93,44	1
	12,67	13,97	14,89	16,64	18,39	20,41	6,67	7,38	111,02	2
	15,17	16,63	17,70	19,60	21,67	24,17	7,87	8,71	131,52	3
	17,84	19,46	20,84	23,23	25,61	28,57	9,31	10,27	155,13	4
	21,00	22,95	24,48	27,18	30,04	33,47	10,94	12,07	182,14	5
L2 to SDRAM cache time [ms]	0,60	0,64	0,63	0,65	0,66	0,65	0,63	0,62	5,08	1
	0,72	0,76	0,74	0,77	0,79	0,78	0,75	0,74	6,04	2
	0,86	0,91	0,88	0,90	0,93	0,92	0,88	0,87	7,15	3
	1,01	1,06	1,04	1,07	1,10	1,09	1,05	1,03	8,44	4
	1,19	1,25	1,22	1,25	1,29	1,27	1,23	1,21	9,91	5
data copy time [ms]	0,53	0,56	0,55	0,56	0,58	0,57	0,55	0,54	4,45	1
	0,63	0,67	0,65	0,67	0,69	0,68	0,66	0,65	5,29	2
	0,75	0,79	0,77	0,79	0,81	0,81	0,77	0,77	6,26	3
	0,88	0,93	0,91	0,94	0,96	0,95	0,92	0,90	7,38	4
	1,04	1,09	1,07	1,10	1,13	1,12	1,08	1,06	8,67	5
required time Appl-Partition [ms]	12,93	14,22	14,92	16,42	17,91	19,59	7,98	8,53	112,51	1
	15,35	15,49	16,37	18,17	19,96	21,96	8,17	8,87	124,35	2
	18,38	18,44	19,46	21,41	23,53	26,01	9,64	10,46	147,33	3
	21,61	21,58	22,92	25,37	27,81	30,75	11,41	12,33	173,77	4
	25,45	27,63	29,05	31,88	34,87	38,25	15,56	16,61	219,31	5
							available time [ms]		334,05	
							utilization [%]		33,68%	1
									37,22%	2
									44,10%	3
									52,02%	4
									65,65%	5

Table D.3: Scheme-1, Core#3 Utilization

D.1.4 Utilization of Core#4

$$FDP\,time = \Bigg(4channel * (\#pulses * \#\frac{RG}{pulse}(COT50 + RMY50) + \#\frac{RG}{pulse} * FFT64) + 2 * (\#\frac{rangegates}{pulse} * 64 * MAG256)\Bigg) * OSoverhead * cycle\,time$$

$$= (4 * (64 * 103(12 + 15) + 103 * 2550) + 2 * (103 * 64 * 20)) * 1.3 * 1.25ns$$

$$= \ 3.29ms \tag{D.4}$$

$$Data\ size\ after\ FDP = 64pulses * 103\frac{rangegates}{pulse}(2 * 8 + 2 * 4) = 158,208\ byte$$

