Post-silicon Validation of

Radiation Hardened Microprocessor and SRAM arrays

by

Sai Bharadwaj Medapuram

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2017 by the
Graduate Supervisory Committee:

Lawrence T. Clark, Chair
David R. Allee
John S. Brunhaver

ARIZONA STATE UNIVERSITY

August 2017

ABSTRACT

Digital systems are increasingly pervading in the everyday lives of humans. The security of these systems is a concern due to the sensitive data stored in them. The physically unclonable function (PUF) implemented on hardware provides a way to protect these systems. Static random-access memories (SRAMs) are designed and used as a strong PUF to generate random numbers unique to the manufactured integrated circuit (IC).

Digital systems are important to the technological improvements in space exploration. Space exploration requires radiation hardened microprocessors which minimize the functional disruptions in the presence of radiation. The design highly efficient radiation-hardened microprocessor for enabling spacecraft (HERMES) is a radiation-hardened microprocessor with performance comparable to the commercially available designs. These designs are manufactured using a foundry complementary metal-oxide semiconductor (CMOS) 55-nm triple-well process. This thesis presents the post silicon validation results of the HERMES and the PUF mode of SRAM across process corners.

Chapter 1 gives an overview of the blocks implemented on the test chip 25. It also talks about the pre-silicon functional verification methodology used for the test chip. Chapter 2 discusses about the post silicon testing setup of test chip 25 and the validation of the setup. Chapter 3 describes the architecture and the test bench of the HERMES along with its testing results. Chapter 4 discusses the test bench and the perl scripts used to test the SRAM along with its testing results. Chapter 5 gives a summary of the post-silicon validation results of the HERMES and the PUF mode of SRAM.

ACKNOWLEDGMENTS

First and fore-most, I would like to thank my parents for their unwavering support throughout my master's education.

I would like to sincerely thank my professor Dr. Lawrence Clark for giving me this post-silicon validation opportunity and providing guidance and support throughout my master's studies. I would like to extend my gratitude to Dr. David Allee and Dr. John Brunhaver for taking their time to serve as committee members. I am indebted to my colleague Divya Kiran Kadiyala for his invaluable contribution, discussion and support throughout the post silicon testing period. I would also like to thank my colleagues Chandrasekharan Ramamurthy, Ankita Dosi, Lovish Masand, Anudeep Reddy Gogulamudi, Parshant Rana, Manoj Vangala, and Vinay Vashistha for their invaluable contributions, discussions, and support during the thesis work. I must also thank the graduate advisors Toni, Sno, and Lynn for their help with all the administrative procedures. Finally, I would like to thank Fujitsu for funding this research.

TABLE OF CONTENTS

Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1. INTRODUCTION

1.1. SRAM PUF and Radiation Effects Overview

Digital systems are increasingly pervasive in the everyday lives of humans especially with the onset of the internet of things (IOT) era. The security of these systems is a concern due to the sensitive data stored in them. The physically unclonable function (PUF) provides a way to protect these systems. PUF hardware produces a challenge-response mapping based on the uncertainties of the transistor variations due to the manufacturing process [Edward07]. It provides a digital fingerprint to the device which helps in device authentication and identification.

The static random-access memory (SRAM) is a type of semiconductor memory that is primarily used in complementary metal-oxide semiconductor (CMOS) circuits due to its speed of operation. SRAMs are usually built using new minimum width and length transistors to obtain high density. This makes them highly sensitive to the transistor variations of the manufacturing process. Therefore, SRAMs can be used as a PUF to generate random numbers unique to the manufactured integrated circuit (IC). SRAMs can be designed to be used both for data storage and as PUF [Chellappa11]. The SRAM based PUF and data storage mode results across slow-slow (SS), fast-fast (FF) and typical-typical (TT) corners of test chip 25 (TC25) are discussed in chapter 4.

Radiation particles such as alpha particles, neutrons cause damage to the electronic circuits by striking their sensitive nodes. The scaling of the transistor sizes and supply voltages in the lower technology nodes have made the electronic circuits more

susceptible to radiation effects. Radiation hardening is the technique used in the design and fabrication of the electronic systems to minimize the damage due to the radiation.

The two major radiation effects on CMOS devices are total ionizing dose (TID) [Barn06] and single event effects (SEEs) [Mavis02]. The SEE is a localized effect, caused when a high-energy particle travelling through a semiconductor leaves an ionized track behind. This effect may cause a glitch in an output of a circuit, or a bit flip in a memory or register. TID effect deposits charge into CMOS devices resulting in the threshold voltage (V_{th}) shifts, noise, mobility and leakage[Barn06]. The post silicon testing results of the highly efficient radiation-hardened microprocessor for enabling spacecraft (HERMES) v2.55 is discussed in chapter 3. It is not TID hardened.

1.2. Test Chip 25 Overview

TC25 was designed to understand the behavior of SRAM arrays across different body bias voltages and process corners on Fujitsu's 55-nm triple-well bulk deeply depleted channel (DDC) CMOS process. To utilize the die area and the effort spent on the test chip effectively two more blocks are added to it. Hence, the TC25 is made up of three main blocks namely HERMES v2.65; SRAM block; radiation-hardened by design (RHBD) double data rate (DDR) based phase locked loop (PLL).

The top-level functional block diagram of the test chip contains the blocks HERMES v2.65, SRAM arrays and DDR PLL as shown in figure 1-1. The logic around these blocks include the exclusive-or (XOR) clock multiplier, divide-by-8 clock divider, 2:1 multiplexer (mux), 4:1 mux, level shifters, input/output (I/O) pads, analog pads and power pads. The input pads of TC25 are shared physically among the three blocks. The outputs from all the

three blocks are connected to the output pads using the 4:1 Mux. A two bit select signal (BLK_SEL) is used to control the 4:1 mux.

