Performance Analysis of Arm Cortex-R Series Processors

Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

In Electronics & Communication Engineering

(VLSI Design)

By

ANISHA HAZRA

19MECV02

Electronics & Communication Engineering Department

Institute of Technology

Nirma University

Ahmedabad - 382 481

May, 2021

Performance Analysis of Arm Cortex-R Series Processors

Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

In Electronics & Communication Engineering

(VLSI Design)

By

ANISHA HAZRA

19MECV02

Institute Guide:

Dr. Amisha P. Naik

Associate Professor (EC)

Institute of Technology,

Nirma University,

Ahmedabad

Industrial Guide:

Rajendra Saini

Staff Engineering/Manager

System Analysis and Benchmarking

Arm Embedded Technologies Pvt Ltd

Bangalore

Electronics & Communication Engineering Department Institute of Technology

Nirma University

Ahmedabad - 382 481

May, 2021

Certificate

This is to certify that the Project Report entitled "Performance Analysis of Arm Cortex-R Series Processors" submitted by Anisha Hazra, Roll No. – 19MECV02, towards the partial fulfillment of the requirements for the award of degree in Master of Technology in the field of VLSI Design in Electronics & Communication Engineering department of Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this major project Part-I to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Institute Guide
Dr. Amisha P. Naik
Associate Professor (EC)
Institute of Technology,
Nirma University,
Ahmedabad

Industrial Guide
Rajendra Saini
Staff Engineer/Manager
System Analysis and Bechmarking
Arm Embedded Technologies Pvt Ltd
Bangalore

Head of Department
Dr. Dhaval Pujara
Professor and Head,
EC Department,
Institute of Technology,
Nirma University,
Ahmedabad

Director
Institute of Technology,
Nirma University.
Ahmedabad

Acknowledgement

In the current scenario of competitive life there is a race of existence in which those are having will to come forward succeed. A project work acts like a bridge between theoretical and practical working. With this willing I took this specific task. The ultimate result of this project needed much direction and help from many individuals and I am extremely privileged to have got them all for my project.

With immense pleasure I express my sincere thanks and regards to my Institute guide Dr. Amisha Naik for her exceptional guidance, extremely useful advice and continuous encouragement at all the phases of the task. I would like to express my gratitude and sincere thanks to Dr. Usha Mehta, P.G. Coordinator of M.Tech VLSI Design for her valuable guidance and constant support during review process.

It gives me an immense pleasure to thank Dr.Dhaval Pujara, Hon'ble Head of Electronics and Communication Engineering Department, Institute of Technology, Nirma University, Ahmedabad and for his kind support and providing basic infrastructure and healthy research environment and a special thank you is expressed wholeheartedly to Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation he has extended throughout course of this work.

Sincere gratitude to my manager Mr. Rajendra Saini. His constructive criticisms at different stages of my work were thought-provoking and has helped me focus on my ideas. I would like to express my gratitude and sincere thanks to my Director Mr. Kaleshwar Vemuri for his valuable guidance and advice during the project work. I also take this opportunity to convey sincere thanks to my colleagues at Arm Bangalore for their smiles and friendship making every work assignment enjoyable and memorable. None of this would have been possible without their love and support.

Last but not the least, nobody has been more important to me in the pursuit of this project than the members of my family. I would like to thank my parents, whose love and support always lie with me in whatever I pursue. They are the ultimate role models.

ANISHA HAZRA
19MECV02

Abstract

This project is based on the Performance Analysis of Cortex-R series processors which are mainly used in Real Time Applications. The Cortex-R family is intended to be utilized in applications where time and safety are critical issues.

The Performance Analysis of such processors is done with the help of certain Industry standard benchmarks like CoreMark, Dhrystone, EEMBC, Automotive, Consumer, Networking, Office, Telecom, FPMark, Stream etc. These tests are written in C and cannot run on cores and hence compilers like Arm Compilers or GNU Compiler Collection are used to convert those tests from c files into binary executable files. The performance can be optimized or enhanced with the help of optimization flag options during the binary build.

Running Benchmarks on our models provides us with a cycle count. This cycle count is a measure of CPU cycles and reflects the time taken by that CPU core to complete certain task. Those cycles are used to calculate the performance scores. We compare those scores with the golden data in our Dashboards. In case when much deviation in performance is observed from the golden data then debugging takes place. Debug can be performed either with Tarmac or with Waveforms. Debug with Tarmac is also done when we are not able to pass some test cases. Waveforms can be used to reconfirm the delays that we have considered. Verdi Synopsys is used for the Waveform generation purpose. Once we have all the performance data, then Dashboard creation takes place for all Cortex-R series processors. A dashboard refresh is done whenever there is a new Milestone release or there is a new Toolchain release. This dashboard is shared with the Marketing group for external purposes.

Performance Analysis can be either Emulation or Simulation based. Emulator used in this project is Veloce/Strato which is an Emulation tool by Mentor Graphics. Simulation based analysis is done on Cycle Models. Improvements to the existing performance numbers is done by updating the platform files.

In this project the Cortex-R8 processor has been added to the Unified R-Class System by tuning its Read and Write Latencies to that of the common latency of the external system. After fixing the latencies, performance and code size calculation is done. To automate the process of speed and code size calculation, a python script has been used which calculated the performance numbers and code sizes and prints them in a CSV format file. This file can be uploaded to Tableau Software to create graphical representation of data and dashboards.

During this project, the Dashboard refresh has been done each time whenever there is a new compiler version release. The current data is always compared with the previous version which keeps a check on the CPU core performance as well as the Compiler performance. Sometimes huge difference in data can be either due to Performance bug or due to Compiler bug.

Abbreviations

CPAK	Cycle Model Performance Analysis Kit
DMIPS	Dhrystone Million Instructions Per Seconds
HPM	High Performance Mobile
SIMD	Single Instruction Multiple Data
MMU	Memory Management Unit
CSD	Computational Storage Device
SSD	Solid State Device
ECC	Error Correcting Code
FPP	Fast Path Port
ITCM	Instruction Tightly Coupled Memory
DTCM	Data Tightly Coupled Memory
I Cache	Instruction Cache
D Cache	Data Cache
MPU	Memory Protection Unit
FPU	Floating Point Unit
ACP	Accelerator Coherency Port
SCU	Snoop Control Unit
AXI	Advanced eXtensible Interfaces
LLPP	Low Latency Peripheral Port
LLRAM	Low Latency RAM Port
EEMBC	Embedded Microprocessor Benchmark Consortium
HDD	Hard Disk Drive
SDD	Solid State Drive
LTO	Link Time Optimization

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Team Profile

System Engineering Group of Arm is responsible for the design and delivery of System IP, Security IP and Imaging IP. It also ensures all Arm IP works well together in a holistic view through the creation of evaluation systems and test chips. It consists of System Integration, Performance Analysis and Benchmarking, Implementation, Tooling, IP Verification etc. Early System Integration is done to prove IP Interoperability and System Architecture. Functional Verification at System Level is done for IP bug hunting and soak testing. IP Performance (CPU) at System Level to characterize and analyze performance.

1.2 Motivation

Cortex-R series processors are meant for quick and deterministic handling of a process. It assures high performance during scenarios having real time constraints within a pace of circumstances. All these benefits are consolidated into a balanced package that gives performance, power and area optimization. These features make Cortex-R series processors a favorable choice in the industry for reliable systems that requires maximum resistance to error.

The purpose of this team and this project is to serve system level performance data and analysis to the stakeholders. The key focus is the debugging of outliers and report the performance bugs.

The motivation for a system level performance analysis comes from an idea to see how the Arm IPs are working together when then are all integrated as a system and not just working independently. As independently the constraints could be less hence the performance could be better, but when stitched together a lot more constraints get added and hence the Analysis becomes more interesting to understand.

1.3 Objective

- To make Arm IP performant and power efficient.
- To serve system level performance data.
- To report and fix the performance bugs.
- To showcase performance of Cortex-R series processors using various benchmarks, various compilers and on various platforms

1.4 Scope of Project

The idea of this project is to understand and analysis the performance of Arm Cortex-R series processors at system level and not at component level. Since individually components like interconnect, memory controller, memory may give best of their performances, but while working in a system bound environment the real time constraints like time constraint, memory constraint might impact the performance as whole.

To do the performance analysis in an efficient way, industry standard benchmarks are used. These benchmarks are designed in such a way that they can provide an idea to fulfill the practical life purposes of a CPU core. Based on the performance number achieved, comparison between any cores can be done, as a specific benchmark will focus on a particular real life application.

As there is always a requirement of balance between performance power and area, once the performance is achieved, further it is handled to take care of the power consumptions. Hence power, performance and area should be always balanced.

Chapter 2

Literature Review

2.1 Industry Standard Benchmarks

- **CoreMark** - CoreMark is a simple but sophisticated benchmark that is used to test the functionality of a microcontroller (MCU) or central processing unit (CPU). The main functionality which Coremark checks for an MCU/CPU is the pipeline operation, memory access (or cache) and handling of integer operations.