FFT, CFAR, Correlation by Core 4

	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	SUM	Look Dir.
data copy time [ms]	0,53	0,56	0,55	0,56	0,58	0,57	0,55	0,54	4,45	1
	0,63	0,67	0,65	0,67	0,69	0,68	0,66	0,65	5,29	2
	0,75	0,79	0,77	0,79	0,81	0,81	0,77	0,77	6,26	3
	0,88	0,93	0,91	0,94	0,96	0,95	0,92	0,90	7,38	4
	1,04	1,09	1,07	1,10	1,13	1,12	1,08	1,06	8,67	5
SDRAM to L2 cache time [ms]	0,60	0,64	0,63	0,65	0,66	0,65	0,63	0,62	5,08	1
	0,72	0,76	0,74	0,77	0,79	0,78	0,75	0,74	6,04	2
	0,86	0,91	0,88	0,90	0,93	0,92	0,88	0,87	7,15	3
	1,01	1,06	1,04	1,07	1,10	1,09	1,05	1,03	8,44	4
	1,19	1,25	1,22	1,25	1,29	1,27	1,23	1,21	9,91	5
Corner Turning, Weighting, FFT and Power Calculation (S, G ch. only) [ms]	3,29	8,88	8,15	7,65	7,22	6,50	5,77	12,10	59,56	1
	9,33	9,10	8,37	7,88	7,46	6,74	13,81	12,33	75,02	2
	9,60	9,38	8,64	8,14	17,93	16,03	14,07	12,58	96,37	3
	9,89	9,68	8,94	19,46	18,25	16,35	14,38	12,88	109,83	4
	10,23	23,16	21,21	19,81	18,62	16,71	14,73	13,23	137,69	5
Burst data after FFT (and power calc) [byte]	158.208	168.696	164.160	169.320	173.712	171.360	165.672	163.488	1.334.616	1
	187.872	199.584	194.400	201.192	206.976	204.000	196.824	194.688	1.585.536	2
	224.952	237.600	231.120	237.048	243.936	241.536	232.224	229.632	1.878.048	3
	264.504	277.992	272.160	280.872	288.288	285.600	274.704	270.816	2.214.936	4
	311.472	327.888	319.680	328.680	338.184	334.560	322.848	318.240	2.601.552	5
L2 to SDRAM cache time [ms]	0,45	0,48	0,47	0,48	0,50	0,49	0,47	0,47	3,81	1
	0,54	0,57	0,56	0,57	0,59	0,58	0,56	0,56	4,53	2
	0,64	0,68	0,66	0,68	0,70	0,69	0,66	0,66	5,37	3
	0,76	0,79	0,78	0,80	0,82	0,82	0,78	0,77	6,33	4
	0,89	0,94	0,91	0,94	0,97	0,96	0,92	0,91	7,43	5
SDRAM to L2 cache time [ms]	0,45	0,48	0,47	0,48	0,50	0,49	0,47	0,47	3,81	1
	0,54	0,57	0,56	0,57	0,59	0,58	0,56	0,56	4,53	2
	0,64	0,68	0,66	0,68	0,70	0,69	0,66	0,66	5,37	3
	0,76	0,79	0,78	0,80	0,82	0,82	0,78	0,77	6,33	4
	0,89	0,94	0,91	0,94	0,97	0,96	0,92	0,91	7,43	5
Area Averages with threshold calcul.	1,11	2,14	1,95	1,80	1,67	1,47	1,28	2,25	13,66	1
Thresholding comparison and Alarmlist generation	2,23	2,14	1,95	1,80	1,67	1,47	2,55	2,25	16,05	2
	2,23	2,14	1,95	1,80	3,33	2,94	2,55	2,25	19,19	3
	2,23	2,14	1,95	3,59	3,33	2,94	2,55	2,25	20,98	4
	2,23	4,28	3,89	3,59	3,33	2,94	2,55	2,25	25,07	5
Monopulse Measurement [ms]	0,05	0,05	0,05	0,05	0,05	0,05	0,05	0,05	0,42	
Range Unfolding [ms]	0,15	0,16	0,16	0,17	0,18	0,20	0,22	0,25	1,51	
M/N Binary Integration of Range [ms]	0,32	0,35	0,35	0,38	0,40	0,45	0,51	0,56	3,32	
Velocity Unfolding [ms]	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,07	
I/N Binary Integration of Velocity [ms]	3,33	4,03	4,03	4,79	5,62	7,49	9,62	12,01	50,92	
Target List Generation [ms]	0,0026	0,0026	0,0026	0,0026	0,0026	0,0026	0,0026	0,0026	0,02	
ta to tracking (once every dwell) [ms]								0,01		
summation correlation processing [ms]	3,87	4,61	4,60	5,41	6,27	8,21	10,41	12,89	56,26	
required time Appl-Partition [ms]	10,31	17,80	16,81	17,03	17,39	18,39	19,58	29,34	146,65	1
	17,84	18,42	17,42	17,67	18,06	19,05	29,30	29,97	167,73	2
	18,59	19,19	18,16	18,39	30,67	30,28	30,02	30,67	195,98	3
	19,39	20,00	18,99	32,07	31,57	31,17	30,87	31,50	215,56	4
	20,34	36,26	33,82	33,03	32,57	32,16	31,85	32,46	252,48	5
							available time [ms]		334,05	
							utilization [%]		43,90%	1
									50,21%	2
									58,67%	3
									64,53%	4
									75,58%	5