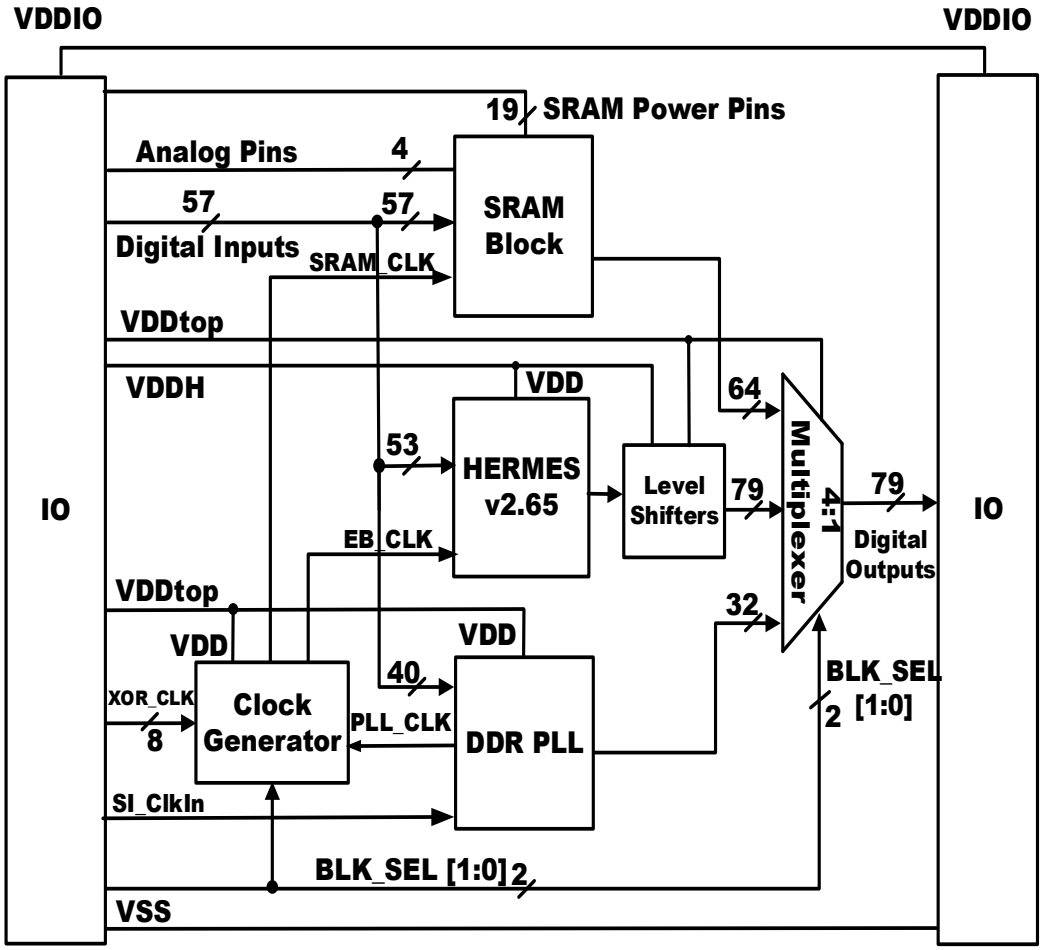

Figure 1-1 Top level functional block diagram of the test chip TC25

1.2.1. Clock Generation

Clock signal to the HERMES and the SRAM blocks can be generated using either the XOR clock multiplier or the DDR PLL. A one bit select signal, CLK_SEL is used to multiplex between the clocks produced from these two sources to the output clock (Figure

1-1). During the beam testing only the XOR clock multiplier is used to generate a high frequency clock, since the PLL is more vulnerable to radiation upsets [Gogula15].

The XOR clock multiplier takes eight clocks as inputs and generates an output clock by xoring every pair of input clocks until a single output clock is obtained (Figure 1-2 (a)). To generate an output clock with a multiplied-by-2 frequency and a duty cycle of 50%, two input clocks must have a phase shift of 90 degrees (180 degrees/2) between them and the remaining six clocks must be held constant. Similarly, to synthesize other frequency multiples of the output clock, the input clocks should be phase-shifted accordingly.

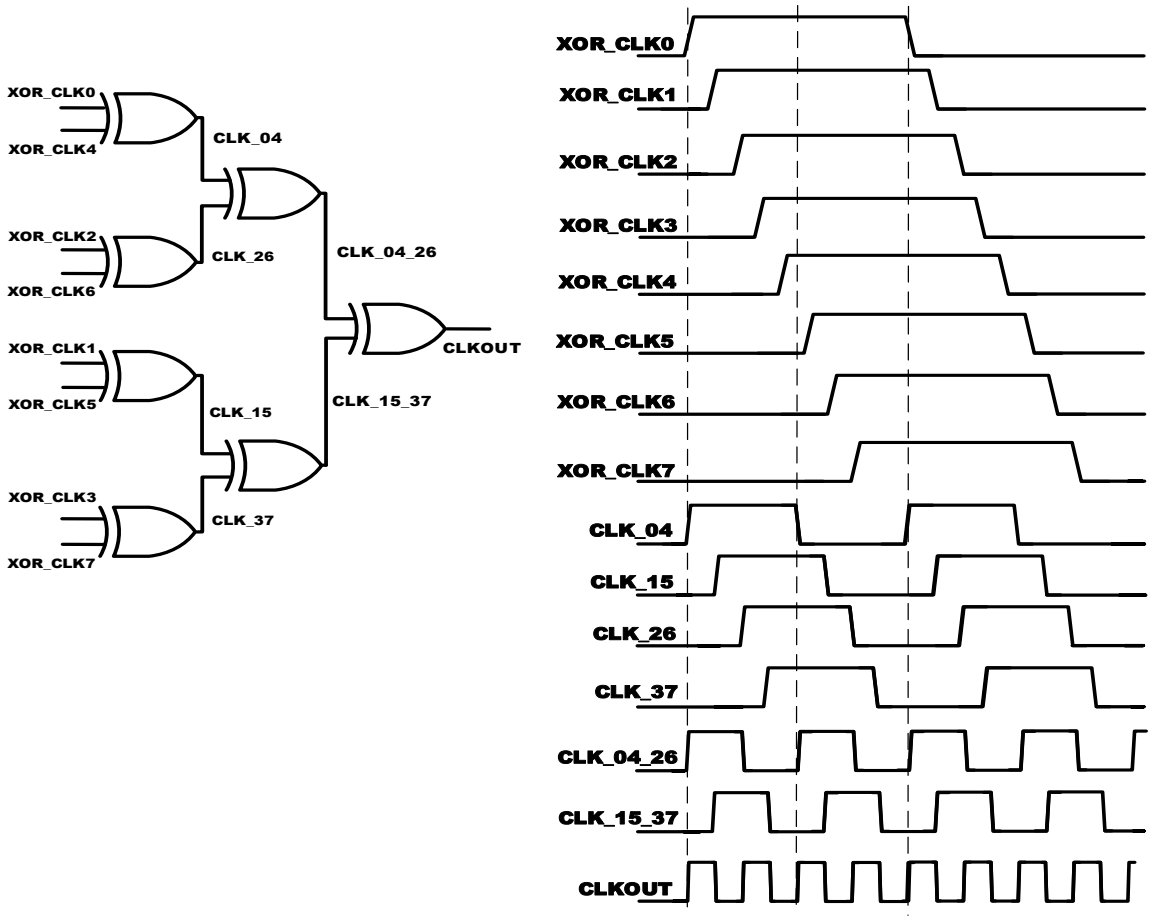

Figure 1-2 (a) XOR clock multiplier logic diagram (b) XOR clock multiplier waveform