- **Dhrystone** - This benchmark checks the general performance of a CPU in Dhrystone per second. The test reports general performance in Dhrystone per second. Dhrystone consists of standard code and focuses on string handling. It does not have any floating-point operations. It is efficiently influenced by hardware and software design, compiler and linker options, code optimizing, cache memory, wait states, and integer data types.

- **EEMBC**- It stands for Embedded Microprocessor Benchmark Consortium. It consists of industry standard benchmarks like:-
  - Automotive – This consists of genereic workload tests, basic automotive algorithms and signal processing algorithms also. Signal processing tests includes algorithms which are important for sensors used in engine knock detection, vehicle stability control, and occupant safety systems.
  - Consumer – It consists of tests like JPEG compression and JPEG decompression tests. It is used to test the performance of processors used in digital cameras and other consumer electronic devices.
  - Networking – It is used to check the performance of processors during tasks of moving packets in networking applications.
  - Office – It is used to test the performance of processors in printers, plotters, and other office automation systems that are used for text and image processing tasks.
  - Telecom – It is used to find the performance of processors in modem and related fixed-telecom applications. The tests of this benchmark suite, also serve as a representation of traditional DSP algorithms.
  - FPMark – As the name suggests, it’s a floating point benchmark suite. It is used to check performance of CPUs in embedded applications such as audio, video,

DSP, graphics, automotive, and motor control. The benchmark tests in this suite have options of floating-point precision: single-precision (SP) and double-precision (DP). The tests can be also differentiated by dataset size which can be small, medium or large. These combinations of tests are useful for a broad range of microcontrollers and platforms.

- **Stream** - The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels which are Copy, Scale, Sum and Triad. It focuses on data movements than data calculations. It is designed to work with datasets much larger than the available cache for a system, so that the results are more indicative of the performance of very large, vector style applications.

2.2 Arm Compiler

Arm Compiler is a convenient choice for converting or compiling source codes written in high level languages like C into a set of machine language instructions because Arm compiler is supported with all the latest releases of Arm which consists of the latest features of Arm's architecture. Arm Compiler supports all the latest Arm Cortex as well as the processors that are in development phase.

The Arm Compiler also has options of optimization techniques which helps in saving space and enhances the performance. This includes scheduling of low-level microarchitecture-specific instructions, elimination of unused sections, and Link-Time Optimization (LTO).

Workflow of Arm Compiler consists of few main stages which are Compilation stage, Linking stage and the Execution stage:-

- Compile – In this phase, all the C sources, C++ Sources and the GNU syntax assembly are passed on to armclang. Armclang is compatible with the source code originally written for GCC and also supports GNU syntax assembly.

- Link – In this phase we have object files being converted to images, with the help of user libraries. It consists of an embedded linker that can combine objects and libraries into executable files. Armar is an archiver that helps a set of object files to be combined together. Arm C libraries and Arm C++ libraries are also being used. Arm C libraries by Arm are used for performance and code density optimization.

- Execute – In this stage the image is converted into a binary using 'fromelf', Fromelf is an utility for image converser and is used as a disassembler.

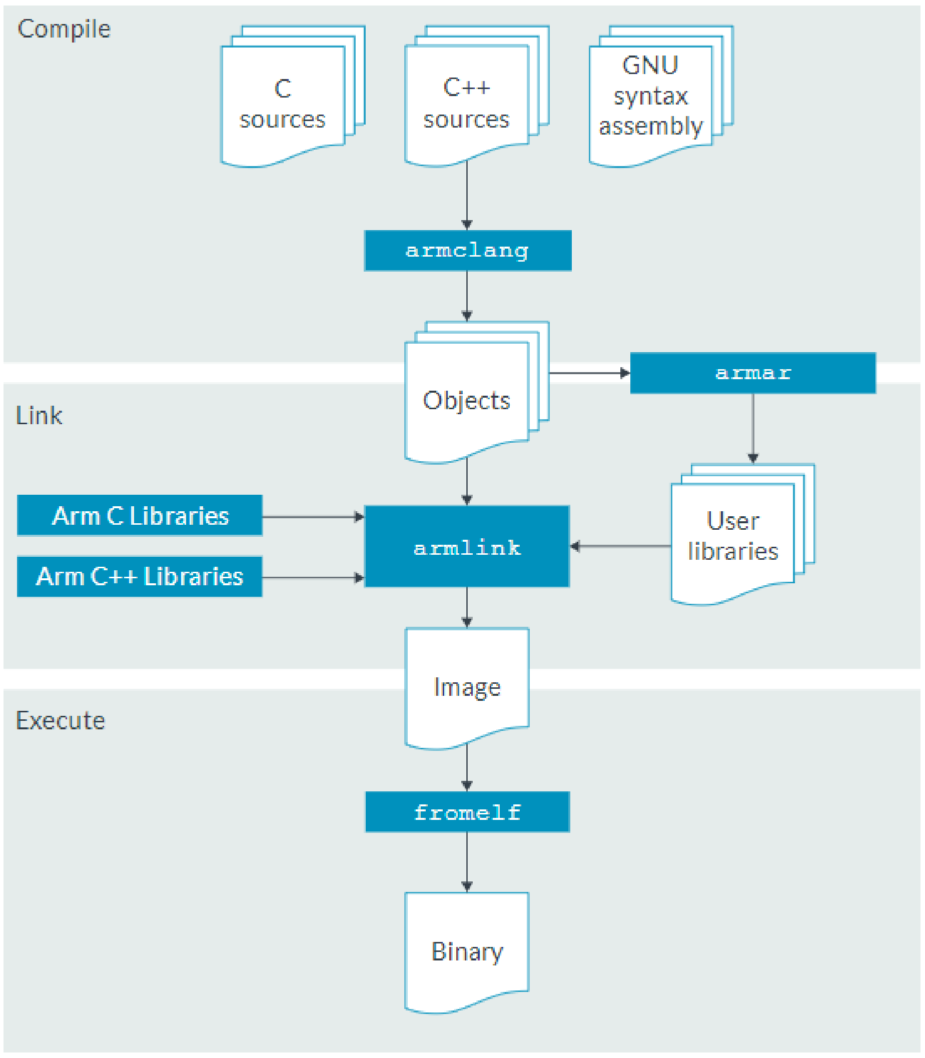


Figure 3.1: Flowchart of Arm Compiler

2.3 Applications of Cortex-R Series Processors

- **Cortex-R82:** This processor has performance which is highest among the processors which are working in real time scenarios from Arm. The architecture that is uses is the Armv8-R AArch64 architecture. It is also the first processor to implement this mode of architecture. Being a powerful real time processor, it is used in smart storage applications. It has outstanding features of larger memory capacity, rich operating system support such as Linux and can help to accelerate ML Workloads. Its field of application lies in datacenters and Enterprise-grade SSDs.

- **Cortex-R52+:** It provides virtualization for real time for applications working for functional safety, along with maintenance of the compatibility of software. Its advantages includes deterministic computation for high performance real time responsiveness. Its areas of applications are Medical Robots and Industrial Automation

- **Cortex-R52:** It delivers the highest level of integrated capability for functional safety of any Arm processor. Its main features are software separation, multiple OS support and real time responsiveness with lowest Cortex-R latency.

- **Cortex-R8:** It uses the Armv7-R architecture and delivers highest performance in its category of processors using the similar architecture. Major benefits of this processor are coherency upto four cores, 11-stage out-of-order pipeline and configurable ports. It is used in applications of WiFi, LTE, 5G Modems and also for Mass Storage.

- **Cortex-R7:** It is a real-time dual core processor with advanced dynamic and static branch prediction.

- **Cortex-R5:** It uses the Armv7-R architecture and provides extended fault containment for real-time applications. It can be said to be a seamless real time embedded processor. It has functional safety support with advanced safety features, efficient high performance processing because of dual core, and system level integration. It used used in mass storage systems for HDD and SDD and in Medical Systems for e.g in remote telemedicine applications, bioinformatics and robotics.

- **Cortex-R4:** It is the smallest deeply embedded real-time processor among all the R-Class processors. It has real time deterministic microarchitecture, high performance efficient processing and build-in error handling. It is used in Hybrid Automotive Clusters that combine traditional meters with graphical displays, in space constrained applications such as HDD and SDD controllers.

2.4 Arm Cortex-R Processor Specifications

- **Instruction Set Architecture –** Cortex-R4, R5, R7 and R8 uses the Armv7-R architecture while Cortex-R52, R52+ and R82 uses the advanced Armv8-R architecture.

- **Pipeline Depth –** Cortex-R4 and R5 has similar pipeline i.e 8 stage in-order and dual issue. Cortex-R7 and R8 has similar pipeline i.e 11 stage out-of-order, superscalar pipelines. Cortex-R52, R52+ and R82 has 8 stage, in-order pipeline. Cortex-R52 and R52+ has superscalar while R82 has triple issue.

- **Address Bits -** All R-Class cores except R82 has 32 Address Bits. Only R82 has 40 address bits.

- **Addressable Memory -** All R-Class processors except R82 has 4GB Addressable memory. Only R82 has 4TB addressable memory.

