Free ARM Compatible Softcores on FPGA

Technical Report · December 2018

CITATIONS
0

READS
1,538

1 author:

Shanmugapriyan Manoharan
Bio Consult SH
13 PUBLICATIONS **43** CITATIONS

SEE PROFILE

Free ARM Compatible Softcores on FPGA

by

Shanmugapriyan Manoharan

Submitted as Masters' Degree Project Report

Technische Universität Chemnitz
Department of Electrical Engineering and Information Technology

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Department of
Electrical Engineering
and Information
Technology
Technische Universität Chemnitz
https://www.tu-chemnitz.de/

Project Topic:
Free ARM Compatible Softcores on FPGA

Project Period:
Winter Semester 2018 / 19

Author:
Shanmugapriyan Manoharan

Matriculation Number:
484321

Project Adviser(s):
Dr.-Ing. Erik Markert
Dipl.-lng. Marcel Putsche

Course of Study:
Masters Embedded Systems

Date of Completion:
17.12.2018

Task Description

ARM is a well-known processor family often used in current systems. The original ARM cores are under license of the ARM Company. Nevertheless there exist some free alternatives that are instruction compatible with the commercial cores. The task is to make a literature survey on available ARM-compatible softcores, select a VHDL version of those cores and implement this on our FPGA platform (Xilinx Artix 7) including debug facilities.

The following subtasks shall be solved:

- Analyzing the existing ARM compatible cores available in the internet
- Selection of one core available in VHDL language
- Extend the code in a way that it is testable on the given FPGA board
- Synthesize the VHDL code and check the result using a backannotated simulation
- Transfer the synthesis result to the FPGA
- Testing the FPGA design using ARM compatible binary files

The sources shall be well documented and shall have a structure that might be easily extended with new features like other instruction execution methods.

Contents

List of Figures

List of Tables

Abstract

ARM processor is a 32-bit Reduced Instruction Set Computer (RISC) processor, Instruction Set Architecture (ISA). The ARM architecture has the best MIPS to Watts ratio as well as best MIPS to euros ratio in the industry; the smallest CPU size; all the necessary computing capability couples with low power consumption of which a highly flexible and customizable set of processors are available with options to choose from, all at a low cost.

In fact, the small size, low cost, and low power usage leads to one of the most common uses for an ARM processor today, embedded applications. Embedded environments like cell phones or Personal Digital Assistants (PDAs) require those benefits that this architecture provides.

In this project ARM softcore - **STORM** is emulated on an **Artix-7 FPGA.** We are using the FPGA Board **TE0711** from **Trenz Electronic.**

The existing core is checked using an external module of memory and IO controller. In the project, the author extends the core by adding the memory and IO controller to the main core. The LCD controller design is also added to the core to display the output in the LCD display from **Electronic Assembly - DOGM204A.**

The data from the IO controller is extracted from STORM core which is running on the FPGA. The extracted data is displayed on LCD display connected to the FPGA on a **Debug board.** The Debug has the push button control switches **Start, Stop, Step and Reset** to control the execution of program running on the STORM core.

Glossary

ABT: Abort
ACP: Accelerator Coherence Port
ALU: Arithmetic and Logical Unit
APB: Advanced Peripheral Bus
ARM: Advanced RISC Machine
BX: Branch and eXchange instruction
CISC: Complex Instruction Set Computer
CMSR: Current Machine Status Register
CPSR: Current Program Status Register
CPU: Central Processing Unit
DAB: Data fetch Abort Interrupt
DAC: Digital to Analog Converter
DBX: Direct Bytecode eXecution
DC: Data Cache
DMIPS: Dynamic Microprocessor without Interlocked Pipelined Stages
DSP: Digital Signal Processing
ECC: Error Correcting Code
FIQ: Fast Interrupt Request
FPGA: Field Programmable Gate Array
FPU: Floating Point Unit
GIC: Generic Interrupt Controller
IAB: Instruction fetch Abort Interrupt
IC: Instruction Cache
ICE: In-circuit Emulator
IO: Input Output
IOC: IO Controller
IP: Intellectual Property
IRQ: Interrupt Request
ISA: Instruction Set Architecture
JTAG: Joint Test Action Group
L1 cache: Level 1 cache
L2 cache: Level 2 cache
LCD: Liquid Crystal Display
LDM: Load Memory
LDR: Load Register
LFSR: Linear Feedback Shift Register
LPAE: Large Physical Address Extensions
LUT: Look-Up Table
LUTRAM: Look-Up Table distributed as Random Access Memory

MAC:	Multiply Accumulate
MCS:	Machine Control System
MEM/IO:	Memory / Input Output
MEMC:	Memory Controller
MHZ:	Mega Hertz
MIPS:	Microprocessor without Interlocked Pipelined Stages
MIPs:	Million Instruction per second
MMU:	Memory Management Unit
MPU:	Memory Processing Unit
MUL:	Multiply
NZCV:	Negative, Zero, Carry, Overload flag
PC:	Program Counter
PDA:	Personal Digital Assistants
PU:	Protection Unit
RAM:	Random Access Memory
RISC:	Reduced Instruction Set Computer
RTOS:	Real-Time Operating System
SCU:	Snoop Control Unit
SIMD:	single instruction multiple data
SMP:	Symmetric Multi-Processing
SMSR:	Saved Machine Status Register
SPI:	Serial Peripheral Interface
SPSR:	Saved Program Status Registers
STM:	Store Memory
STR:	Store Register
SVP:	Supervisor
SYS:	System mode
TCM:	Tightly-Coupled Memory
TDMI:	16 bit Thumb + JTAG Debug + fast Multiplier + enhanced ICE
TLB:	Translation Lookaside Buffer
UART:	Universal Asynchronous Receiver/Transmitter
UND:	Undefined Instruction
USR:	User mode Memory
VFU:	Vector Floating-point Unit
VHDL:	Very high speed Hardware Description Language
VIDC:	Video Controller

Chapter 1

Introdution

Advanced RISC Machine (ARM) holdings [1] incorporated was founded as Advanced Reduced Instruction Set Computer (RISC) Machines is the main Central Processing Unit (CPU) design strategy implemented in its processors at 1990. ARM is a semiconductor company that develops system-on-chips, processors and software.

Intellectual Property's (IP's) of ARM include its low cost, low power, efficiency of RISC microprocessors is high, other peripherals and system on chips. It doesn't manufacture semiconductor devices or processors but rather licenses the cores as IP to other companies like ATMEL, NXP, Samsung etc.

Processors developed by ARM IP's which are used in embedded systems like tablet computers, smart watches, phones and TV's. ARM Processors are almost in many domains like consumer electronics, automation, handheld devices and robotics.

1.1 Features of ARM

ARM Processors are based on RISC architecture. But the processor design based on the current embedded systems requirements, with some improvements to the RISC architecture are made. The instructions for accessing the memory and data processing on registers are different. The ARM's instruction set is fixed in length and uniform. ARM 32-bit Processors have two instruction sets:

- General 32-bit ARM Instruction Set, and
- 16-bit Thumb Instruction Set

It has the following features [1]:

1. 3-address data processing instructions
2. Conditional execution of every instruction
3. The load and store multiple register instructions are included
4. Performs a general shift operation and a general Arithmetic and Logical Unit (ALU) operation in a single instruction that executes in a single clock cycle
5. A 16-bit compressed representation of the instruction set in the Thumb architecture
6. Pipelining: Instruction pipelining is one of the key features which reduces execution time
7. A high clock rate with single-cycle execution

1.2 Why ARM [1]

1. At decoding instructions, ARM is better than Intel chips
2. ARM architecture uses RISC which makes CPU cores about 20 percent smaller and more power efficient.
3. Instruction set allowed for a pipelined, RISC processor superscalar design achieve nearly 2 to 4 times the performance of complex instruction set computer (CISC) processors
4. As the RISC processor's instruction set is so simple, it uses less chip space; extra functions, such as floating point arithmetic units or memory management units, which are placed on the same chip
5. A simple processor should take less design effort and therefore have a lower design cost and be better matched to the process technology when it is launched that is, the shorter development time.
6. The combination of the simple hardware with an instruction set that is grounded in RISC and also retains a few key CISC features, and thereby achieves a significantly better code density than a pure RISC.

1.3 ARM Nomenclature [2]

ARM follows the nomenclature shown in the Figure 1.1 to describe the implementation of the implementations. The letters after "ARM" are used to indicate the features of a processor.

Figure 1.1: ARM Nomenclature [2]

- x – Family/series
- y – Memory Management Unit (MMU)/Protection Unit (PU)
- z – Cache
- T – 16 bit Thumb decoder
- D – Joint Test Action Group (JTAG) Debugger
- M – Fast Multiplier
- I – Embedded In-circuit Emulator (ICE) Macrocell
- E – Enhanced Instructions for DSP (assumes 16 bit Thumb + JTAG Debug + fast Multiplier + enhanced ICE (TDMI))
- J – Jazelle (for accelerated JAVA execution)
- F – Vector Floating-point Unit
- S – Synthesizable

*T – **T**humb Instruction Set*

ARM Processors supports the 32-bit ARM and 16-bit Thumb Instruction Set. The 32-bit ARM Instructions consists of 32-bit opcodes which turns out to be a 4-byte binary pattern. The 16-bit Thumb Instructions consists of 16-bit opcodes or 2-byte binary pattern which improves the code density.

*D – JTAG **D**ebug*

Joint Test Action Group (JTAG) is a serial protocol used by ARM to transfer the debug information between the test equipment and processor.

*M – Fast **M**ultiplier*

Previously, ARM Processors used a small and simple multiplier unit and it required more clock cycles to complete a single multiplication. With the introduction of Fast Multiplier unit in the ARM Processors, the clock cycles required for multiplication are significantly reduced and modern ARM Processors are capable of calculating a 32-bit product in a single cycle.