Table D.4: Scheme-1, Core#4 Utilization

$$L2cache\ to\ SDRAM = 158,208byte * 2.29\frac{cycle}{byte} * 1.25\frac{ns}{cycle} = 0.45\,ms$$

$$Average\ Calculation = 2channel * 103\frac{rangegates}{pulse} * 64(FFTsize) * 20(AVG100) * 2$$
$$= 527,360\ cycle$$

$$
\begin{aligned}
Comparison &= 2channel * 103\frac{rangegates}{pulse} * 64(FFTsize) * 7(CMPR100) \\
&= 92,288\, cycle \\
Detection &= 1channel * 103\frac{rangegates}{pulse} * 64(FFTsize) * 10(DET100) \\
&= 65,920\, cycle \\
Time\, required &= (Average\, calculation + Comparison + Detection) \\
&\quad * OSoverhead * cycle\, time \\
&= (527,360 + 92,288 + 65,920) * 1.3 * 1.25ns = 1.1ms
\end{aligned} \tag{D.5}
$$

$$
\begin{aligned}
Monopulse\, Measurement &= 2channel * \#\frac{alarms}{burst} * \#\frac{cycle}{alarm} * OSoverhead * cycle\, time \\
&= 2 * 32 * 500 * 1.3 * 1.25ns = 0.05\, ms \\
Range\, Unfolding &= \left(RG_{max} * \#\frac{cycle}{rangegate} + \#Unfoldings * \#\frac{alarms}{burst} * \#cycle\right) \\
&\quad * OSoverhead * cycle\, time \\
&= (987 * 30 + 10 * 32 * 200) * 1.3 * 1.25ns = 0.15\, ms \\
M/N\, Range\, Integration &= \left(\#RangeGates * \#\frac{cycle}{rangegate} + \#Unfoldings * \#\frac{alarms}{burst}\right. \\
&\quad \left. * \#\frac{cycle}{alarm}\right) * OSoverhead * cycle\, time \\
&= (988 * 40 + 32 * 10 * 500) * 1.3 * 1.25ns = 0.32\, ms \\
Velocity\, Unfolding &= \left(\#Unfoldings * \#\frac{alarms}{burst} * \#\frac{cycle}{alarm}\right) \\
&\quad * OSoverhead * \#cycle\, time \\
&= (8 * 32 * 30) * 1.3 * 1.25ns = 0.01\, ms \\
M/N\, Velocity\, Integration &= \left(\#\frac{alarms}{burst} * \#Unfoldings^2 * \#bursts * \#\frac{cycle}{alarm}\right) \\
&\quad * OSoverhead * cycle\, time \\
&= (32 * 10^2 * 8 * 80) * 1.3 * 1.25ns = 3.33\, ms \\
Target\, List\, Generation &= \left(\#targets * \#\frac{cycle}{target}\right) * OSoverhead * cycle\, time \\
&= (16 * 100) * 1.3 * 1.25ns = 0.0026\, ms \\
Correlation\, Processing[ms] &= 0.05 + 0.15 + 0.32 + 0.01 + 3.33 + 0.0026 = 3.87\, ms
\end{aligned}
$$

$$Required\ time[ms] = 0.53 + 0.60 + 3.29 + 0.45 + 0.45 + 0.11 + 3.87 = 10.31\ ms \tag{D.6}$$

D.1.5 Processing Latency

$$\begin{aligned} Processing\ latency &= IO + Beamforming + Pulse\ compression + FFT, CFAR, Correlation \\ &= 32.84 + 41.67 + 112.51 + 146.65 = 333.67\ ms \end{aligned}$$

$$\#Dwells\ transmitted = \frac{processing\ latency}{dwell\ time} = \frac{333.67}{27.84} = 11.99 \tag{D.7}$$