- **ECC on Memories –** Error Control Correction is present in all R series processors.

- **MPU or MMU -** All R-Class processors except R82 has Memory Protection Unit. Only R82 has both, i.e Memory Protection Unit and Memory Management Unit.

- **Maximum MPU Regions –** This is 12 in case of R4, and 16 in case of R5 and R7. R8 has 24 PMU regions, R52 and R52+ has 24+24 and R82 has 32+32.

- **Symmetric Multi Processing Support (SMP) –** R4 has 1 core and no coherency, R5 has 2 cores and IO coherency, R7 has support upto MP2. R8, R52 and R52+ has support upto MP4 with R52 having no coherency. R82 on the other hand has upto MP8.

- **Floating Point Unit (FPU) –** This unit is optional across all R-Class processors.

- **SIMD (Neon) –** There is no neon feature available in R4, R5, R7 and R8. While in R52, R52+ and R82 it is optional.

- **Maximum External Interrupts –** This valus is upto 480 in case of R4, R5, R7 and R8 while R53 and R52+ has upto 960 external interrupts. R82 on the other hand has 56K+ interrupts.

- **Bus Protocol –** R4, R5, R7 and R8 uses AXI3 bus protocol, R52 and R52+ uses AXI4 and R82 uses the AXI5 bus protocol.

- **L2 Cache –** This feature is only present in case of R82 having 0KB – 4MB L2 Cache.
- **Dual Core Lock-Step (DCLS) –** This feature is available only in R5, R52, R52+ and R82.
- **Safety Documentation Package -** This feature is available only in R5, R52, R52+ and R82.
- **Software Test Library-** This feature is available only in R5, R52, R52+ and R82.

Chapter 3

Hardware Design

2.5 Arm Cortex-R8 Processor

The Cortex-R8 processor is the highest performance Armv7-R processor used for real time purposes. It has a 32-bit CPU core with the Armv7-R architecture. The important features includeds 11-stage pipeline and superscalar out-of-order execution. It has widespread applications in real time scenarios like in LTE and 5G models and mass storage applications such as SSDs and HDDs.

The architecture of Cortex-R8 processor includes major components like Error Correcting Code block, Fast Path Port, 1MB Instruction Tightly Coupled Memory Interface, 1MB Data Tightly Coupled Memory interface, Memory Protection Unit, Floating Point Unit with both Single Precision and Double Precision options. Other blocks include Accelerator Coherency Port, Snoop Control Unit, Advanced eXtensible Interfaces (Slave and Master), Low Latency Peripheral Port and Low Latency RAM Port.

- The Instruction Set Architecture used by Cortex-R8 is Armv7-R. Pipeline depth used in it is a 11 stage out-of-order and superscalar pipeline. Pipeline is basically a microarchitecture feature of such processors. It has 32 address buts and a 4GB addressable memory. It has the support of Error Correction Code on Memory. There is an optional Fast Path Port which is a 32-bit AMBA AXI port provided for each core. This is used to closely integrate peripherals which are latency-sensitive with a specific core within the processor.

- It includes the Memory Protection Unit but does not support the Memory Management Unit. The maximum MPU regions supported by R8 are 24. It also supports Symmetric Multi Processing (SMP) which is upto MP4. The optional Floating Point Unit can work for either single precision-only or both single and double precision. Floating point operations taking place are addition, subtraction, multiplication, division, multiplication and accumulation, square root, conversions between fixed and floating-point, and floating-point constant instructions etc.

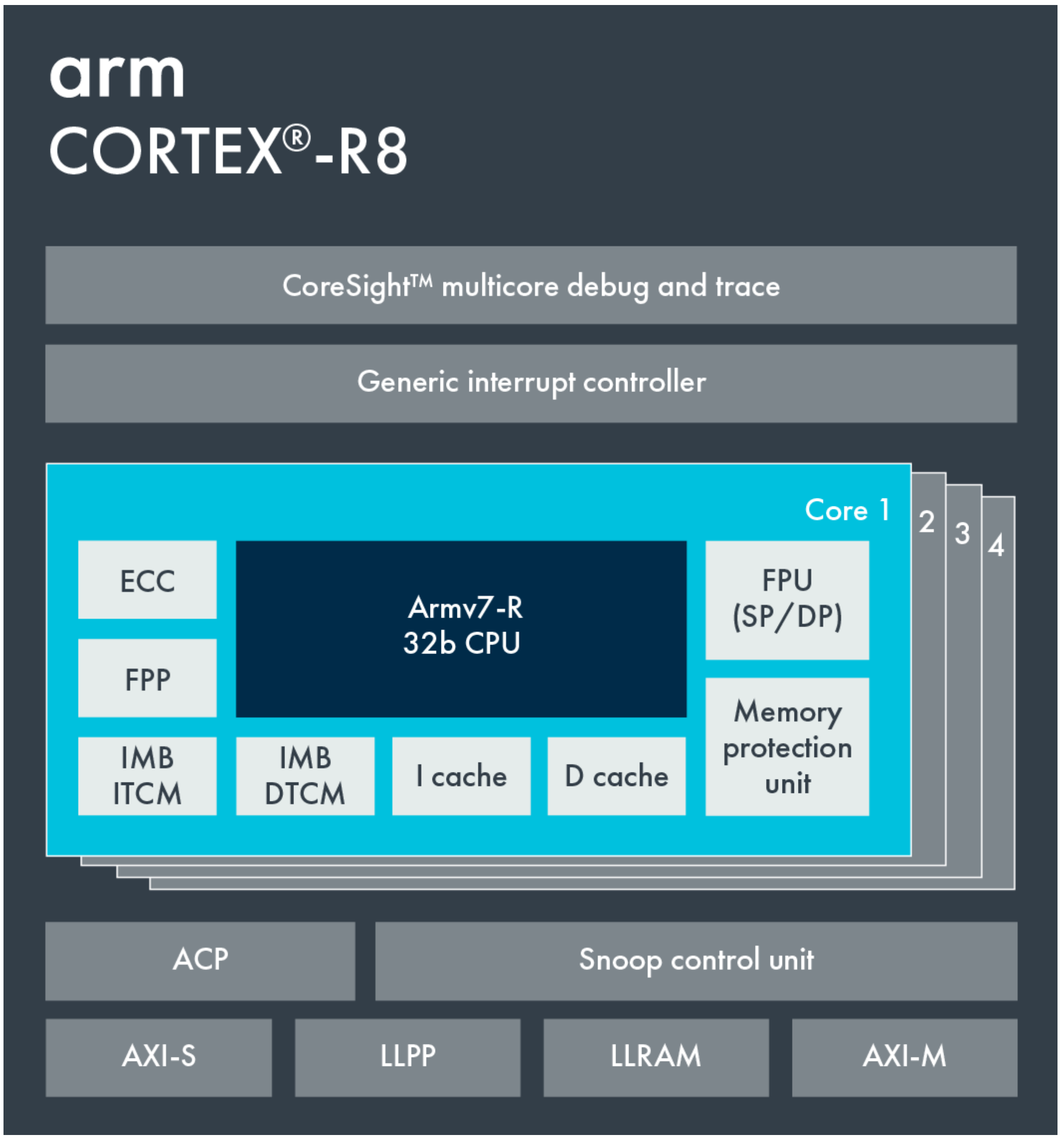


Figure 3.1: Architecture of Cortex-R8 Processor

- **1 MB ITCM -** 1 MB Instruction Tightly Coupled Memory interface
- **1 MB DTCM -** 1 MB Data Tightly Coupled Memory interface
- **I Cache -** Instruction Cache is 0KB – 64KB
- **D Cache -** Data Cache is 0KB – 64KB
- **L2 Cache –** Not avaliable

- SIMD (Neon) is not present in case of R8. The maximum external interrupts allowed are upto 480. Bus protocol used by R8 is AXI4. Also, features like Dual Core Lock-Step (DCLS), Safety Documentation Package and Software Test Library are not available with Cortex-R8 processor.

2.6 Features and Benefits

- **Four-Core Coherency –** Coherency upto four cores helps in the simplification of software development. The updates are synchronized since the data exchange between the cores is maintained. Furthermore, the accelerator coherency port are synchronized with the non coherent accelerators and peripherals in the system.

- **11-Stage Out-of-Order Pipeline –** This leads to faster processing as the 11-Stage Out-of-Order Pipeline makes the most use of the execution bandwidth. Because of the longer pipeline, there is high clock speeds and the out-of-order takes care of minimized processors and forward progress.

- **Configurable Ports –** There are multiple ports offered by Cortex-R8 processor to the system. These are configurable and hence leads to flexible design options and low latency control of accelerators and peripherals. Advantage of this feature is lowest latency and highest throughputs in the end products.

2.7 Applications

- **WiFi, LTE and 5G Modems –** The most recent WiFi, high category LTE, and 5G requires increased transmission speeds. To meet such needs, Cortex-R8 provides, low latency, high performance and power efficient applications. This supports the highest LTE categories and 5G. It creates flexible options to increase feature sets and carries additional workloads reliably.