*I – Embedded **I**CE Macrocell*

ARM Processors have on-chip debug hardware that allows the processor to set watchpoints and breakpoints.

*E – **E**nhanced Instructions*

ARM Processors with the Enhanced instruction mode supports the extended DSP Instruction Set for high performance DSP applications. With these extended DSP instructions, the DSP performance of the ARM Processors can be increased without high clock frequencies.

*J – **J**azelle*

ARM Processors with the Jazelle Technology can be used in Java bytecodes with accelerated execution. Jazelle Direct Bytecode eXecution (DBX) is used in consumer devices and mobile phones for high performance Java execution without affecting battery or memory.

*F – Vector **F**loating-point Unit*

The ARM Processors with the Floating Point Architecture provides the execution of floating point arithmetic operations. The Precision and Dynamic Range offered by the ARM Processors with the Floating Point Architecture are used in many real time applications in the automotive and industrial areas.

*S – **S**ynthesizable*

The source code are available for the ARM Processor Core. By using EDA tools, this software core can be compiled into a format that can be easily understood. It is possible to modify the architecture of the ARM Processor using the processor source code.

1.4 ARM Classifications

The primary processor groups [3] used at ARM are: Classic, Embedded, and Applications. These are characterized by the added capabilities from advanced features, as well as increasing functionality and performance which is demonstrated by the graph shown in Figure 1.2.

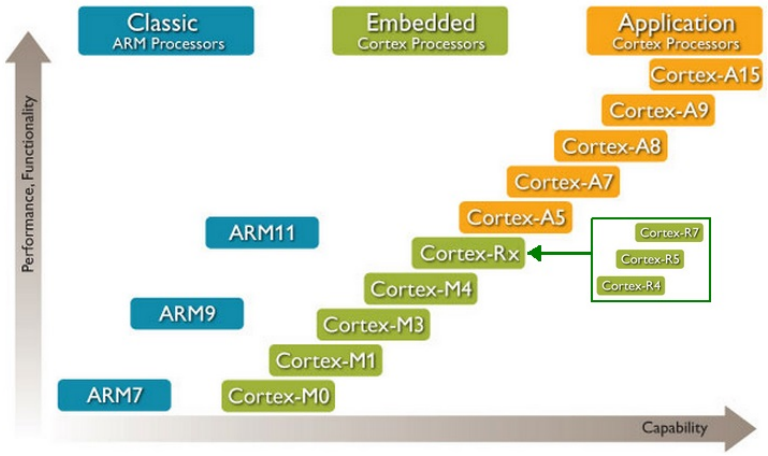


Figure 1.2: ARM Processor Performance, Functionality Vs Capability [3]

The three major ISA versions of ARM architecture were included in the classic processors. ARM7 (Actually the ARM7TDMI or ARM7EJ), using ARMv4, is almost entirely antiquated at this point, and the company look forward for the Cortex counterparts. For DSP and java applications, ARM9 (ARMv5) is still in use as a low-end single processor. Modern development are still widely using ARM11which is based on ARMv6. These older processors have a binary compatible where utilized in the Cortex family for design upgrades and which does not require a software redesign in large scale.

The next classification of ARM processors used in the modern embedded processors, Cortex-R for real-time applications and Cortex-M for microcontroller applications. These processor utilizes the ARMv7, as indicated by the Cortex, and includes the Thumb-2 Instruction set. The difference from the applications series, both of these processor families utilize the MPU for memory control. They also operate on a Real-Time Operating System (RTOS) in conjunction with user generated code. The Cortex-R features deeper pipelines and uses high clock frequencies. It also utilizes Tightly-Coupled Memory (TCM) for fast access to important data or instructions that are needed for immediate access. TCM is level 1 memory, and in some cores it entirely replaces the cache. In contrast, the Cortex-M is designed with low-power, code density, and interruption management as focus points. The Cortex-M series exclusively uses Thumb-2 and does not have the ARM instruction set. Thumb-2 allows it to maintain the low impact design requirements of its 8/16-bit competitors while still keeping the performance advantage offered a 32-bit machine. Due to this instruction set it is able to function as the industry standard by vastly outperforming competition in a MIPs per MHZ comparison.

The final classification of ARM processors is the applications series, Cortex-A. These are used for high functionality, and have the ability to run complex and complete operating systems. The main difference from the embedded classification Cortex processors is the applications series uses the MMU for memory control instead of the MPU. These cores supports a fully coherent L1 cache. The certain features that are used as options in the other processor families are automatically included in all Cortex-A processors, namely Jazelle and NEON.

A complete observation of the different features that were discussed and their availabilities for the different processor families may be observed in Figure 1.3. The top half of the image indicate the

processor classification, and the processors with their respective architecture are listed above. The bottom half indicates the various options are available to the specific architecture.

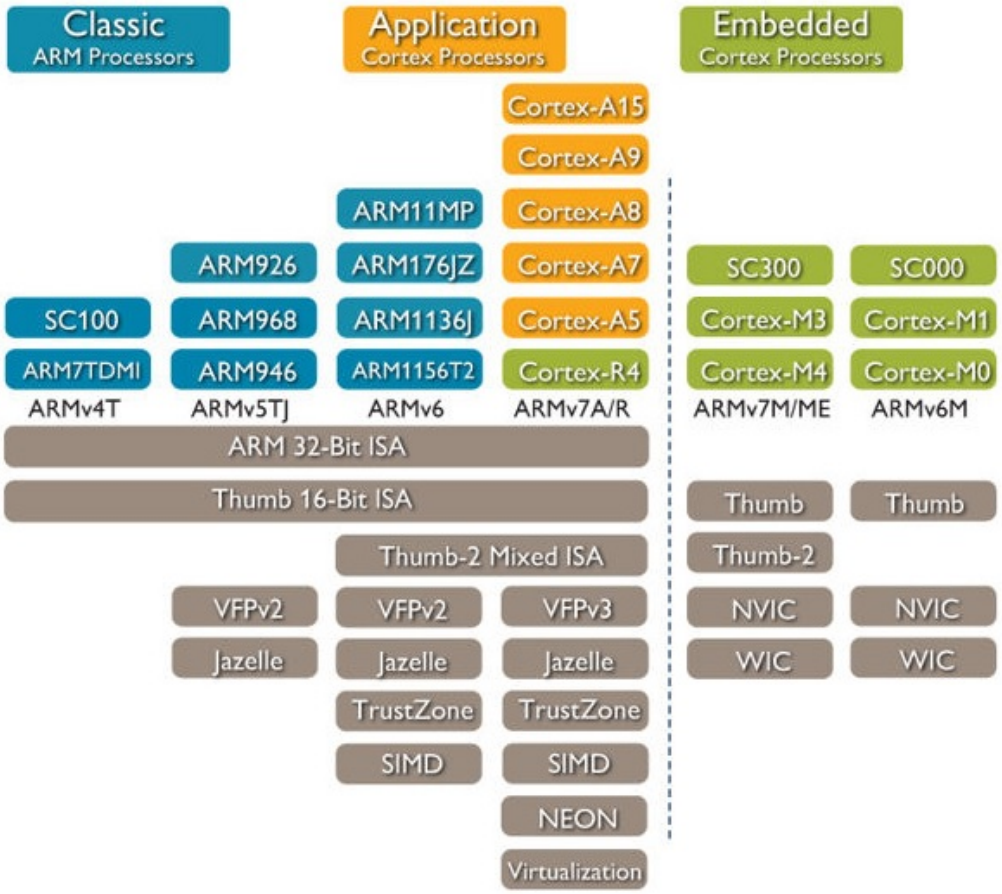


Figure 1.3: ARM Processors and its Features [3]

1.5 ARM Architecture

ARM shares all of the relevant characteristics of an instruction set of RISC. However, it was necessary to expand and enhance the capabilities of a typical RISC machine. ARM still uses the simple addressing modes, load/store architecture, fixed instruction width, and uniform register files common to RISC machines.

There are 37 registers in the ARM in which 30 registers used for general purpose, 6 used for status registers, and 1 for program counter. The 15 out of 30 general registers are accessible along with the program counter and the status register at any given time. The registers which are accessible depends on the operating mode used by the processor. The seven operating modes are used in the ARM, six out of seven are privileged and seventh is the user mode which is unprivileged. The first two privileged modes are used for interrupt handling:

- Low-priority normal interrupts, IRQ and
- Immediate needs interrupts, FIQ

For the memory access violations and unrecognized instructions, the Abort and undefined modes are used respectively. When the system is reset and for software interrupts, the Supervisor mode is used. The user and system mode uses the exact same registers. Figure 1.4 illustrates the different register swaps and modes that accompany them.