D.1.6 Memory Utilization

SDRAM Memory Usage [Byte]	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	Sum
input-buffer Core 1	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
output-buffer Core 1	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
input-buffer Core 2	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
output buffer Core 2	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
input buffer Core 3	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
output buffer Core 3	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
input buffer Core 4	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
output-buffer Core 4	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	8.192
Core 2 Processing: 4 x Processing Buffer for biggest burst size [Byte]	1.868.832	1.967.328	1.918.080	1.972.080	2.029.104	2.007.360	1.937.088	1.909.440	15.609.312
Core 3 Processing: 4 x Processing Buffer for biggest burst size [Byte]	1.868.832	1.967.328	1.918.080	1.972.080	2.029.104	2.007.360	1.937.088	1.909.440	15.609.312
Core 4 Processing: 4 x Processing Buffer for biggest burst size [Byte]	1.868.832	1.967.328	1.918.080	1.972.080	2.029.104	2.007.360	1.937.088	1.909.440	15.609.312
								SUM	73.285.240
								available memory	1.000.000.000
								utilization ratio [%]	7%

Table D.5: Scheme-1, Memory Utilization

D.1.7 Interface Utilization

The DGPMs are identical, thus only one DGPM's utilization is listed here.

Interface [MByte/s]	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	max. average data rate [MByte/s]	Available bandwidth [Mbyte/s]	average utilization ratio [%]	peak data rate [Mbytes/s]	peak utilization ratio [%]
iCON1 output to PSM1	72,31	72,35	71,60	72,01	71,79	71,97	71,79	71,51	72,35	100	72,35%	72,35	72,35%
DGPM input from PSM1	12,05	12,06	11,93	12,00	11,97	12,00	11,97	11,92	12,06	100	12,06%	72,35	72,35%
CPU input	6,0255	6,0291	5,9670	6,0009	5,9829	5,9976	5,9826	5,9592	6,03	100	6,03%	72,35	72,35%
CPU output	0,0260	0,0260	0,0260	0,0260	0,0260	0,0260	0,0260	0,0260	0,03	100	0,03%	0,31	0,31%
DGPM output to PSM1	0,0520	0,0520	0,0520	0,0520	0,0520	0,0520	0,0520	0,0520	0,05	100	0,05%	0,62	0,62%
iCON2 input from PSM1	0,31	0,31	0,31	0,31	0,31	0,31	0,31	0,31	0,31	100	0,31%	10,24	10,24%

Table D.6: Scheme-1, Interface Utilization

$$
\begin{aligned}
iCON1\ output\ to\ PSM1 &= \#\frac{byte}{sample} * \#channel * range\ gate * frequency \\
&= 4 * 9 * 103 * 19.5kHz = 72.31\ MB/s \\
DGPM\ input\ from\ PSM &= \frac{data\ rate}{\#DGPMs} = \frac{72.31}{6} = 12.05\ MB/s \\
CPU\ input &= \frac{DGPM\ data\ rate}{\#CPUs} = \frac{12.05}{2} = 6.02\ MB/s \\
DGPM\ output\ to\ PSM &= \frac{data\ rate}{\#DGPMs} = \frac{0.31}{6} = 0.0526\ MB/s \\
CPU\ output &= \frac{DGPM\ output}{\#CPUs} = \frac{0.0526}{2} = 0.026\ MB/s
\end{aligned} \tag{D.8}
$$

