



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.

Ulm, 16-Dec-2015

.....
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.

I thank my parents Mr. Shanmugam Maarappan and Mrs. Boopathy Shanmugam for supporting me in all the ways to pursue master studies. I also 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.

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 vorgestellt.

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
CFAR:	Constant False Alarm Rate
CMV:	Correlation Matrix 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. $1\text{GHz} = 10^9 \text{ Hz}$
GiB:	GibiByte. $1\text{GiB} = 2^{30} \text{ Bytes}$
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. $1\text{KiB} = 2^{10} \text{ Bytes}$
MHz:	Mega Hertz. $1\text{MHz} = 10^6 \text{ Hz}$
MiB:	MibiByte. $1\text{MiB} = 2^{20} \text{ Bytes}$
nm:	nautical mile. $1\text{nm} = 1.852\text{km}$
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
SAR:	Synthetic Aperture Radar
SATA:	Serial AT Attachment
SDRAM:	Synchronous Dynamic Random Access Memory
SPM:	Signal Processing Module
SIMD:	Single Instruction Multiple Data
SSD:	Solid State Drive
TFLOP:	Tera FLoating point OPerations per second
UAV:	Unmanned Airborne Vehicles
VM:	Virtual Machine
WCET:	Worst Case Execution Time

Contents

List of Figures	xvii
List of Tables	xxi
1 Introduction	1
1.1 Radar Introduction	1
1.1.1 Principle of Radar	1
1.1.2 Terminologies	3
1.2 Motivation	4
1.3 Problem Statement	5
1.4 Contributions	5
1.5 Thesis Outline	6
2 Background and Related Work	7
2.1 IMA Processor Architecture	7
2.1.1 Platform Support Modules (PSM)	8
2.1.2 Data Graphics Processing Module (DGPM)	8
2.1.3 Radar Processor Design	9
2.2 Experiment Setup	10
2.2.1 Target Hardware	10
2.2.2 Software Development Platform	11
2.3 Air to Air Mode Processing Chain	12
2.3.1 Time Domain Processing	13
2.3.2 Frequency Domain Processing (FDP)	13
2.3.3 Detection Processing (DET)	14
2.3.4 Correlation Processing	15
2.4 Benchmark Results	15
2.5 Air to Air Mode Radar Characteristics	16
2.6 Related Work	18

3 Existing Analysis	21
3.1 Scheme 1 - Space Partitioning	21
3.1.1 A/A Mode Results	23
3.2 Scheme 2 - Time Partitioning	33
3.2.1 A/A Mode Results	35
3.2.2 Summary	36
4 Test Bed and Design Decisions	37
4.1 Pros and Cons of the Existing Analysis	37
4.1.1 Pros	37
4.1.2 Cons	38
4.1.3 Other Comments	41
4.1.4 Measured Values of Existing Analysis	42
4.2 Test Procedure	44
4.2.1 Memory Transfer Bandwidth	44
4.2.2 Peak Memory Utilization	45
4.2.3 Processing Latency (Worst Case Execution time)	45
4.2.4 CPU Load	47
4.2.5 A/A Mode Sequence of Execution	47
4.3 Performance Comparison of Single Core vs Four Cores	49
4.4 Design Decisions	50
4.4.1 Parallel Execution	50
4.4.2 Balanced Utilization	50
4.4.3 Space Partition	50
4.4.4 Dedicated Communication Channels	51
5 Mode Mapping	53
5.1 Scheme 3	53
5.1.1 Data distribution	53
5.1.2 Hypothesis	54
5.1.3 Processing Latency	56
5.1.4 CPU Load	59
5.1.5 Memory Transfer Bandwidth	62
5.1.6 Memory Utilization	62
5.1.7 Summary	62
5.2 Scheme 4	64
5.2.1 Hypothesis	64

5.2.2	Data Distribution	64
5.2.3	CPU Load	69
5.2.4	Processing Latency	73
5.2.5	Memory Utilization	73
5.2.6	Memory Transfer Bandwidth	74
5.2.7	Summary	75
5.3	Overview	75
5.4	Verification of Results	77
6	Future Scope and Conclusion	79
6.1	Some Implementation Details	79
6.1.1	Synchronization	79
6.1.2	Thread Safety	80
6.1.3	Performance of the SQLite Library	80
6.2	Future Scope	80
6.2.1	Parallel Executable Modules	81
6.2.2	Architectural Features	81
6.2.3	Cache Performance	81
6.2.4	A/G Mode	82
6.3	Conclusion	82
Bibliography		83
A Block Diagrams		85
A.1	Nitrogen6X Development Kit	85
A.2	iMX6Quad CPU	86
B Description of the Benchmark Functions		87
C Code Snippets		89
C.1	set_core_affinity()	89
C.2	wait_for_all_threads()	90
C.3	mem_util.sh	91

List of Figures

1.1	Block Diagram of Pulse Doppler Radar [3]	2
1.2	Transmitter's Pulse and Echo Signal from Target [4]	2
1.3	Azimuth and Elevation of the Sun [6]	3
1.4	Example of Burst	4
2.1	Platform Support Module [8]	8
2.2	Data Graphics Processing Module [8]	9
2.3	Block Diagram of Radar Processor [8]	10
2.4	Details of the A/A Mode Scan	12
2.5	Block Diagram of Air to Air Mode Processing	13
2.6	A/A Mode Time Domain Processing	13
2.7	Corner Turning	14
2.8	A/A Mode Frequency Domain Processing	14
2.9	A/A Mode Detection Processing	15
2.10	A/A Mode Correlation Processing	15
2.11	A/A Mode Radar Characteristics	17
3.1	Scheme-1, Mode Mapping	22
3.2	Scheme-1, A/A Mode Processing	24
3.3	Scheme-1, Core1 Utilization	25
3.4	Scheme-1, Core2 Utilization	26
3.5	Scheme-1, Core3 Utilization	27
3.6	Scheme-1, Core4 Utilization	28
3.7	Scheme-1, Memory Utilization	31
3.8	Scheme-1, Interface Utilization	31
3.9	Scheme-2, Time Partition	33
3.10	Scheme-2, Values for Time Partition	33
3.11	Scheme-2, Mode Mapping	34
3.12	Scheme-2, Memory Utilization	36

4.1	Time Partition of Scheme 3	38
4.2	Comparison of Context Switch Time	40
4.3	Clashing Data Streams	41
4.4	Before and After Spike Removal	46
4.5	A/A Mode, Sequence of Function Execution	48
4.6	Performance of Single Core vs Four Cores	49
4.7	Serial Execution of A/A Mode Data in Existing Analysis	50
4.8	Dedicated Communication Channels	51
5.1	Scheme-3, Mode Mapping	54
5.2	Scheme-3, Data Distribution	55
5.3	Comparison of Data Distribution Schemes	55
5.4	Scheme-3, Input Output, Beamforming and Pulse Compression	56
5.5	Scheme-3, FFT and CFAR Processing	57
5.6	Scheme-3, Processing Time	57
5.7	Scheme-3, Elapsed Time Calculation	58
5.8	Scheme-3, CPU Utilization	59
5.9	Scheme-3, Execution Time of Correlation Processing	60
5.10	Look Direction-1, Utilization of the Correlation Processing CPU	61
5.11	Scheme-3, Memory Utilization Footprint	62
5.12	A/A Mode - Data Dependency of the Radar Application	64
5.13	Scheme-4, Mode Mapping	65
5.14	Scheme-4 Data Distribution	65
5.15	Scheme-3 vs Scheme-4	66
5.16	Structure of RCI and CMV	68
5.17	Comparison of Correlation Processing Schemes	68
5.18	Correlation Processing - Data Dependency and Mode Mapping	69
5.19	Scheme-4, CPU Utilization (1/2)	70
5.20	Scheme-4, CPU Utilization (2/2)	71
5.21	Scheme-4, Utilization of Correlation Processing CPU	72
5.22	Look Direction-1, Utilization of the Correlation Processing CPU	73
5.23	Scheme-4, Memory Utilization Footprint	74
5.24	Scheme-4, Relationship Between Number of CPUs and Utilization	75
5.25	Speedup, Scheme-4 vs Scheme-1	76
6.1	Race Condition in <code>wait_for_all_threads()</code> Function	80

A.1	Nitrogen6X Development Kit [21]	85
A.2	Internals of iMX6Quad CPU [9]	86

List of Tables

2.1	Measured Execution Cycle of the Benchmarks	16
2.2	Execution Cycle of Correlation Processing [8]	18
3.1	Scheme-1, A/A Mode Processing Latency	30
3.2	Scheme-1, CPU Utilization	32
3.3	Scheme-2, CPU Utilization	35
3.4	Scheme-2, A/A Mode Processing Latency	35
4.1	Measured Context Switch Time	39
4.2	Comparison of Old Execution Cycle vs Measured Execution Cycle of A/A Mode Benchmarks	42
4.3	Measured CPU Utilization of Scheme-1, A/A Mode	43
4.4	Measured Processing Latency of Scheme-1, A/A Mode	43
4.5	Measured CPU Utilization of Scheme-2, A/A Mode	43
4.6	Measured Processing Latency of Scheme-2, A/A Mode	43
4.7	Idle Memory Transfer Bandwidth	44
4.9	A/A Mode - Benchmark Selection	48
5.1	Scheme-3, Processing Latency	59
5.2	Utilization of Correlation Processing CPU	61
5.3	Recorded Memory Transfer Bandwidth of the STREAM Benchmark Running as Background Task	62
5.4	Comparison of Scheme-1 vs Acceptable Values vs Scheme-3	63
5.5	Scheme-4, Processing Latency	73
5.6	Recorded Memory Transfer Bandwidth of the STREAM Benchmark Running as Background Task	74
5.7	Comparison of Scheme-1 vs Acceptable Values vs Scheme-4	75

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 [1], 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 [2].

1.1.1 Principle of Radar

Radar can be classified as Pulsed Radar or Continuous Wave Radar based on the operating principle. Pulsed Radar or also 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 antenna. Up to this point, the antenna acts as a transmitter. As soon as the pulse releases from the transmitter, it is switched to receiver mode and is listening for echo from the target. Figure 1.1 and 1.2 illustrates the block diagram 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 distance to the

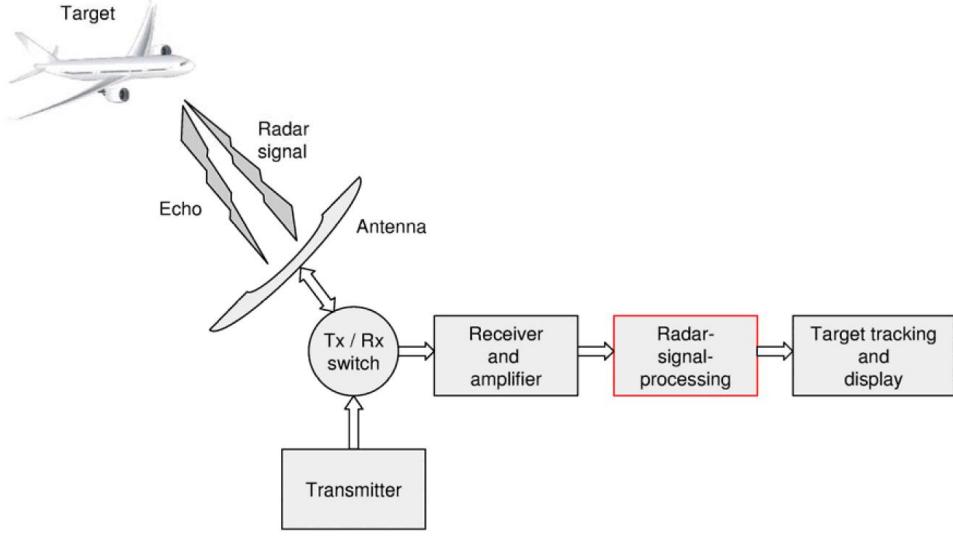


Figure 1.1: Block Diagram of Pulse Doppler Radar [3]

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

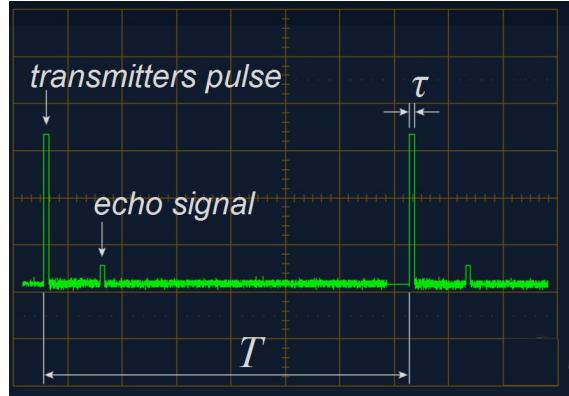


Figure 1.2: Transmitter's Pulse and Echo Signal from Target [4]

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 [5].

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.

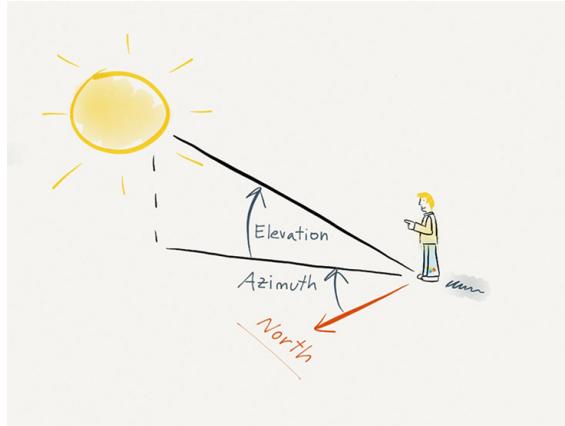


Figure 1.3: Azimuth and Elevation of the Sun [6]

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 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.

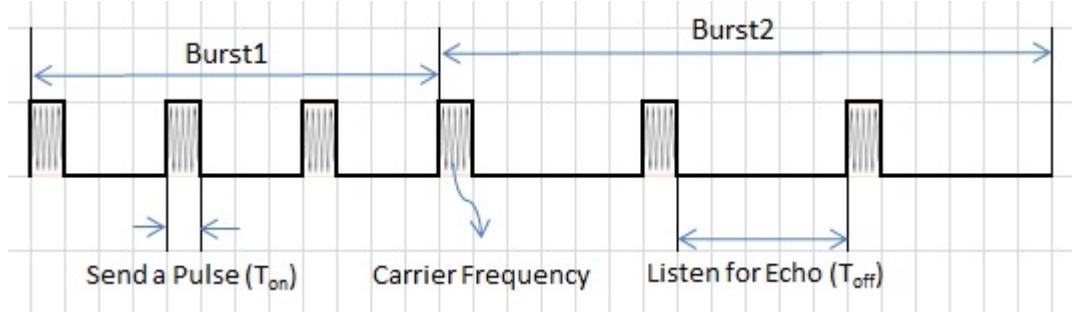


Figure 1.4: Example of Burst

1.2 Motivation

Radar is one of the core components of a combat air system. Range and accuracy of the Radar is important for missile system, guidance system, early warning system etc. Medium range (40nm) Radars are considered as suitable candidates to adopt low cost, compact ARM processors. They do not need several TFLOP of processing power to adhere to the requirements. The replaced Radar processor should be able to process Air to Air (A/A) Mode data and Air to Ground Mode (SAR) data. As a safety critical software, it should stick to the industrial standards DO-178B/C, ARINC-653 which have impact on design decisions.

The Radar modes shall be mapped to an Integrated Modular Avionics (IMA) processor and the following assessments are carried out.

- Resulting processing latency
- Memory transfer bandwidth
- Memory utilization
- CPU utilization

Mode mapping activity covers the definition of

- Mode allocation
- Task allocation
- Data distribution
- Communication schemes

Results and possible improvements are discussed for further optimization.

1.3 Problem Statement

The Airborne Radar is subjected to operate on different Pulse Repetition Frequencies (PRF) to resolve ambiguity in distance and velocity. In the examined Radar application, it is fixed that 8 different bursts form a dwell. That is on every time the Radar scans a complete azimuth, several dwells are transmitted, echoes are received and analysed. The resulting worst case latency of the existing analysis is 15x dwell time (see Table 4.4). A typical dwell time is 54ms. An Eurofighter Typhoon moving at a speed of Mach 2 would move a distance of 560 meters during this latency time. Half a kilometer difference between detection and display is very high for a typical Airborne Radar processor.

The acceptable latency range is maximum 2x dwell time. Since the system is in the early stage of development and to allow room for further requirements, the CPU utilization shall not exceed 50%. The same is applicable for memory utilization and memory transfer bandwidth. The investigation of this thesis is to determine the number of processors required, mode mapping, data distribution, resulting processing latency and exploit the parallelism to achieve best result.

1.4 Contributions

Benchmarks of the performance critical radar functions were given as inputs. They were meant for single core processors and thus single-threaded. One of the major works of this thesis is extending the benchmark program to replicate the Radar application, targeted for multicore and multi-threaded application. In summary, the main contributions of this thesis are:

- **Multicore Performance**

- Scalability and performance of the application on multicore environment are estimated. Shared resources, resource contention, race condition are considered.
- All the threads are set to execute the benchmarks simultaneously to ensure the maximum memory transfer bandwidth.

- **Mode Mapping**

- Data dependencies are identified and evaluated by executing them in parallel using POSIX Threads.
- Constraints on non-thread-safe benchmarks are discovered.

- Optimal mode mapping schemes are proposed and their results are assessed.

- **Measurement Tools**

- Developed automated scripts to measure peak memory utilization, CPU load, memory transfer bandwidth and processing latency.
- Worst case execution time is computed by measuring the maximum execution time of 1000 runs.

- **Verification of Results**

- Results of multicore application are verified against the single core application.

1.5 Thesis Outline

The rest of this thesis is organized as follows:

- **Chapter 2:** Explains about IMA architecture, experimental setup and related work concerning ARM processors in Radar application.
- **Chapter 3:** Discusses results of the existing analysis done by Airbus Defence and Space GmbH.
- **Chapter 4** Explains pros and cons of the existing analysis, test bed information and important design choices.
- **Chapter 5:** Proposes new schemes of mode mapping and verifies their correctness via implementation. Results of the new schemes are analysed.
- **Chapter 6:** Summarizes this thesis with a conclusion and proposes future work regarding the new analysis.

Chapter 2

Background and Related Work

Minimizing size, weight and power consumption of the Radar processor without compromising performance is the topic under research. This low cost, portable 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 [7]. 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)

¹These 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.

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

or any combination of these two.

- 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.