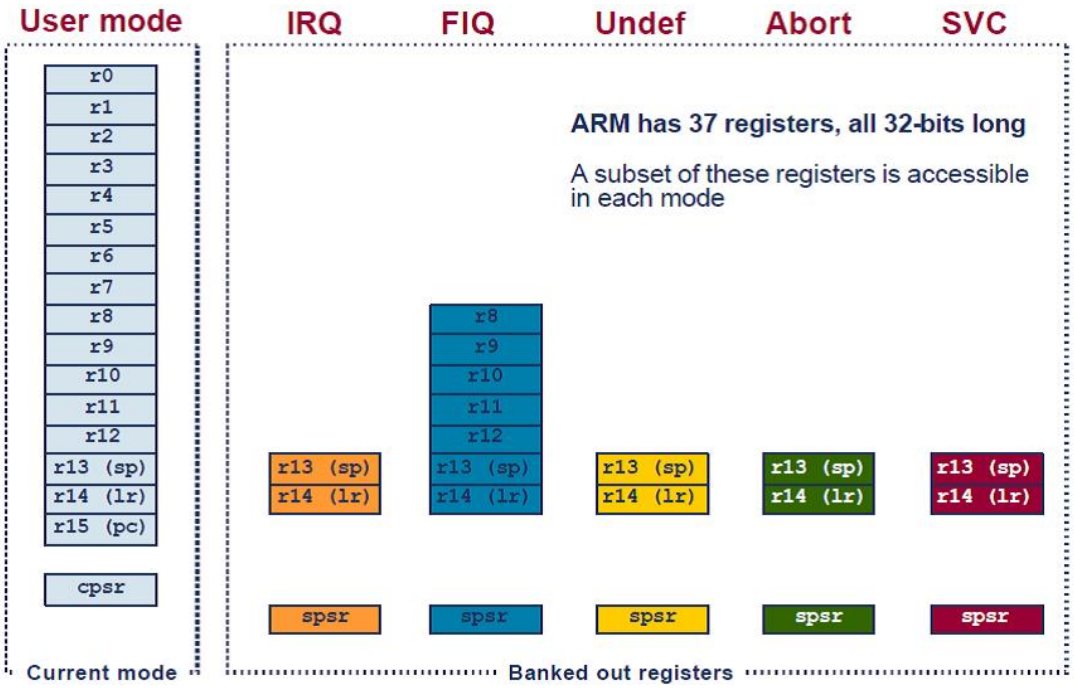


Figure 1.4: ARM Registers [3]

The six status registers consist of a five Saved Program Status Registers (SPSRs) and single Current Program Status Register (CPSR). The CPSR which contains the current state of the machine was used by the system and user modes. The content of the CPSR is preserved into the corresponding SPSR, whenever there is a change of mode. When the handling of the exception or interrupt that prompted the mode change, then the state is stored in the SPSR which allows a return to the previous state. The full analysis of the program status registers can be seen in Figure 1.5.

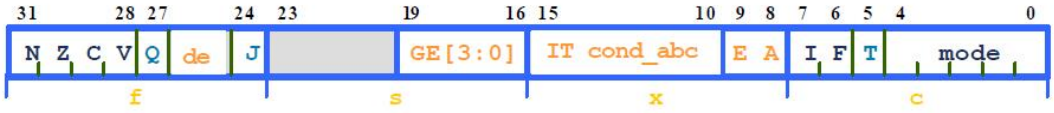


Figure 1.5: ARM Status Register [3]

The current operating modes are indicated by the bottom 5 bits. The "T" is the 6th bit, read-only bit used to determine whether the machine is operating in the ARM ISA or Thumb ISA. The enabling or disabling of low priority and high priority interrupts was done by using "I" and "F" bits respectively. The 25th bit labeled "J", indicates whether the processor is in a Jazelle state or not. The NZCV is the most significant four bits and referred as the condition flags. These are flagged for the following conditions:

- Negative result from ALU,
- Result of zero from ALU,
- ALU operation carried out, and
- ALU operation overflowed.

Chapter 2

Literature Survey

2.1 List of Open Soft Cores

ARM Holdings major business is vending IP cores, these licenses are used to create CPUs, systems-on-chips and microcontrollers base on the IP cores. It offers various licensing terms which varies in cost and deliverables. It also provides all licenses an integrate-able hardware description as well as software development toolset.

ARM Holdings have lend out few of the IP cores which are available as open cores for the students to learn about the ARM architecture and for third to use in their products. The few o the available ARM Holdings are discussed below.

The **ARM2** is a reimplementation of the ARM1 on a smaller process along with the addition of a number of additional enhancements. The ARM2 was capable of exceeding 10 MIPS when not bottlenecked by memory with an average of around 6 MIPS. The ARM2 was designed to work as an embedded controller or a coprocessor or as a stand-alone microprocessor system.

The **ARM250** is a very high integration chip - incorporating the ARM3 core along with most of the new MMU logic that was developed for the ARM6 along with all the support chips that were previously needed for the ARM3/2 - the MEMC chip (Memory Controller), VIDC chip (Video Controller), IOC/IOEB (I/O Controller).

The **ARM3** is a higher performance processor through the introduction of on-die cache but without any major changes to the core itself. It can operate at up to 25 MHz with a peak performance of 25 MIPS and a sustainable performance of 12 MIPS.

The **Cortex-A8** [4] processor is a low-power, cached application, high-performance processor that provides full virtual memory capabilities. The full implementation of the ARMv7-A instruction set. The integer instructions are executed in a pipeline. ARMv7 debug with breakpoint and watchpoint registers and a 32-bit Advanced Peripheral Bus (APB) slave interface to a Core Sight debug system.

The **Cortex-A9** [5] processor is a low-power, high-performance, ARM macrocell with an L1 cache subsystem which provides full virtual memory capabilities. This processor implements the ARMv7 A architecture and runs 16-bit and 32-bit Thumb instructions, 32-bit ARM instructions, and 8-bit Java bytecodes in Jazelle state.

The **Cortex-A15** [6] MPCore processor is a low-power, high-performance multiprocessor that implements the ARMv7-A architecture. This processor has one to four processors in a single multiprocessor device with L1 and L2 cache subsystems.

The **Cortex-A53** [7] processor is a low-power, mid-range processor that implements the Armv8-A architecture. This processor has one to four cores, each with a single shared L2 cache and an L1 memory system.

The **Cortex-A57** [8] processor is a low-power, high-performance processor that also implements ARMv8-A architecture. It consist of one to four cores in a single processor with L1 and L2 cache subsystems.

The **Cortex-A73** [9] processor is a low-power, high-performance, ARM macro cell that also implements the Armv8 A architecture. It also contains up to four cores, each with one shared Level 2 (L2) cache and a Level 1 (L1) memory system. The maximum throughput of two instructions per cycle were handled by each core. This processor includes a variable-length, superscalar, out-of-order pipeline.

The Table 2.1 has the list of microarchitectures based on the instruction sets designed by ARM Holdings and 3rd parties, ARM instruction set and name are sorted by the version. It also provides the vendors who implement ARM cores in their design.

Table 2.1: Open Softcores of ARM [10]

ARM Holdings	**Opencores / Third Party**	**ARM architecture**	**Features**
ARM2, ARM250, ARM3	Amber, STORM	ARMv2	ARMv2 added the multiply (MUL) instruction ***Cache/Memory Management Unit (MMU):*** None ***Microprocessor without Interlocked Pipelined Stages (MIPS) @ MHz:*** 4 MIPS @ 8 MHz 0.33 DMIPS/MHz
ARM8	StrongARM, FA526, ZAP	ARMv4	double-bandwidth memory, static branch prediction ***Cache/MMU:*** 8 KB unified, MMU ***MIPS @ MHz:*** 84 MIPS @ 72 MHz
Cortex-A8	Qualcomm Krait, Scorpion, PJ4/ Sheeva, Apple Swift	ARMv7-A	ARM / NEON / Thumb / Thumb-2 / Jazelle Randomized Control Trial (RCT) / Vector Floating Point version 3 (VFPv3) Floating Point Unit (FPU) and Digital to Analog Converter (DAC), 13-stage superscalar pipeline ***Cache/MMU:*** 0–1 MB L2 optimized Error Correcting Code (ECC) memory, 16–32 KB / 16–32 KB L1, MMU + TrustZone (embedded security option for the ARM Cortex-based processor systems) ***MIPS @ MHz:*** Up to 2000 (2.0 Dynamic MIPS (DMIPS)/MHz in speed from 600 MHz to greater than 1 GHz)
Cortex-A9			ARM / DSP / Optional VFPv3 FPU / Thumb / Thumb-2 / Jazelle RCT and DBX / Optional NEON, out-of-order speculative issue superscalar, Microprocessor Core, 1–4 SMP cores, Accelerator Coherence Port (ACP), Snoop Control Unit (SCU), Generic Interrupt Controller (GIC) ***Cache/MMU:*** 0–8 MB L2 opt. parity, 16–64 KB / 16–64 KB L1, MMU + TrustZone ***MIPS @ MHz:*** 2.5 DMIPS/MHz per core, 10,000 DMIPS @ 2 GHz on Performance Optimized TSMC 40G(dual-core)

ARM Holdings	Opencores / Third Party	ARM architecture	Features
Cortex-A15	Qualcomm Krait, Scorpion, PJ4/ Sheeva, Apple Swift	ARMv7-A	ARM / DSP / VFPv4 FPU / NEON / Thumb / Thumb-2 / integer divide / Jazelle RCT / hardware virtualization / fused Multiply Accumulate (MAC) , out-of-order speculative issue superscalar, PCore, Large Physical Address Extensions (LPAE), 1–4 SMP cores, SCU, GIC, 15-24 stage pipeline, ACP ***Cache/MMU:*** 0–4 MB L2, L2 has ECC, 32 KB w/parity / 32 KB w/ECCL1, MMU + TrustZone ***MIPS @ MHz:*** At least 3.5 DMIPS/MHz per core (up to 4.01 DMIPS/MHz depending on implementation)
Cortex-A53	X-Gene, Nvidia Project Denver, Cavium Thunder X, AMD K12, Apple Cyclone/ Typhoon/ Twister/ Hurricane/ Zephyr, Qualcomm Kryo, Samsung M1 and M2	ARMv8-A	AArch32 and AArch64, NEON advanced SIMD, 1–4 SMP cores, TrustZone, hardware virtualization, dual issue, VFPv4, in-order pipeline ***Cache/MMU:*** 128 KB–2 MB L2 shared, 40-bit physical addresses, 864 KB w/parity / 864 KB w/ECC L1 per core ***MIPS @ MHz:*** 2.3 DMIPS/MHz
Cortex-A57			AArch32 and AArch64, NEON advanced Single Instruction Multiple Data (SIMD), 1–4 Symmetric Multi-Processing (SMP) cores, TrustZone, hardware virtualization, dual issue, VFPv4, in-order pipeline ***Cache/MMU:*** 128 KB–2 MB L2 shared, 40-bit physical addresses, 864 KB w/parity / 864 KB w/ECC L1 per core ***MIPS @ MHz:*** 2.3 DMIPS/MHz
Cortex-A73			AArch32 and AArch64, 2-way superscalar, deeply out-of-order pipeline, VFPv4, 1–4 SMP cores, TrustZone, NEON advanced SIMD, hardware virtualization ***Cache/MMU:*** 256 KB–8 MB L2 shared w/ optional ECC, 64 KB / 3264 KB L1 per core, 44-bit physical addresses ***MIPS @ MHz:*** 4.8 DMIPS/MHz

The above table summarizes the available ARM cores as opencores and with the Third party. As per the task requirement the available opencores are analyzied in section 3.1 and out of the available core, one core is selected to implement in the given FPGA board.