Appendix E

Scheme-3 Calculations

E.1 IO Processing

I/O									
	CPU1				CPU2				
	CORE1 PRF1	CORE2 PRF2	CORE3 PRF3	CORE4 PRF4	CORE5 PRF5	CORE6 PRF6	CORE7 PRF7	CORE8 PRF8	Look Dir.
Burst data input [byte]	2,37,312	2,53,044	2,46,240	2,53,980	2,60,568	2,57,040	2,48,508	2,45,232	1
	4,67,208	4,91,832	4,79,520	4,93,020	5,07,276	5,01,840	4,84,272	4,77,360	5
data receive time [ms]	3.28	3.50	3.44	3.53	3.63	3.57	3.46	3.43	1
	6.46	6.80	6.70	6.85	7.07	6.97	6.75	6.68	5
data copy time [ms]	0.57	0.60	0.59	0.61	0.62	0.61	0.59	0.58	1
	1.11	1.17	1.14	1.18	1.21	1.20	1.15	1.14	5
Beamforming									
Burst data input [byte]	2,37,312	2,53,044	2,46,240	2,53,980	2,60,568	2,57,040	2,48,508	2,45,232	1
	4,67,208	4,91,832	4,79,520	4,93,020	5,07,276	5,01,840	4,84,272	4,77,360	5
SDRAM to L2 cache time [ms]	0.65	0.69	0.67	0.69	0.71	0.70	0.68	0.67	1
	1.27	1.34	1.31	1.34	1.38	1.37	1.32	1.30	5
Beamforming processing [ms]	7.09	7.56	7.36	7.59	7.79	7.68	7.43	7.33	1
	13.97	14.70	14.34	14.74	15.17	15.00	14.48	14.27	5
Burst data after Beamforming [byte]	2,10,944	2,24,928	2,18,880	2,25,760	2,31,616	2,28,480	2,20,896	2,17,984	1
	4,15,296	4,37,184	4,26,240	4,38,240	4,50,912	4,46,080	4,30,464	4,24,320	5
L2 to SDRAM cache time [ms]	0.57	0.61	0.60	0.62	0.63	0.62	0.60	0.59	1
	1.13	1.19	1.16	1.19	1.23	1.22	1.17	1.16	5
Pulse Compression									
SDRAM to L2 cache time [ms]	0.57	0.61	0.60	0.62	0.63	0.62	0.60	0.59	1
	1.13	1.19	1.16	1.19	1.23	1.22	1.17	1.16	5
Pulse Compression [ms]	10.98	12.08	12.70	14.00	15.29	16.76	6.15	6.57	1
	21.61	23.48	24.73	27.18	29.77	32.72	11.98	12.80	5
L2 cache to SDRAM time [ms]	0.57	0.61	0.60	0.62	0.63	0.62	0.60	0.59	1
	1.13	1.19	1.16	1.19	1.23	1.22	1.17	1.16	5

Table E.1: Scheme-3, Input Output, Beamforming and Pulse Compression

E.2 FFT and CFAR Processing

FFT, CFAR	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	Look dir.
SDRAM to L2 cache time [ms]	0.57	0.61	0.60	0.62	0.63	0.62	0.60	0.59	1
	1.13	1.19	1.16	1.19	1.23	1.22	1.17	1.16	5
COT, FFT, Power Calculation [ms]	7.46	12.38	11.63	11.34	11.10	10.42	9.63	16.77	1
	17.58	32.49	30.29	29.10	28.15	26.10	23.77	22.12	5
Range Doppler, AVG with Thresholding	3.39	6.51	5.92	5.46	5.06	4.47	3.88	6.84	1
	6.77	13.02	11.84	10.92	10.13	8.94	7.76	6.84	5
Burst data after Alarmlist Generation [byte]	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1
	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1,024	5
L2 cache to SDRAM time [ms]	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	1
	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	5
data copy time [ms]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5
required time Appl-Partition [ms]	35.72	45.78	44.70	45.69	46.73	46.72	34.23	44.57	1
	73.31	97.78	94.99	96.08	97.79	97.17	71.91	69.77	5

Table E.2: Scheme-3, FFT and CFAR Processing

E.3 CPU Utilization

	CPU1				CPU2				Look Direction
burst	1	2	3	4	5	6	7	8	
core	1	2	3	4	1	2	3	4	
Sum of required time[ms]	35.72	45.78	44.70	45.69	46.73	46.72	34.23	44.57	1
	49.61	51.77	50.74	52.22	53.74	53.91	51.79	49.65	2
	56.72	59.15	58.08	59.58	77.95	76.68	57.44	55.34	3
	64.31	66.98	66.29	86.28	87.28	86.39	64.23	62.05	4
	73.31	97.78	94.99	96.08	97.79	97.17	71.91	69.77	5
utilization [%]	43%	55%	54%	55%	56%	56%	41%	53%	1
	50%	52%	51%	53%	54%	54%	52%	50%	2
	48%	50%	49%	51%	66%	65%	49%	47%	3
	46%	48%	48%	62%	63%	62%	46%	45%	4
	45%	60%	58%	59%	60%	60%	44%	43%	5
Max util [%]	50%	60%	58%	62%	66%	65%	52%	53%	66%