- **Mass Storage –** Because of the increasing need of storage speed and capacity, storage drivers need to deliver performance reliability, latency reduction and improvement in error correction. Cortex-R8 Processors provide the responsive power required for high performance HDD and SSD controllers.

Chapter 4

Software Design

4.1 Performance Analysis and Benchmarking

Performance Analysis and Benchmarking explains the measurement of performance of CPU cores in various ways. Based on the analysis results, any inefficient section of code can be improved. This procedure includes running benchmarks on the Arm processors, after that the profiling facilities are used to improve the performance of our code as well as the size. When developing application software or comparing the Arm with another processor, following parameters can be measured:

- code and data sizes
- overall execution time
- time spent in specific parts of an application.

With the help of above information, Arm's performance can be compared against other processors in benchmark tests. The comparison helps to identify performance-critical sections of code that can be optimized. The optimization can be done by using a more efficient algorithm, or rewriting in assembly language can also help.

4.2 Code Size

Code size is an important parameter To decide the effectiveness of the compiler for generating code. This is helpful when we need to compare one compiler with other compilers. During the code size calculation and the code size comparison among compilers the following key metrics are used:

- Total RO Size = Code + RO Data
- Total RW Size = RW Data + ZI Data
- Total ROM Size = Code + RO Data + RW Data

Here RO Data (Read-Only Data) consists of constant strings and constant variables, RW Data (Read-Write Data) has initialized variables and zero initialized data and ZI Data (Zero-Initialized Data) has uninitialized global variables.

4.3 Optimization for Code Size and Performance

For optimizing our code, the compiler and associated tools use a lot of optimization techniques. Some of the techniques can help in improving the performance of the code while other techniques reduce the size of the code. Hence often, these optimizations techniques work with opposite effects to each other. It means that if by the use of some optimization technique the code performance is improving it might lead to increased code size and if by some technique the code size reduce, it might reduce the performance. It explains that to achieve higher performance, the compiler can unroll small loops but this in return will create a disadvantage of increased code size. By default there is no optimization offered by armclang. The default optimization level is -O0.

Optimization	Effect
-O0 | -O1 | -O2 | -O3	Specify the level of optimization to be used when compiling source files, where -O0 is the minimum and -O3 is the maximum.
-Ofast	Enables all the optimizations from -O3 along with other aggressive optimizations that might violate strict compliance with language standards.

Table 4.1: Optimization for code performance

Optimization	Effect
-Os	Performs optimizations to reduce the image size at the expense of a possible increase in execution time. This option balances code size against performance.
-Oz	Optimizes for smaller code size.

Table 4.2: Optimization for code size

Optimization	Effect
-flto	Enables link time optimization, which lets the linker make additional optimizations across multiple source files.

Table 4.3: Optimization for both code size and code performance

Optimization can also be done by making following choices during coding:-

- Loop Termination Conditions - Optimizing loop termination conditions can help in improving both code size and performance. Loops with decrementing counters that decrement to zero usually produce smaller, faster code than loops with incrementing counters.

- Unrolling Loops – Performance can be improved by manual unrolling of loops. It means that the number of loop iterations are getting reduced, but the amount of work done in each iteration getting increased. It improves performance at the expense of code size.

- Reducing Debug Information – Image size can be reduced by reducing debug information in objects and libraries. This is useful because we do not need debug information while checking the performance. Debug information can be traced when root causing for outliers.

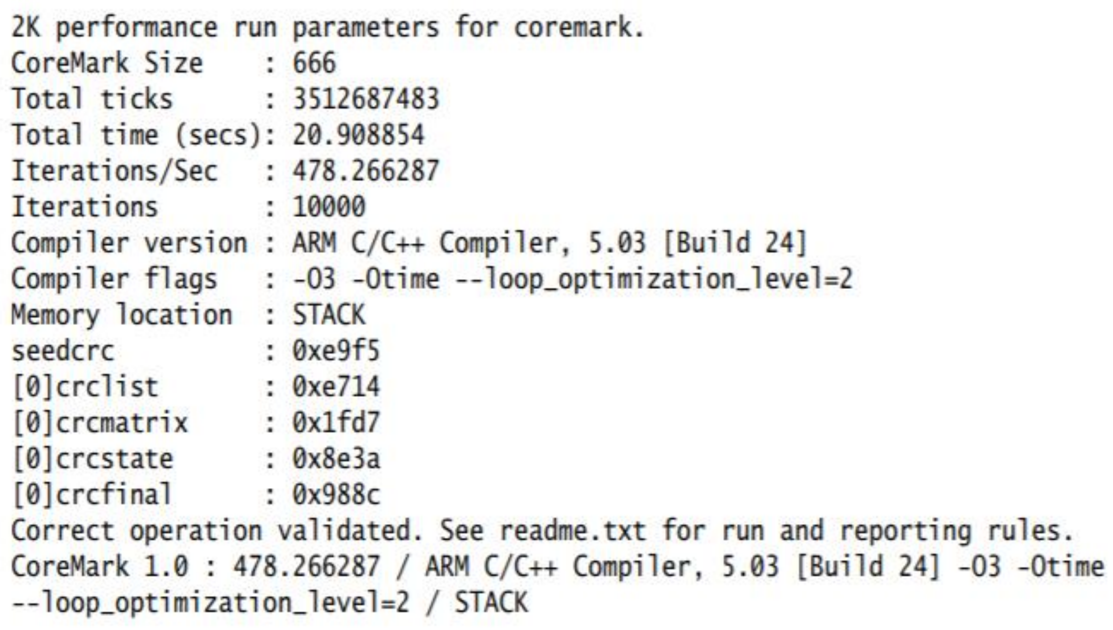
- Inline Functions – Use of inline functions offers a trade-off between code size and performance.

Chapter 5

Calculation

5.1 Performance

5.1.1 CoreMark Performance



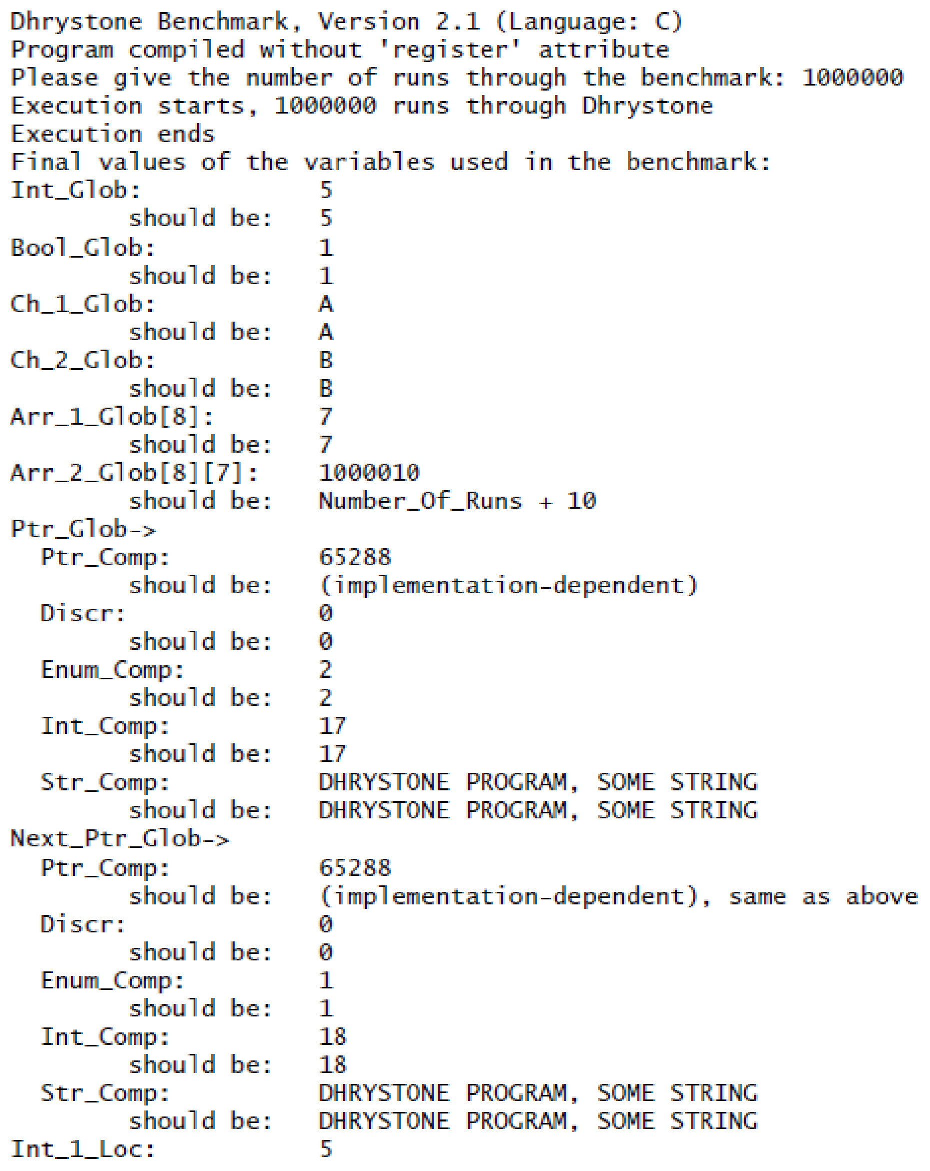
```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 3512687483
Total time (secs): 20.908854
Iterations/Sec   : 478.266287
Iterations       : 10000
Compiler version : ARM C/C++ Compiler, 5.03 [Build 24]
Compiler flags   : -O3 -Otime --loop_optimization_level=2
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 478.266287 / ARM C/C++ Compiler, 5.03 [Build 24] -O3 -Otime
--loop_optimization_level=2 / STACK
```