2.2 STORM Core Processor [11]

STORM core operation codes, functionality, and programmer's models is similar to ARM's 32-bit processor family (instruction architecture – ARMv2).

2.2.1 Features

- Function and Opcode compatible to ARMv2 architecture
- 32-bit RISC open source soft-core processor

- 8 stage-Pipelined instruction execution
- Single cycle execution except for branch and multi-cycle memory operations
- 7 operating modes with unique privileges and register sets
- 4 external interrupt request signals
- Internal coprocessor for system management
- Internal 32-bit Linear Feedback Shift Register (LFSR)
- IO port of system (16x in, 16x out)
- Described in behavioral VHDL - no instantiated hardware primitives
- Configurable I-cache and D-cache as well as D-cache coherency strategy
- 32-bit pipelined Wishbone bus interface
- 85% device utilization on a Xilinx Spartan-3 XC3S400A
- Compatible with arm-elf assembler and WinARM tool chain

2.2.2 System Architecture

The performance of the core, the system is increased by providing with two cache units: an instruction cache and a data cache. Both data and instruction caches are fully associative and can store data from/to any MEM/IO location. The number of cache pages, page size and its coherency strategies can be configured for both the cache independently. Together with a bus unit, connects the cache via a Wishbone interface to the rest of the system. Figure 2.1 shows the architecture of the STORM Core.

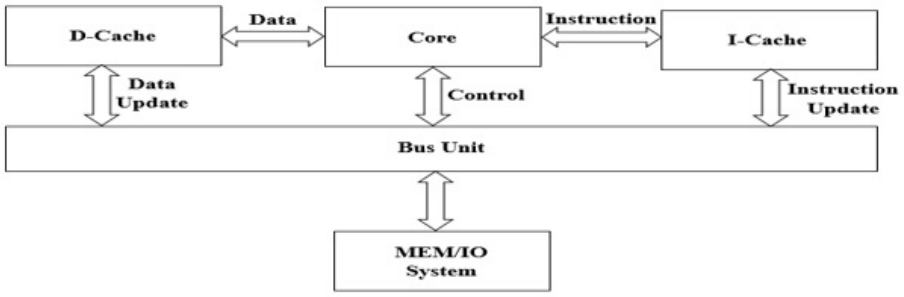


Figure 2.1: STORM Core Architecture [11]

2.2.3 ARM Vs STORM Core

The STORM Core uses completely a new approach to create an ARM processor, there are few differences. The critical difference needs adaption of code while running programs on STORM Core. The noncritical difference does not affect the behavior of the processor which is ARM-compatible, therefor no adaption of code is not necessary.

Critical Differences

- Multiply-accumulate-long and multiply-long instructions are not implemented. Undefined instruction trap will trigger if such instructions are executed.

- If bit location 0 of Register Rn is '0', then Branch and exchange instruction (BX) will execute a normal jump to the address stored in Rn. Whereas bit location 0 of of Rn is '1', then undefined instruction trap will be triggered as the processor does not support short instruction format.
- The pre-fetch abort interrupt is termed as Instruction fetch Abort Interrupt (IAB).
- The data abort interrupt is termed as Data fetch Abort Interrupt (DAB).
- When executing shift operations with a given register shift offset, or performing MAC operations, there is no necessary to fetch additional data from the register file. So, if R15 is an operand, then its value will always be the address plus 8 bytes of the corresponding data processing operation.

Noncritical Differences

- No restrictions for the use of any register as operand/destination for all instructions
- While performing single memory access operations, shift value, which is applied to the offset register value, can also be specified by the content of the data register (not intended in ARM code).
- Data bits 8 and 9 of the machine status register are not reserved/undefined, used for disabling the IAB and DAB external interrupts (when set to '1').

2.2.4 Operating Modes

The STORM Core supports six different operation modes. Once the processor is reset, it starts operation always in System mode. The changing of different mode, then the corresponding mode code has to be written to the lowest 5 bit of the CMSR. This is not possible with unprivileged mode and only possible when the processor is in privileged mode.

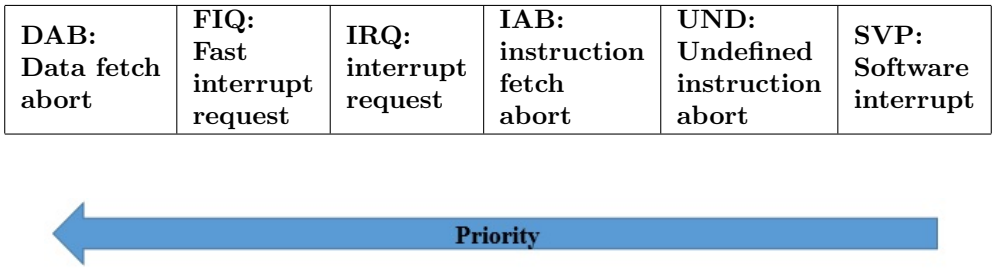


Figure 2.2: Operating modes and its priorities (priority increasing from right to left) [11]

User mode is the execution state, and is used for executing most application programs. System mode is same as the user mode for operation system, but privileged. Supervisor mode used for the protection of the operating system. When a data or instruction Pre-fetch Abort is occurred, then the system enters the Abort mode. General-purpose interrupts are handled by IRQ mode where data transfer or channel process are handled by FIQ mode. The Table 2.2 has the list of the operating modes of the STORM Core.

Table 2.2: Operating Modes [11]

Mode	Interrupt base address	Mode code
User, USR	-	"10000"
System, SYS	0x00000000	"11111"
Undefined Instruction, UND	0x00000004	"11011"
Supervisor, SVP	0x00000008	"10011"
(Instruction) Abort, ABT (IAB)	0x0000000C	"10111"
(Data) Abort, ABT (DAB)	0x00000010	"10111"
reserved	0x00000014	-
Interrupt Request, IRQ	0x00000018	"10010"
Fast Interrupt Request, FIQ	0x0000001C	"10001"

2.2.5 Registers

Each operation mode has the Current Machine Status Register (CMSR) and a Saved Machine Status Register (SMSR), a unique register set, including data registers implying a Link Register (LR, always R14) and the Program Counter (PC, always R15).

Table 2.3: Register for different Modes [11]

Mode	Accessible data registers	Accessible machine registers
USR	R0, . . . , R14	PC, CMSR
SYS	R0, . . . , R14	PC, CMSR, SMSR_SYS
FIQ	R0, . . . , R07, R08_FIQ,. . . , R14_FIQ	PC, CMSR, SMSR_FIQ
IRQ	R0, . . . , R12, R13_FIQ, R14_FIQ	PC, CMSR, SMSR_IRQ
SVP	R0, . . . , R12, R13_SVP, R14_SVP	PC, CMSR, SMSR_SVP
ABT	R0, . . . , R12, R13_ABT, R14_ABT	PC, CMSR, SMSR_ABT
UND	R0, . . . , R12, R13_UND, R14_UND	PC, CMSR, SMSR_UND

- System mode has a unique saved machine status registers, but User mode (USR) and System mode (SYS) share the same data registers
- System mode is one of the privileged mode
- Writing to R15 Program Counter (PC) which results in a jump operation to the written address
- While reading from R15, then the result is the PC value (address) of the corresponding operation, which is read from R15 plus 8 bytes
- R14 of each mode as the corresponding Link Register value which is used to store the jump-back address
- Register R13 of every mode is used as Stack Pointer

2.2.6 Machine Status Register

In a 32 bit ARM processor, the Program Counter and Processor Status Register both share the register R15. The Processors status register is categorized into CPSR and SPSR. CPSR is the same for all the processor modes whereas SPSR is specific to each mode with an exception that User and System modes does not have a SPSR. SPSR can be altered for the mode currently in, which is

done by entering the mode related to SPSR need to be updated. SREG_O_FLAG, SREG_C_FLAG, SREG_Z_FLAG, SREG_N_FLAG allows to alter the flag bits without affecting the control bits. In User mode, can only alter the condition flags as the control bits of CPSR are protected. And also specifying the R15 as a source or destination register is not possible.

Table 2.4: Current Machine Status Register and its Values, Function [11]

CMSR bit #	Name	Default	Function
0 ... 4	SREG_MODE_x	11111	Mode register, SYS after reset
6	SREG_FIQ_DIS	1	Fast interrupt request disable
7	SREG_IRQ_DIS	1	Interrupt request disable
8*	SREG_DAB_DIS	1	Data fetch abort disable
9*	SREG_IAB_DIS	1	Instruction fetch abort disable
28	SREG_O_FLAG	0	Overflow flag
29	SREG_C_FLAG	0	Carry flag
30	SREG_Z_FLAG	0	Zero flag
31	SREG_N_FLAG	0	Negative flag

In ARM processors, these bits the corresponding interrupts are always enabled but in the STORM Core these bits are reserved. So, these functionality are not compatible to ARM processor.