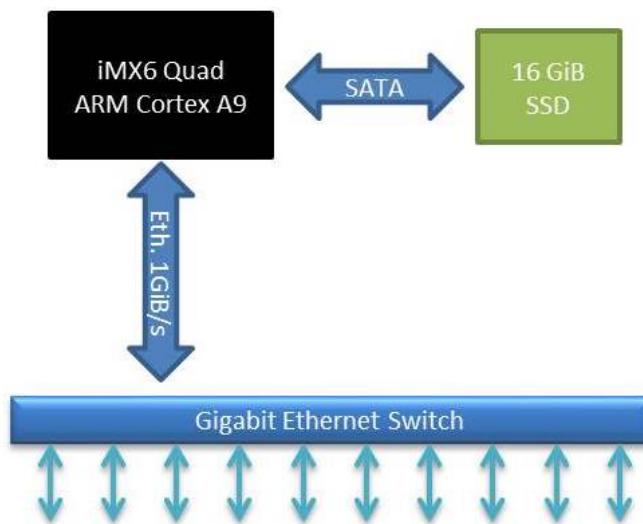


Figure 2.1: Platform Support Module [8]

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

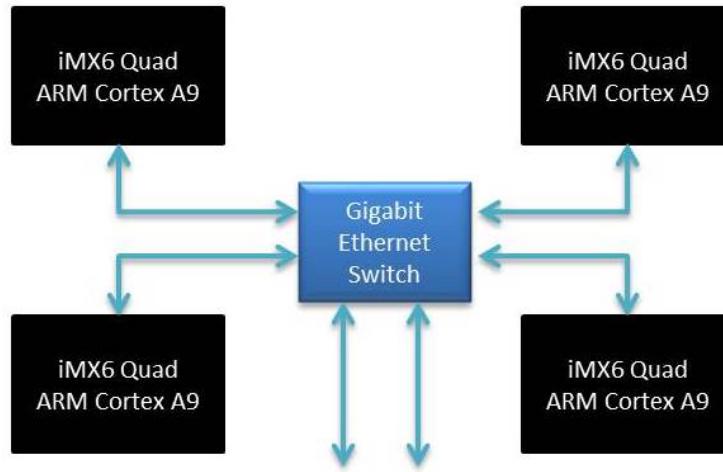


Figure 2.2: Data Graphics Processing Module [8]

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 Graphics Processing Unit, one common 533MHz 64-bit DDR3 memory interface, one common 1GBit/s Ethernet interface and several other standard interfaces [9], (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.

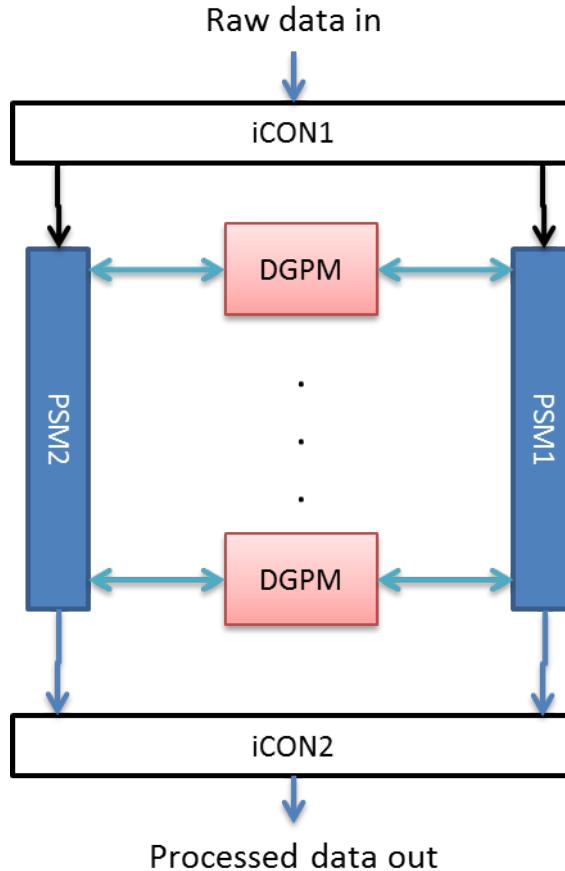


Figure 2.3: Block Diagram of Radar Processor [8]

2.2 Experiment Setup

To mimic the IMA processor architecture on a single processor level, a Nitrogen6X developer kit is used. The Nitrogen6X is built with iMX6Quad CPU, clocked 1GHz. It is connected via Ethernet interface to a Virtual Machine containing the development environment. Measurements of the execution time are done on the target and exported in `.csv` format to the VM. The results are scaled down to 800MHz to match the IMA architecture. Mode mapping and latency analysis are based on the single iMX6Quad CPU result measured from the Nitrogen6X board.

2.2.1 Target Hardware

Target hardware is a Nirtogen6X development board (Appendix A.1) from Boundary Devices Inc. It is a low cost development kit built with iMX6Quad CPU. 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

2.2.2 Software Development Platform

Linaro Linux is installed on the Nitrogen6X board along with Lightweight X11 Desktop Environment. The development board is connected to a Virtual Machine via Ethernet. The VM is accessed from Windows Server PC. Other details are:

- Linaro Linux 3.0.35-02828-g5cedf96 is installed on Nirtogen6X board.
- Debian 3.2.0-4-686-pae is installed on VM.
- 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 [10].

A simulation of the Radar application is written in C and compiled with the following parameters:

```
-O3 -funsafe-math-optimizations -ftree-vectorize -mvectorize-with-neon-
quad -ftree-vectorizer-verbose=2 -funroll-loops -fprefetch-loop-
arrays "-Wa,-ahl=$*.s" -Wall -march=armv7-a -mcpu=cortex-a9 -mfpu=
neon -c -mfloating-abi=hard -fmessage-length=0
```

FFTW 3.3.3 library is compiled with the following parameters:

```
./configure CC="arm-linux-gnueabihf-gcc -march=armv7-a -mcpu=cortex-a9
-mfpu=neon -mfloating-abi=hard -funsafe-math-optimizations -funroll-
loops -fprefetch-loop-arrays -ftree-vectorize -mvectorize-with-neon-
quad -O3" --host=arm-linux-gnueabihf --enable-float --enable-neon --
enable-threads --prefix=/home/e001919d/fftw-3.3.3/version06
```

2.3 Air to Air Mode Processing Chain

Previous work has been carried out by Airbus DS in Future Combat Air System (FCAS) project, see [8]. 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 boresight. The azimuth scan of the antenna is sub-divided into 2×5 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. 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 determines the data receive rate. Table 2.1 shows the maximum execution time of the Radar functions running on a single core, 1GHz processor. The mode mapping analysis scales down the results to 800MHz to match the IMA processor architecture.

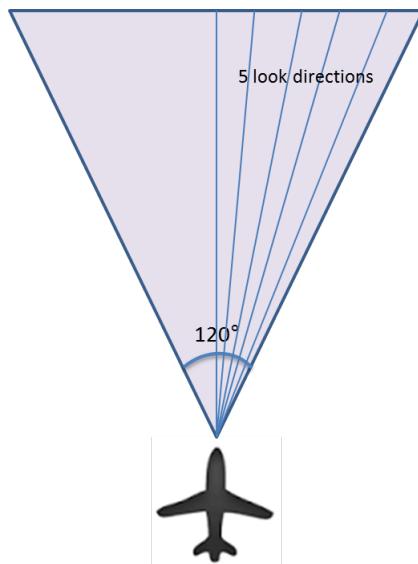


Figure 2.4: Details of the A/A Mode Scan

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

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

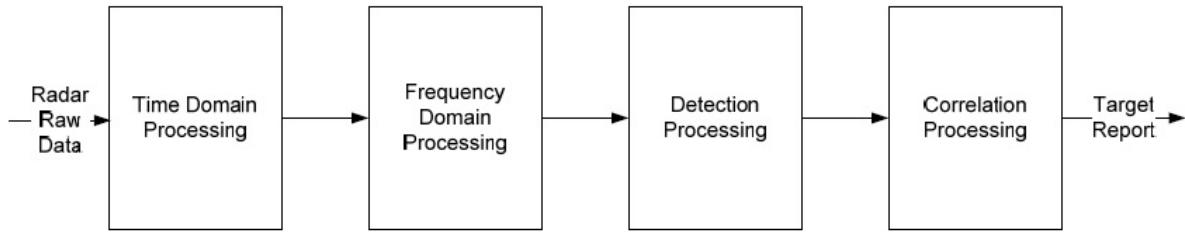


Figure 2.5: Block Diagram of Air to Air Mode Processing

2.3.1 Time Domain Processing

During Sum channel Beamforming, the input vectors of 8 antenna channels are multiplied with pre-calculated Sum channel beamforming 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 beamforming, only requires float conversion. Each of the processed four channel data are multiplied by pre-calculated weighting vector.

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, Range Multiply followed by IFFT. This step is called pulse compression.

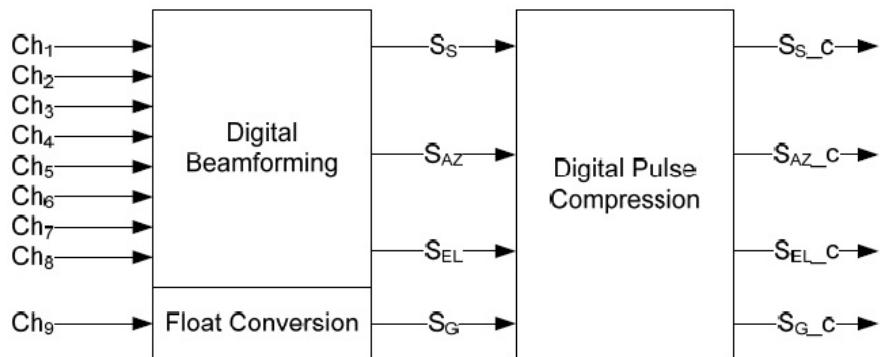


Figure 2.6: A/A Mode Time Domain Processing

2.3.2 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] ma-

trix as shown in Figure 2.7, then FFT is computed for the Sum, Guard, Azimuth and Elevation channel followed by magnitude of the Sum channel and Guard channel are computed.

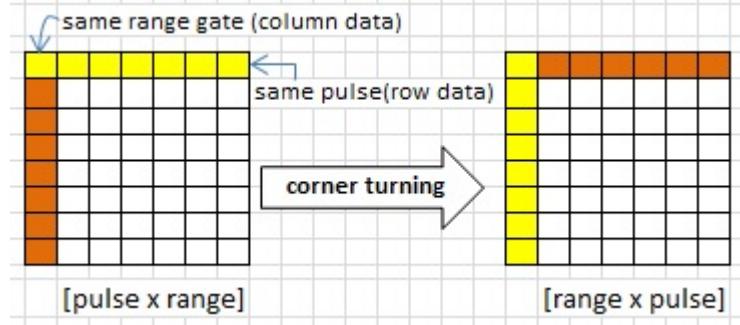


Figure 2.7: Corner Turning

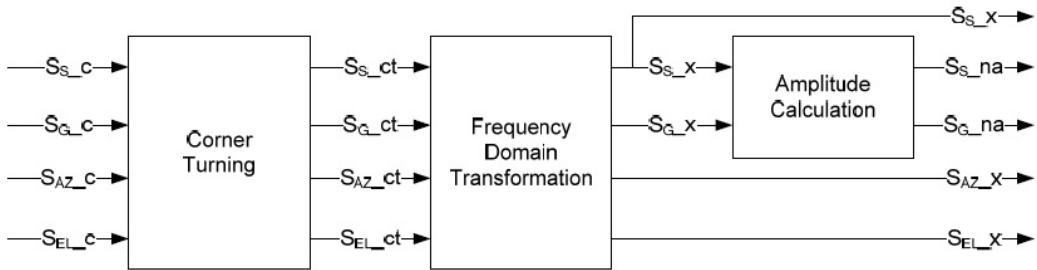


Figure 2.8: A/A Mode Frequency Domain Processing

2.3.3 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. 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.

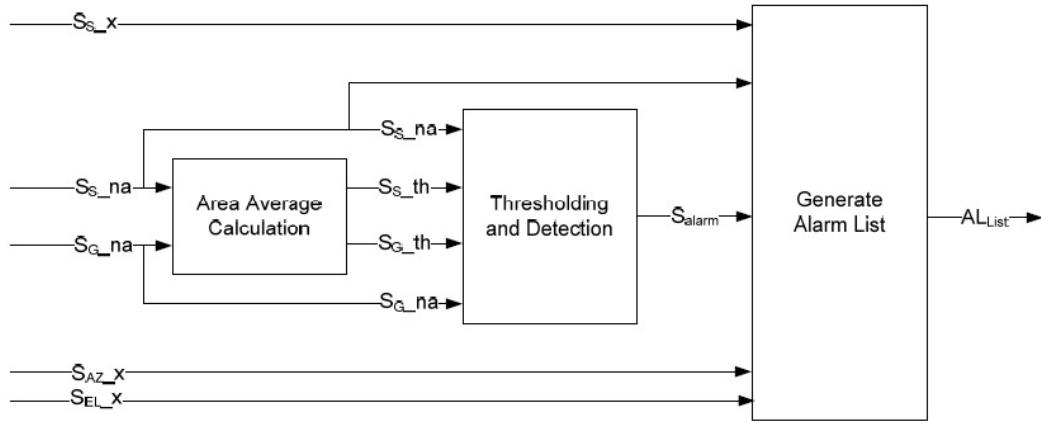


Figure 2.9: A/A Mode Detection Processing

2.3.4 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.

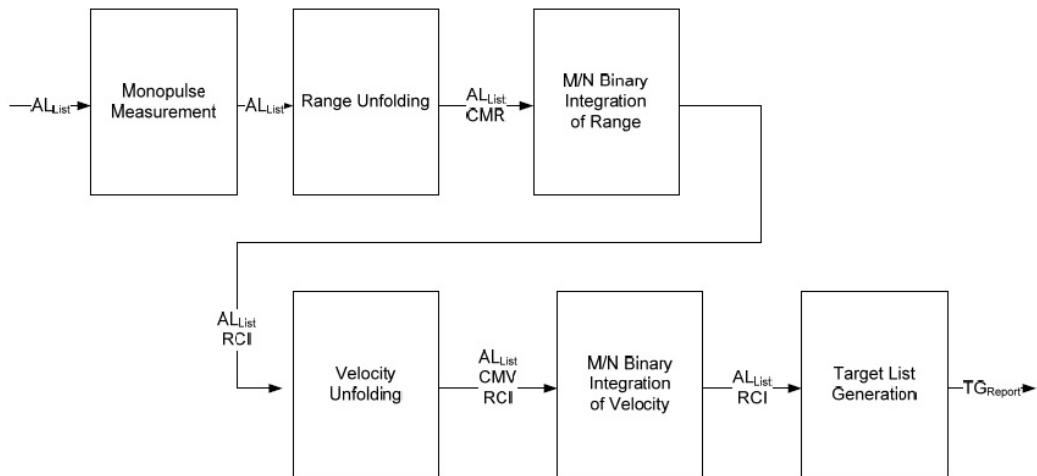


Figure 2.10: A/A Mode Correlation Processing

2.4 Benchmark Results

Performance critical functions related to the Radar processing algorithm are benchmarked by Airbus DS. The benchmarks are executed in the ARM Cortex A9 processor, clocked 1GHz and their execution times are listed in Table 2.1. Explanation about the benchmarks can be found in Appendix B.

Function Name	#cycles	Unit
100CMYACC8	79	per 8 element
100RMY50	15	per complex element
100CONV64	6,500	per 64-pt vector
100CONV128	24100	per 128-pt vector
150COT50	12	per complex element
100FFT64	2,550	per 64-pt vector
100FFT128	10,600	per 128-pt vector
50FFT256	29,700	per 256-pt vector
50MAG256	20	per complex element
64AVG100	20	per element
64CMPR100	7	per element
64DET100	10	per element
50FTFW1k	110,000	per 1024-pt vector
50FTFW2k	248,000	per 2048-pt vector
1CONV1k	190,300	per 1024-pt vector
50FFT1k	96,000	per 1024-pt vector
50FFT2k	219,300	per 2048-pt vector
10MAGSQ1k	10	per complex element
100LOG256	12	per element
100LIM256	8	per element

Table 2.1: Measured Execution Cycle of the Benchmarks

2.5 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 $\pm 3\text{Mach}$. A burst shall have a maximum of 32 alarms and a maximum of 16 targets, formed out of the 32 alarms. Each sample of the received echo consists of a complex int_16 value, Real and Imaginary parts, stored in 4 bytes.

Figure 2.11 shows the delicate 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

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
Range gates	103	99	90	83	77	68	59	52
Burst time[ms] = PRF/pulse count	3.28	3.50	3.44	3.53	3.63	3.57	3.46	3.43
	3.90	4.14	4.07	4.19	4.32	4.25	4.11	4.08
	4.67	4.93	4.84	4.94	5.10	5.03	4.85	4.82
	5.49	5.76	5.70	5.85	6.02	5.95	5.74	5.68
	6.46	6.80	6.70	6.85	7.07	6.97	6.75	6.68
#range unfoldings	10	11	11	12	13	15	17	19
#velocity unfoldings	8	8	6	6	6	6	4	4

Figure 2.11: A/A Mode Radar Characteristics

have 103 range gates. Range unfolding and Velocity unfolding are performed to compute the unambiguous range and unambiguous velocity of the target. Burst time and Dwell time calculations for the same is done here.

$$\text{Burst time} = \frac{1}{19.5 * 10^3 \text{ Hz}} * 64$$

$$= 3.28 \text{ ms}$$

$$\text{Dwell time} = \sum_{n=1}^8 \text{Burst time}_n$$

$$= 3.28 + 3.50 + 3.44 + 3.53 + 3.63 + 3.57 + 3.46 + 3.43$$

$$= 27.84 \text{ ms}$$

$$\text{Unambiguous Range} = \frac{1}{2} \frac{\text{velocity of the radar signal}}{\text{PRF frequency}}$$

$$\text{Range Unfoldings} = \frac{\text{max.range}}{\text{unambiguous range}} = \frac{(40 * 1852) * 19.5 * 10^3 * 2}{3 * 10^8} = 10$$

$$\text{Unambiguous Velocity} = \frac{1}{2} \frac{\text{velocity of the radar signal}}{\text{carrier frequency}} * \text{PRF frequency}$$

$$\text{Velocity unfolding} = 2 * \text{ROUNDUP}\left(\frac{\text{max.velocity}}{\text{unambiguous velocity}}\right)$$

$$= 2 * \text{ROUNDDUP} \left(\frac{1020 * 9 * 10^9 * 2}{3 * 10^8 * 19.5 * 10^3} \right) = 8 \quad (2.1)$$

According to the STREAM benchmark, 400 MiB/s data transfer rate is measured to copy data from one memory location of the SDRAM to another memory location, when 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 [8]

2.6 Related Work

Deploying ARM processors in fighter aircraft is in the research stage at the moment. Footprints of the low cost Radar processor has been considered as related work of this thesis. Many innovative architectures have been investigated in past to reduce the size, weight and power 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 [11], 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.