Figure 5.1: Coremark Performance Output

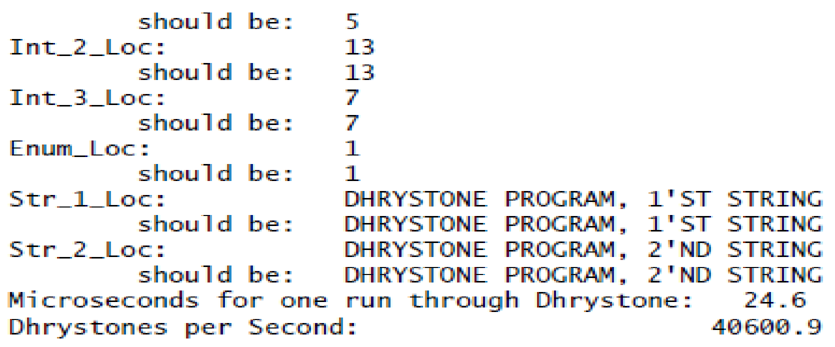
CoreMark: Measurement characteristics

- Above result is the CoreMark output from a development board containing a Cortex-M4 processor running at 168 MHz.
- The CoreMark benchmark number is the number of iterations per second.
- In the output shown, the CoreMark number is 478.266287.
- Score = CoreMarks / MHz.
- Score = CoreMark number (478.266287) / Processor Speed in MHz (168)
- Score = 478.266287 / 168 = 2.8468

5.1.2 Dhrystone Performance



```
Dhrystone Benchmark, Version 2.1 (Language: C)
Program compiled without 'register' attribute
Please give the number of runs through the benchmark: 1000000
Execution starts, 1000000 runs through Dhrystone
Execution ends
Final values of the variables used in the benchmark:
Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    1000010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          65288
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          65288
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
        should be:   18
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:           5
```



```
        should be:   5
Int_2_Loc:           13
        should be:   13
Int_3_Loc:           7
        should be:   7
Enum_Loc:            1
        should be:   1
Str_1_Loc:           DHRYSTONE PROGRAM, 1'ST STRING
        should be:   DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:           DHRYSTONE PROGRAM, 2'ND STRING
        should be:   DHRYSTONE PROGRAM, 2'ND STRING
Microseconds for one run through Dhrystone:    24.6
Dhrystones per Second:                       40600.9
```

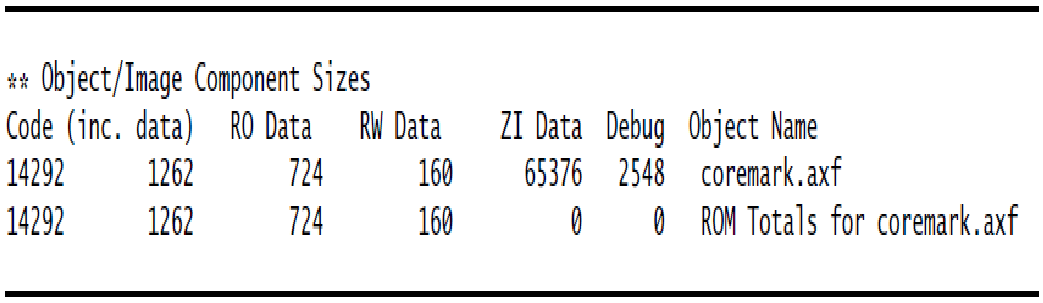
Figure 5.2: Dhrystone Performance Output

Dhrystone: Measurement characteristics

- Above result is the Dhrystone output from a development board containing a Cortex-M3 processor running at 18.5 MHz.
- DMIPS (Dhrystone MIPS) = Dhrystones per second / 1757.
- 40600.9 / 1757 = 23.11.
- Score = DMIPS / MHz.
- Score = 23.11 / 18.5 = 1.25.

5.2 Code Size

5.2.1 Code Size for CoreMark:



```
** Object/Image Component Sizes
Code (inc. data)   RO Data    RW Data    ZI Data  Debug  Object Name
14292      1262        724        160      65376   2548   coremark.axf
14292      1262        724        160          0      0   ROM Totals for coremark.axf
```

Figure 5.3: Coremark Code Size Output

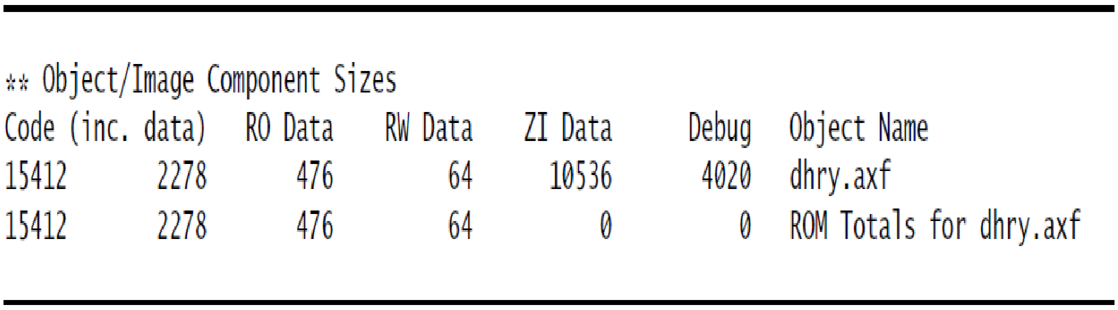
1. First column : It gives the application code size in bytes. It includes the size of inline data.
2. Second column : It shows the size of the inline data that has been included in the application code size. The inline data is located in the code section and it consists of literal pools, case-branch offset tables, and short strings.
3. Third column : It gives the of the read-only data size.
4. Fourth column : It gives read-write data size.
5. Fifth column : It gives the zero initialized data.

- Total RO Size = Code + RO Data = 14292 + 724 = 15016 Bytes

- Total RW Size = RW Data + ZI Data = 160 + 65376 = 65536 Bytes

- Total ROM Size = Code + RO Data + RW Data = 14292 + 724 + 160 = 15176 Bytes

5.2.2 Code Size for Dhrystone:



```
** Object/Image Component Sizes
Code (inc. data)   RO Data    RW Data    ZI Data      Debug   Object Name
15412       2278       476         64      10536       4020   dhry.axf
15412       2278       476         64          0          0   ROM Totals for dhry.axf
```

Figure 5.4: Dhrystone Code Size Output

- Total RO Size = Code + RO Data = 15412 + 476 = 15888 Bytes
- Total RW Size = RW Data + ZI Data = 64 + 10536 = 10600 Bytes
- Total ROM Size = Code + RO Data + RW Data = 15412 + 476 + 64 = 15952 Bytes

Chapter 6

Experimental Results and Discussion

6.1 Block Diagram of R-Class External System

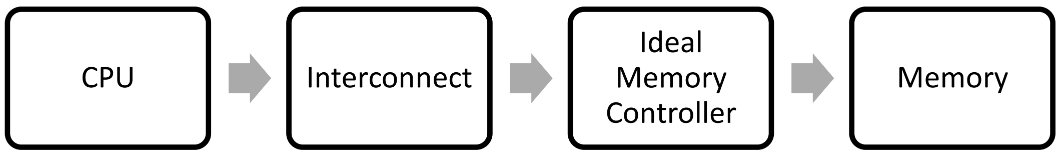


Figure 6.1: Flow of R-Class External System

Major components of R-Class External System are CPU, Interconnect, Ideal Memory Controller and Memory. From CPU, the data request will go to interconnect. Interconnect in return sends that request to Memory Access. The interconnect based on the data received from CPU decides the peripherals. There are many slaves, but interconnect to choose/send data to which slave. Once the interconnect has the information about the peripheral, it puts the data on memory controller, which is a peripheral.

It was observed that latency is not common across all the R-class cores. Difference in system latencies does not lead to a fair comparison of performances between two processors. Hence to tune the latency of all R-class cores, IMC settings are updated. This helps to obtain a common latency so that an unified system can be achieved.

6.2 External System Latency

6.2.1 External System Read Latency :

- External System Latency can be calculated or even verified with Waveform Dump. Software tool used for waveform generation here is Synopsys Verdi.
- External System Latency is the time taken by the request to traval from CPU output back to CPU input. It is calculated in terms of Read Latency and Write Latency.