The maximum frequency of the output clock that can be produced is 8x compared to the frequency of the input clocks. To generate this, the eight input clocks must be phase shifted by 22.5 degrees (180 degrees / 8) relative to each other (Figure 1-2 (b)). However, there is an inherent jitter on the input clocks relative to each other due to the FPGA (~160ps – 200ps) and the wire delay on the PCB Board. This limits the maximum frequency and duty cycle of the output clock that can be synthesized by the XOR clock multiplier.

The clock divider on the test chip uses the multiplexed clock from the 2:1 mux and generates a divide-by-8 clock as output (Figure 1-1). This is used to check the functionality of the clocks produced using the XOR clock multiplier and the DDR PLL on the manufactured test chip. Validating the functionality of the clocks is the first test done on any manufactured digital chip because clocks are crucial to any digital design. Hence the clock divider plays a key role in the post-silicon testing of the manufactured test chip.

1.2.2. HERMES Processor

The HERMES is a radiation hardened MIPS32 4Kc compliant embedded microprocessor [MIPS00]. The design of this block incorporates multiple radiation hardening techniques at the circuit, layout, micro-architecture and architecture levels. To achieve radiation hardening at the circuit design level, self-correcting triple-mode redundant (TMR) circuits are used to protect the key architectural state of the microprocessor [Gogula15] which includes the program counter (PC), write buffers, configuration registers and the bus interface. In addition to that, instruction execution pipeline, data and instruction cache subsystems, memory management unit (MMU) and register file (RF) are made dual-mode redundant (DMR). Radiation hardening at the layout

level is accomplished using a special automated place and route (APR) flow. This flow separates the nodes of the circuit by a certain distance to mitigate the soft errors induced due to multiple node charge collection (MNCC) [Hind11].

Radiation hardening at the micro-architectural and architectural levels of HERMES is realized using soft error detection and recovery. Soft errors are detected based on the mismatch between the two copies of DMR speculative instruction execution pipeline right at their commission to the architectural state [Vash15]. Soft error recovery is software controlled and achieved by the addition of new instructions.

1.2.3. SRAM Block

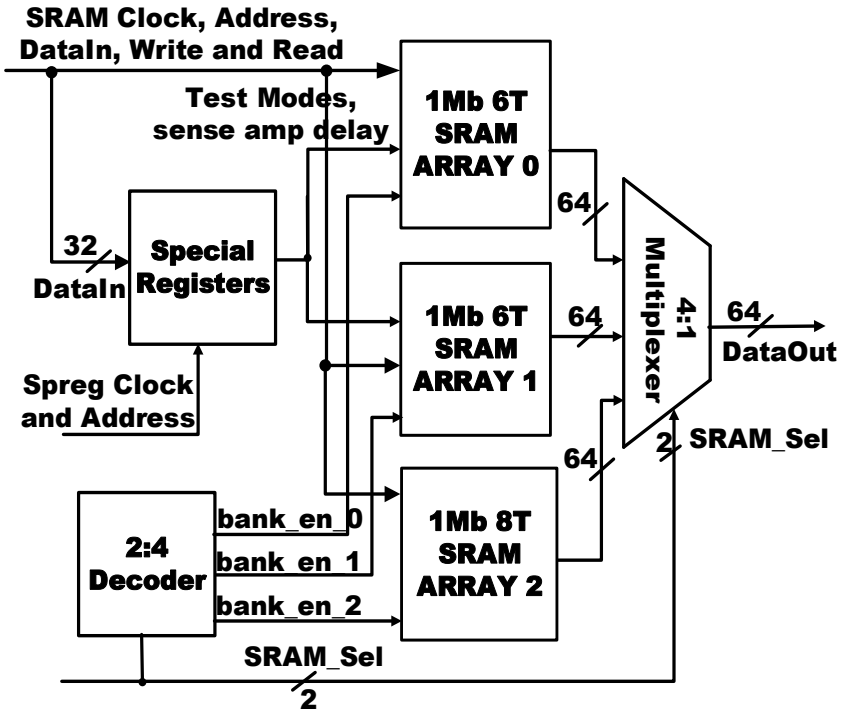

Figure 1-3 Top Level Functional Block diagram of the SRAM Block

The SRAM block consists of three 1mega bit (Mb) SRAM arrays and three 32-bit special registers (Figure 1-3). Out of these three arrays, two of them are made up of 6-

transistor (6T) bit cells and the third one is built using 8-transistor (8T) bit cells. The 6T bit cell based array (SRAM 6T array) shares a single port for read and write operations whereas the 8T bit cell based array (SRAM 8T array) has independent ports for read and write operations.

Three special registers are used to configure the SRAM 6T arrays in various test modes. The test modes supported by the SRAM 6T arrays are sense amplifier test (satest), direct access test (DAT) and PUF. Both satest and DAT modes are enabled simultaneously to measure the offset voltage of the sense amplifiers used in the SRAM arrays. PUF mode is used to generate a random number based on process variations in the manufactured SRAM array. Only the DAT mode is enabled to measure the currents on the transistors of the bit cells. Special registers are also used to configure the programmable delay value to turn off the sense amplifiers during a read operation.