Table E.3: Scheme-3, CPU Utilization

E.4 Correlation Processing CPU Utilization

Correlation Processing	Core1									
	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	sum	Look Dir.
data receive time [ms]	0.01				0.01				0.02	
data copy time [ms]	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002		1
	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002		2
	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.020	3
	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002		4
	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002		5
SDRAM to L2 cache time [ms]	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003		1
	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003		2
	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.022	3
	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003		4
	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003		5
Monopulse Measurement [ms]	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.32	
Range Unfolding [ms]	0.12	0.13	0.13	0.13	0.14	0.16	0.17	0.19	1.16	
M/N Binary Integration of Range [ms]	0.25	0.27	0.27	0.29	0.31	0.35	0.39	0.43	2.55	
Velocity Unfolding [ms]	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.06	
M/N Binary Integration of Velocity [ms]	2.56	3.10	3.10	3.69	4.33	5.76	7.40	9.24	39.17	
Target List Generation [ms]	0.010	0.010	0.010	0.010	0.010	0.010	0.010	0.010	0.08	
				summation correlation processing [ms]					43.34	

Table E.4: Scheme-3, Execution Time of Correlation Processing

Appendix F

Scheme-4

F.1 M/N BIV - Pseudo Code

Data structures of the Range Correlation Information(RCI) and Correlation Matrix Velocity(CMV) are depicted in Figure F.1. These two data structures are the results of previous correlation processing steps namely Monopulse measurement, Range unfolding, M/N range integration and Velocity unfolding.

```
typedef struct
{
    uint32_t BID;    /*! Burst ID */
    uint32_t AL;     /*! Alarm ID */
} BI;

typedef struct
{
    uint32_t Nbc;    /*! #Previous bursts correlating */
    uint32_t AI;     /* Alarm Index */
    uint32_t RG;     /*! Range Gate. Max = 987 */
    float V;         /*! Alarm velocity */
    float Qr;        /*! Quality of range correlation */
    float Qv;        /*! Quality of velocity correlation */
    BI bi[NUM_PREV_BURSTS];
} RCI_PREV;

typedef struct
{
    uint32_t Nrc;    /*! #Alarms correlating */
    RCI_PREV prev[MAX_RG];
} RCI;
```

```
typedef struct
{
    uint32_t AI;            /*! Alarm Index into ALI of current burst */
    uint32_t Nunf;          /*! #Unfoldings */
    float V[NUM_BURSTS];    /*! Unfolded Alarm Velocity */
} AI;

typedef struct
{
    uint32_t Nal;       /*! #Alarms in the current burst */
    AI ai[MAX_ALARMS];  /*! Matching alarm info */
} CMV_PREV;

typedef struct
{
    uint32_t bi;    /*! Current Burst ID */
    CMV_PREV prev[NUM_PREV_BURSTS];
} CMV;
```

Figure F.1: Structure of RCI and CMV

Algorithm 1 Pseudocode - Binary Integration of Velocity[4].

Input:

M_v: Binary integration threshold from constant table
W_{size}: Velocity window size from constant table
RCI and CMV from previous processing steps

Output:

Updated RCI

```
1: b_ref = burst_id;
2: n_rc = RCI.N_rc
3: for n:=1 to n_rc do
4:     ai_ref = RCI.RCI_PREV[n].AI;
5:     n_burst = RCI.RCI_PREV[n].N_bc;
6:     best_count = M_v-1;
7:     best_v = 0;
8:     count = 0;
9:     for ref_unf:=1 to CMV.CMV_PREV[b_ref].AI[ai_ref].N_unf do
10:        v_ref = CMV.CMV_PREV[b_ref].AI[ai_ref].V[ref_unf];
11:        for m:=1 to n_bursts do
12:            b_cmp = RCI.RCI_PREV[n].BI[m].BID;
13:            ai_cmp = RCI.RCI_PREV[n].BI[m].AL;
14:            cmp_unf = CMV.CMV_PREV[b_cmp].AI[ai_cmp].N_unf;
15:            for k:=1 to cmp_unf do
16:                v_cmp = CMV.CMV_PREV[b_cmp].AI[ai_cmp].V[k];
17:                diff_v = v_cmp - v_ref;
18:                if ((diff_v <= W_size) && (sign(v_cmp)==sign(v_ref))) then
19:                    count = count +1;
20:                    break for;
21:                end if
22:            end for
23:        end for
24:        if (count > best_count) then
25:            best_count = count;
26:            best_v = v_ref;
27:        end if
28:    end for
29:    RCI.RCI_PREV[n].V = best_v;
30:    RCI.RCI_PREV[n].Q_v = count;
31: end for
```