ARM type processor has been used by Zeng Y, Xu J and Peng D in Velocity Measuring System [12]. 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.

Salama Y, Fitzgerald D et al [13], 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 results 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 [14], 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.

Chapter 3

Existing Analysis

The Radar processor should be capable of processing both the Air to Air mode data as well as Air to Ground mode data (SAR). Scope of this thesis is limited to A/A Mode processing. Mode mapping analysis in this chapter are based on the timing information provided in the Table 2.1. Existing work by Airbus DS[8] configures the mode mapping 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 time-frame 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

This section explains the formulas and calculations in the excel document provided by Airbus DS. A RADAR echo received by the Radar antenna is distinguished as A/A mode or A/G mode by the iCON1 module.

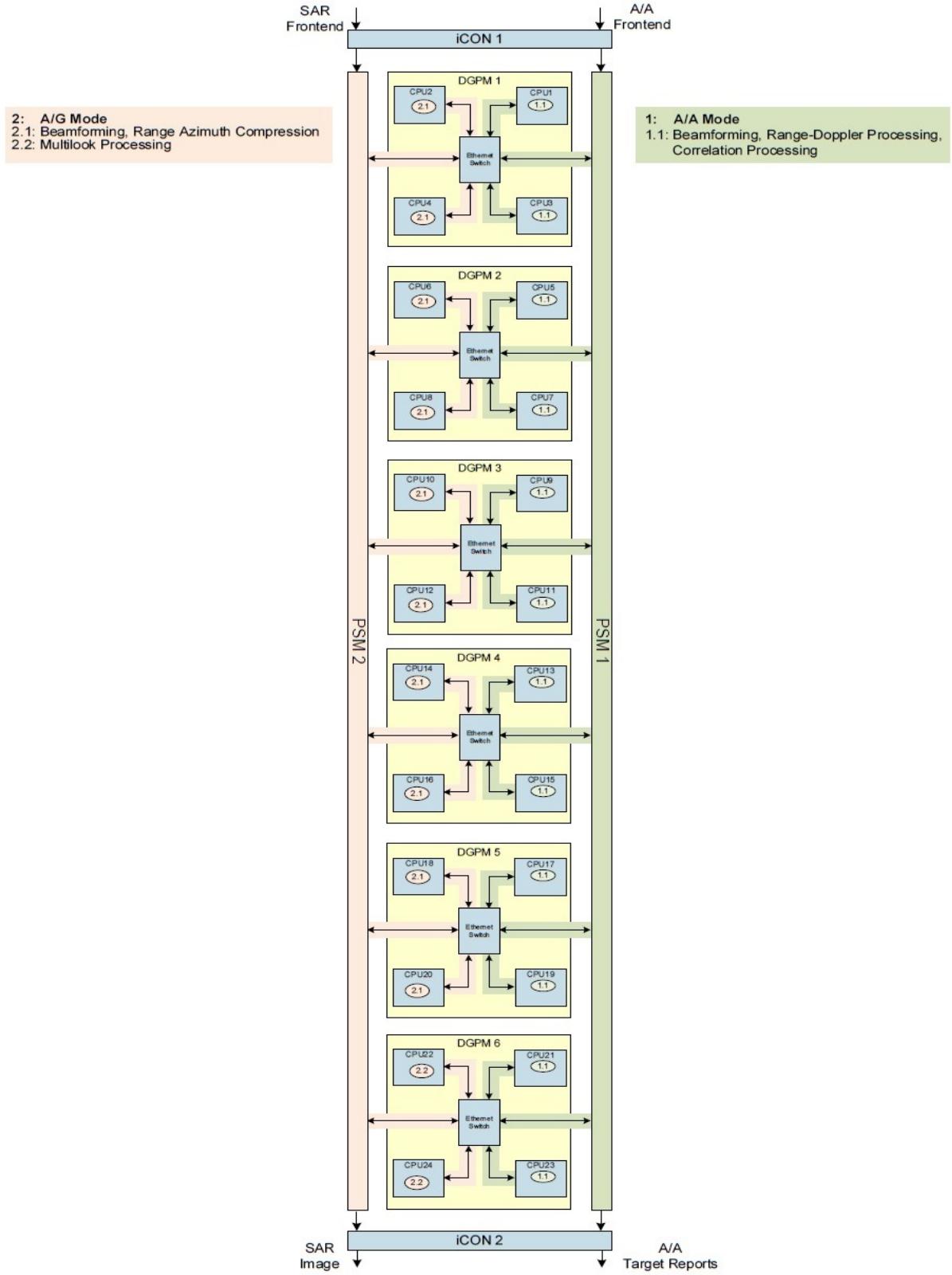


Figure 3.1: Scheme-1, Mode Mapping

A/A mode data are redirected to PSM1 and A/G mode data are sent to PSM2. The two odd numbered CPUs(CPU1 and CPU3) of a DGPM are allocated to the processing of the A/A mode. The two even numbered CPUs(CPU2 and CPU4) of a DPGM are alloctated 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: $6 \times 4 = 24$

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 indepentent to the processing of the successor dwell data, there is no data dependency between CPUs processing different dwell data.

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 2.5. As depicted in Figure 3.3, 19% of the Core1's time is utilized in worst-case scenario. Calculations for the first burst of the look direction-1 is explained for each profiling.

Core1 receives incoming data from the Ethernet and stores them into Core1's buffer memory in SDRAM. Then the data is copied to Core2's buffer memory.

$$\begin{aligned} \text{Burst data input} &= 64 \text{ pulses} * 103 \frac{\text{rangeegates}}{\text{pulse}} * 9 \text{ channel} * 4 \frac{\text{byte}}{\text{sample}} \\ &= 237,312 \text{ byte} \end{aligned}$$

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

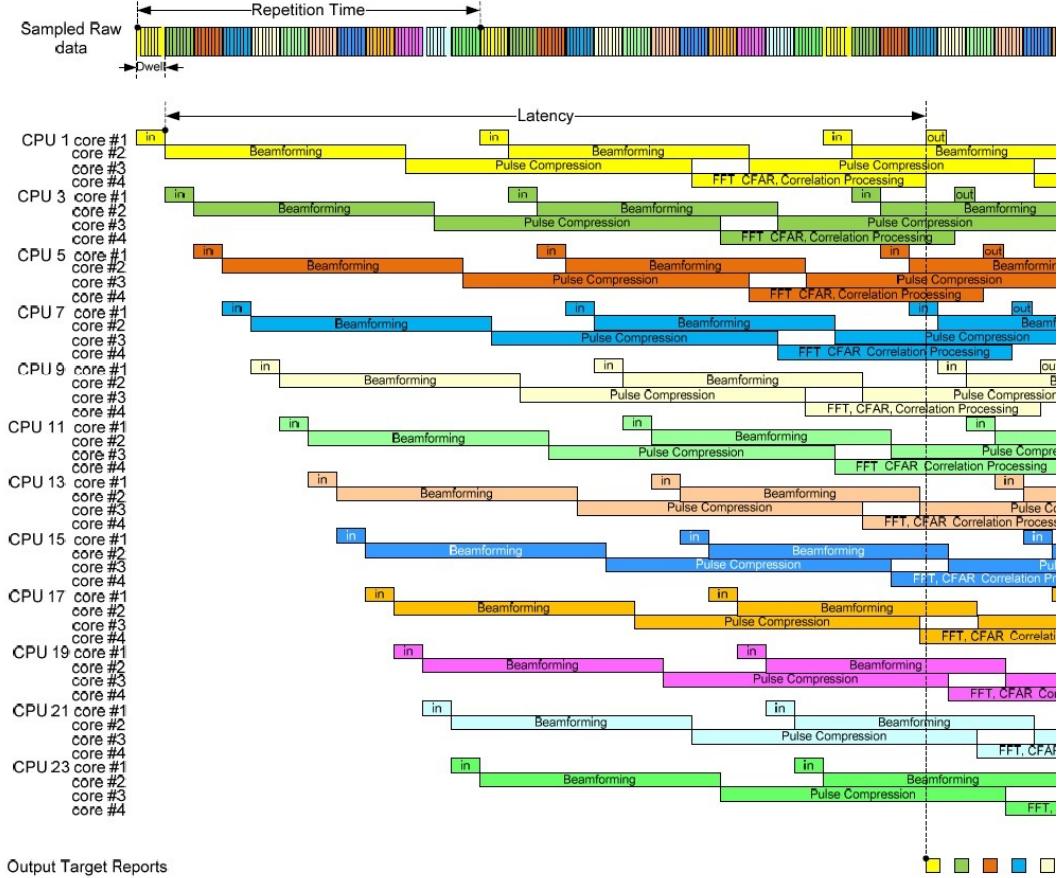


Figure 3.2: Scheme-1, A/A Mode Processing

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

$$\text{Required time} = 3.28 \text{ ms} + 0.59 \text{ ms} = 3.88 \text{ ms}$$

$$\text{Available time} = 12 * \text{min.dwelltime} = 12 * 27.84 \text{ ms} = 334.05 \text{ ms} \quad (3.1)$$

Core2 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 core3's buffer memory in SDRAM. As shown in Figure 3.4, maximum utilization of Core2 reaches upto 24% of the available time. The burst data input and data copy time are the same as the Equation 3.1. Every beamformed sample occupies 8 byte of memory.

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

I/O by Core 1									Look Dir.	
	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%
										11,68%
										13,83%
										16,32%
										19,16%

Figure 3.3: Scheme-1, Core1 Utilization

$$\text{Beamforming processing} = 3\text{channel} * 64\text{pulses} * 103 \frac{\text{rangegeates}}{\text{pulse}} * 79 \frac{\text{cycle}}{8\text{element}}$$

$$* 1.25 \frac{\text{ns}}{\text{cycle}} * 1.3(\text{OSoverhead}) = 2.54\text{ms}$$

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

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

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

$$\text{Required time} = 0.59\text{ms} + 0.68\text{ms} + 2.54\text{ms} + 0.60\text{ms} + 0.53\text{ms} = 4.94\text{ms} \quad (3.2)$$

Core3 reads the data from its SDRAM partition and performs Pulse Compression. The results of the processing are stored back to the SDRAM. Taken from Figure 3.5, maximum utilization of Core3 is 65% of the available time. Time required to copy the input data and store back pulse compressed data are the same as the Equation 3.2. The input data set [64 x 103] is zero padded to the next power of 2 [64 x 128] to perform 128-point convolution.

$$\text{Pulse compression} = \left(\# \text{channel} * \# \text{pulses} (\text{CONV128} + \# \frac{\text{rangegeates}}{\text{pulse}} * \text{RMY50}) \right)$$

Beamforming by Core 2									Look Dir.
	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	
Burst data input [byte]	237.312	253.044	246.240	253.980	260.568	257.040	248.508	245.232	2.001.924
	281.808	299.376	291.600	301.788	310.464	306.000	295.236	292.032	2.378.304
	337.428	356.400	346.680	355.572	365.904	362.304	348.336	344.448	2.817.072
	396.756	416.988	408.240	421.308	432.432	428.400	412.056	406.224	3.322.404
	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
data copy time [ms]	0.59	0.63	0.62	0.63	0.65	0.64	0.62	0.61	5,00
	0.70	0.75	0.73	0.75	0.78	0.77	0.74	0.73	5,95
	0.84	0.89	0.87	0.89	0.91	0.91	0.87	0.86	7,04
	0.99	1,04	1,02	1,05	1,08	1,07	1,03	1,02	8,31
	1,17	1,23	1,20	1,23	1,27	1,25	1,21	1,19	9,76
SDRAM to L2 cache time [ms]	0.68	0.72	0.70	0.73	0.74	0.73	0.71	0.70	5,72
	0.81	0.86	0.83	0.86	0.89	0.87	0.84	0.83	6,80
	0.96	1,02	0.99	1,02	1,05	1,04	1,00	0.98	8,05
	1,13	1,19	1,17	1,20	1,24	1,22	1,18	1,16	9,49
	1,33	1,41	1,37	1,41	1,45	1,43	1,38	1,36	11,15
Beamforming processing [ms]	2,54	2,71	2,63	2,72	2,79	2,75	2,66	2,62	21,42
	3,01	3,20	3,12	3,23	3,32	3,27	3,16	3,12	25,44
	3,61	3,81	3,71	3,80	3,91	3,88	3,73	3,68	30,14
	4,24	4,46	4,37	4,51	4,63	4,58	4,41	4,35	35,54
	5,00	5,26	5,13	5,27	5,43	5,37	5,18	5,11	41,75
Burst data after Beamforming [byte]	210.944	224.928	218.880	225.760	231.616	228.480	220.896	217.984	1.779.488
	250.496	266.112	259.200	268.256	275.968	272.000	262.432	259.584	2.114.048
	299.936	316.800	308.160	316.064	325.248	322.048	309.632	306.176	2.504.064
	352.672	370.656	362.880	374.496	384.384	380.800	366.272	361.088	2.953.248
	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
L2 to SDRAM cache time [ms]	0.60	0.64	0.63	0.65	0.66	0.65	0.63	0.62	5,08
	0.72	0.76	0.74	0.77	0.79	0.78	0.75	0.74	6,04
	0.86	0.91	0.88	0.90	0.93	0.92	0.88	0.87	7,15
	1,01	1,06	1,04	1,07	1,10	1,09	1,05	1,03	8,44
	1,19	1,25	1,22	1,25	1,29	1,27	1,23	1,21	9,91
data copy time [ms]	0,53	0,56	0,55	0,56	0,58	0,57	0,55	0,54	4,45
	0,63	0,67	0,65	0,67	0,69	0,68	0,66	0,65	5,29
	0,75	0,79	0,77	0,79	0,81	0,81	0,77	0,77	6,26
	0,88	0,93	0,91	0,94	0,96	0,95	0,92	0,90	7,38
	1,04	1,09	1,07	1,10	1,13	1,12	1,08	1,06	8,67
required time [ms]	4,94	5,27	5,13	5,29	5,42	5,35	5,17	5,10	41,67
	5,87	6,23	6,07	6,28	6,46	6,37	6,15	6,08	49,51
	7,02	7,42	7,22	7,40	7,62	7,54	7,25	7,17	58,64
	8,26	8,68	8,50	8,77	9,00	8,92	8,58	8,46	69,16
	9,73	10,24	9,98	10,26	10,56	10,45	10,08	9,94	81,23
									available time [ms]
									334,05
									utilization [%]
									12,48%
									14,82%
									17,56%
									20,70%
									24,32%

Figure 3.4: Scheme-1, Core2 Utilization

$$* \# \text{ cycle time} * O\text{Soverhead}$$

$$= (4 * 64(24100 + 103 * 15)) * 1.25 * 1.3 = 10.67ms \quad (3.3)$$

Pulse Compression by Core 3									Look Dir.	
	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

Figure 3.5: Scheme-1, Core3 Utilization

Core4 gets the data from its SDRAM partition and performs FFT, CFAR, Correlation Processing. The results of the processing are stored back to the SDRAM. As mentioned in Figure 3.6, maximum utilization of the Core4 is reaching 75% of the available time. Time required to copy the input data is same as the Equation 3.2. FFT64 is chosen for Frequency Domain Transformation, as the corner turned data set size is [103 x 64].

$$\begin{aligned}
 FDP\ time &= \left(4\text{channel} * (\#\text{pulses} * \# \frac{RG}{pulse} (COT50 + RMY50) + \# \frac{RG}{pulse} * FFT64) \right. \\
 &\quad \left. + 2 * \left(\# \frac{\text{range gates}}{pulse} * 64 * MAG256 \right) \right) * OS\ overhead * \text{cycle time} \\
 &= (4 * (64 * 103(12 + 15) + 103 * 2550) + 2 * (103 * 64 * 20)) * 1.3 * 1.25ns \\
 &= 3.29ms \tag{3.4}
 \end{aligned}$$

									Look Dir.	
	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	SUM	
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. Thresholding comparison and Alarmlist generation	1.11	2.14	1.95	1.80	1.67	1.47	1.28	2.25	13.66	1
	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	
	0.15	0.16	0.16	0.17	0.18	0.20	0.22	0.25	1.51	
	0.32	0.35	0.35	0.38	0.40	0.45	0.51	0.56	3.32	
	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.07	
	3.33	4.03	4.03	4.79	5.62	7.49	9.62	12.01	50.92	
I/N Binary Integration of Velocity [ms]	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.02	
	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.01	
	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.01	
	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.02	
	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.0026	0.02	
Target List Generation [ms]	3.87	4.61	4.60	5.41	6.27	8.21	10.41	12.89	56.26	
	17.84	18.42	17.42	17.67	18.06	19.05	19.58	29.34	146.65	1
	18.59	19.19	18.16	18.39	30.67	30.28	30.02	30.67	167.73	2
	19.39	20.00	18.99	32.07	31.57	31.17	30.87	31.50	195.98	3
	20.34	36.26	33.82	33.03	32.57	32.16	31.85	32.46	215.56	4
ta to tracking (once every dwell) [ms]									252.48	5
									252.48	5
									252.48	5
									252.48	5
									252.48	5
									available time [ms]	334.05
									utilization [%]	43.90%
										50.21%
										58.67%
										64.53%
										75.58%

Figure 3.6: Scheme-1, Core4 Utilization

After Frequency Domain Transformation and Magnitude Calculation, The Sum channel and Guard channel data have 4 byte per magnitude sample, Azimuth channel and Elevation channel data have 8 byte per complex sample.