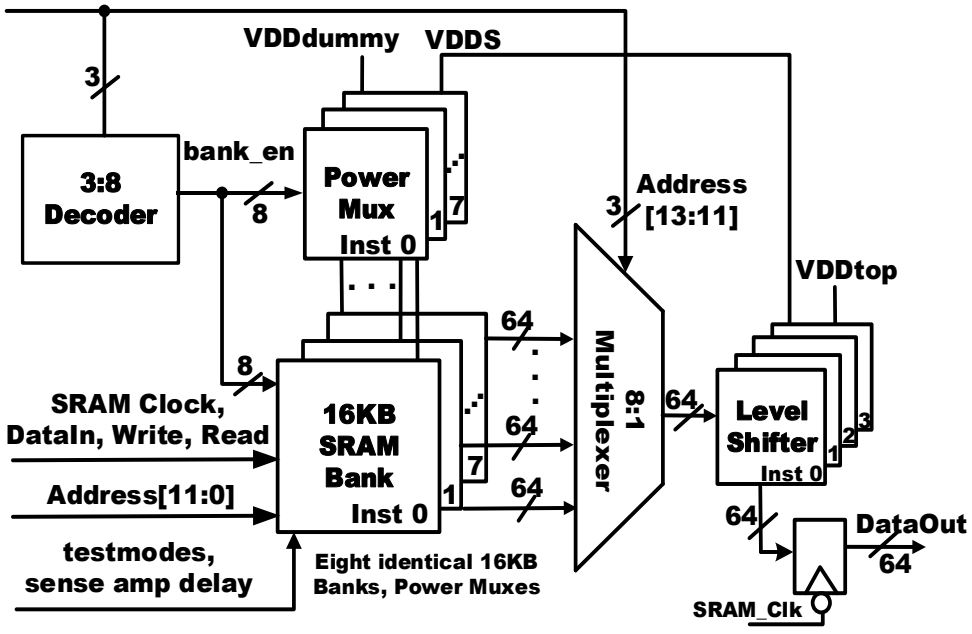

Figure 1-4 Structural block diagram of the 1Mb SRAM 6T array

The SRAM 6T array is built using eight 16 kbytes (kB) banks, eight 2:1 power muxes and four 16-bit level shifters. The eight 2:1 power muxes gate the power to the bank when it is not selected to perform any operations. A three-bit signal, banksel is used to select one of the eight banks. Each bank has a clock gater module in it. Depending on the bank selected, clock signal is gated to the rest of the banks using the clock gater modules in them. Clock gating and power muxing help in the low power operation of the SRAM 6T array.

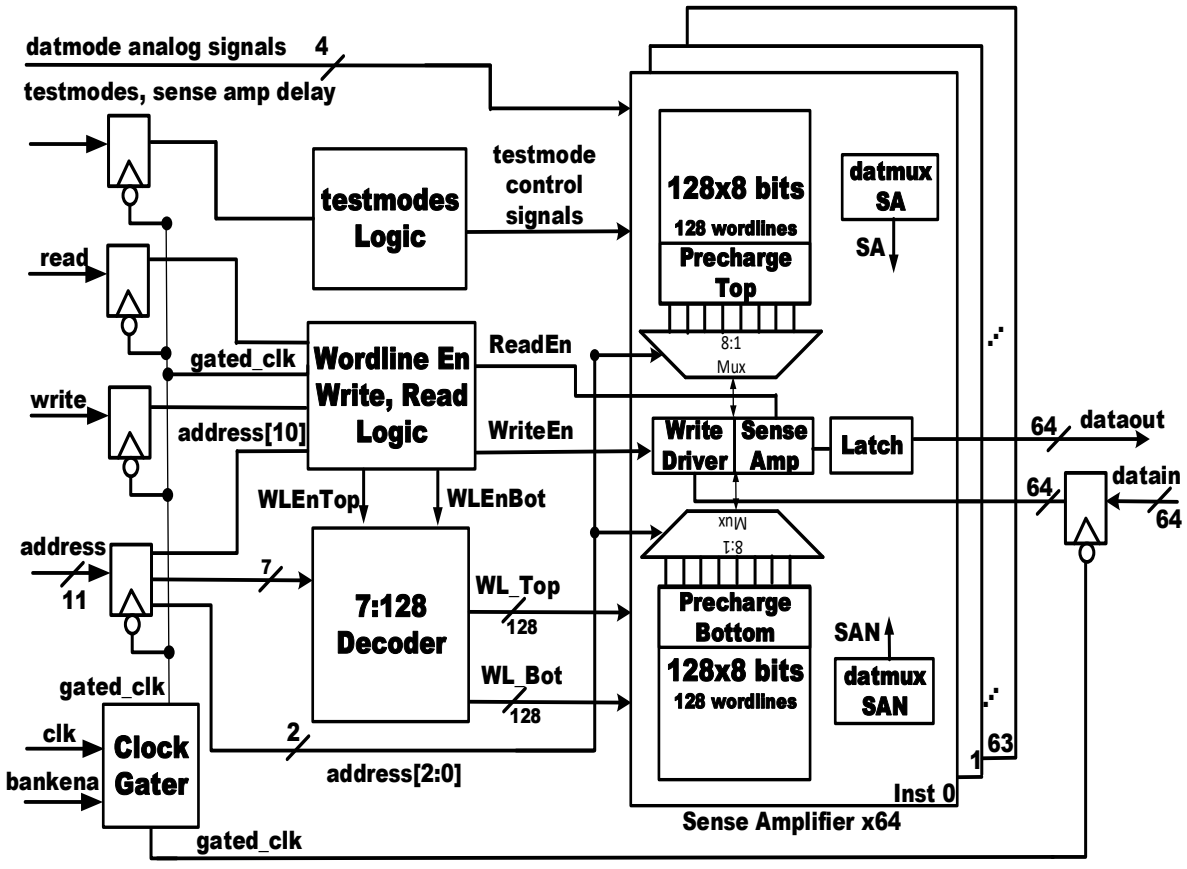

Figure 1-5 Functional block diagram of the 16kB bank of SRAM 6T array

The 16kB bank of the SRAM 6T array has 7:128 address decoder, write, read and word line logic, clock gater module, 64 column groups and test modes logic as shown in figure 1-5. Every column group in the bank is divided into top and bottom halves with the write driver, sense amplifier and latch shared between them. The top and the bottom halves of every column group both have a 8:1 mux for selecting one of the eight columns and receive 128 word lines each from the address decoder.