2.2.7 Data Flow

STORM Processor has 8 stage pipeline. Single cycle execution of instructions with no stalls. The STORM pipeline are functional description is given in the Table 2.5.

Table 2.5: Pipeline Stage and Functional description of STORM Core [11]

Stage	Functional Description
Instruction Access (IA) (program counter)	Instruction cycle starts with the new value of the program counter, which is old value + 4, since every instruction is 32 bit wide.
Instruction Fetch (IF) (I-cache access)	The instruction cache outputs the requested data and accepts the instruction request. When the requested cache line is missing, the required data set gets updated in the new cache page.
Instruction Decode (ID)	In the next cycle, loading the instruction into the instruction register and the instruction decoder will decode the instruction and applies opcode into internal control signals.
Operand Fetch (OF)	The control information from the decoder loads the needed registers from the register bank. If there are any data conflicts, the forwarding system takes action of the cycle to fetch operand.
Multiplication/Shift (MS)	A shift or multiplication of the operands can be applied in this stage of the pipeline.

Stage	Functional Description
Execution (EX)	The ALU operations take place in this stage of the pipeline. Also, the values from the coprocessors or the machine status registers can be loaded and also the condition check is done. All instructions even with a not fulfilled condition code are valid, if they were not marked as invalid by the branch control or the instruction arbiter.
Memory Access (MA)	This stage of the pipeline can update the machine status registers and also performs the memory access, the coprocessor registers and the PC. All needed control signals and data address are sent to the Data Cache. Also write-data is brought to the data memory interface and gets aligned if necessary.
Write Back (WB)	The final stage of the pipeline is the data write back stage. From the D-cache, the data is read into this stage, where it gets aligned which depends on the address offset and quantity of data read. Data from the WB stage - either the stage output data of the previous stage or the read memory data – is written directly during the next rising clock edge to the destination register. The data flow resumes in the operand fetch stage of the pipeline.

2.2.8 Cache Access

If the data entry request is not present in the cache memory, then from the memory/IO system a new cache page will be downloaded. This operation takes several cycles, depending on the speed of the accessed device (e.g. memory), the cache's page size, and the speed of the bus system.

When the access time of the device is longer than the maximum value, which is specified by the system coprocessor and IAB interrupt is taken. During the data fetch, DAB interrupt is taken. Maximum value for maximum bus cycle length = x"FFFF".

When updating (all cache entries from the memory/IO system are made invalid) and flushing (memory/IO system copies all the pages of cache) the cache manually can be done by the system control coprocessor. The strategy of page replacement is "least used".

MEM / IO → Cache coherency

The STORM Core that can access the memory system and if the user want other device to access the memory, then the user has to take care. The cache contains the recent data from the memory system. By using an external interrupt (IRQ), other master devices can show that data in the memory system are changed.

Communication devices (Universal Asynchronous Receiver/Tranmitter (UART), Serial Peripheral Interface (SPI), ...), should not be cached, because this leads to incoherent data. Use of the IO area definition generics defines the address space, and where IO devices are located. Nevertheless, IO devices can be cached to increase the data transport speed for streaming devices.

Cache → MEM / IO coherency

During "Write-Thru" coherency strategy, any alteration of a cache entry (only within the d-cache) leads to a write back of the corresponding complete cache page to the memory/IO system. This triggers the operation on write-access bus cycles and might cause problems for IO devices (e.g. UART). To avoid this problem, mapping the IO devices to a specific address area.

An altered cache page is only written back to the memory/IO system by disabling the "Write-Thru" strategy in the system control coprocessor introduces the standard coherency strategy, when it is replaced by the bus unit.

2.3 ZAP Processor [12]

ZAP is a 32-bit RISC processor and synthesizable open source core capable of executing ARMv4T binaries at both the supervisor and user level. The 10-stage pipeline that allows the processor to reach reasonable operating frequencies. The processor supports standard memory management and I/D cache that may be controlled using coprocessor. Both the cache and Translation Lookaside Buffer (TLB) are direct mapped. Branch memory, Caches and TLBs are implemented as generic fully synchronous RAMs that can efficiently map to native FPGA block RAM to save FPGA resources. The memory bus is fully compliant with Wishbone B3. To improve the performance of the processor a store buffer is implemented.

2.3.1 Features

- Fully synthesizable core
- Improved performance by store buffer
- Execute ARMv4T code at both the supervisor and user level
- Burst access supported by Cache unit. Wishbone B3 compatible interface
- Bypass network in pipeline to resolve dependencies. 10-stage pipeline
- Two write ports for the register file to allow Load Register (LDR)/Store Register (STR) with writeback to be executed as a single instruction
- 2-bit state for each branch and branch prediction is supported
- Split Data and Instruction writeback cache
- Split Data and Instruction MMUs
- To simplify data abort handling a base restored abort model is used

2.3.2 CPU Clock and Reset

ZAP uses a core clock to drive the entire design. The clock must be supplied to the port CLK. ZAP expects a rising edge synchronous RESET (active high) to be applied i.e., the reset signal changes only during the rising edges of clock. The external synchronizes of reset to the core clock before being applied to the processor.

The Zap processor is connected with Data and Instruction Cache for fetching the data and instruction for the processing within the processor. The processed data is stored in the Data cache. Both the Data and Instruction cache are connected to the wishbone bus interface through the MMU. The Figure 2.3 show the block diagram of the ZAP processor.

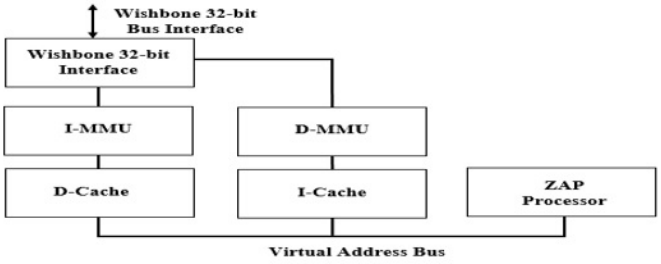


Figure 2.3: Block Diagram of Zap Processor [12]

2.3.3 Pipeline Overview

ZAP processor uses a 10 stage pipeline. The pipeline stalls are minimized by using an extensive bypass network. Forwarding the data from memory a cycle early with the help of a load accelerator. Most instructions can be executed in a single clock cycle expect non-multiply instructions, multiplies or non-trivial shifts are exceptions.

Table 2.6: Pipeline Stage and Functional description of ZAP Processor [12]

Stage	Description
Fetch	Data from I-cache is locked into instruction register. And the branch predictor memory is also read out during this stage of the pipeline execution.
FIFO	Clocks a shallow buffer with instructions and corresponding PC+8 values.
Thumb Decoder	Conversion of 16-bit thumb instructions into 32-bit ARM instructions. Pipeline to change to the new predicted target as instructions predicted are taken.
Pre-decode	Handles coprocessor Load Memory (LDM)/Store Memory (STM), instructions and SWAP.
Decode	Decodes ARM instructions.
Issue	Bypass network used to extract the Operand values. When data is not available from the bypass network, then the register file is read.
Shift	This pipeline stage performs multiply and shift operations. Single level bypass network used to optimize certain dependencies. Multiple clock cycles were utilized by multiplication operations.
Execute	The Execute stage contains the ALU. The ALU operations are executed is single cycle.
Memory	Data from the data cache fetched into the pipeline. Read data is aligned, if necessary.
Writeback	Data is written into the register file. Two writes were performed per clock cycle although the only use for the feature is to accelerate Load Register (LDR) performance in the current implementation.

As per the task requirement literature survey has been done for the available open cores of the ARM holdings. The chapter 3 will deal with the selection of the core and implementation of the core on the given FPGA Board.

Chapter 3

Implementation

3.1 Selection of Core

After analyzing the given task Requirements, the author made the list of open ARM softcores available in the internet and which are listed in the Table 3.1.

Table 3.1: List of available Softcores, its Design Language and Look-Up Table (LUT)

Core	Language	LUT
STORM [13]	VHDL	6006
ARM4U [14]		1419
16 Bit ARM Thumb [1]		830
ZAP [12]	VERILOG	Not proceeded
AMBER [15]		
ARM7 [16]		

As per the task Requirement, the author has to select a core in VHDL language and which is need to be implemented in the given FPGA board. The following are the factors used to select the core to implement:

- Core should be in VHDL language
- The design should replicate the functions of ARM Processor
- ARMv2 instruction architecture

Even though all the three cores **ARM4U, 16 Bit ARM** and **STORM** core can be implemented in the given FPGA Board.

ARM4U does not include the Abort and interrupt mode and it also does not support for compressor and related instruction. Register R15 is a Program counter in ARM architecture but in **ARM4U** R15 can be used as an input for every instruction.

16 bit ARM Thumb processor is a 16 bit processor and does not include the Branch prediction unit, which makes the target of the branch is not known until end of Execute unit in the pipeline stages. The Thumb core requires 70% of space and uses 40% more instructions than ARM core. But, with 32-bit memory, ARM core is 40% faster than the Thumb core.

As the ARM core is faster than Thumb core for 32 bit memory, by analyzing the **STORM** core and its features in section 2.2.1. The **STORM** Core design is selected to implement in the FPGA Board.

3.2 STORM Softcore – Existing Design

The existing STORM softcore was taken from the **https://opencores.org/projects/ storm_core** website which is available to anyone to download with no charges.