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

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

$$\begin{aligned} \text{Average Calculation} &= 2\text{channel} * 103 \frac{\text{rangeagates}}{\text{pulse}} * 64(\text{FFTsize}) * 20(\text{AVG100}) * 2 \\ &= 527,360 \text{ cycle} \end{aligned}$$

$$\begin{aligned} \text{Comparison} &= 2\text{channel} * 103 \frac{\text{rangeagates}}{\text{pulse}} * 64(\text{FFTsize}) * 7(\text{CMPR100}) \\ &= 92,288 \text{ cycle} \end{aligned}$$

$$\begin{aligned} \text{Detection} &= 1\text{channel} * 103 \frac{\text{rangeagates}}{\text{pulse}} * 64(\text{FFTsize}) * 10(\text{DET100}) \\ &= 65,920 \text{ cycle} \end{aligned}$$

$$\begin{aligned} \text{Time required} &= (\text{Average calculation} + \text{Comparison} + \text{Detection}) \\ &\quad * \text{OSoverhead} * \text{cycle time} \\ &= (527,360 + 92,288 + 65,920) * 1.3 * 1.25 \text{ ns} = 1.1 \text{ ms} \end{aligned} \quad (3.5)$$

$$\begin{aligned} \text{Monopulse Measurement} &= 2\text{channel} * \# \frac{\text{alarms}}{\text{burst}} * \# \frac{\text{cycle}}{\text{alarm}} * \text{OSoverhead} * \text{cycle time} \\ &= 2 * 32 * 500 * 1.3 * 1.25 \text{ ns} = 0.05 \text{ ms} \end{aligned}$$

$$\begin{aligned} \text{Range Unfolding} &= \left(RG_{max} * \# \frac{\text{cycle}}{\text{rangegate}} + \# \text{Unfoldings} * \# \frac{\text{alarms}}{\text{burst}} * \# \text{cycle} \right) \\ &\quad * \text{OSoverhead} * \text{cycle time} \\ &= (987 * 30 + 10 * 32 * 200) * 1.3 * 1.25 \text{ ns} = 0.15 \text{ ms} \end{aligned}$$

$$\begin{aligned} \text{M/N Range Integration} &= \left(\# \text{RangeGates} * \# \frac{\text{cycle}}{\text{rangegate}} + \# \text{Unfoldings} * \# \frac{\text{alarms}}{\text{burst}} \right. \\ &\quad \left. * \# \frac{\text{cycle}}{\text{alarm}} \right) * \text{OSoverhead} * \text{cycle time} \\ &= (988 * 40 + 32 * 10 * 500) * 1.3 * 1.25 \text{ ns} = 0.32 \text{ ms} \end{aligned}$$

$$\begin{aligned} \text{Velocity Unfolding} &= \left(\# \text{Unfoldings} * \# \frac{\text{alarms}}{\text{burst}} * \# \frac{\text{cycle}}{\text{alarm}} \right) \\ &\quad * \text{OSoverhead} * \# \text{cycle time} \\ &= (8 * 32 * 30) * 1.3 * 1.25 \text{ ns} = 0.01 \text{ ms} \end{aligned}$$

$$\begin{aligned} \text{M/N Velocity Integration} &= \left(\# \frac{\text{alarms}}{\text{burst}} * \# \text{Unfoldings}^2 * \# \text{bursts} * \# \frac{\text{cycle}}{\text{alarm}} \right) \\ &\quad * \text{OSoverhead} * \text{cycle time} \end{aligned}$$

$$\begin{aligned}
&= (32 * 10^2 * 8 * 80) * 1.3 * 1.25ns = 3.33 \text{ ms} \\
\text{Target List Generation} &= \left(\# \text{targets} * \# \frac{\text{cycle}}{\text{target}} \right) * \text{OS overhead} * \text{cycle time} \\
&= (16 * 100) * 1.3 * 1.25ns = 0.0026 \text{ ms} \\
\text{Correlation Processing}[ms] &= 0.05 + 0.15 + 0.32 + 0.01 + 3.33 + 0.0026 = 3.87 \text{ ms} \\
\text{Required time}[ms] &= 0.53 + 0.60 + 3.29 + 0.45 + 0.45 + 0.11 + 3.87 = 10.31 \text{ ms}
\end{aligned} \tag{3.6}$$

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. In case of Active Electronically Scanned Array (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. Derivation of Look direction-1's processing latency is explained below.

$$\begin{aligned}
\text{Processing latency} &= \text{IO} + \text{Beamforming} + \text{Pulse compression} + \text{FFT, CFAR, Correlation} \\
&= 32.84 + 41.67 + 112.51 + 146.65 = 333.67 \text{ ms} \\
\#\text{Dwell latency} &= \frac{\text{processing latency}}{\text{dwell time}} = \frac{333.67}{27.84} = 11.99
\end{aligned} \tag{3.7}$$

Look direction	Dwell time[ms]	Latency[ms]	#Dwell latency
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: Scheme-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 are explained in each core's CPU utilization section. Figure 3.7 shows that 7% of the available capacity is sufficient for A/A Mode processing.

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%

Figure 3.7: Scheme-1, Memory Utilization

Interface Utilization

Interfaces are the data routing paths in the Radar processor. Nominal bandwidth of 100MiB/s data rate is assumed for the interfaces. The DGPMs are identical, thus only one DGPM's utilization is listed here. As calculated in Figure 3.8, peak interface utilization is inferred as 72%.

Interface														
[MByte/s]	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8		max. average data rate [MByte/s]	Available bandwidth [Mbytes/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%

Figure 3.8: Scheme-1, Interface Utilization

$$\begin{aligned}
 iCON1 \text{ output to PSM1} &= \# \frac{\text{byte}}{\text{sample}} * \# \text{channel} * \text{range gate} * \text{frequency} \\
 &= 4 * 9 * 103 * 19.5kHz = 72.31 MB/s \\
 \text{DGPM input from PSM} &= \frac{\text{data rate}}{\# \text{DGPMs}} = \frac{72.31}{6} = 12.05 MB/s
 \end{aligned}$$

$$\begin{aligned}
 CPU\ input &= \frac{DGPM\ data\ rate}{\#CPUs} = \frac{12.05}{2} = 6.02\ MB/s \\
 DGPM\ output\ to\ PSM &= \frac{data\ rate}{\#DGPMs} = \frac{0.31}{6} = 0.0526\ MB/s \\
 CPU\ output &= \frac{DGPM\ output}{\#CPUs} = \frac{0.0526}{2} = 0.026\ MB/s
 \end{aligned} \tag{3.8}$$

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.

	Core1	Core2	Core3	Core4
Utilization	19.16%	24.32%	65.65%	75.58%

Table 3.2: Scheme-1, CPU Utilization

3.2 Scheme 2 - Time Partitioning

This section explains the formulas and calculations in the excel document provided by Airbus DS. Every CPU 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 3.10.
- Partition change time is 0.5ms.

Figure 3.9 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.

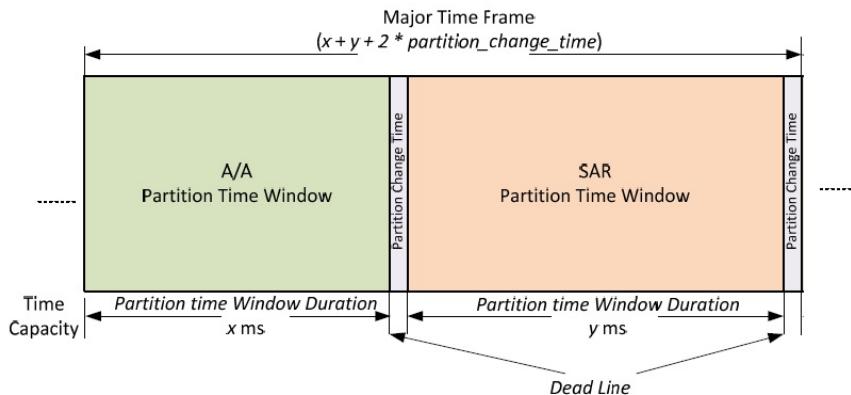


Figure 3.9: Scheme-2, Time Partition

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 transferred 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 3.10: Scheme-2, Values for Time Partition

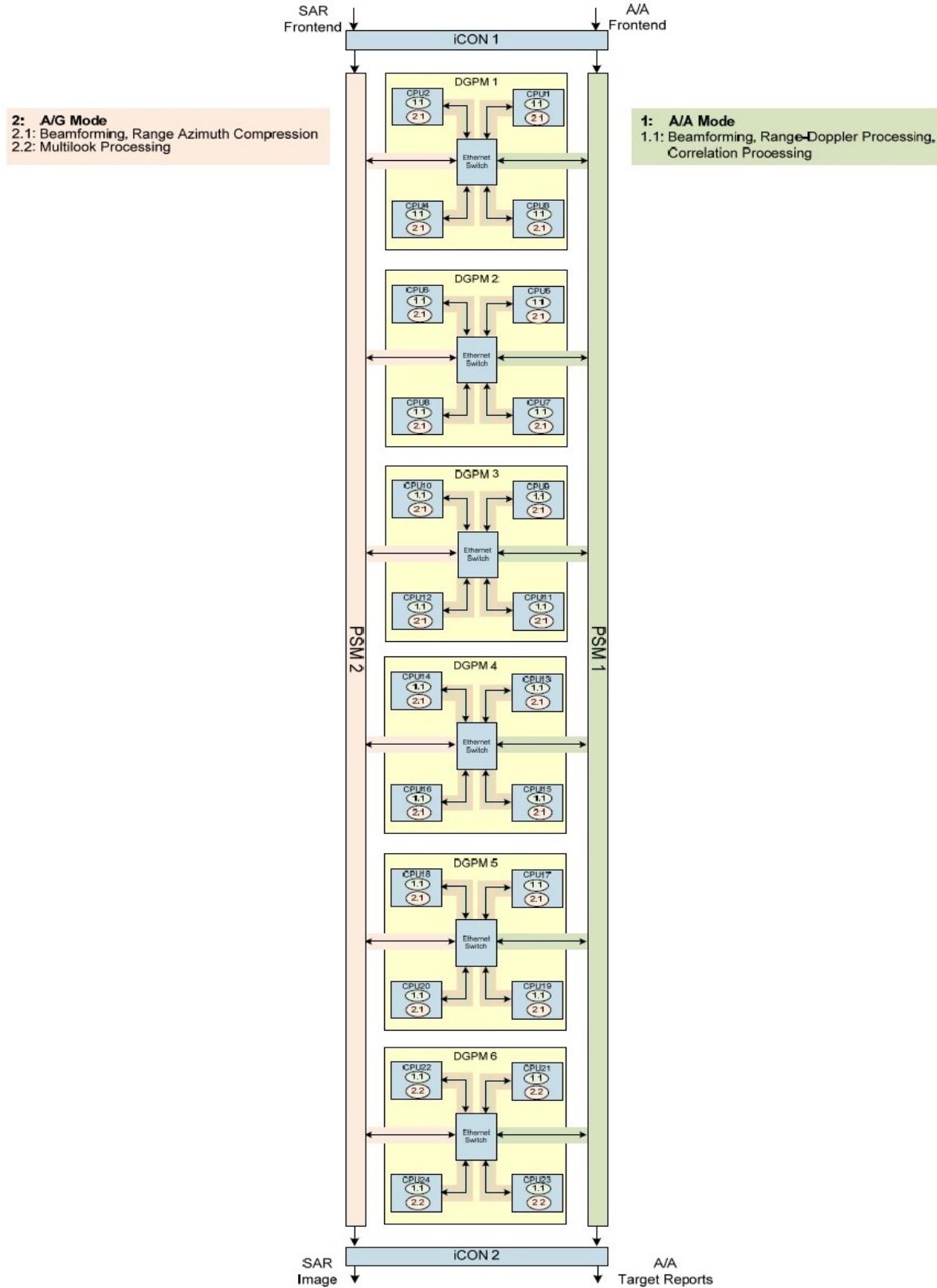


Figure 3.11: Scheme-2, Mode Mapping

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 34th 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.

	Core1	Core2	Core3	Core4
Core Utilization	25%	53%	71%	63%

Table 3.3: Scheme-2, 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.

Look direction	Dwell time[ms]	Latency[ms]	#Dwell latency
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: Scheme-2, 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. As seen in Figure 3.12, peak memory utilization reaches upto 9% of the available memory.

SDRAM Memory Usage [Byte]									Sum
	PRF1	PRF2	PRF3	PRF4	PRF5	PRF6	PRF7	PRF8	
system-buffer Core 1	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
partition-buffer Core 1	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
input system-buffer Core 2	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
input partition-buffer Core 2	467.208	491.832	479.520	493.020	507.276	501.840	484.272	477.360	3.902.328
output system-buffer Core 2	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
output partition-buffer Core 2	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
input system-buffer Core 3	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
input partition-buffer Core 3	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
output system-buffer Core 3	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
output partition-buffer Core 3	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
input system-buffer Core 4	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
input partition-buffer Core 4	415.296	437.184	426.240	438.240	450.912	446.080	430.464	424.320	3.468.736
output system-buffer Core 4	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	8.192
output partition-buffer Core 4	1.024	1.024	1.024	1.024	1.024	1.024	1.024	1.024	8.192
Core 2 Processing: 2 x Processing Buffer [Byte]	934.416	983.664	959.040	986.040	1.014.552	1.003.680	968.544	954.720	7.804.656
Core 3 Processing: 4 x Processing Buffer [Byte]	1.661.184	1.748.736	1.704.960	1.752.960	1.803.648	1.784.320	1.721.856	1.697.280	13.874.944
Core 4 Processing: 8 x Processing Buffer [Byte]	3.322.368	3.497.472	3.409.920	3.505.920	3.607.296	3.568.640	3.443.712	3.394.560	27.749.888
								SUM	92.805.072
								available memory	1.000.000.000
								utilization ratio [%]	9%

Figure 3.12: Scheme-2, Memory Utilization

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. Since every CPU is executing A/A mode processing as well as A/G mode processing, the combined results have to be taken into consideration to estimate peak values of the aforementioned profiling.

Chapter 4

Test Bed and Design Decisions

4.1 Pros and Cons of the Existing Analysis

The contribution of this thesis starts with scrutinizing the Existing Analysis. Upsides and downsides are investigated and verified to implement the best practice.

4.1.1 Pros

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

Cache Miss

Memory access time is different for different memory hierarchy levels. For instance, register memory can be accessed in one cycle, L1 cache in few cycles, L2 cache in 10 cycles and local DRAM needs approximately 100 cycles to be accessed by the CPU. To compute magnitude square root of 20 elements and store back the results to L1 cache, a core requires 280 cycles if the data are in L1 cache, 2140 cycles if the data are in SDRAM, assuming 4 cycles L1 access time and 10 cycle execution time per element. This kind of major variations are unfavourable for determinism. For safety critical systems the Worst Case Execution Time(WCET) needs to be analyzed and verified. The cache miss behaviour needs to be considered in the determination of the WCET. To achieve the worst case memory access times, it is assumed in the Existing Analysis that a cache miss is produced for each memory access.

One of the ways to calculate Worst Case Execution Time (WCET) is to assume cache miss always. This is a pessimistic approach as the cache hit ratio is not always 0%.

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 load balancing. In Scheme-3 A/A mode analysis, 81% of the core4's L2 partition is utilized whilst only 6% of the core1's L2 partition is utilized. This quantitative information helps to reconfigure the partition values to have good load balance.

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, core4 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 core4. 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 load balancing.

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 (Scheme2 and Scheme3), 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.

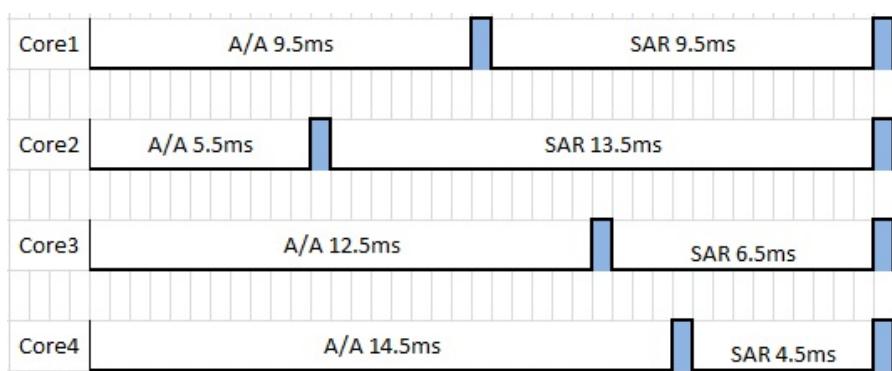


Figure 4.1: Time Partition of Scheme 3

Assuming the order of SDRAM access is core1, core2, core3 and core4. Core2 waits for core1 to complete (0.5ms), core3 waits for core1,2 (0.5ms + 0.5ms), core4 waits for core1,2,3

(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. Worstcase waiting time for four cores is

$$\begin{aligned}
 &= \frac{4 * \text{ContextSwitchTime} + 2 * \text{OverheadInWaitingTime}}{\text{AvailableTime}} \\
 &= \frac{4 * 2 * 0.5\text{ms} + 2 * 3\text{ms}}{4 * 20\text{ms}} = 12.5\%
 \end{aligned} \tag{4.1}$$

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[15] 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. 7.43 μs 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

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

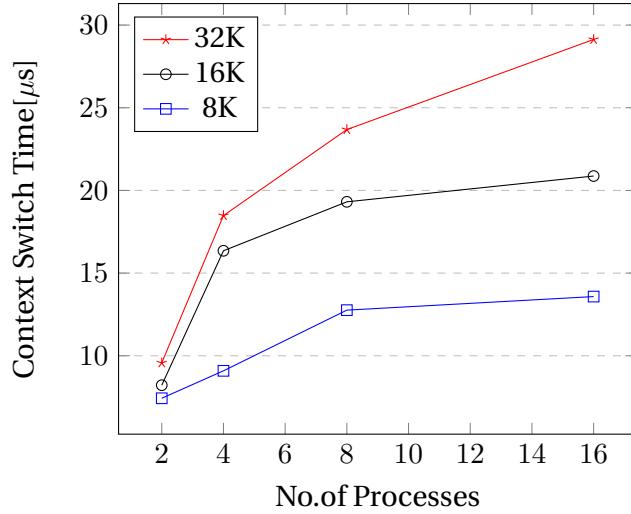


Figure 4.2: Comparison of Context Switch Time

CPU utilization. So, no changes are made in the Existing Analysis with respect to the context switch time.

Clashing Data Streams

In time partition scheme, assume that the core1 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 existing analysis, as a result it leads to data loss if no countermeasure is provided.