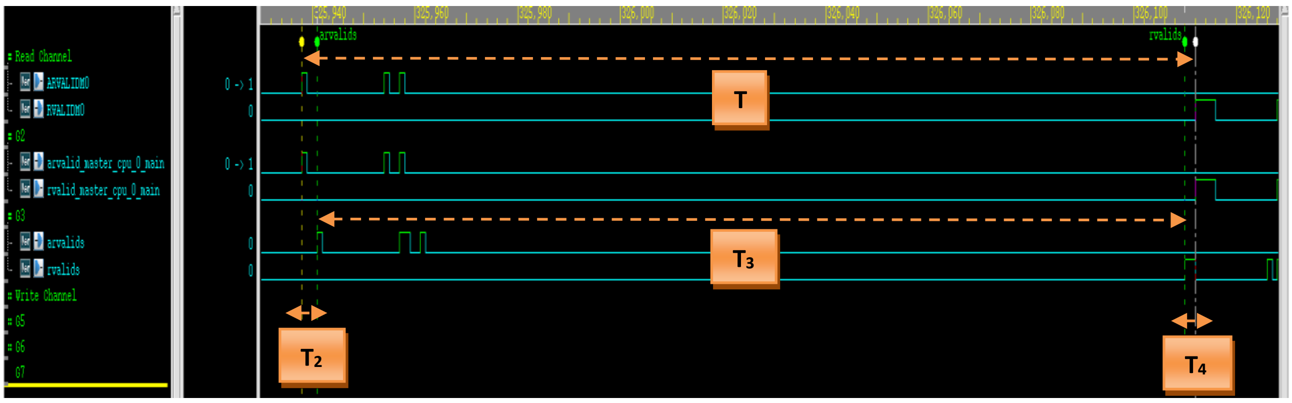


Figure 6.2: External System Read Latency for Cortex-R8

T = CPU (output) to CPU (input) i.e External System Read Latency

T_1 = CPU (output) to Interconnect (input)

T_2 = Interconnect (input) to Memory Controller (input)

T_3 = Memory Controller (input) to Memory Controller (output)

T_4 = Memory Controller (output) to Interconnect (output)

T_5 = Interconnect (output) to CPU (input)

- Total Time (T) = $T_1 + T_2 + T_3 + T_4 + T_5$
- T_3 = Total Time taken by Interrupt Memory Controller.
- Here T_1 and T_2 are 0 ns because the request from CPU to interconnect is reaching instantly and there is no delay. Vice versa from interconnect to CPU.

6.2.2 External System Write Latency :

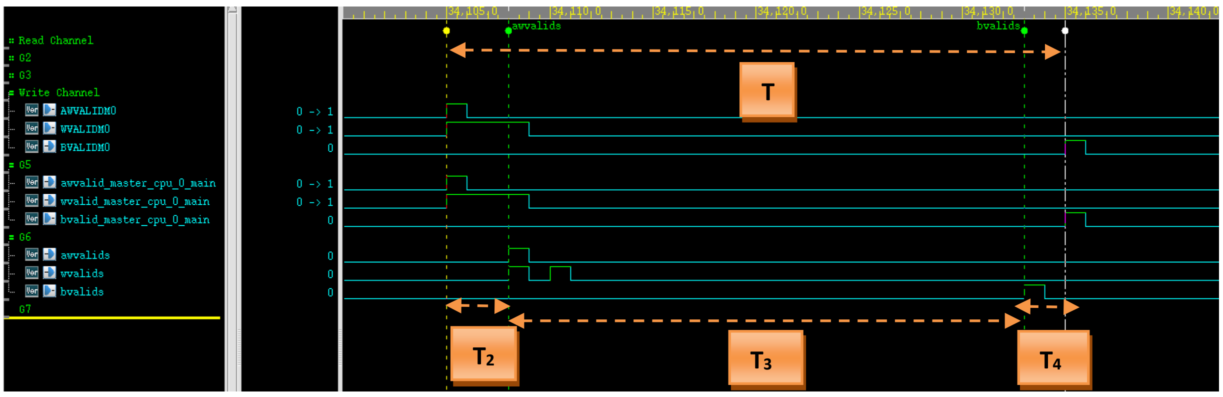
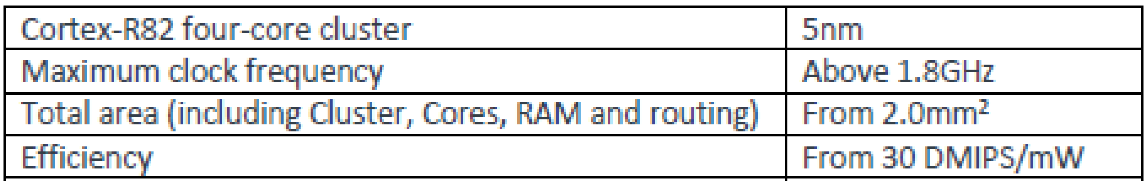


Figure 6.3: External System Write Latency for Cortex-R8

6.3 Characteristics

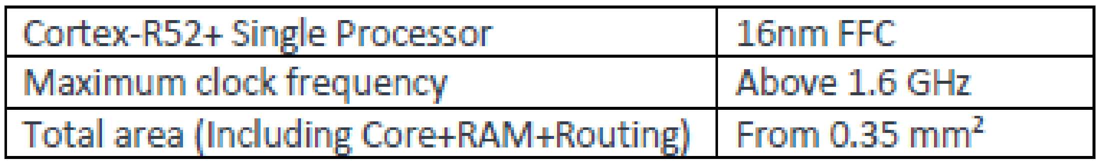
Processor area, frequency, and power consumption are highly dependent on process, libraries, and optimizations.



Cortex-R82 four-core cluster	5nm
Maximum clock frequency	Above 1.8GHz
Total area (including Cluster, Cores, RAM and routing)	From 2.0mm^2
Efficiency	From 30 DMIPS/mW

Table 6.1: Characteristics of Cortex-R82 processor

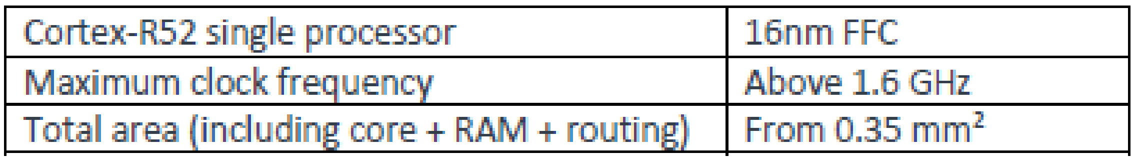
Table 6.1 explains the Cortex-R82 processor characteristics for a four-core cluster implementation. Here 5nm refers that these characteristics are given on a low-power process technology of 5nm with standard-performance cell libraries.



Cortex-R52+ Single Processor	16nm FFC
Maximum clock frequency	Above 1.6 GHz
Total area (Including Core+RAM+Routing)	From 0.35 mm^2

Table 6.2: Characteristics of Cortex-R52+ processor

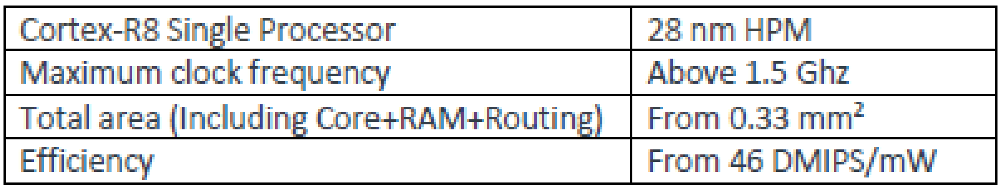
Table 6.2 explains Cortex-R52+ processor as single processor implementation. Here 16 nm refers that the characteristics are given on low-power process technology of 16nm with high-density, standard-performance cell libraries.



Cortex-R52 single processor	16nm FFC
Maximum clock frequency	Above 1.6 GHz
Total area (including core + RAM + routing)	From 0.35 mm^2

Table 6.3: Characteristics of Cortex-R52 processor

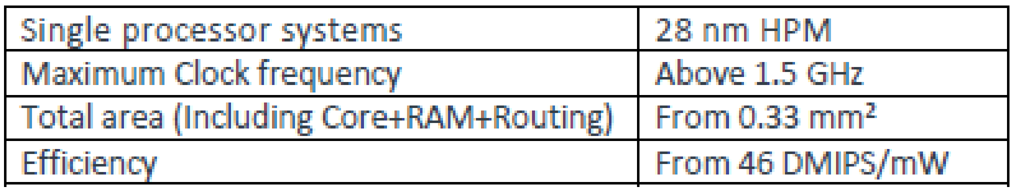
Table 6.3 Explains Cortex-R52 with the single processor implementation. 16 nm refers that these characteristics are given on low-power process technology of 16nm with high-density, standard-performance cell libraries.