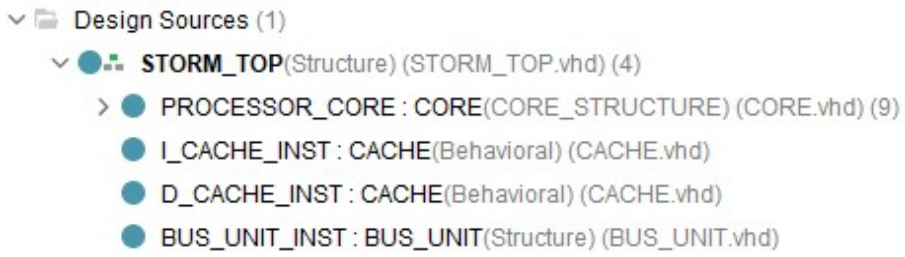


Figure 3.1: Existing Design source code [13]

The Figure 3.1 shows the design source of the STORM core and for checking the correct operation of the core a test bench is designed with an internal memory and IO Controller. Internal Memory has the assembly instruction with corresponding Hex value which is sent using wishbone bus to core. The processed output of the core sent back to IO controller through wishbone bus. The design source of STORM Test bench is shown in the Figure 3.2.

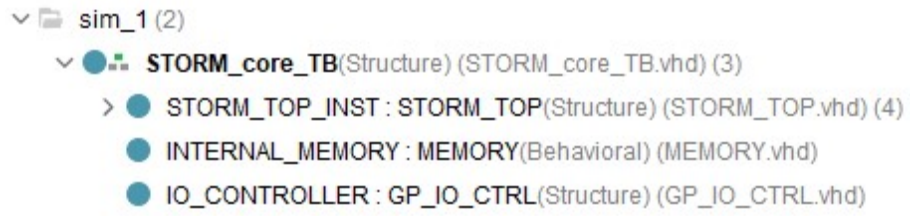


Figure 3.2: Existing Test bench Design [13]

Run Criteria

The existing STORM softcore was simulated on Vivado tool with a simple VHDL Test bench file. The Test bench file drives the clock and reset signal to the IP core to determine it's working condition.

STORM Softcore Development

The main task can be divided into three sub-tasks:

1. Removing the internal memory and IO controller design from the test bench design and adding the same to main STORM core
2. Controlling the clock and reset signals of the STORM IP softcore with the push button switches START, STOP, STEP and RESET so that the program running on STORM core could be controlled as per the wish of the user
3. Fetch the content of the IO Controller's output to be displayed on LCD. For this, created a LCD controller VHDL design file which will interact with the STORM Softcore to fetch the required data for user

3.3 STORM Softcore - New Design

The new implementation merged internal memory and IO controller with the core as shown in Figure 3.3.

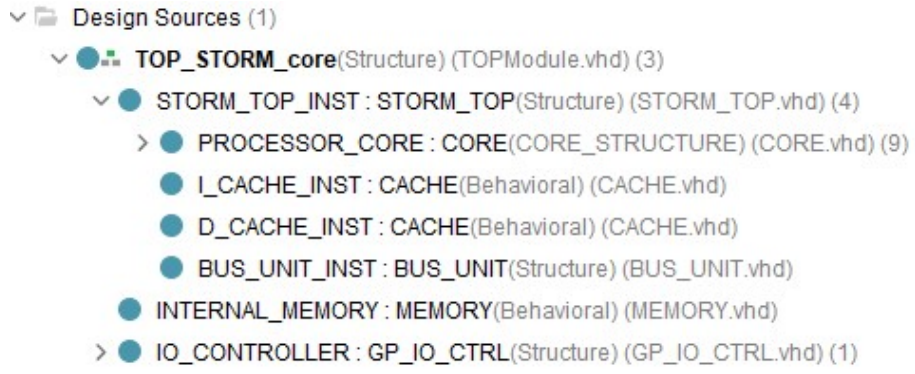


Figure 3.3: Modified Design Sources

Then LCD demo and LCD controller VHDL design are created and RESET has changed from ACTIVE HIGH to ACTIVE LOW.

The Figure 3.4 shows the block diagram of the modified STORM Core. Core is the top entity of the processor system. It includes the processing core, data and instruction cache and a Wishbone compatible bus interface.

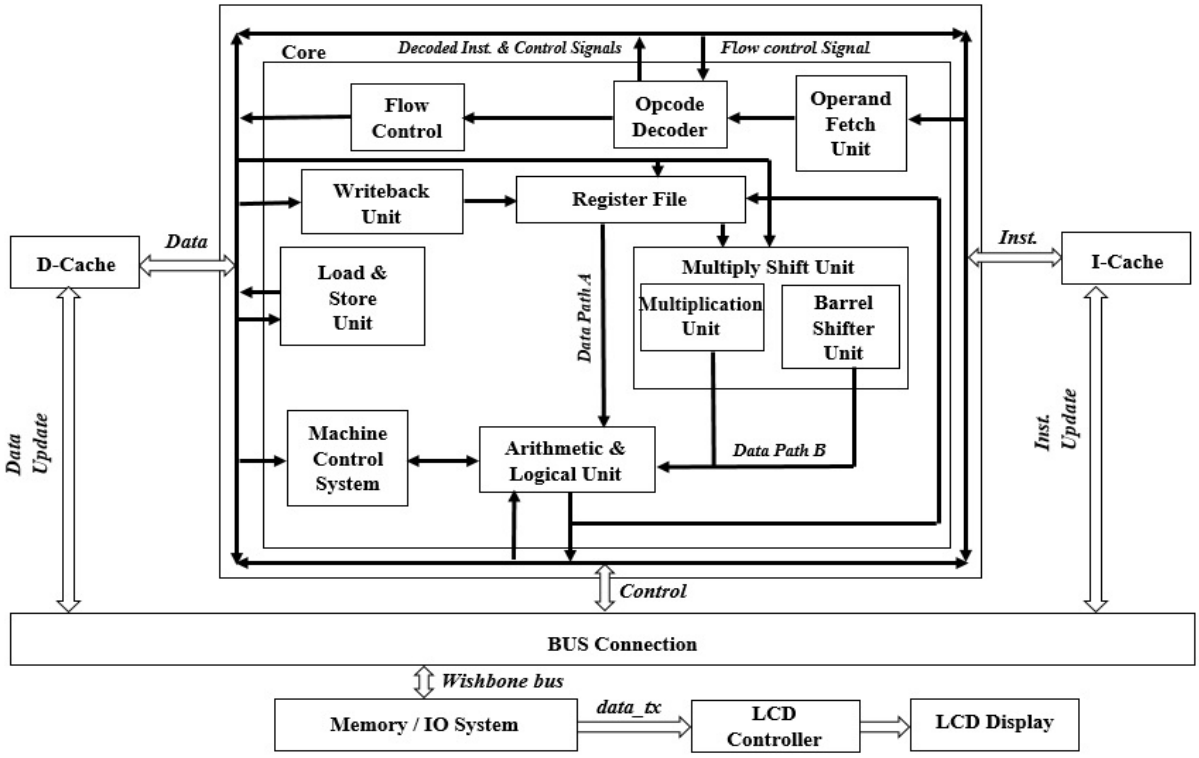


Figure 3.4: Block Diagram of modified STORM Softcore

Wishbone bus interface represented by the bus unit. Instruction and Data fetch to/from the cache memories are organized by this unit. It can operate at different clock compared to the core.

There are instruction cache (IC) and data cache (DC). These caches are fully associative and mapped to the dedicated memory blocks.

Operand Fetch performs fetching of the operand for all the 3 operand-slots. It loads immediate values from the instruction decoder and register values from the register file. Also detects the conflict in pipeline data and forward the data to the next stage. Opcode decoder used to decodes the ARM 32-bit opcode into processor control signals.

The flow control generates the control signals for each module and every stage within the pipeline. The decoded instruction data is brought to this unit where it triggers all internal operations. Furthermore the the cycle operation, instruction operation, which solves temporal pipeline conflicts, the branch operations and the condition check system.

The Machine Control System (MCS) holds the machine control circuits, which include the current and saved machine status register, program counter, and also the interrupt handler, the context change system and the branch system.

Register file consists of 32 registers, out of which 16 are accessible at any time, depends on the operating mode. Mapping the registers to three memory blocks that creates three data ports (read) for efficient use the hardware.

The ALU performs the arithmetic and logical operations. It also handles data access to/from the system coprocessor and to/from the machine control registers. The multiplication unit used to calculate 32x32 bit operation and lower 32 bits of the output is given to the data path B of the ALU. The shifting of the data in data path B of the ALU is performed by the barrel shifter. The shift value can be either from the register or opcode value.

The address and control signals are generated by load-store unit to access the data cache port. The data written back to the register file by the write back unit. It also read data from the data cache interface. The LCD controller is briefly discussed in the section 3.5.

3.4 System Design

For the Softcore enhancement, the existing design is modified. The modified design includes the implementation of the push button to control the operation of the core. The push buttons are used to reset and give clock pulse to the core. Also the modified design includes the LCD controller to display the output data of the Softcore.

The Top level system design of Debug Board components communicating with FPGA Softcore is explained with the help of Figure 3.5.