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

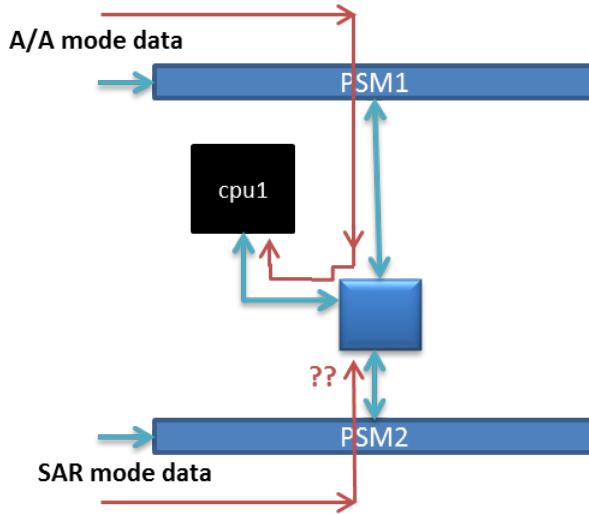


Figure 4.3: Clashing Data Streams

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

Typo

Decimal conversion format(1000 Byte = 1KB) was used for byte conversion. This is changed to Binary conversion format (1024 Byte = 1KiB) to comply the standards. This has significant impact in the analysis results. An example is shown for interface utilization calculation of Scheme-1 in Chapter 3.1.1.

$$\begin{aligned}
 iCON1 \text{ output to PSM1} &= \# \frac{\text{byte}}{\text{sample}} * \# \text{channel} * \text{range gate} * \text{frequency} \\
 &= 4 * 9 * 103 * 19.5 \text{kHz} = 72306000 \frac{\text{byte}}{\text{sec}} = 72.3 \text{ MB/s} \\
 \text{Actual data rate} &= \frac{72306000}{1024 * 1024} = 69 \text{ MiB/s}
 \end{aligned} \tag{4.2}$$

Ideal Data Set Size

Benchmarked functions have ideal data set range. For instance, matrix size of [1000 x 8] elements is processed to measure the execution time of complex multiply (CMY) function. Assuming 4 byte for a complex sample, 128-bit advanced SIMD unit (NEON) can hold four

elements of CMYACC in a Q register and four multiplier elements in another Q register. Results of four multiplications are computed at the cost of one multiplication operation. But in reality, size of the data set is not always multiple of 128-bit. To compute CMY of [1000 x 9] data set, first eight elements of a row can be multiplied in SIMD lanes, last one element can be processed as normal multiplication operation. By doing so, it requires same execution time to process [1000 x 9] data set and [1000 x 12] data set, neglecting intrinsic load-store time. This performance constraint has to be taken into account for the full-fledged application development.

OS Overhead

Operating System overhead(OS Overhead) of 1.3 was included in the analysis. As the execution time are measured in Linaro OS, redundant *OSoverhead* factor is removed for further analysis.

Execution Cycle Mismatch

Execution cycle reported in Chapter 2.1 and the measured execution cycle of the benchmark functions do not match. The reported values are much lower than the actual measured values. These values are amended according to the measured values for Mode Mapping analysis. Table 4.2 shows the comparison of old values versus self measured values of the A/A Mode benchmarks.

Benchmark	Old value [cycles]	Measured value [cycles]
CMYACC8	79	287
RMY50	15	68
CONV128	24100	27300
COT50	12	98
FFT256	29700	37490
MAG256	20	41
AVG256	20	51
CMPR256	7	55
DET256	10	97

Table 4.2: Comparison of Old Execution Cycle vs Measured Execution Cycle of A/A Mode Benchmarks

4.1.4 Measured Values of Existing Analysis

The following processing latency and CPU utilization values are obtained for Existing Analysis, after updating the execution cycle values and removing OS overhead factor.

Scheme-1

	Core1	Core2	Core3	Core4
Utilization	19.03%	46.20%	65.78%	108.07%

Table 4.3: Measured CPU Utilization of Scheme-1, A/A Mode

Look direction	Dwell time[ms]	Latency[ms]	#Dwell latency
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 4.4: Measured Processing Latency of Scheme-1, A/A Mode

Scheme-2

	Core1	Core2	Core3	Core4
Utilization	25.00%	103.33%	71.43%	83.33%

Table 4.5: Measured CPU Utilization of Scheme-2, A/A Mode

Look direction	Dwell time[ms]	Latency[ms]	#Dwell latency
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 4.6: Measured Processing Latency of Scheme-2, A/A Mode

Inference

The Existing Analysis will not meet real-time requirements as the CPU cores are utilized more than 100%. Another down side is the 15x dwell latency, which is not in the acceptable range.

4.2 Test Procedure

Executable code is transferred to the Nitrogen6X board via Ethernet. Benchmark profiling are done as follows.

4.2.1 Memory Transfer Bandwidth

STREAM benchmark[16] is used to measure memory bandwidth of the machine. The STREAM benchmark is a simple, synthetic benchmark designed to measure sustainable memory bandwidth (in MB/s) and a corresponding computation rate for four simple vector kernels namely Copy, Scale, Add and Triad [17]. 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 STREAM benchmark is compiled and executed as follows:

```

1 user$ gcc -fopenmp -D_OPENMP stream.c -o stream
2 user$ export OMP_NUM_THREADS=4
3 user$ ./stream

```

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 4.7, the iMX6Quad is capable of transferring **1048 MiB/s** when all the four cores are running in parallel.

Function	Best Rate [MB/s]	Avg time[s]	Min time[s]	Max time[s]
Copy	1048.7	0.177625	0.152575	0.217059

Table 4.7: 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. Command for running STREAM benchmark at low priority:

```

1 user$ sudo nice -n 19 ./stream >> ${OUTPUT_FILE}

```

4.2.2 Peak Memory Utilization

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

```
1 user$ sudo nice -n 19 ./mem_util.sh &
```

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 will execute the same benchmark, generating maximum communication bandwidth, L2 cache contention and memory contention. To ensure that all the cores are executing the same benchmark, `wait_for_all_threads()` 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 benchmark execution 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 benchmark 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

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

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 benchmark 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) benchmark with spike and the resulting execution time after removing the spike are shown in Figure 4.4.

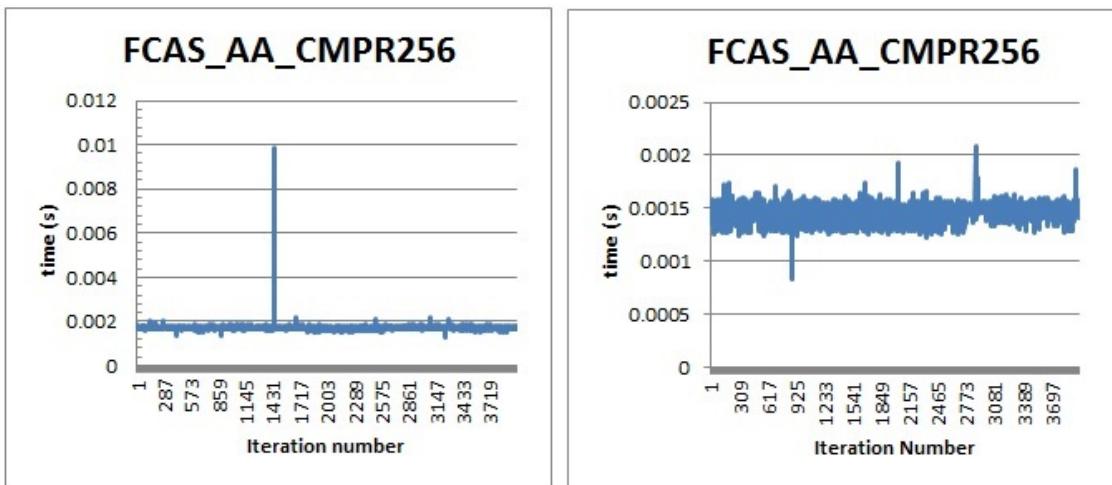


Figure 4.4: Before and After Spike Removal

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 benchmarks, but the magnitude has reduced from 5x to 2x. As expected, some benchmarks 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 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 core1, thread2 on core2, thread3 on core3 and thread4 on core4. The Radar application is given higher priority than other user space tasks to give precedence. The following command is issued on Linaro Linux.

```
1 user$ sudo nice -n -20 ./AA_space_partition
```

4.2.4 CPU Load

CPU Load 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 load says the percentage of utilization as well as the remaining buffer time for future growth. 50% spare time is a healthy value for CPU load. 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.

4.2.5 A/A Mode Sequence of Execution

By matching the A/A Mode Processing Chain explained in Chapter 2.3 and the list of benchmarks in Chapter 2.4, the following benchmarks are chosen to mimic the A/A mode Radar processing chain.

Processing Chain			Benchmark	Description
Time Domain Processing	Beamforming	Beamforming	CMYACC	Sum, Azimuth and Elevation channel data are complex multiplied and the results are summed up
		Weighting	RMY	4 channel data are multiplied with weighting vector.
	Pulse compression		CONV	Pulse compression in frequency domain.
Frequency Domain Processing	Corner turning		COT	4 channel input data are corner turned.
	Frequency Domain Transformation	Windowing	RMY	4 channel data are multiplied with windowing vector.
		FFT	FFT	Transform time domain data to frequency domain.
	Amplitude calculation		MAG	Magnitude calculation of the Sum channel and Guard channel data.

Processing Chain		Benchmark	Description
Detection Processing	Area average calculation	AVG	Area average calculation of the Sum channel and Guard channel data.
	Thresholding & Detection	CMPR	Compare magnitude and Area average to produce pre-alarm matrix.
	Detection	DET	Detects if the alarms entered through sidelobes by comparing Sum channel alarms against Guard channel alarms.

Table 4.9: A/A Mode - Benchmark Selection

The benchmarks shall be executed as shown in Figure 4.5 to mimic the Radar processing chain. Functions of the Correlation processing are not included here. The burst, leading to the large data set ($N_{pri}=205$, $N_{rg}=64$) is processed as per the sequence diagram and the results are used for mode mapping analysis. In the following figure, 3x CMYACC means, computing CMYACC for three channels.

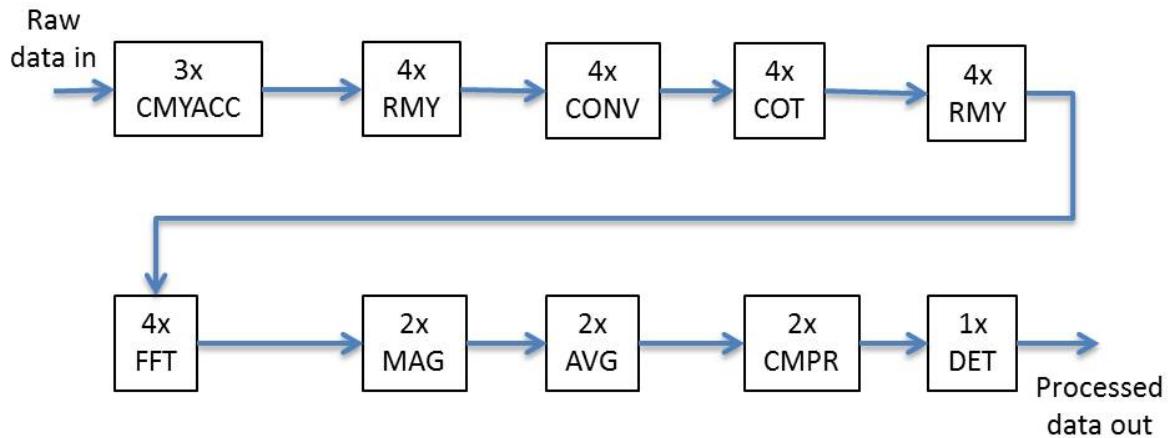


Figure 4.5: A/A Mode, Sequence of Function Execution

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 the Radar benchmarks. In Figure 4.6, measurements of 1-thread, CMYACC means running CMYACC benchmark on core1 while other cores are idle. 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 benchmarks when running on four cores. The values used for the mode mapping analysis belong to the measurements of four core implementation.

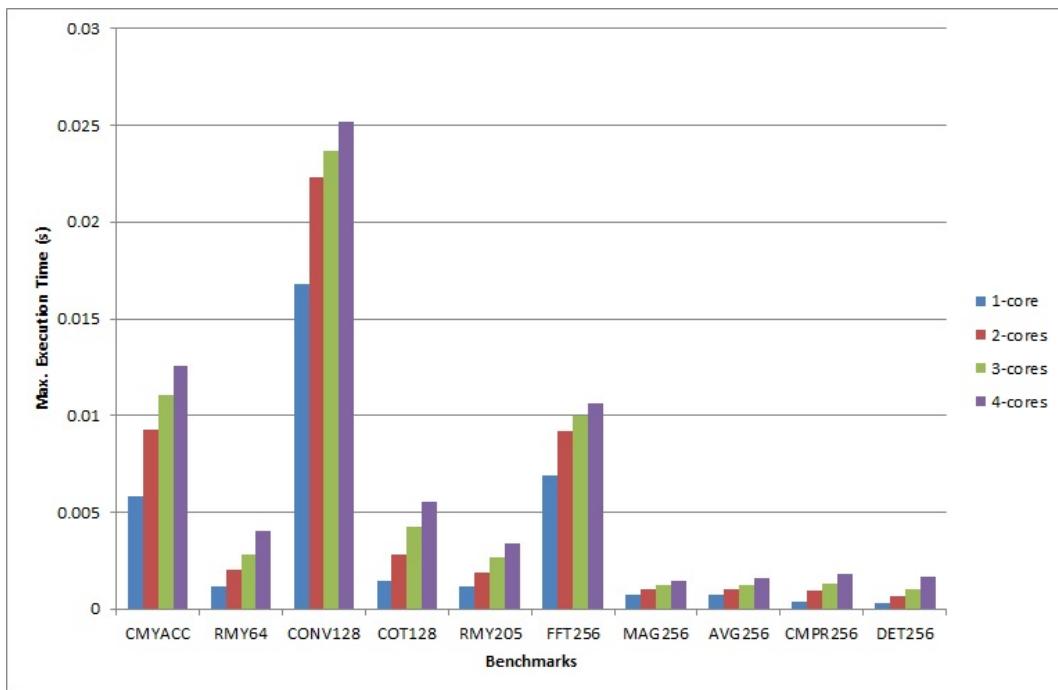


Figure 4.6: Performance of Single Core vs Four Cores

4.4 Design Decisions

4.4.1 Parallel Execution

Figure 4.7 illustrates the data distribution of Scheme-3 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 4.7: Serial Execution of A/A Mode Data in Existing Analysis

4.4.2 Balanced Utilization

In all the schemes of the Existing Analysis, core1 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 load is a gesture of a healthy system. Core1 should also take part in processing the Radar application for load balancing as well as latency reduction.

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 mode mapping analysis.

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 4.8, 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.

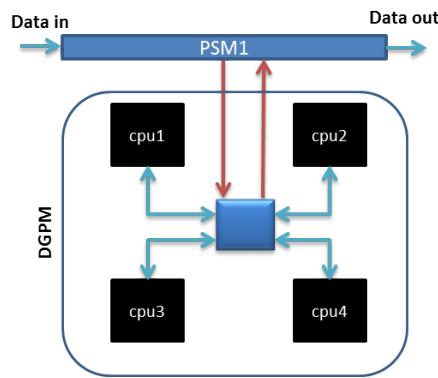


Figure 4.8: Dedicated Communication Channels

Chapter 5

Mode Mapping

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 be 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 Data distribution

Burst processing comprises of Beamforming, Pulse Compression, FFT and CFAR processing. A burst processing CPU gets four bursts and executes one instance of burst processing per core. As shown in Figure 5.1, CPU1...6 are allocated to process four bursts each and CPU7 executes four instances of Correlation Processing in four cores, performing one instance per core.

No.of cores for burst processing: $6 \times 4 = 24$

No.of cores for correlation processing: $1 \times 4 = 4$

Data distribution-burst processing: One burst per core

Data distribution-correlation processing: One dwell per core

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.

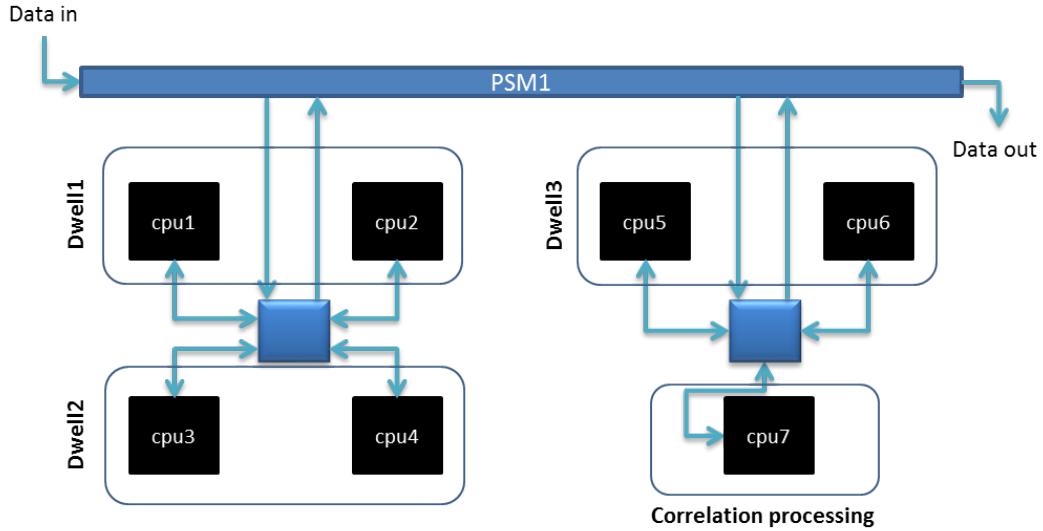


Figure 5.1: Scheme-3, Mode Mapping

3. In a burst processing CPU, a core completing the processing steps last will transfer all the cores(core1...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. Data distribution scheme is shown in Figure 5.2.

5.1.2 Hypothesis

In case of existing analysis, as illustrated in Figure 5.2, beamforming of the first burst will start only after receiving a complete dwell. Though the first burst is ready for Beamforming, 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, Beamforming, Pulse compression, FFT and CFAR processing. Nothing has changed in terms of Correlation Processing, hence it will not contribute to the latency reduction.