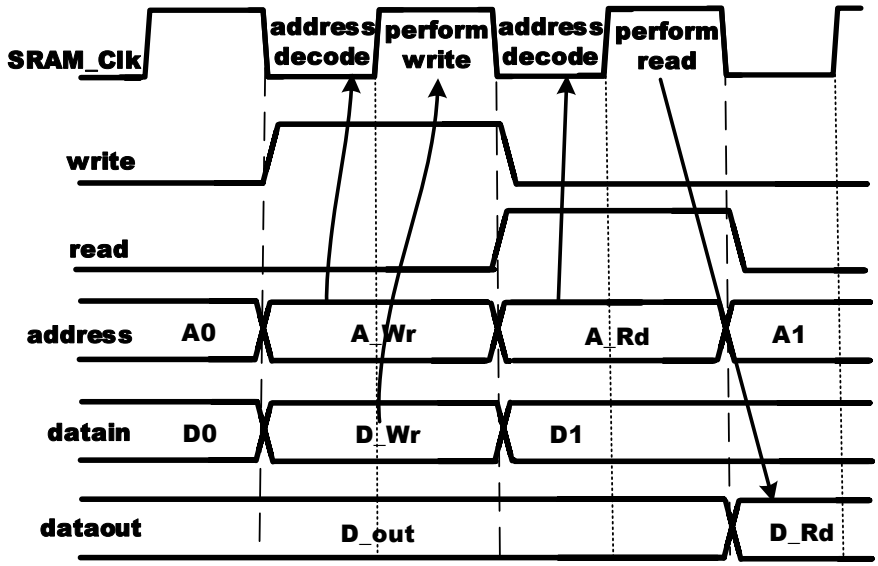

Figure 1-6 Timing diagram of the write and read operations to the 16kB bank

The write and read operations are performed on the bank selected as shown in figure 1-6. Depending on the write or read operation, the write or read signal is asserted after the negative edge of the clock. Then, address is decoded when the clock is low. Based on the address decoded, the corresponding word line is asserted when the clock goes high. For write operation, data present on datain is written and for read operation the data is read out from the appropriate bits in the column groups. The read data is captured and pipelined out using negative edge triggered flipflops.

1.2.4. DDR PLL

The DDR PLL block is a digital PLL which can generate clock frequencies for DDR based memory integrated circuits (IC) such as synchronous dynamic-random access memory (SDRAM). This block contains the configuration registers, custom built PLL and the clock divider logic as shown in figure 1-7. There are three configuration registers present in the block. Out of these three, two of them are used to set the delay through the coarse control and fine control units of the PLL. Depending on these settings, the PLL synthesizes different clock frequencies. The third register is used to set the division factor of the clock divider logic. The output of the clock divider logic is given as clock to the other blocks of the test chip depending on the value of the clock select signal.

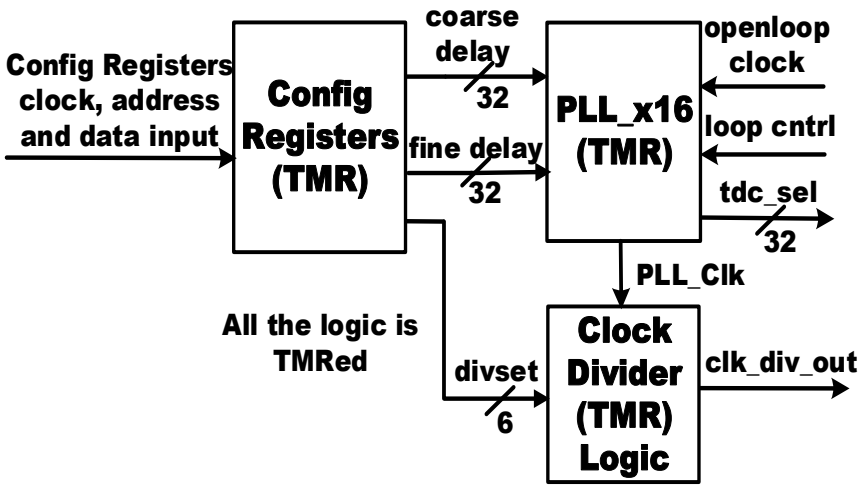

Figure 1-7 TC25 Functional block diagram of the DDR PLL

1.2.5. Pads of Test Chip 25

TC25 consists of 208 pads in total placed on a 28 mm x 28 mm quad flat package. They are made up of four pad types such as input, output, analog and power. All the digital I/O pads can handle LVCMOS signals with a maximum voltage of 3.3V. The pad

distribution among the pad types is as follows: 153 digital I/O pads, 4 analog pads and 51 power pads (Table 1-1).

Type of pads	HERMES	SRAM	DDR PLL	Others	Total number
Digital input	66	65	48	7	73
Digital output	79	64	32	1	80
Analog	-	4	-	-	4
Power	-	-	-	-	51
Total number	145	133	80	8	208

Table 1-1 TC25 Pad distribution

1.2.6. Power Supplies of Test Chip 25

TC25 uses ten distinct power pins to provide voltage and current to all the blocks (Table 1-2). The test chip is manufactured using a triple-well process, hence three different power pins are used for the three wells. The I/O pads and the top-level mux logic of the test chip use a power pin each. The SRAM block uses two power pins for the peripheral logic and power muxes. The bit cells of the SRAM 6T array use a power pin. All the three blocks share two power pins for supply voltage and ground.

Name	Description
vdda_array	power pin of the N-Well of SRAM 6T arrays
Vdda	power pin of the N-Well of all the blocks except SRAM 6T arrays
Vssa	power pin of the global P-Well
VDDtop	V_{DD} of the glue logic at the top level of TC25
VDDH	V_{DD} of the HERMES, SRAM and DDR PLL blocks
VDDS	V_{DD} of the peripheral logic of the selected SRAM array
VDDdummy	V_{DD} of the unselected SRAM arrays of the SRAM block
VDDarray	V_{DD} of the bit cells used in SRAM 6T arrays
VDDIO	power pin of the I/O block at the top level of TC25
VSS	ground pin for all the blocks