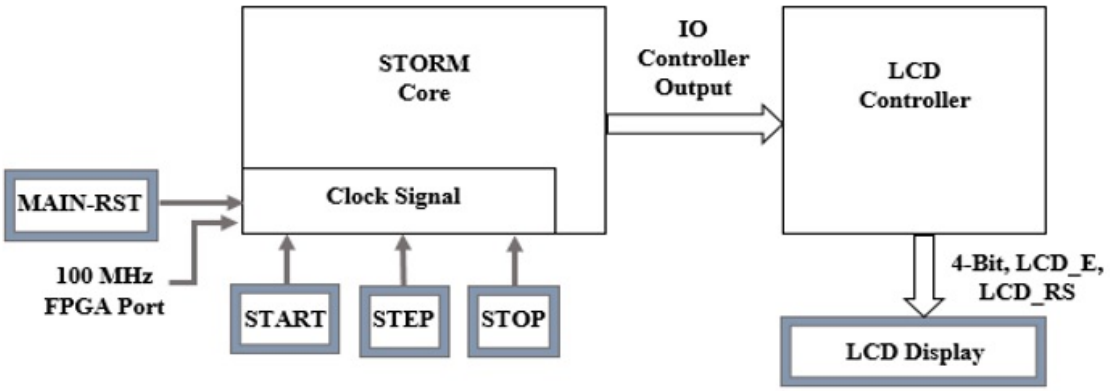


Figure 3.5: System Design

The Clock Control Logic gets the command from four push button keys: "Start", "Stop", "Step" and "Reset".

Operation of Push Button Switches

- ***RESET*** – Reset the Softcore. It sends Active low Reset signal to the STORM core running on the FPGA
- ***START*** – Starts the program execution. It sends continuous clock pulses to the Clk input of the Softcore
- ***STOP*** – Stops the execution of program. The supply of the clock signal to the Softcore is stopped
- ***STEP*** – Sends one clock pulse to the Softcore for one press

Depending on these push button key inputs and 100 MHz clock input (MAIN_CLK), the actual Clock (CLK) input to the STORM Core is controlled. The required data from the STORM Softcore's IO Controller are passed on to the LCD Controller module.

3.5 LCD Controller

LCD Controller IP Core was taken from the **https://opencores.org/projects/16x2_lcd_ controller** website which is available to anyone to download with no charges.

The available IP Core modified to work for 20 x 4 LCD display instead of 16 x 2 LCD display. This LCD controller can operate at 50 MHZ which implemented in a way that by introducing the required delay in the design to match with operating frequency of 2 KHz LCD display from **Electronic Assembly - DOGM204A.**

The LCD controller has been designed into two modules:

1. ***LCD demo***
2. ***LCD Controller***

LCD demo design does the operation of dividing a 32 bit value of the IO controller output into 8 values of 4 bit each. Each 4 bit value are converted into ASCII values using BIT_ASCII function as shown in the below figure corresponding to the LCD display used.

```
function BIT_ASCII (
    data    : in std_logic_vector(3 downto 0))
    return std_logic_vector is
   variable out1 : std_logic_vector(7 downto 0);
  begin
   case data is
    when "0000" => out1:=X"30";
    when "0001" => out1:=X"31";
    when "0010" => out1:=X"32";
    when "0011" => out1:=X"33";
    when "0100" => out1:=X"34";
    when "0101" => out1:=X"35";
    when "0110" => out1:=X"36";
    when "0111" => out1:=X"37";
    when "1000" => out1:=X"38";
    when "1001" => out1:=X"39";
    when "1010" => out1:=X"41";
    when "1011" => out1:=X"42";
    when "1100" => out1:=X"43";
    when "1101" => out1:=X"44";
    when "1110" => out1:=X"45";
    when "1111" => out1:=X"46";
    when others => out1:=X"2D";
    end case;
    return out1;

  end function BIT_ASCII;
```

Figure 3.6: BIT to ASCII Conversion function

Likewise, **LCD demo** does for all the output values of the IO controller and assign it to the 4 line buffers.

LCD Controller design does the operation of the LCD Configuration as per the given order in **Appendix D**. Configured LCD will display the ASCII data from the line buffer LCD demo module. When all the 4 lines are displayed, then LCD Controller will clears the display to show the next set of values.

Line buffers are updated using counter in the LCD demo module such that it prevents of displaying same set of values again and again.

The output of the IO Controller is first 30 Fibonacci numbers in Hex values, each has the size of 32 bit.

LCD can display 20 x 4 = 80 characters, then display is cleared before displaying the next 80 characters. Hence, there are 9 pages in total including the starting page to display all the 30 Fibonacci numbers.

The Flowchart in Figure 3.7 shows the implementation of how the IO Controller's output data are displayed.

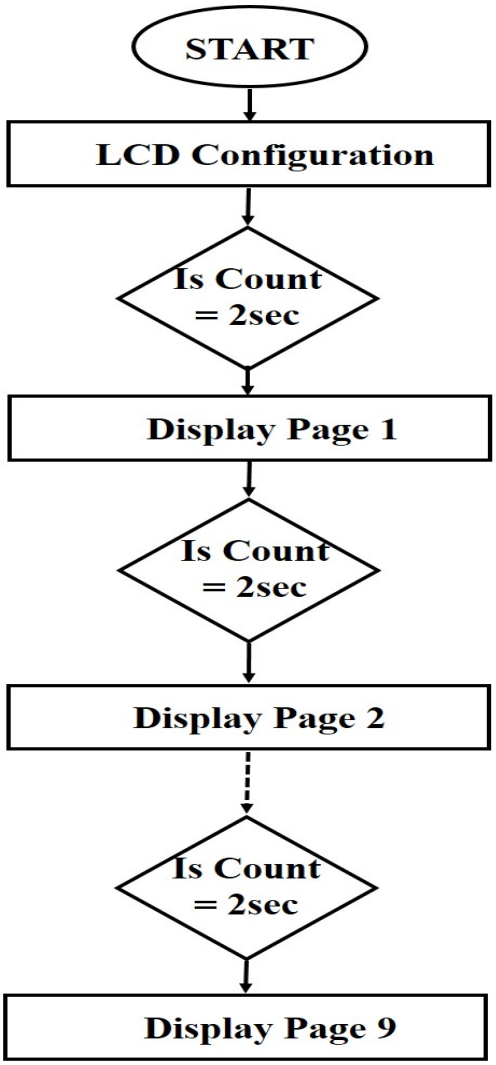


Figure 3.7: Flowchart – IO Controller Output Display

3.6 Resource Utilization Report

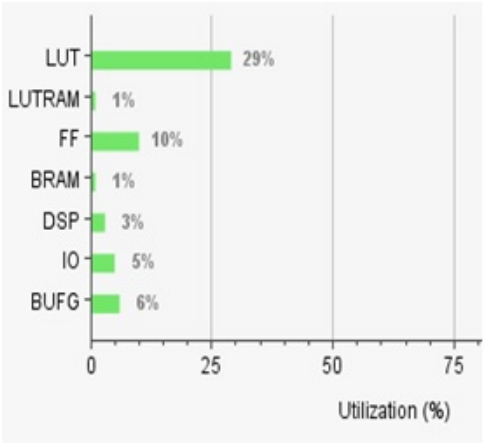
In the Xilinx-Vivado project FPGA device selected is XC7A35TCSG324-2.

As we can see from the Figure 3.8, IO resource utilization is just **5%**.

The low IO usage is due to direct fetching the output of the IO Controller to the LCD demo. Hence, there arises no need to have an output port to connect with the LCD controller.

This is done by adding the LCD Controller to the new added **TOP Module** of the STORM Core.

Look-Up Table (LUT) utilized is almost 1/4th of the total FPGA Resource which indicates that STORM Core fits in the given FPGA Board.



Resource	Utilization	Available	Utilization %
LUT	6006	20800	28.88
LUTRAM	136	9600	1.42
FF	4178	41600	10.04
BRAM	0.50	50	1.00
DSP	3	90	3.33
IO	11	210	5.24
BUFG	2	32	6.25

Figure 3.8: Resource Utilization Report

The resource utilization for the flip flops is 10% and all other resources utilization is less than 10% such as Block RAM is 1%, DSP is 3.33% and LUTRAM is 1.42%.

3.7 Timing Analysis

In the timing analysis it was found that the existing design would not run on 100 MHz clock frequency input.

By using **Create_Generated_Clock** command syntax and create a clock frequency of 50 MHz and the STORM core run successfully.

The Figure 3.9 shows the Timing analysis report of the implemented STORM Core on given FPGA Board.

Figure 3.9: Timing Analysis Report

From the above timing Report, the Worst Negative Slack, Worst Host Slack has the positive value which indicates the design runs without any timing violations.

Chapter 4

Results

4.1 Simulation Output

The new TOP Module is used to merge the internal memory, IO Controller to the main STORM Softcore. The Behavioral simulation is ran to ensure that the introduction of the new module does not affect the STORM Softcore process.

The Figure 4.1 shows the behavioral simulation output of the core which indicates the update of the Instruction Cache when a MISS state attained by the STORM Core.

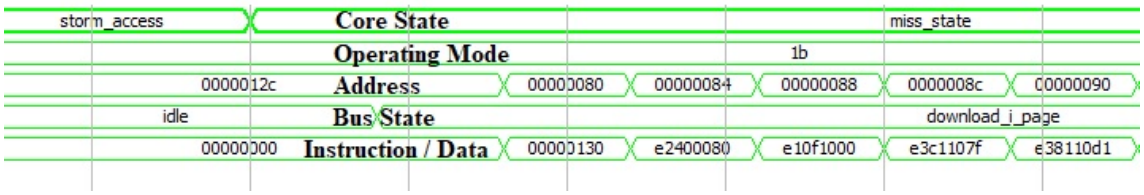


Figure 4.1: Simulation Output of Core

The main purpose of the STORM Softcore enhancement is to display the output on the LCD display to ensure that the core is working properly in the given FPGA Board. The LCD Display is controlled by the enable, Register select signal.

The Figure 4.2 shows the behavioral simulation output of the LCD Controller of the STORM Softcore.