Cortex-R8 Single Processor	28 nm HPM
Maximum clock frequency	Above 1.5 Ghz
Total area (Including Core+RAM+Routing)	From 0.33 mm^2
Efficiency	From 46 DMIPS/mW

Table 6.4: Characteristics of Cortex-R8 processor

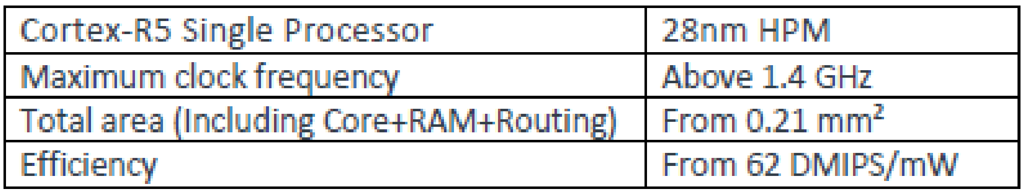
Table 6.4 given the characteristics of Cortex-R8 processor as a single processor implementation. Here 28nm refers that these specifications are given on low-power process technology of 28nm HPM with high-density, standard-performance cell libraries.



Single processor systems	28 nm HPM
Maximum Clock frequency	Above 1.5 GHz
Total area (Including Core+RAM+Routing)	From 0.33 mm^2
Efficiency	From 46 DMIPS/mW

Table 6.5: Characteristics of Cortex-R7 processor

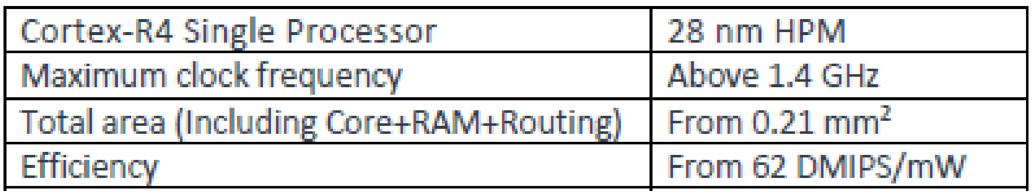
Table 6.5 gives Cortex-R7 with a single processor implementation. These specifications are given as high performance characteristics for mobile process technology (28nm HPM). The instruction cache and data cache are both 32KB in this case.



Cortex-R5 Single Processor	28nm HPM
Maximum clock frequency	Above 1.4 GHz
Total area (Including Core+RAM+Routing)	From 0.21 mm^2
Efficiency	From 62 DMIPS/mW

Table 6.6: Characteristics of Cortex-R5 processor

Table 6.6 explains Cortex-R5 with a single processor implementation. It is given on low-power process technology (28 nm HPM) with high-density, standard-performance cell libraries. The instruction cache and data cache are both 32KB in this case.



Cortex-R4 Single Processor	28 nm HPM
Maximum clock frequency	Above 1.4 GHz
Total area (Including Core+RAM+Routing)	From 0.21 mm^2
Efficiency	From 62 DMIPS/mW

Table 6.7: Characteristics of Cortex-R4 processor

6.4 Performance and Speed

Benchmark Name (Data Unit)	R82	R52+	R52	R8	R7	R5	R4
DhryLegal (DMIPS/MHz)	3.41	2.09	2.09	2.50	2.50	1.67	1.68
DhryInline (DMIPS/MHz)	4.32	2.72	2.72	2.90	2.90	2.02	2.03
DhryInlineLto (DMIPS/MHz)	8.67	5.99	5.99	3.77	3.77	2.45	2.45
CoreMark (CoreMark/MHz)	5.82	4.3	4.3	4.62	4.62	3.47	3.47

Table 6.8: Coremark and Dhrystone Performance

For Dhrystone we run different variants of the benchmark by enabling different flags. Some of the Variants are DhryLegal, DhryInline and DhryInlineLto. DhryLegal result follows all the 'ground rules' which are mentioned in the Dhrystone documentation. DhryInline result allows function inlining. DhryInlineLto result in addition allows optimizations at the time of linking which is known as LTO or Link Time Optimization. LTO performs excessive optimization which results in the removal of unused variables and functions in the source code. Hence the scores are usually very high with this variant.

Coremark Scores indicates about the pipeline of that CPU core. better the pipeline, better will be the performance. From the results it can be observed that though Cortex-R8 is an older core compared to R52 and R52+, still it has better numbers because the pipeline is better. Maximum coremark performance can be observed in Cortex-R82 as it has the best pipelining out of all R-Class processors.

6.5 Performance (Speed) Comparison

The graphical representation for performance analysis can be useful to compare performances between different compilers like GCC and Arm Compilers and between compiler versions. The graphs below reflects the percentage change in CoreMark and Dhrystone Performance from the oldest R-Class CPU Cortex-R4. Hence the baseline considered for the comparison is Cortex-R4.

6.5.1 CoreMark Performance:

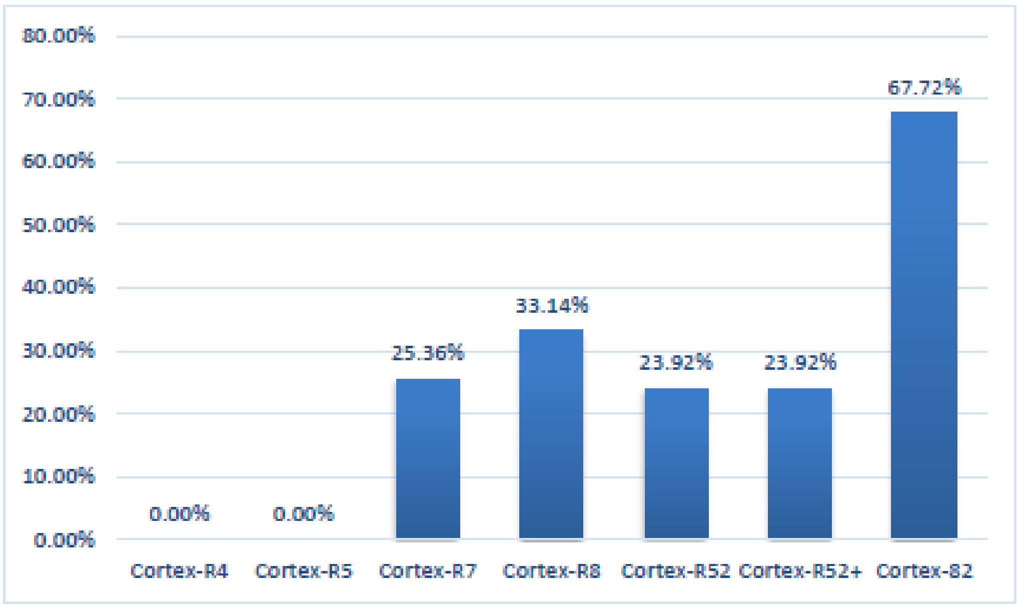


Figure 6.4: Percentage change in Coremark Performance w.r.t Cortex-R4

6.5.2 Dhrystone Performance:

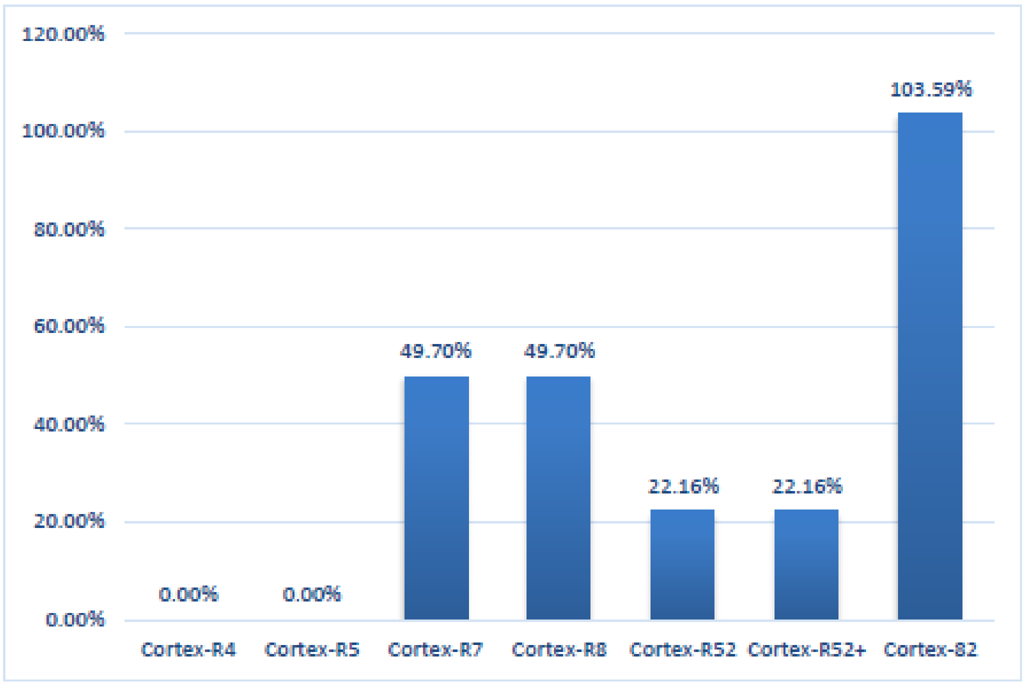


Figure 6.5: Percentage change in Dhrystone Performance w.r.t Cortex-R4

Chapter 7

Conclusion

- Performance Analysis of R-Class cores with the Industry Standard Benchmark CoreMark helps to understand the pipeline of the CPU Core. Hence it gives more details about the Microarchitecture of the CPU core.