Table 1-2 TC25 Power Pins distribution

1.2.7. Die Photo of the Manufactured Test Chip 25

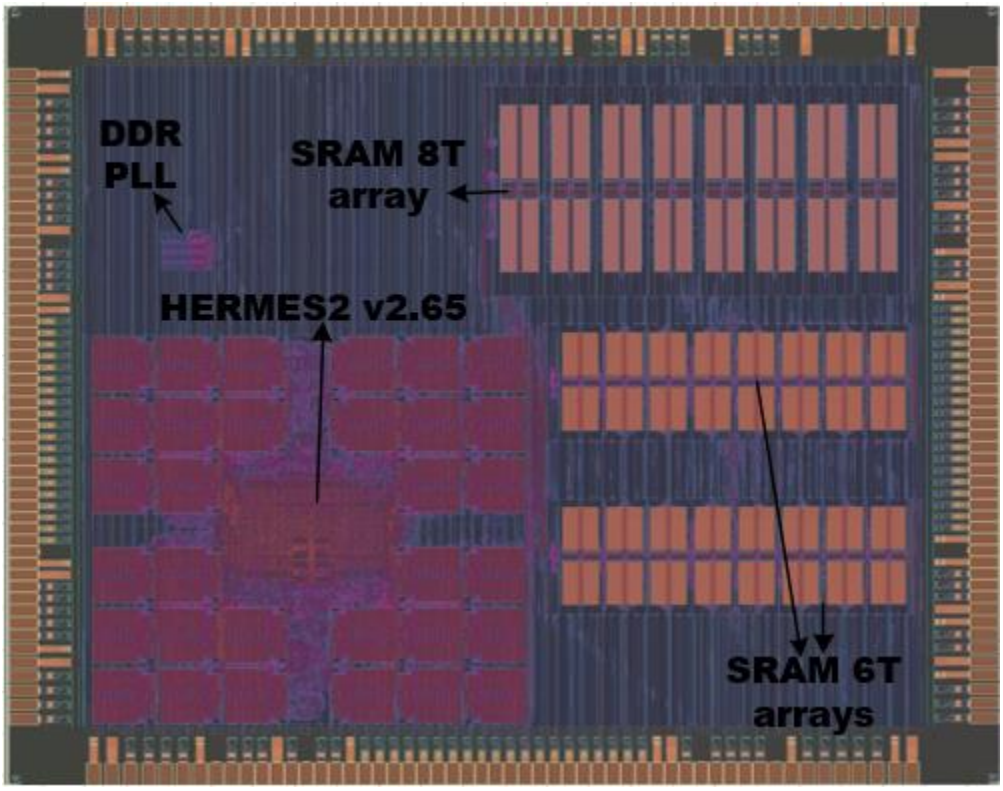

Figure 1-8 Die photo of metal M2 layer of the manufactured test chip TC25

1.3. Pre-Silicon Functional Verification of TC25

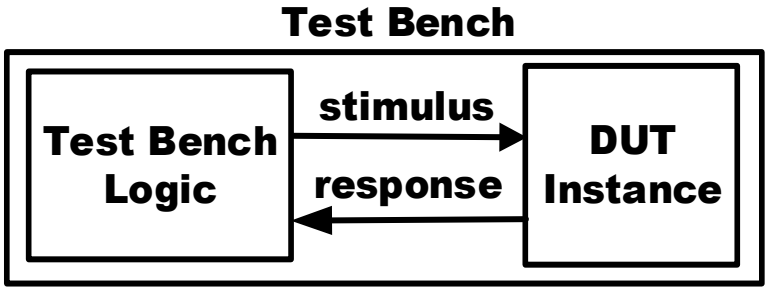

Figure 1-9 Block diagram of the generic test bench used for verification of a design

Digital designs are described using hardware description languages (HDL) like Verilog, VHDL, system verilog. A digital design needs to be tested for its functionality

before it is manufactured. The design being tested is referred to as the design under test (DUT). Test bench helps in verifying the correctness of the DUT. It instantiates the DUT, provides the stimulus to the DUT and records the response from the DUT (Figure 1-9). The register transfer level (RTL) and the test bench of the test chip, TC25 are written using a mix of Verilog and VHDL.

1.3.1. Simulation setup using Modelsim

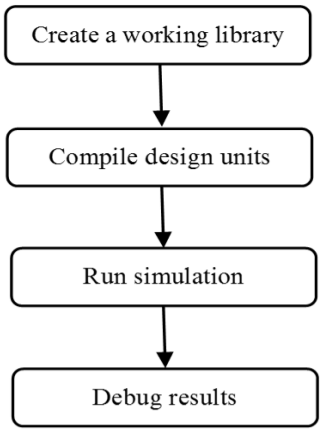

Figure 1.10 Simulation flow in ModelSim [Mentor04]

Modelsim is used to simulate the RTL and the test bench of TC25. To simulate a design in modelsim, four basic steps need to be followed (Figure 1-10). The first step is the creation of a working library into which all the HDL files are compiled. By default, modelsim compiles all the files into a destination library called “work”. The second step involves compilation of all the design files into the library specified earlier. The third step is to simulate the design by invoking the simulator on top-level module and running the simulations on it. The fourth step is to debug the results obtained if they don’t match the expectations.

1.3.2. Test Bench setup of TC25

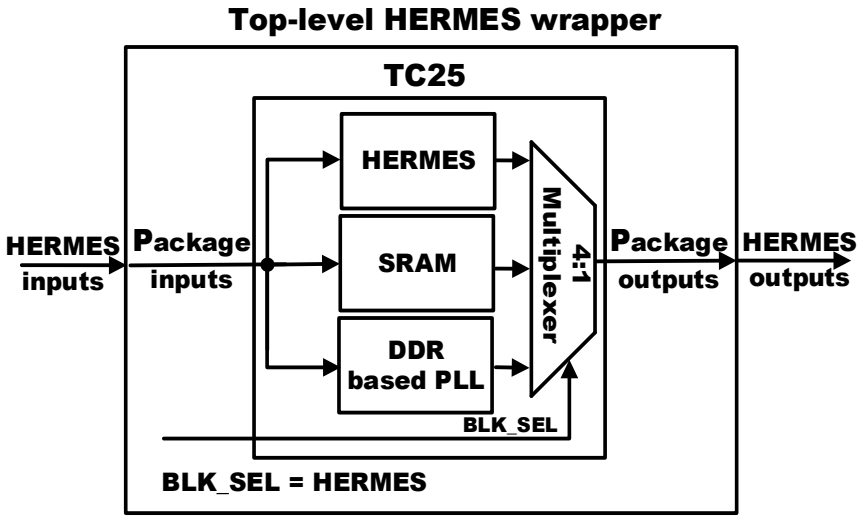

Figure 1-11 TC25 Top-level HERMES Wrapper

The functionality of every block is verified using a test bench specific to it. To use the same test bench on the block level as well as the chip level RTL, a wrapper using the top-level module of TC25, specific to the block is created. The wrapper module for a block is created by, assigning the input and output package pins of the test chip to the respective input and output signal names of the block, setting the BLK_SEL and unused pins to the right values. For example, in the case of the HERMES block, BLK_SEL is set to select HERMES, all package pins are assigned to signals of HERMES (Figure 1-11).