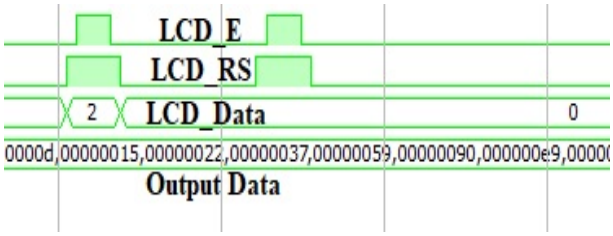


Figure 4.2: Simulation Output with LCD Controller

When the LCD enable and register select is high, then the LCD Controller will display the corresponding ASCII value of the output data.

4.2 Hardware Output

The STORM Softcore is synthesis and implemented successfully in the Xilinx-Vivado tool. As a next step Bit stream file must be generated with the correct settings as per the requirement.

The following constraint is added in order to generate the correct Bit stream file:

set_property CFGBVS Vcco [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]

The generated Bit stream file is then programmed into the given FPGA Board. The Figure 4.3 shows the LCD display embedded on the FPGA Board displays Page 9 of the IO Controller's output.

Figure 4.3: LCD Display – IO Controller Output

The output on the LCD display indicates the successful implementation of the STORM Softcore on the given FPGA Board.

4.3 Design Comparison

The existing STORM Core was implemented in the Xilinx FPGA Spartan 3 XC3S400A which has the design utilization of 85% as shown in the below Figure 4.4.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3255	3584	90%
Number of Slice Flip Flops	2250	7168	31%
Number of 4 input LUTs	6109	7168	85%
Number of bonded IOBs	152	221	68%
Number of BRAMs	8	16	50%
Number of MULT18X18s	3	16	18%
Number of GCLKs	1	8	12%

Figure 4.4: Design Utilization of existing STORM Core in Spartan 3 Board [13]

The existing STORM Core resource utilization of the Block RAM is 50% whereas usage of Block RAM in the enhanced design is only 1%. Similarly, Flip flops used by the existing core is 31% whereas only 10% used in the enhanced design.

The enhanced softcore of the Storm is implemented in the Xilinx FPGA Artix 7 XC7A35T which has the design utilization of only 29% which is briefly discussed in the section 3.6.

Chapter 5

Issues Faced and Solution

5.1 Fetching Output

The Figure 5.1 shows the simulation output of IO Controller STORM Core updates even when the address remains the same. Hence, it becomes not possible to fetch the output with address as the reference.

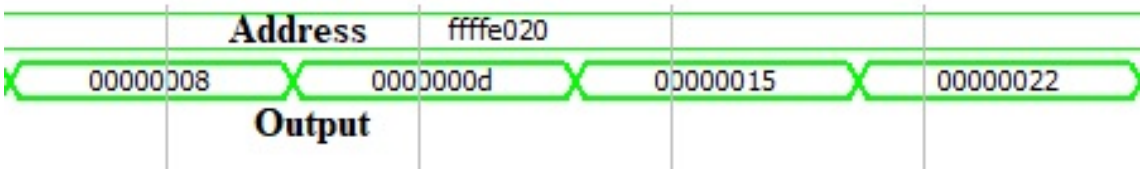


Figure 5.1: Problem faced to fetch output from STORM Core

The solution was implementing a process to store the value in the memory array, which is then sent to the LCD controller to display.

5.2 LCD Display Working frequency

The frequency of the main process which controls to STORM Core clock to the softcore was 50 MHZ and which is much higher for the LCD display operating at 2 KHz. When the Core clock frequency reduced to 2 KHz to match with operating frequency of LCD, then STORM core is not functioning at that low clock frequency.

The solution was using a LCD Controller IP Core which is implemented with required delays to match the LCD working frequency.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

All the subtasks mentioned in the task are achieved:

- Analyzed the existing ARM compatible cores available in the Internet
- Selected a core in VHDL language
- Code was extended and modified to test on given FPGA board
- Synthesized the code and checked the result of the simulation
- The enhancement of the existing STORM Softcore is implemented so that IO Controller output can be monitored on LCD screen

6.2 Future Work

At present: If the program to be run on the STORM core, internal memory and IO controller are added to the main core, then all the design files needs to be synthesized and a new Bitsream file (.bit) and output is generated on the given FPGA Board.

Future Work: As discussed in section 5.1, the design file has to be modified such that the address is updating with the corresponding output generation. With the modified design, it is possible to fetch the register and memory content with respect to the address.

Bibliography

[1] Design and Implementation of Thumb Processor on FPGA [Accessed on December 11, 2018]:
url: https://docs.google.com/file/d/0B2TRi3T8BtdkWmVBcngyd0ZBb19J
VzFDdm5XTUIzbVE4QW9z/edit

[2] ARM Introduction [Accessed on December 11, 2018]:
url: https://www.electronicshub.org/arm-introduction/

[3] Seth WIilliams, "BENCHMARKING ARM-BASED APPLICATION INTEGRATED SYSTEMS", Oklahoma State University, December 2011.

[4] Cortex-A8 [Accessed on December 11, 2018]:
url: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344k/DDI0344K_cortex_a8_r3p2_trm.pdf

[5] Cortex-A9 [Accessed on December 11, 2018]:
url: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388i/DDI0388I_cortex_a9_r4p1_trm.pdf

[6] Cortex-A15 [Accessed on December 11, 2018]:
url: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0438i/DDI0438I_cortex_a15_r4p0_trm.pdf

[7] Cortex-A53 [Accessed on December 11, 2018]:
url: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500j/DDI0500J_cortex_a53_trm.pdf

[8] Cortex-A57 [Accessed on December 11, 2018]:
url: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0488h/DDI0488H_cortex_a57
_mpcore_trm.pdf

[9] Cortex-A73 [Accessed on December 11, 2018]:
url: http://infocenter.arm.com/help/topic/com.arm.doc.100048_0002_04_en/cortex_a73
_trm_100048_0002_04_en.pdf

[10] ARM Infocenter [Accessed on December 11, 2018]:
url: http://infocenter.arm.com/help/index.jsp

[11] STORM CORE Datasheet [Accessed on December 10, 2018]:
url: https://opencores.org/websvn/filedetails?repname=storm_core&path=%2Fstorm_core
%2Ftrunk%2Fdoc%2FSTORM+CORE+datasheet.pdf

[12] ZAP Processor [Accessed on December 10, 2018]:
url: https://opencores.org/projects/zap

[13] STORM Core [Accessed on December 10, 2018]:
url: https://opencores.org/projects/storm_core

[14] ARM4U [Accessed on December 10, 2018]:
url: https://opencores.org/projects/arm4u

[15] AMBER[Accessed on December 10, 2018]:
url: https://opencores.org/projects/amber

[16] ARM7 [Accessed on December 10, 2018]:
url: https://github.com/chsasank/ARM7

Appendix A

Test Program

	Assembly Code	Equivalent Hex value
Reset:	MOV R0, #0	0000A0E3
	MOV R1, #1	0110A0E3
	MOV R2, #0	0020A0E3
	MOV R3, #100	6430A0E3
LOOP:	CMP R3, #220	DC0053E3
	BEQ NEXT	FEFFFF0A
	STR R0, [R3], #4	040083E4
	ADD R2, R0, R1	012080E0
	MOV R0, R1	0100A0E1
	MOV R1, R2	0210A0E1
NEXT:	.....	

Appendix B

STORM Core Directory Structure

Appendix C

Constraints

Signal Name and Pin

Signal Name	FPGA Pin
START	F3
STOP	F4
STEP	J3
MAIN_RST	D5
MAIN_CLK	P17
lcd_e	A1
lcd_rs	B3
lcd_db[7]	D3
lcd_db[6]	H1
lcd_db[5]	G1
lcd_db[4]	F1

Timing Constraints

create_clock -period 10.000 **-name** MAIN_CLK **-waveform** 0.000 5.000 [**get_ports** MAIN_CLK]

create_generated_clock -name Clk **-source** [**get_ports** MAIN_CLK] -divide_by 2 [**get_pins** Clk_reg/Q]

Appendix D

LCD Initialization

Command	RS	HEX	Remarks
Synchronize 1	0	33h	Make sure to switch to 8 bit data length
Synchronize 2	0	32h	Switch to 4 bit data length
Function Set	0	2Ah	4 bit data length extension Bit RE = 1; REV = 0
Extended Function Set	0	09h	4 line display
Entry mode set	0	06h	Bottom view
Bias setting	0	1Eh	4 bit data length extension Bit RE = 0; IS = 1
Function Set	0	29h	4 bit data length extension Bit RE = 0; IS = 1
Internal OSC	0	1Bh	4 bit data length extension Bit RE = 0; IS = 1
Follower control	0	6Eh	Divider on and set value
Power control	0	57h	Booster on and set contrast (DB1 = C5, DB0 = C4)
Contrast Set	0	72h	Set contrast (DB3 – DB0 = C3 – C0)
Function Set	0	28h	4 bit data length extension Bit RE = 0; IS = 0
Display on	0	0Fh	Display on, cursor on, blink on

Appendix E

LCD Output

Decimal	Hex Value	LCD Display Page
0	00000000	**PAGE 2**
1	00000001	
1	00000001	
2	00000002	
3	00000003	**PAGE 3**
5	00000005	
8	00000008	
13	0000000D	
21	00000015	**PAGE 4**
34	00000022	
55	00000037	
89	00000059	
144	00000090	**PAGE 5**
233	000000E9	
377	00000179	
610	00000262	
987	000003DB	**PAGE 6**
1597	0000063D	
2584	00000A18	
4181	00001055	
6765	00001A6D	**PAGE 7**
10946	00002AC2	
17711	0000452F	
28657	00006FF1	
46368	0000B520	**PAGE 8**
75025	00012511	
121393	000DA31	
196418	0002FF42	
317811	0004D973	**PAGE 9**
514229	0007D8B5	
832040	000CB228	