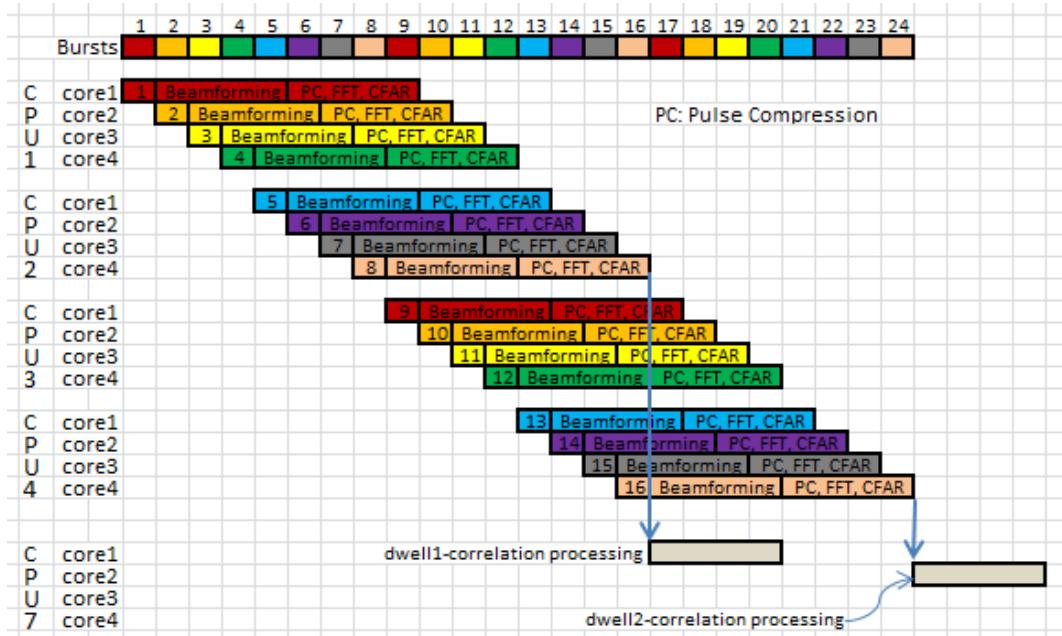


Figure 5.2: Scheme-3, Data Distribution

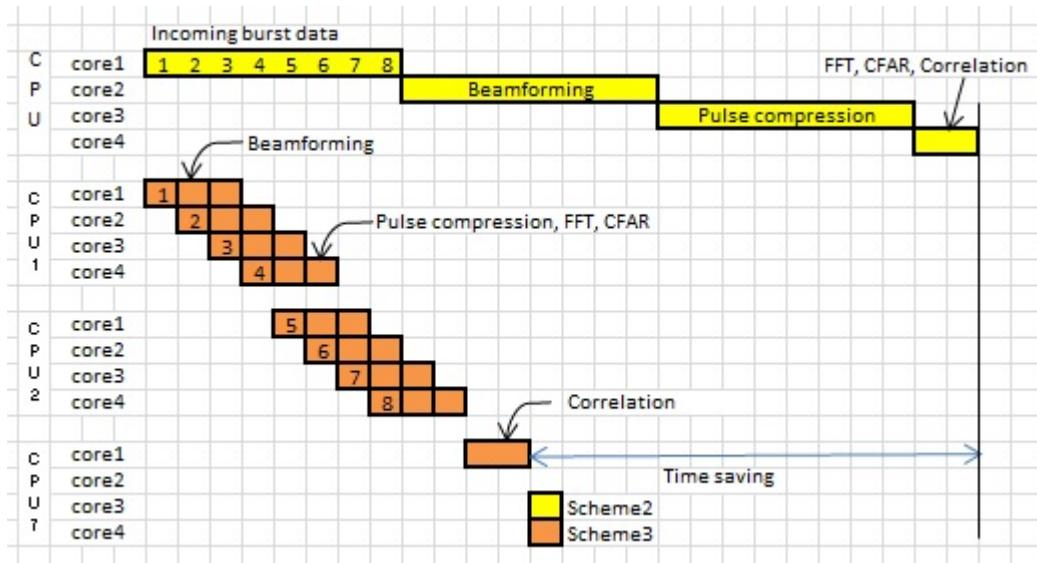


Figure 5.3: Comparison of Data Distribution Schemes

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-4 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. Total time required to process one burst until Thresholding and Detection stage is same as Scheme-1, the differences here are

- Four bursts are distributed to a CPU.
- OS overhead factor is removed.
- Data copy times among the cores are ignored, as the Time Domain Processing, Frequency Domain Processing and Detection Processing of a burst are performed in individual cores.
- The *Sum of required time* values in Figure 5.6 are different from Scheme-1 because the execution cycle of the benchmarks are evaluated and updated (see Chapter 4.1.3).

Processing latency calculations are same as explained in Chapter 3.1.1. Details of the look direction2,3,4 are not shown for simplicity.

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

Figure 5.4: Scheme-3, Input Output, Beamforming and Pulse Compression

The stated values in Figure 5.5 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* in the Figure 5.6. Example calculations for the look direction-1 are explained here with reference to the Figure 5.7. A core waits for the pre-defined burst data to be received, i.e. a core processing PRF5 should wait till the PRF1...5 are distributed by the iCON. Burst receive time are derived from the Radar characteristics (see Chapter 2.5). PRF7 requires lesser time to process a burst than PRF6, because $N_{rg}=59$ for PRF7, requires 64-point(next power of 2) convolution, on the other hand $N_{rg}=68$ for PRF6, requires 128-point convolution.

FFT, CFAR									
SDRAM to L2 cache time [ms]	0.57	0.61	0.60	0.62	0.63	0.62	0.60	0.59	1
COT, FFT, Power Calculation	1.13	1.19	1.16	1.19	1.23	1.22	1.17	1.16	5
Range Doppler AVG with threshold calcul.	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
Burst data after Alarmlist Generation	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
L2 cache to SDRAM time [ms]	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
data copy 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
Sum.all req.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
	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

Figure 5.5: Scheme-3, FFT and CFAR Processing

	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

Figure 5.6: Scheme-3, Processing Time

$$\text{Elapsed time} = \text{MAX}((\text{Previous burst receive time} + \text{Execution time}), \text{Elapsed time})$$

$$\text{PRF1} = \text{MAX}((0 + 35.72), 0) = 35.72 \text{ ms}$$

$$\begin{aligned}
PRF2 &= \text{MAX}((PRF1 \text{ receive time} + \text{Execution time}), \text{Elapsed time}) \\
&= \text{MAX}(3.28 + 45.78), 35.72) = 49.07 \text{ ms} \\
PRF3 &= \text{MAX}((PRF1...2 \text{ receive time} + \text{Execution time}), \text{Elapsed time}) \\
&= \text{MAX}(3.28 + 3.50 + 44.7), 49.07) = 51.47 \text{ ms} \\
PRF4 &= \text{MAX}((3.28 + 3.50 + 3.44 + 45.69), 51.47) = 55.90 \text{ ms} \\
PRF5 &= \text{MAX}((3.28 + 3.50 + 3.44 + 3.53 + 46.73), 55.90) = 60.48 \text{ ms} \\
PRF6 &= \text{MAX}((3.28 + 3.50 + 3.44 + 3.53 + 3.63 + 46.72), 60.48) = 64.09 \text{ ms} \\
PRF7 &= \text{MAX}((3.28 + 3.50 + 3.44 + 3.53 + 3.63 + 3.57 + 34.23), 64.09) = 64.09 \text{ ms} \\
PRF8 &= \text{MAX}((3.28 + 3.50 + 3.44 + 3.53 + 3.63 + 3.57 + 3.46 + 44.57), 64.09) \\
&= 68.98 \text{ ms}
\end{aligned} \tag{5.1}$$

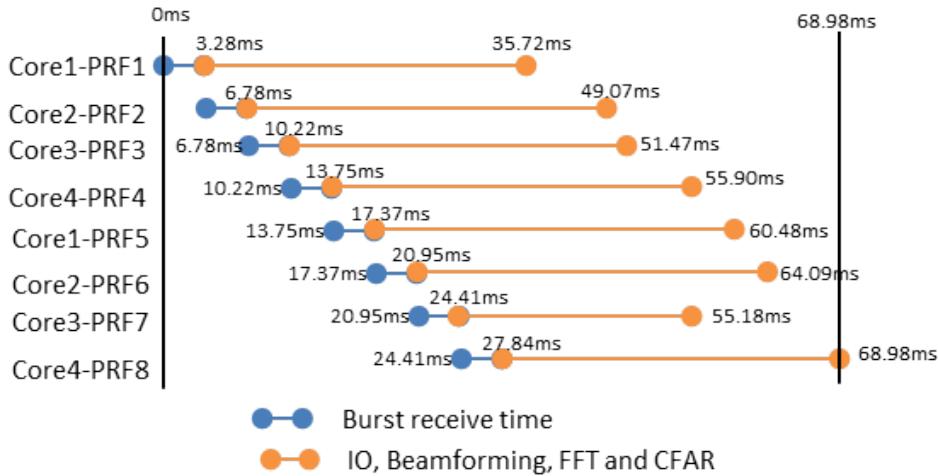


Figure 5.7: Scheme-3, Elapsed Time Calculation

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.

$$\begin{aligned}
\text{Latency} &= \text{Burst processing} + \text{Data transfer to CPU7} + \text{Correlation processing} \\
&\quad + \text{Transfer results} \\
&= 68.98 + 0.02 + 43.34 + 0.2 = 112.5 \text{ ms}
\end{aligned} \tag{5.2}$$

The processing latency between 112.52ms and 174.58ms are achieved depending on the look direction. Table 5.1 lists the processing latency of every look direction.

Look direction	Dwell time[ms]	Latency[ms]	#Dwell latency
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 5.1: Scheme-3, Processing Latency

5.1.4 CPU Load

CPU load 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. From the Figure 5.8, maximum core utilization can reach up to 66%. Calculations for the look direction-1, PRF1 are explained here.

burst	CPU1				CPU2				Look Direction
	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%

Figure 5.8: Scheme-3, CPU Utilization

$$\text{Required time} = 35.72 \text{ ms}$$

$$\text{Available time} = 3 * \text{dwell time} = 3 * 27.84 = 82.2 \text{ ms}$$

$$\text{Utilization} = \frac{\text{Required time}}{\text{Available time}} = \frac{35.72}{82.2} = 43.4\% \quad (5.3)$$

Utilization of Correlation Processing CPU

Updated correlation processing time values are listed in Figure 5.9. The calculations are the same as used in Scheme-1, the difference is that the operating system overhead factor is removed from the analysis.

	CPU1 PRF1	CPU2 PRF2	CPU3 PRF3	CPU4 PRF4	CPU5 PRF5	CPU6 PRF6	CPU7 PRF7	CPU8 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	0.02	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	0.02	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]	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.02	
	summation correlation processing [ms]								43.34	

Figure 5.9: Scheme-3, Execution Time of Correlation Processing

Every core of the correlation processing CPU is in idle state until it receives results of the 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. From the Table 5.2, peak utilization of the CPU7 is 39%.

Core1 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 Figure 5.6, look direction-1 needs 68.98ms to do burst processing. Until this time, core1 of the CPU7 is in idle state. Correlation processing takes place for the next 43.34ms in core1 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). Utilization calculation for the look direction-1 is depicted in Figure 5.10, and the core1's utilization is derived as follows.

$$\text{Utilization at dwell1} = \frac{\text{processing time}}{\text{processing time} + \text{Idle time}} = \frac{43.34}{43.34 + 68.98} = 38\%$$

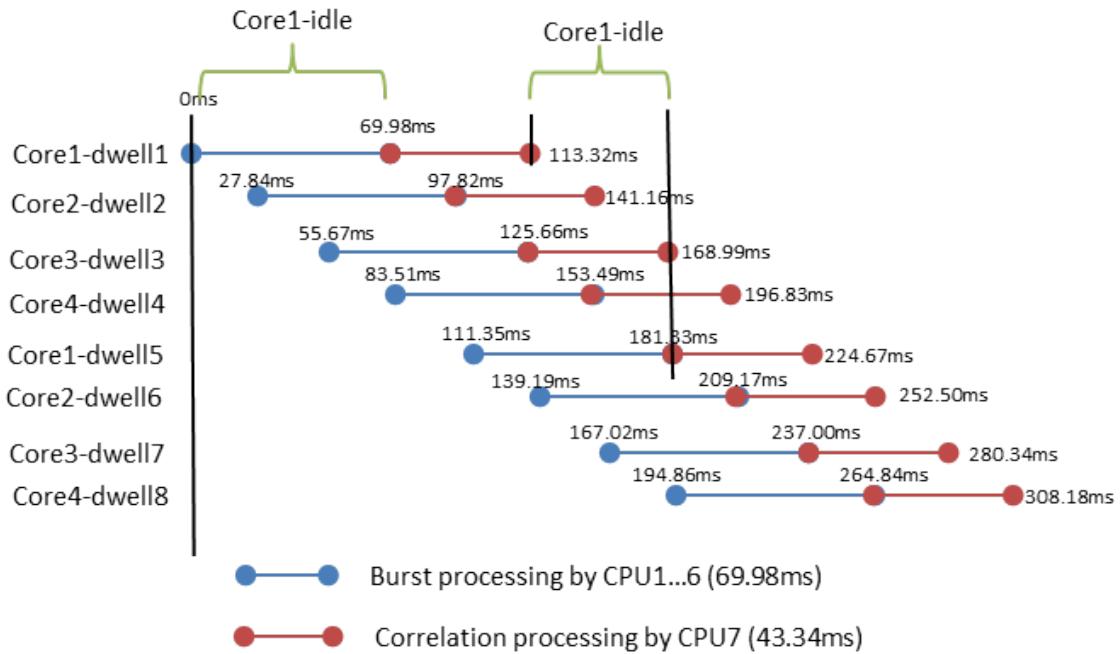


Figure 5.10: Look Direction-1, Utilization of the Correlation Processing CPU

$$\begin{aligned}
 \text{Next dwell time} &= \text{Dwell5 input at burst processing} + \text{Burst processing time} \\
 &= 4 * 27.84 + 69.98 = 181.3 \text{ ms} \\
 \text{Idle time} &= \text{Next dwell in} - \text{Last dwell out} \\
 &= 181.33 - (69.98 + 43.34) = 68.01 \text{ ms} \\
 \text{Utilization at dwell5} &= \frac{43.34}{43.34 + 68.01} = 39\%
 \end{aligned} \tag{5.4}$$

Look direction	Max CPU Utilization
1	39%
2	35%
3	30%
4	27%
5	25%

Table 5.2: Utilization of Correlation Processing CPU

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.

5.1.5 Memory Transfer Bandwidth

The 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 [MB/s]	Avg time[s]	Min time[s]	Max time[s]
Copy	635.0	0.264075	0.251953	0.272976

Table 5.3: Recorded Memory Transfer Bandwidth of the STREAM Benchmark Running as Background Task

$$\begin{aligned}
 \text{Peak BW of the Radar application} &= \text{Idle BW} - \text{Lowest recorded BW} \\
 &= 1048 - 635 = 413 \text{ MiB/s} \\
 &= \frac{413}{1048} = 39.4\%
 \end{aligned} \tag{5.5}$$

5.1.6 Memory Utilization

From the `mem_util.sh` script, it is measured that the Radar application is consuming only 0.9% of the available 879MiB memory. Memory utilization footprint of the Radar application is shown in Figure 5.11.

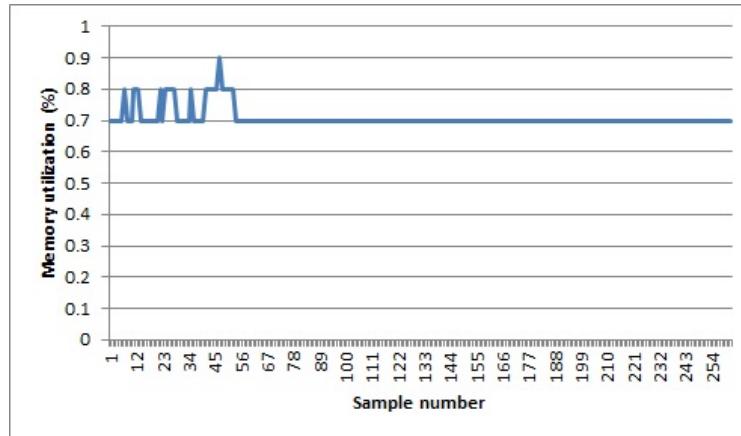


Figure 5.11: Scheme-3, Memory Utilization Footprint

5.1.7 Summary

Scheme-3 has 4x dwell time latency, 66% CPU utilization, 41% 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
Dwell latency	14.96	2	4.04
CPU load	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

- Taking a closer look into the processing chain in Chapter 2.3, 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. Figure 5.12, shows the data dependency diagram for A/A mode processing.

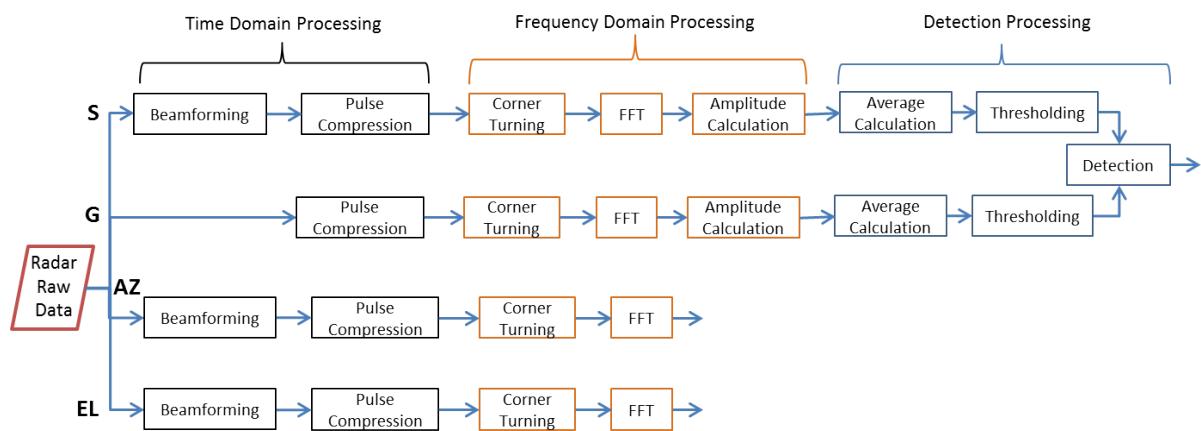


Figure 5.12: A/A Mode - Data Dependency of the Radar Application

Note: The data dependency diagram doesn't consider constant table values such as Sum channel beamforming vector, Azimuth channel beamforming 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 Data Distribution

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 routes 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 5.13 and the data distribution is illustrated in Figure 5.14.