F.2 Utilization - Burst Processing CPUs

I/O	CPU1 PRF1	CPU2 PRF2	CPU3 PRF3	CPU4 PRF4	CPU5 PRF5	CPU6 PRF6	CPU7 PRF7	CPU8 PRF8	Look Dir.
Burst data input [byte]	2,37,312	2,53,044	2,46,240	2,53,980	2,60,568	2,57,040	2,48,508	2,45,232	1
	4,67,208	4,91,832	4,79,520	4,93,020	5,07,276	5,01,840	4,84,272	4,77,360	5
data receive time [ms]	3.28	3.50	3.44	3.53	3.63	3.57	3.46	3.43	1
	6.46	6.80	6.70	6.85	7.07	6.97	6.75	6.68	5
data copy time [ms]	0.57	0.60	0.59	0.61	0.62	0.61	0.59	0.58	1
	1.11	1.17	1.14	1.18	1.21	1.20	1.15	1.14	5
Beamforming									
Burst data input [byte]	2,37,312	2,53,044	2,46,240	2,53,980	2,60,568	2,57,040	2,48,508	2,45,232	1
	4,67,208	4,91,832	4,79,520	4,93,020	5,07,276	5,01,840	4,84,272	4,77,360	5
SDRAM to L2 cache time [ms]	0.65	0.69	0.67	0.69	0.71	0.70	0.68	0.67	1
	1.27	1.34	1.31	1.34	1.38	1.37	1.32	1.30	5
Beamforming processing [ms]	2.36	2.52	2.45	2.53	2.60	2.56	2.48	2.44	1
	4.66	4.90	4.78	4.91	5.06	5.00	4.83	4.76	5
Burst data after Beamforming [byte]	52,736	56,232	54,720	56,440	57,904	57,120	55,224	54,496	1
	1,03,824	1,09,296	1,06,560	1,09,560	1,12,728	1,11,520	1,07,616	1,06,080	5
L2 to SDRAM cache time [ms]	0.14	0.15	0.15	0.15	0.16	0.16	0.15	0.15	1
	0.28	0.30	0.29	0.30	0.31	0.30	0.29	0.29	5
Pulse Compression									
SDRAM to L2 cache time [ms]	0.14	0.15	0.15	0.15	0.16	0.16	0.15	0.15	1
	0.28	0.30	0.29	0.30	0.31	0.30	0.29	0.29	5
Pulse Compression [ms]	2.74	3.02	3.17	3.50	3.82	4.19	1.54	1.64	1
	5.40	5.87	6.18	6.79	7.44	8.18	3.00	3.20	5
L2 cache to SDRAM time [ms]	0.14	0.15	0.15	0.15	0.16	0.16	0.15	0.15	1
	0.28	0.30	0.29	0.30	0.31	0.30	0.29	0.29	5

Figure F.2: Scheme-4, CPU Utilization (1/2)

Nothing has changed in I/O processing and Burst data input of the Beamforming. CPU time spent on processing is explained for PRF1, look direction-1. Listed calculations illustrate one channel processing per core, as shown in the data dependency diagram 5.11.

$$SDRAM\ to\ L2cache = 237,312byte * 2.18\frac{cycle}{byte} * 1.25\frac{ns}{cycle} = 0.65\,ms$$

$$Beamforming = 1channel * 64pulses * 103\frac{rangegates}{pulse} * 287\frac{cycle}{8element}$$

$$* 1.25\frac{ns}{cycle} = 2.36\,ms$$

$$Beam formed\ data = 1channel * 64pulses * 103\frac{rangegates}{pulse} * 8\frac{byte}{sample} = 52,736\ byte$$

$$L2cache\ to\ SDRAM = 52,736byte\ *\ 2.18\frac{cycle}{byte}\ *\ 1.25\frac{ns}{cycle} = 0.14\ ms$$

$$Pulse\ compression = \left(\#channel * \#pulses(CONV128 + \#\frac{rangegates}{pulse} * RMY50)\right) * cycle\ time$$
$$= (1 * 64(27300 + 103 * 68)) * 1.25ns = 2.74\ ms \quad \text{(F.1)}$$