- Benchmarking of CPU cores with the Industry Standard Benchmark FPMark which is a floating point benchmark suite is useful in emphasizing embedded applications like Audio, Video, DSP and Graphics. It focuses on operations like Fast Fourier Transform (used in Spectral Analysis and Image Compression), Linear Algebra, Fourier Coefficients etc.

- In most systems the constraint on performance is memory as the memory data rates are lower than the computing rates from cache. Stream Benchmarking helps in overcoming the problems with memory bandwidth on large computer systems as it focuses on data movement over the data calculation.

- Performance Analysis is done for all Cortex-R series processors using various benchmarks like Coremark, Dhrystone, EEMBC and Stream, on various compilers like GCC and Arm specific Compilers and on various platforms like CPAK and Emulation. It helps in verifying the compiler performance as well as the processors' strengths and weak points.

- Fixing the Latency Issue on all R-class cores with a common latency helps in achieving a Unified R-Class System. This becomes as a common platform for all processor comparisons.

- Automated flow with a python script helps in giving all the performance data and code size data into a CSV Format output. This CSV can be uploaded to Tableau Software and useful data representations can be created in the Dashboard.

References

[1] Gal-On, Shay, and Markus Levy. "Exploring coremark a benchmark maximizing simplicity and efficacy." *The Embedded Microprocessor Benchmark Consortium* (2012).

[2] Poovey, Jason A., Thomas M. Conte, Markus Levy, and Shay Gal-On. "A benchmark characterization of the EEMBC benchmark suite." *IEEE micro* 29, no. 5 (2009): 18-29.

[3] Weiss, Alan R. "Dhrystone benchmark: History, analysis, scores and recommendations." (2002).

[4] McCalpin, John D. "A survey of memory bandwidth and machine balance in current high performance computers." *Newsletter of the IEEE Technical Committee on Computer Architecture (TCCA)(December 1995)* (1997).

[5] McCalpin, John D. "Sustainable memory bandwidth in current high performance computers." *Silicon Graphics Inc* (1995).

[6] www.arm.com

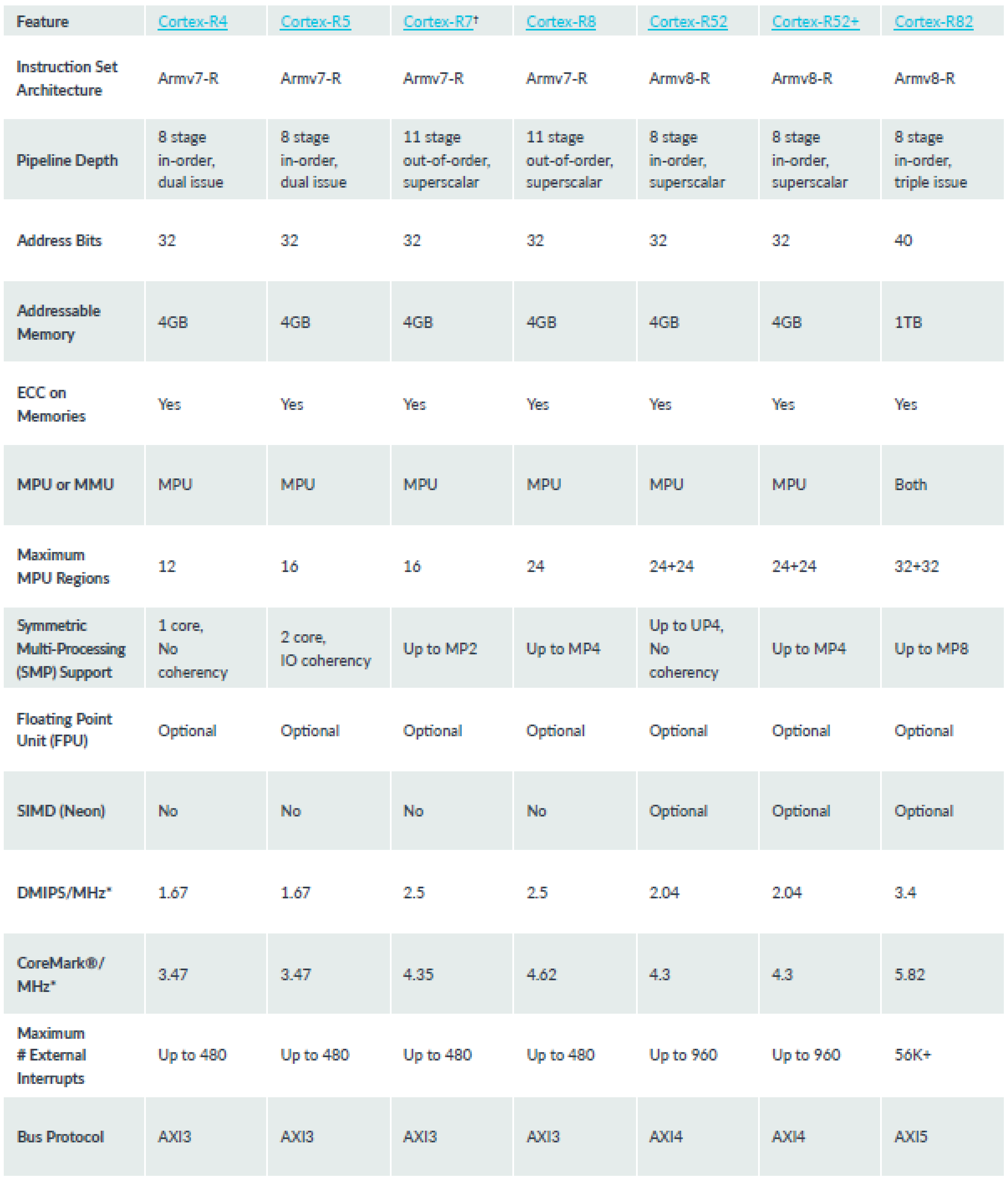
[7] www.developer.arm.com

[8] www.eembc.org

[9] www.cs.virginia.edu

Appendix

Arm Cortex-R Processor Comparison



Feature	Cortex-R4	Cortex-R5	Cortex-R7†	Cortex-R8	Cortex-R52	Cortex-R52+	Cortex-R82
Instruction Set Architecture	Armv7-R	Armv7-R	Armv7-R	Armv7-R	Armv8-R	Armv8-R	Armv8-R
Pipeline Depth	8 stage in-order, dual issue	8 stage in-order, dual issue	11 stage out-of-order, superscalar	11 stage out-of-order, superscalar	8 stage in-order, superscalar	8 stage in-order, superscalar	8 stage in-order, triple issue
Address Bits	32	32	32	32	32	32	40
Addressable Memory	4GB	4GB	4GB	4GB	4GB	4GB	1TB
ECC on Memories	Yes	Yes	Yes	Yes	Yes	Yes	Yes
MPU or MMU	MPU	MPU	MPU	MPU	MPU	MPU	Both
Maximum MPU Regions	12	16	16	24	24+24	24+24	32+32
Symmetric Multi-Processing (SMP) Support	1 core, No coherency	2 core, IO coherency	Up to MP2	Up to MP4	Up to UP4, No coherency	Up to MP4	Up to MP8
Floating Point Unit (FPU)	Optional	Optional	Optional	Optional	Optional	Optional	Optional
SIMD (Neon)	No	No	No	No	Optional	Optional	Optional
DMIPS/MHz*	1.67	1.67	2.5	2.5	2.04	2.04	3.4
CoreMark®/ MHz*	3.47	3.47	4.35	4.62	4.3	4.3	5.82
Maximum # External Interrupts	Up to 480	Up to 480	Up to 480	Up to 480	Up to 960	Up to 960	56K+
Bus Protocol	AXI3	AXI3	AXI3	AXI3	AXI4	AXI4	AXI5

Feature	Cortex-R4	Cortex-R5	Cortex-R7[†]	Cortex-R8	Cortex-R52	Cortex-R52+	Cortex-R82
Instruction TCM	0-8MB	0-8MB	0-8MB	0-1MB	0-1MB	0-1MB	0-1MB
Data TCM	0-8MB	0-8MB	0-8MB	0-1MB	0-1MB	0-1MB	0.16-1MB
Instruction Cache	0, 4KB-64KB	0, 4KB-64KB	0, 4KB-64KB	0KB-64KB	0KB-32KB	0KB-32KB	16KB-128KB
Data Cache	4KB-64KB	4KB-64KB	4KB-64KB	0KB-64KB	0KB-32KB	0KB-32KB	16KB-64KB
L2 Cache	N/A	N/A	N/A	N/A	N/A	N/A	0KB-4MB
Dual Core Lock-Step (DCLS)	No	Yes	No	No	Yes	Yes	No
Safety Documentation Package	No	Yes	No	No	Yes	Yes	No
Software Test Library	No	Yes	No	No	Yes	Yes	No