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

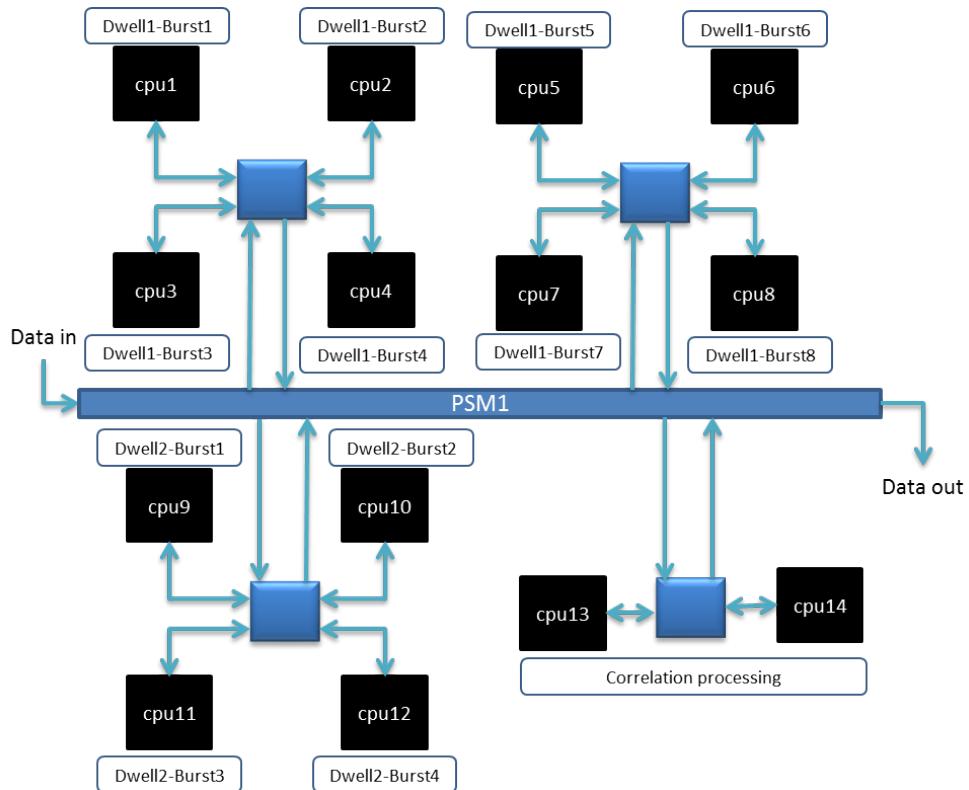


Figure 5.13: Scheme-4, Mode Mapping

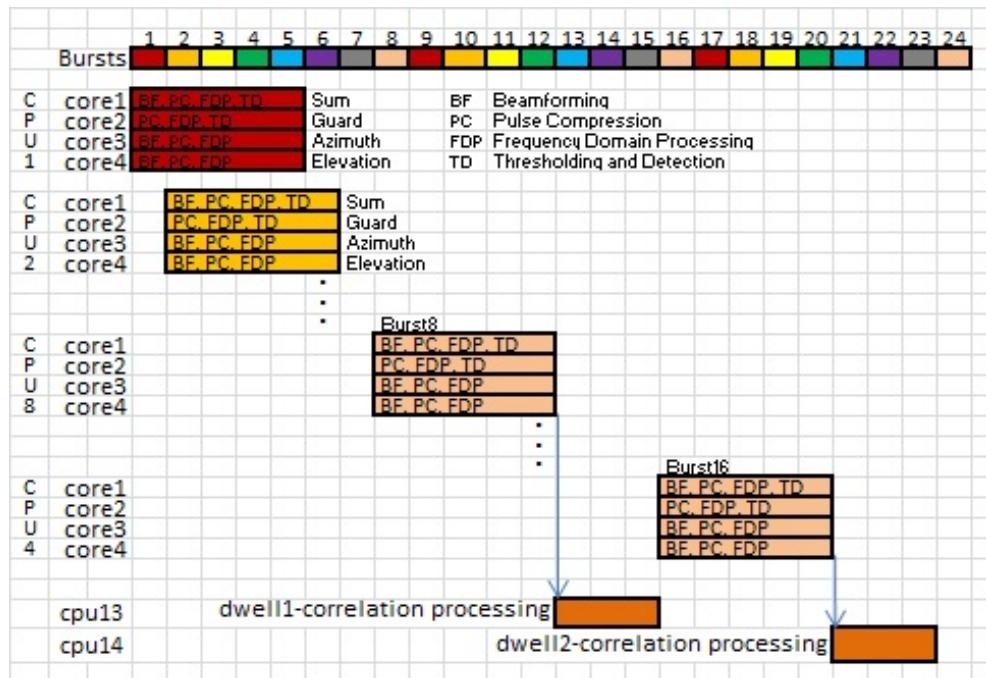


Figure 5.14: Scheme-4 Data Distribution

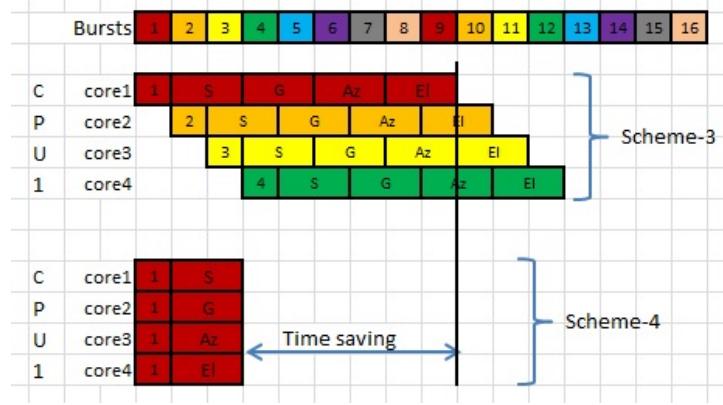


Figure 5.15: Scheme-3 vs Scheme-4

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 5.15.

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 time. Best result cannot be achieved without optimizing the correlation processing. According to the derivations of measurement time, M/N Binary Integration of Velocity consumes 39.16ms of total 43.34ms. It is a good candidate for optimization. M/N Binary Integration of Velocity is implemented in C program according to the pseudo-code shown in Algorithm 1 and the execution time is measured as 31.72ms when four instances are running on four cores. The loop iterations(Line 3) of a single instance of M/N velocity integration are independent of the previous loop processing and next loop processing. They are broken into four loops to run on four cores of a CPU in parallel. 4x speedup 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. Data distribution scheme and time saving are shown in Figure 5.17, data dependency is shown in Figure 5.18. Data structures of the Range Correlation Information(RCI) and Correlation Matrix Velocity(CMV) are depicted in Figure 5.16. These two data structures are the results of previous correlation processing steps namely Monopulse measurement, Range unfolding, M/N range integration and Velocity unfolding.

At first, core1 receives the dwell data for correlation processing, it does Monopulse measurement, Range unfolding, M/N range integration and Velocity unfolding serially. After-

Algorithm 1 Pseudocode - Binary Integration of Velocity[8].

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.Nrc
3: for n:=1 to n_rc do
4:   ai_ref = RCI.RCI_PREV[n].AI;
5:   n_burst = RCI.RCI_PREV[n].Nbc;
6:   best_count = Mv-1;
7:   best_v = 0;
8:   count = 0;
9:   for ref_unf:=1 to CMV.CMV_PREV[b_ref].AI[ai_ref].Nunf 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].Nunf;
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 <= Wsize) && (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].Qv = count;
31: end for

```

```

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 5.16: Structure of RCI and CMV

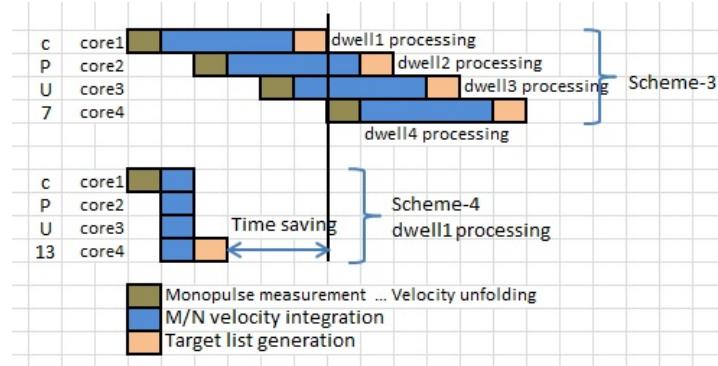


Figure 5.17: Comparison of Correlation Processing Schemes

wards four threads are spawned to perform M/N velocity integration in parallel; each tread is allocated to a core. The thread completing M/N range integration last, will perform Target list generation followed by sending out the results to tracking/display processor via PSM1. 0.02ms overhead 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 5.9)

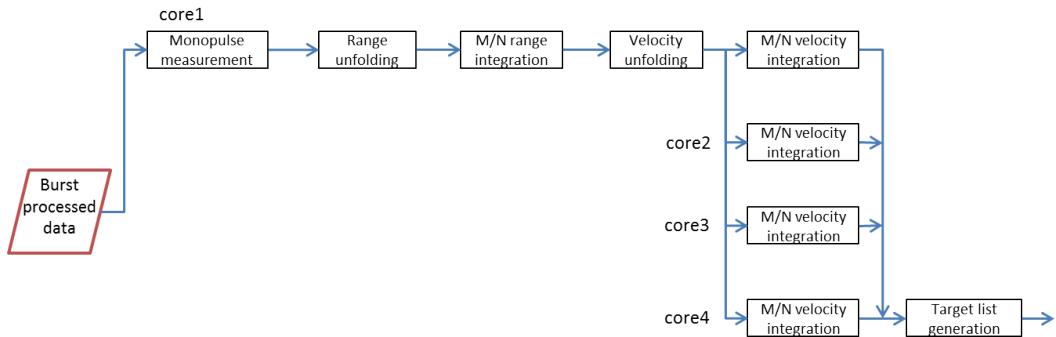


Figure 5.18: Correlation Processing - Data Dependency and Mode Mapping

$$\begin{aligned}
 & \text{Correlation processing time} = \text{Monopulse measurement...Velocity unfolding} \\
 & \quad + M/N \text{ velocity integration} + \text{Target list generation} \\
 & \quad + \text{Overhead} \\
 & = 4.09 + 7.93 + 0.02 + 0.02 = 12.06 \text{ ms}
 \end{aligned} \tag{5.6}$$

5.2.3 CPU Load

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. As shown in Figures 5.19 and 5.20, it is implied that the peak CPU utilization reaches up to 50.8%. Look direction-2 to 4 are not shown in the figures for simplicity. Look direction-5 contributes to the worst case, as the PRF and pulse count configuration yields higher amount of data to be processed.

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 5.12.

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

$$\text{Beamforming} = 1\text{channel} * 64\text{pulses} * 103 \frac{\text{range gates}}{\text{pulse}} * 287 \frac{\text{cycle}}{8\text{element}} \\ * 1.25 \frac{\text{ns}}{\text{cycle}} = 2.36 \text{ ms}$$

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 5.19: Scheme-4, CPU Utilization (1/2)

$$\text{Beamformed data} = 1\text{channel} * 64\text{pulses} * 103 \frac{\text{rangeegates}}{\text{pulse}} * 8 \frac{\text{byte}}{\text{sample}} = 52,736 \text{ byte}$$

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

$$\text{Pulse compression} = \left(\# \text{channel} * \# \text{pulses} (\text{CONV128} + \# \frac{\text{rangeegates}}{\text{pulse}} * \text{RMY50}) \right) * \text{cycle time}$$

$$= (1 * 64(27300 + 103 * 68)) * 1.25 \text{ ns} = 2.74 \text{ ms} \quad (5.7)$$

$$\text{FDP time} = \left(1\text{channel} * (\# \text{pulses} * \# \frac{\text{RG}}{\text{pulse}} (\text{COT50} + \text{RMY50}) + \# \frac{\text{RG}}{\text{pulse}} * \right.$$

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 5.20: Scheme-4, CPU Utilization (2/2)

$$\begin{aligned}
 & FFT64) + 1 * \left(\# \frac{\text{rangeagates}}{\text{pulse}} * 64 * \text{MAG256} \right) * \text{cycle time} \\
 & = (1 * (64 * 103(98 + 68) + 103 * 2550) + 1 * (103 * 64 * 41)) * 1.25ns \\
 & = 2.03 \text{ ms} \\
 \text{Average Calculation} & = 1\text{channel} * 103 \frac{\text{rangeagates}}{\text{pulse}} * 64(\text{FFTsize}) * 51(\text{AVG100}) * 2 \\
 & = 672,384 \text{ cycle} \\
 \text{Thresholding} & = 1\text{channel} * 103 \frac{\text{rangeagates}}{\text{pulse}} * 64(\text{FFTsize}) * 55(\text{CMPR100}) \\
 & = 362,560 \text{ cycle} \\
 \text{Detection} & = 103 \frac{\text{rangeagates}}{\text{pulse}} * 64(\text{FFTsize}) * 97(\text{DET100}) = 639,424 \text{ cycle} \\
 \text{Sum time} & = (\text{Average calculation} + \text{Thresholding} + \text{Detection}) * \text{cycle time} \\
 & = (672,384 + 362,560 + 639,424) * 1.25ns = 2.09 \text{ ms} \tag{5.8}
 \end{aligned}$$

Total time required to process one channel of a burst by a core is calculated as the sum of IO processing, Beamforming, Pulse compression, FFT, CFAR, Thresholding, Detection and data copy time between SDRAM to the core. CPU utilization is the ratio of processing time to the available time. Calculations for the look direction-1, CPU1 is explained here.

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

$$\text{CPU utilization} = \frac{\text{processing time}}{\text{available time}} = \frac{14.31}{41.76} = 34.3\% \quad (5.9)$$

Correlation Processing

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. 0.02ms time delay between burst processing CPUs and correlation processing CPUs is assumed. Timeline diagram is illustrated in Figure 5.22. Utilization calculations for the look direction-1 are listed below.

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 5.21: Scheme-4, Utilization of Correlation Processing CPU

$$\text{Utilization at dwell1} = \frac{\text{processing time}}{\text{processing time} + \text{waiting time}} = \frac{12.06}{12.06 + 42.54} = 22\%$$

$$\begin{aligned} \text{Next dwell time} &= \text{Dwell3 input at burst processing} + \text{Burst processing time} \\ &= 2 * 27.84 + 42.54 = 98.2\text{ ms} \end{aligned}$$

$$\text{Idle time} = \text{Next dwell in} - \text{Last dwell out}$$

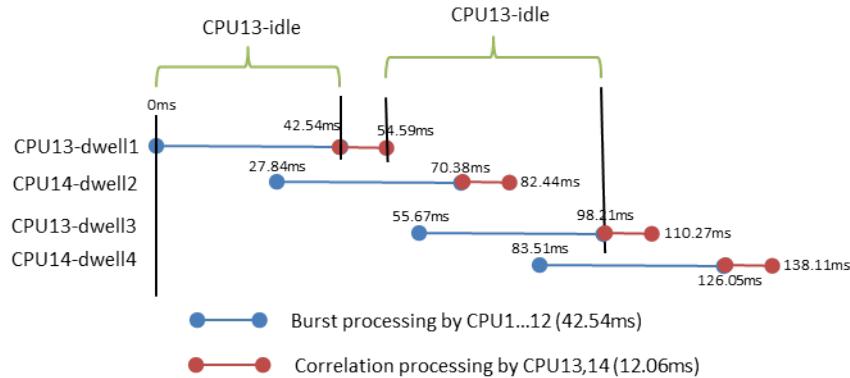


Figure 5.22: Look Direction-1, Utilization of the Correlation Processing CPU

$$\begin{aligned}
 &= 98.2 - (12.06 + 42.54) = 43.6 \text{ ms} \\
 Utilization \text{ at } dwell3 = \frac{12.06}{12.06 + 43.6} &= 22\%
 \end{aligned} \tag{5.10}$$

From the Figure 5.21, peak utilization of the correlation processing CPU reaches upto 22% when using two 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 latency 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]	#Dwell latency
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: Scheme-4, Processing Latency

5.2.5 Memory Utilization

From the `mem_util.sh` script, it is measured that the Radar application is consuming only 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 Radar application is shown in Figure 5.11. Though the memory require-

ment is reduced by a factor of 4, memory utilization of the Radar application is not reduced by the same figure. Since, the Radar application has SQLite library, FFTW library, spawning threads and other processing steps are required to run the application, they consume certain amount of memory regardless of the input data set.

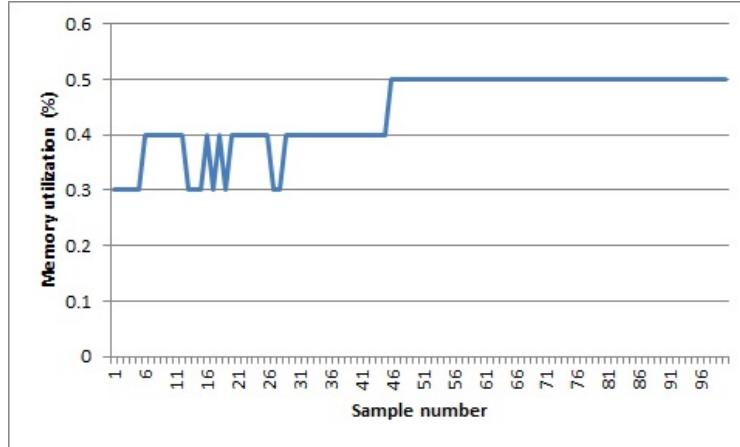


Figure 5.23: Scheme-4, Memory Utilization Footprint

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 [MB/s]	Avg time[s]	Min time[s]	Max time[s]
Copy	729.9	0.253547	0.219210	0.300818

Table 5.6: Recorded Memory Transfer Bandwidth of the STREAM Benchmark Running as Background Task

$$\text{Peak BW of the Radar application} = \text{Idle BW} - \text{Lowest recorded BW}$$

$$\begin{aligned}
 &= 1048 - 729.9 = 318.1 \text{ MiB/s} \\
 &= \frac{318.1}{1048} = 30.3\%
 \end{aligned} \tag{5.11}$$

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 has 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 5.24 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 load. 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.

#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 5.24: Scheme-4, Relationship Between Number of CPUs and Utilization

Parameter	Scheme-1	Acceptable Values	Scheme-4
Dwell latency	14.96	2	1.97
CPU load	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

5.3 Overview

The Existing Analysis has been observed to spot the latency contributors, bottlenecks and data dependencies. Upsides and downsides are evaluated to bring up a best mode mapping scheme. Examined information have supported in design decisions of the mode mapping. 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

Beamforming, 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 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 processing latency is achieved along with 50.8% CPU utilization, provided 13 CPUs are performing computation. From the Table 5.7 data, speedup of the Scheme-4 compared to the best scheme(Scheme-1) in the Existing Analysis is drawn as follows.

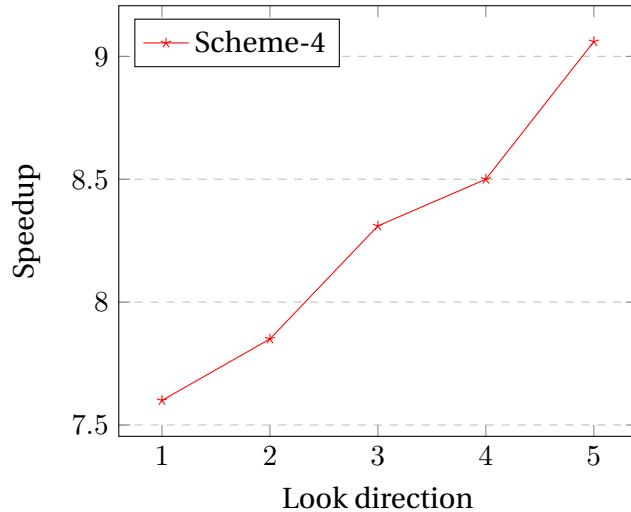


Figure 5.25: Speedup, Scheme-4 vs Scheme-1

The best results of the Existing Analysis are 75.5% peak CPU utilization and 15x dwell 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.