1.3.3. Functional Verification of HERMES

The same design of HERMES is used in test chip 25 as well as test chip 24. Therefore, to reduce the effort spent on verification of the design used in test chip 25, it is validated with the one used in test chip 24. To achieve this, the top-level HERMES wrappers of both the test chips are instantiated in the top-level test bench module (Figure

1-12). The input signals of both the wrappers are driven with the same signals in the HERMES test bench. The output signals from both the wrappers are XORed and stored using negative edge triggered flipflops. Assertions based on this flopped signals are used to check the behavior of the design across all the 70 tests.

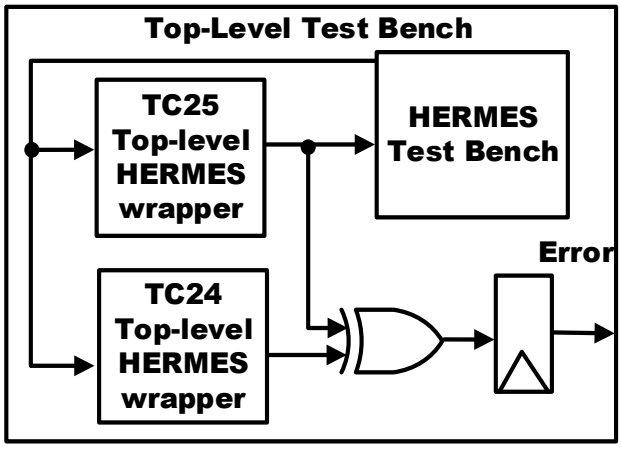

Figure 1-12 TC25 Block diagram of the test bench of HERMES processor

1.3.4. Functional Verification of SRAM Block

The SRAM block contains custom built 16kB banks inside 1Mb arrays. The behavioral model of the 16kB bank is used in RTL simulations of the SRAM block. Since the 16kB bank is custom built, the logic around these banks involving both the decoders, clock gaters, special registers, data output mux is verified for their functionality in RTL simulations. Both the decoders and data output mux are checked for their working by performing write and reads on the banks in the SRAM block.

During the normal mode of operation, the special registers are configured without any test modes. Then, the write operation is performed on an address of a bank. Later, the read is performed on the same address of the same bank and data output is checked with the data input of the write operation. This is repeated for all the banks in all the SRAM

arrays to verify the functionality of both the decoders, data output mux. The functionality of the clock gaters is verified by checking the clock of the banks not selected for any operation. The special registers are configured with test modes and the output of these registers is checked to complete the functionality check.

1.3.5. Functional Verification of DDR PLL

The DDR PLL contains custom built PLL with the clock divider logic and the configuration registers around it. Since the PLL is custom built, the functionality of the clock divider logic and configuration registers is verified in RTL simulations. To achieve this, the behavioral model of the PLL has its input clock connected directly to the output clock. Then, the division factor of the clock divider is fixed by setting the appropriate configuration register. Now, the frequency of the output clock from the clock divider is checked against the input clock frequency divided by the division factor set. This verifies the functionality of the configuration registers and the clock divider logic.

CHAPTER 2. POST SILICON VALIDATION SETUP

2.1. Test Setup Overview

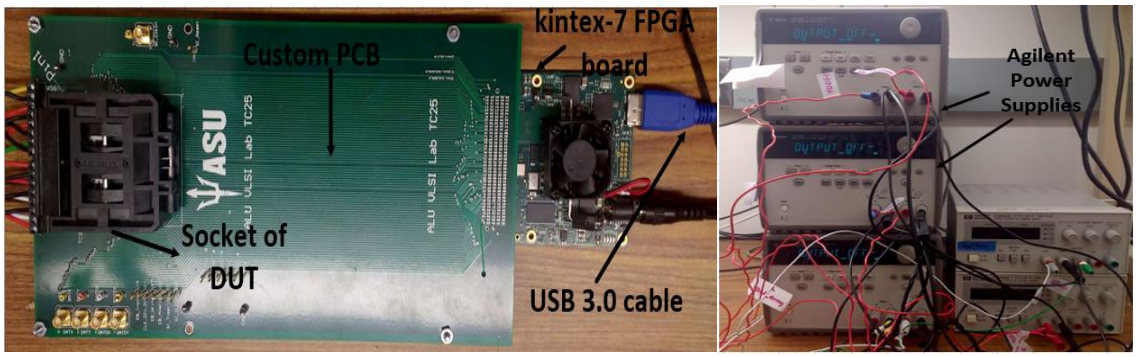

Figure 2-1 TC25 Post silicon testing setup

The post silicon validation setup of TC25 consists of the custom printed circuit board (PCB), the kintex-7 field programmable gate array (FPGA) based XEM7350 Opalkelly board, personal computer (PC) and five agilent E3646A direct current (DC) power supplies as shown in figure 2-1.

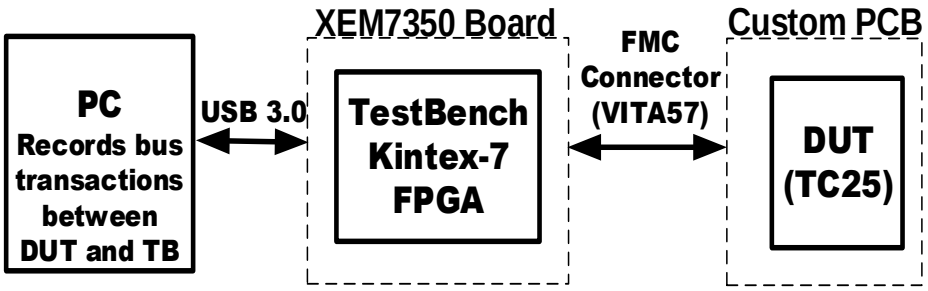

Figure 2-2 Functional block diagram of TC25 post silicon validation setup

The top level functional block diagram of the post silicon validation setup of TC25 is shown in figure 2-2 The test chip TC25 is placed inside a socket on the custom PCB.