FFT, CFAR

SDRAM to L2 cache time [ms]	0.14	0.15	0.15	0.15	0.16	0.16	0.15	0.15	1
	0.28	0.30	0.29	0.30	0.31	0.30	0.29	0.29	5
Corner Turning, FFT, Power Calculation	2.03	3.42	3.20	3.11	3.03	2.83	2.60	4.53	1
	4.73	8.77	8.16	7.82	7.54	6.97	6.33	5.87	5
Area Averages with threshold calcul.	2.09	4.02	3.66	3.37	3.13	2.76	2.40	4.23	1
	4.19	8.05	7.32	6.75	6.26	5.53	4.80	4.23	5
Burst data [byte]	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1
	1,024	1,024	1,024	1,024	1,024	1,024	1,024	1,024	5
L2 cache to SDRAM time [ms]	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	1
	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003	5
data copy time [ms]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1
	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5
required time [ms]	14.31	18.39	17.79	17.96	18.17	17.86	14.35	18.13	1
	19.94	20.66	20.06	20.39	20.76	20.47	21.15	20.18	2
	22.65	23.46	22.82	23.13	29.86	28.98	23.45	22.49	3
	25.54	26.43	25.91	33.19	33.31	32.51	26.21	25.20	4
	28.96	38.10	36.75	36.84	37.19	36.44	29.35	28.33	5
utilization [%]	34.3%	44.1%	42.6%	43.0%	43.5%	42.8%	34.4%	43.4%	1
	40.2%	41.7%	40.4%	41.1%	41.9%	41.3%	42.6%	40.7%	2
	38.5%	39.9%	38.8%	39.4%	50.8%	49.3%	39.9%	38.3%	3
	36.8%	38.1%	37.4%	47.9%	48.1%	46.9%	37.8%	36.4%	4
	35.6%	46.8%	45.2%	45.3%	45.7%	44.8%	36.1%	34.8%	5
Max utilization[%]	40.2%	46.8%	45.2%	47.9%	50.8%	49.3%	42.6%	43.4%	50.8%

Figure F.3: Scheme-4, CPU Utilization (2/2)

$$FDP\ time = \left(1channel * (\#pulses * \#\frac{RG}{pulse}(COT50 + RMY50) + \#\frac{RG}{pulse} * FFT64) + 1 * (\#\frac{rangegates}{pulse} * 64 * MAG256)\right) * cycle\ time$$

$$= (1 * (64 * 103(98 + 68) + 103 * 2550) + 1 * (103 * 64 * 41)) * 1.25ns$$

$$
\begin{aligned}
&= \ 2.03\,ms \\
Average\,Calculation &= 1channel * 103\frac{rangegates}{pulse} * 64(FFTsize) * 51(AVG100) * 2 \\
&= 672,384\,cycle \\
Thresholding &= 1channel * 103\frac{rangegates}{pulse} * 64(FFTsize) * 55(CMPR100) \\
&= 362,560\,cycle \\
Detection &= 103\frac{rangegates}{pulse} * 64(FFTsize) * 97(DET100) = 639,424\,cycle \\
Sum\,time &= (Average\,calculation + Thresholding + Detection) * cycle\,time \\
&= (672,384 + 362,560 + 639,424) * 1.25ns = 2.09\,ms
\end{aligned} \tag{F.2}
$$

$$
Available\,time = \#CPUs * Avg.\,burst\,time = 12 * \frac{27.84ms(dwell\,time)}{8(\#burst)} = 41.76\,ms
$$

$$
CPU\,utilization = \frac{processing\,time}{available\,time} = \frac{14.31}{41.76} = 34.3\% \tag{F.3}
$$

Utilization - Correlation Processing CPUs

	Look direction 1			
	idle time [ms]	data in [ms]	data out [ms]	util %
dwell1	42.54	42.54	54.59	22%
dwell3	43.62	98.21	110.27	22%
dwell5	43.62	153.88	165.94	22%
dwell7	43.62	209.56	221.62	22%
dwell9	43.62	265.23	277.29	22%

Look Direction	Max CPU Utilization
1	22%
2	20%
3	18%
4	16%
5	14%

Figure F.4: Scheme-4, Utilization of Correlation Processing CPU