5.4 Verification of Results

To verify the correctness of the Scheme-3 and Scheme-4 implementations, the results of the 4-core implementations are cross checked against the single core implementation. Both the results are matching, assuring that the application is producing consistent results in single-threaded and multi-threaded environment.

Chapter 6

Future Scope and Conclusion

6.1 Some Implementation Details

The whole application is written in C and uses `pthread`, `SQLite`, `FFTW` external libraries for computation as well as storing the execution measurement results. Some of the important challenges faced during the implementation phase are listed here.

6.1.1 Synchronization

`wait_for_all_threads()` function (Appendix C.2) is providing a synchronization point. Sync point is provided at the beginning of every benchmark (3xCMYACC, 4xRMY, etc) execution to make sure that threads are running simultaneously. A thread reaching this function first locks the mutex variable `mutex` then waits on condition variable `cv_count`. The last(4^{th}) thread reaching this function will release the condition variable by broadcasting followed by releasing the mutex `mutex`. Now one thread(4^{th}) has passed the synchronization point and remaining three threads are waiting for the `mutex` access, called thread wakeup. Those waiting three threads will be released one after another serially. In the program execution, there may be a scenario where the first thread crossing the sync point-1 will reach sync point-2 `wait_for_all_threads()` and waits for the `mutex` access, before all the three threads are released from sync point-1.

Figure 6.1, illustrates the first three threads are waiting at sync1, while the 4^{th} thread crossed sync-1 and waiting for `mutex` access corresponds to sync-2. On such occasions, the order of thread wakeup relies on the scheduling policy of the thread [18]. This goes haywire in a multithreaded environment where the order of thread execution is non deterministic. This issue occurred at sporadic intervals making it hard to debug. Validating the number of awakened threads solved the problem. It acts as a guard for mutex lock mechanism.

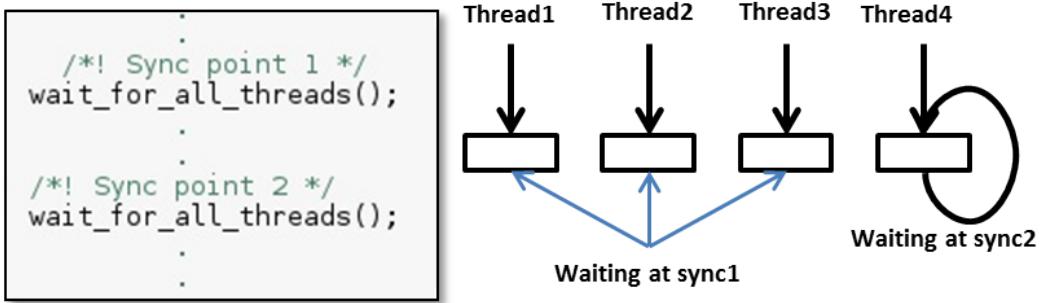


Figure 6.1: Race Condition in `wait_for_all_threads()` Function

6.1.2 Thread Safety

The benchmarks should be thread-safe to be executed in parallel threads. FFTW has `fftw_plan` stage to prepare input parameters of `fftw_execute`. According to the FFTW documentation, `fftw_execute` is thread-safe, but not `fftw_plan` [19]. However, the same plan can be used for multiple threads in parallel. Instead of all threads executing `fftw_plan`, only one thread is allowed to execute `fftw_plan`, then all threads are allowed to do `fftw_execute` in parallel.

6.1.3 Performance of the SQLite Library

SQLite library is used to store execution time measurements of the Radar benchmarks into a database. The Radar algorithm has ten performance critical functions as benchmarks. Every thread will store the execution measurement result at the end of every benchmark processing. The SQLite library performs well in a single threaded application with relatively less number of insertions. The scene is extremely different in a multithreaded environment. Data insertions have to be mutually exclusive as only one database is used through the application. Four threads, running on four cores inserting 40,000 records consumed 15 hours on ARM Cortex A9 platform. Retrieval is also equally slow. On the other hand, .csv implementation of the same did it in couple of minutes. Processing .csv data is also simple and convenient as it can be interpreted by Microsoft Excel.

6.2 Future Scope

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

6.2.1 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-4 executes FFTW library for every channel (Sum, Guard, Azimuth and Elevation) in sequential order, whilst Scheme-5 performs channel computations in parallel. In any case, single instance of FFT library is executed in serial. Breaking the FFT and Convolution benchmarks 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 speedup factor of broken parallel modules.

Benchmarks like Complex Multiply Accumulate (CMYACC), Range Multiply (RMY), Magnitude (MAG), Area Average Calculation (AVG) and Comparison (CMPR) have lot amount of parallelism. Loop iterations in the above benchmarks can be split to run in parallel. Effective speedup will be equal to number of parallel executable modules. Although breaking the application for parallel execution increases speedup, it needs more computing entities to carry out the task, new data distribution schemes and so mode mapping. This is a starting point of the optimization and requires further feasibility study.

6.2.2 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.

6.2.3 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* [20], to improve data locality. This will reduce memory transfer bandwidth as well as execution time.

In Scheme-5, the burst data size after beamforming 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.

6.2.4 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.

6.3 Conclusion

This thesis has presented the Airborne Radar processing chain, IMA processor architecture for safety critical systems and existing mode mapping analysis. Inference from the existing 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 memory utilization and bandwidth utilization have been made.

Couple of optimization schemes have been implemented for Air to Air Mode processing. Based on the new mode mapping analysis, the optimized scheme(Scheme-4) guarantees 7.5x speedup than the existing analysis with lesser number of resources. The achieved 2x dwell 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 processer 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 latency with 87% CPU utilization!

Bibliography

- [1] Medical Applications of Ultra-WideBand (UWB), [Online]. Available: <http://www.cse.wustl.edu/~jain/cse574-08/ftp/uwb/> (visited on Sep. 21, 2015).
- [2] Collision avoidance system, [Online]. Available: https://en.wikipedia.org/wiki/Collision_avoidance_system (visited on Sep. 21, 2015).
- [3] Ludloff and K. Albrecht, *Praxiswissen Radar und Radarsignalverarbeitung*. Springer, 1998.
- [4] Radar Tutorial, [Online]. Available: <http://www.radartutorial.eu/index.en.html> (visited on Sep. 21, 2015).
- [5] Radar Basics, [Online]. Available: <http://www.radartutorial.eu/07.waves/Waves%20and%20Frequency%20Ranges.en.html> (visited on Oct. 07, 2015).
- [6] Understanding Azimuth and Elevation, [Online]. Available: <http://www.photopills.com/articles/understanding-azimuth-and-elevation> (visited on Sep. 22, 2015).
- [7] E. Cary and R. Spitzer, *The Avionics Handbook*. CRC Press LLC, 2001.
- [8] H. P. Keller, “Future Combat Air System - Analysis of IMA Modules,” Airbus Defence and Space GmbH, Tech. Rep., Dec. 2014.
- [9] Freescale-Semiconductor, “Datasheet - i.MX 6Dual/6Quad Applications Processors for Industrial Products,” ARM, Tech. Rep., Mar. 2014.
- [10] FFTW Application Library, [Online]. Available: <http://www.fftw.org/> (visited on Sep. 23, 2015).
- [11] T. Nohara, P. Weber, A. Premji *et al.*, “Affordable Avian Radar Surveillance Systems For Natural Resource Management And BASH Applications,” in *IEEE*, 2005.
- [12] Y. Zeng, J. Xu, and D. Peng, “Radar Velocity-measuring System Design and Computation Algorithm Based on ARM Processor,” in *8th World Congress on Intelligent Control and Automation*, 2010.

- [13] Y. Salama, D. Fitzgerald, G. Bright, and J. Rooks, "Power Efficient Radar Signal Processor," in *Radar Conference, 2005 IEEE International*, 2005.
- [14] 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).
- [15] LMbench - Tools for Performance Analysis, [Online]. Available: <http://lmbench.sourceforge.net/> (visited on Oct. 27, 2015).
- [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] Pthread_cond_wait(3) - Linux man page, [Online]. Available: http://linux.die.net/man/3/pthread_cond_wait (visited on Oct. 08, 2015).
- [19] Thread safety FFTW 3.3.4, [Online]. Available: http://www.fftw.org/fftw3_doc/Thread-safety.html#Thread-safety (visited on Oct. 08, 2015).
- [20] 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.
- [21] Boundary Devices, [Online]. Available: <http://boundarydevices.com/wp-content/uploads/2014/06/Nitrogen6x1.jpg> (visited on Sep. 23, 2015).

Appendix A

Block Diagrams

A.1 Nitrogen6X Development Kit

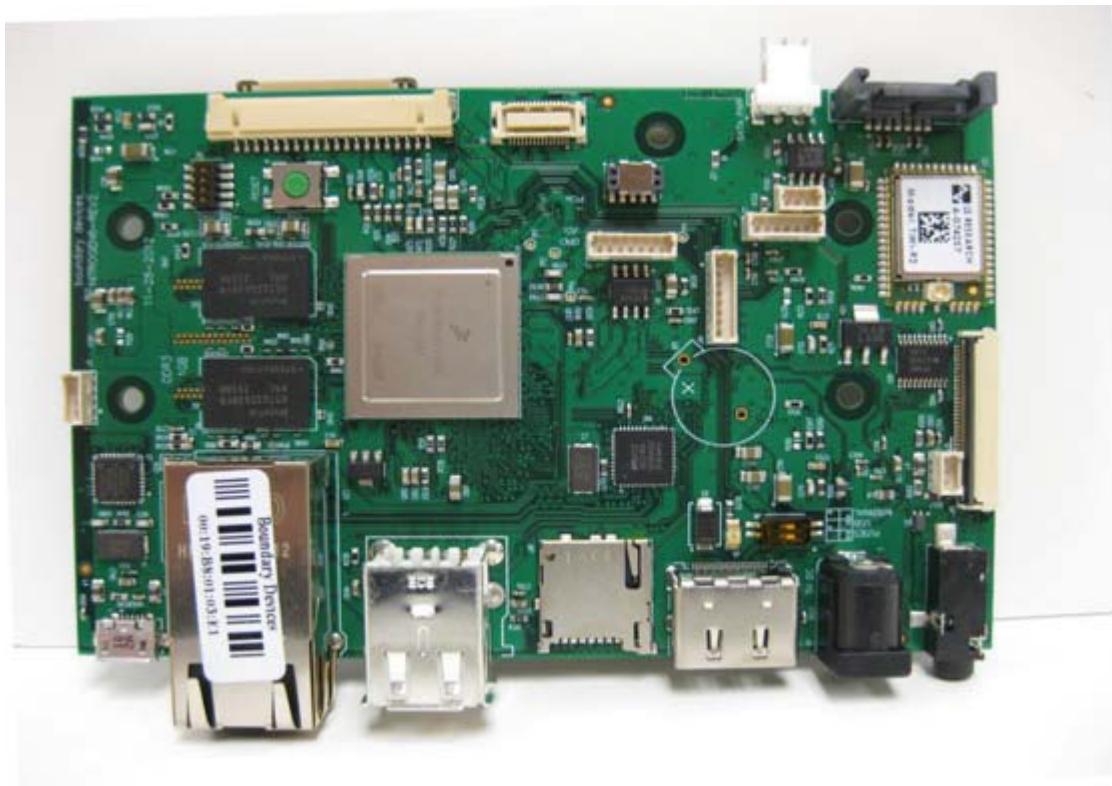


Figure A.1: Nitrogen6X Development Kit [21]

A.2 iMX6Quad CPU

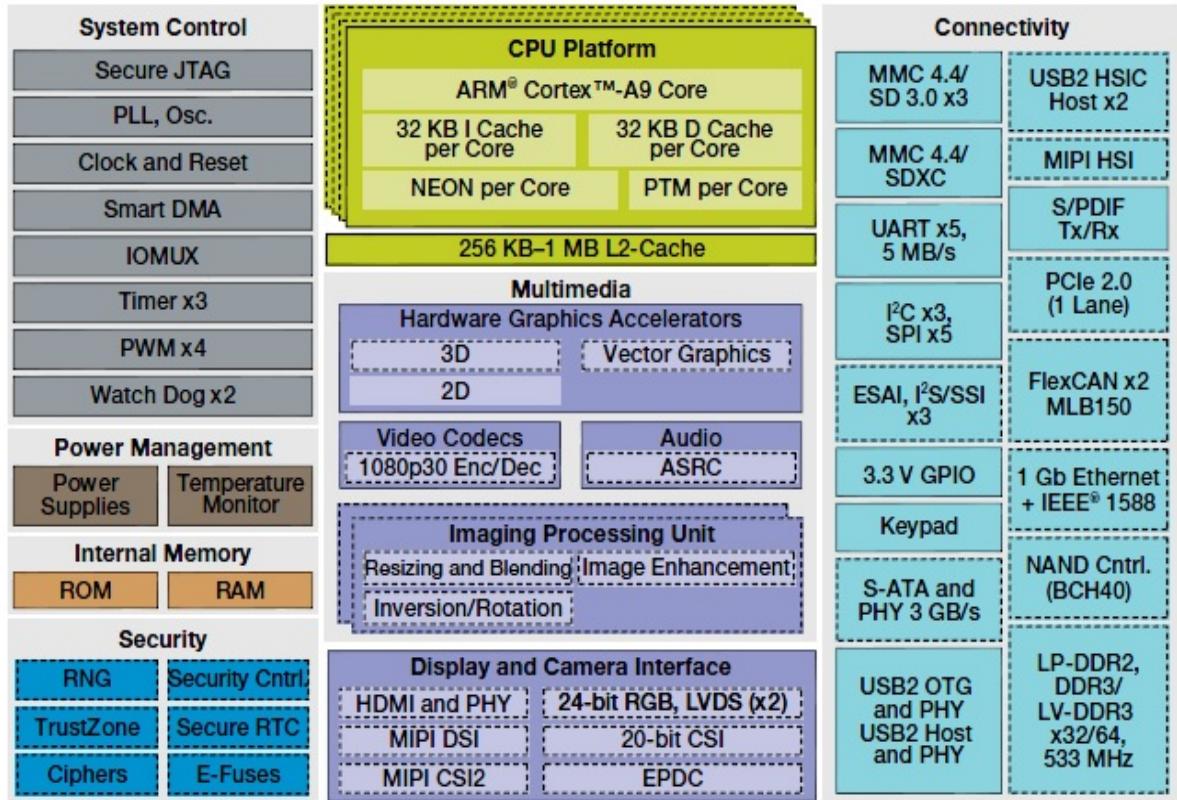


Figure A.2: Internals of iMX6Quad CPU [9]

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 arround 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.
- 64DET100: Calculate Main/Guard comparison with a 64 x 100-element float matrix of the sum-channel and a 64 x 100-element float matrix of the guard-channel. Resulting in a 64 x 100-element boolean matrix.

- 1CMY1k: Multiply a 1 x 1k-element complex matrix with a 1 x 1k-element complex matrix, resulting in a 1 x 1k-element complex matrix.
- 10MAGSQ1k: Perform the magnitude square calculation of 10 x 1k complex matrix, resulting in a 10 x 1k float matrix.
- 10ACCOFFSET1k: Perform the accumulation of the data from 4 separate 10 x 1k-element float matrices. The result shall be a 10 x 640 element float matrix. For the access to the vectors an offset of $k * 128$ elements shall be used (i.e. Novl = 128), where $k = 0, 1, 2, 3$.
- 100LOG256: Perform a $10^{\star}\log_{10}$ calculation of the data from 100 x 256-element float matrix, resulting in a 100 x 256-element float matrix.
- 100LIM256: Perform the upper-limitation of the data of a 100 x 256-element float matrix to a float constant Dmax. All elements in the matrix which are greater than Dmax shall be set to Dmax. Resulting in a 100 x 256-element float matrix.
- 100SCAL256: Perform the scaling algorithm with a 100 x 256-element float matrix. Resulting in a 100 x 256-element byte matrix. Assume Ddisp = 256.

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_setspecific_np(current_thread, sizeof(cpu_set_t), &cpuset)
13        != 0)
14        return CPU_AFFINITY_ERROR;
15
16    return CPU_AFFINITY_OK;
}
```

Listing C.1: Set core affinity of calling thread

C.2 wait_for_all_threads()

```
1 void wait_for_all_threads()
2 {
3     while(awakenedThreads%NUM_THREADS != 0)
4         sleep(0.001);
5
6     pthread_mutex_lock(&mutex);
7     readyThreads += 1;
8
9     if(readyThreads == NUM_THREADS) {
10        pthread_cond_broadcast(&cv_count);
11    }
12
13    while(readyThreads != NUM_THREADS)
14        pthread_cond_wait(&cv_count, &mutex);
15
16    ++awakenedThreads;
17    if(awakenedThreads == NUM_THREADS){
18        awakenedThreads = 0;
19        readyThreads = 0;
20    }
21
22    pthread_mutex_unlock(&mutex);
23 }
```

Listing C.2: Wait for other threads to join

C.3 mem_util.sh

```

1 #!/bin/bash
2 ITER=1
3 PEAK_MEM=0
4 MEM_SIZE='free -m | sed -n 2p | awk '{print $2}''
5 DATE_BEGIN=$(date +%Y-%m-%d" "%H:%M:%S)
6 OUTPUT_FILE="logs/peak_mem_util.csv"
7
8 echo "DATE BEGIN,\${DATE_BEGIN}" > \${OUTPUT_FILE}
9
10 # Check for user input
11 if [ "$#" = "0" ]
12 then
13 # Read from file
14 PID='cat logs/pid'
15 else
16 PID=$1
17 fi
18 echo "PID = \${PID}"
19
20 NAME='cat /proc/\${PID}/cmdline'
21 echo "Application name, \${NAME}" >> \${OUTPUT_FILE}
22 RES='eval top -b -n \$\{ITER\} | grep \${PID}'
23
24 while [ ! -z "\$RES" ]
25 do
26 #echo "res = \${RES}"
27 array=(\$RES)
28 PEAK_MEM=\${array[9]}
29 echo "\${PEAK_MEM}" >> \${OUTPUT_FILE}
30
31 #echo "Peak mem = \${PEAK_MEM}"
32 RES='eval top -b -n \$\{ITER\} | grep \${PID}'
33 sleep 0.1
34 done

```

Listing C.3: Peak Memory Utilization