The kintex-7 FPGA is configured as the test bench to the DUT. The XEM7350 board is connected, to the PCB using the FPGA mezzanine connector (FMC) and to the PC using universal serial bus (USB) 3.0 cable.

2.1.1. Custom PCB

Figure 2-3 Custom PCB of TC25 post silicon validation setup

The custom PCB is built with long traces to keep the FPGA board away from the neutron beam testing as shown in figure 2-3. The board has bypass capacitors (0.1uF) shunted across all the power supplies to decouple the power supply noise from the test chip. All the input XOR clock signals to the test chip are routed with equal wire length to make sure that phase relationship between the clocks is maintained. The board has five test points shared across the analog signals of the SRAM block and the core clock signal of the HERMES processor. The board has probe points related to few of the signals of all the blocks on the test chip.

2.1.2. XEM7350 board

The XEM7350 Opalkelly board is used as the test bench of TC25. The functional block diagram of the FPGA board is shown in the figure 2-4. The board has a 676 pin kintex-7

FPGA device mounted on it [Opalkelly15]. To generate low-jitter clocks to the FPGA device, two crystal oscillators are present on-board. They generate fixed clock frequencies of 100MHz and 200MHz. A cypress FX3 USB 3.0 microcontroller, present on-board makes it a USB 3.0 peripheral. This results in fast data transfers between the board and the PC.

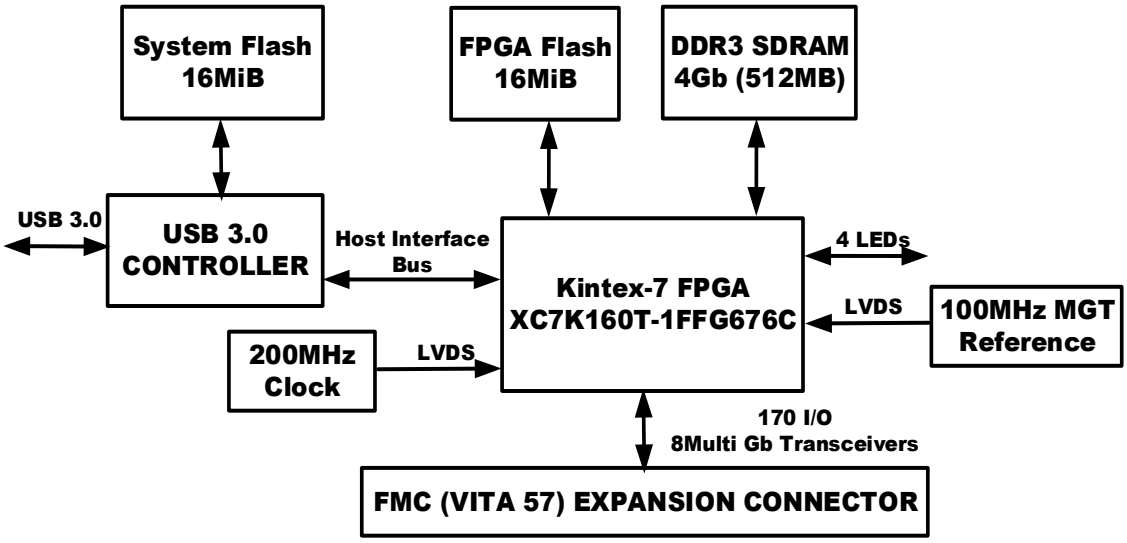

Figure 2-4 Function block diagram of XEM7350 FPGA board [Opalkelly15]

The non-volatile storage devices on the FPGA board include a 16 million bits (Mib) system flash and a 16Mib FPGA flash. The System flash is used to store the device firmware and configuration settings along with FPGA configuration files. The FPGA flash can be used only by the kintex-7 device. The FMC on the board is a high pin count (HPC) version of the VITA 57 specification. This connector helps in interfacing large pin-count designs to the kintex-7 device. The I/O pins on the connector are categorized into LA, HA and HB groups. The pins of the LA and HA groups are routed to FPGA banks which are powered from the XEM7350 board. On the other hand, the pins of the HB group are routed

to FPGA bank 32 which is powered from an external power supply provided through FMC_VIO_B_M2C pin on the FMC connector. In the case of TC25, the power supply which provides VDDIO voltage is used to power the HB group pins.

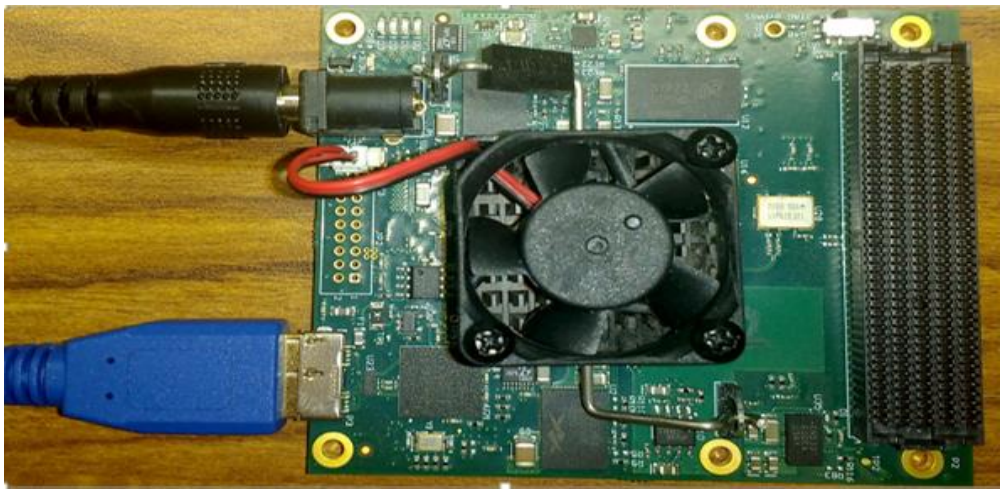

Figure 2-5 XEM7350 FPGA board

The FPGA board is operated using a 5-volt power source supplied using the DC power jack on-board. The heat dissipated by the kintex-7 device is very high and could damage it depending on the application run on the device. This is due to the presence of large density of logic in a small area on the device. Therefore, to protect the device from heat dissipation, a fan is used as an active heat sink. The fan is mounted on top of the FPGA device as shown in figure 2-5.

2.2. Top level Test Bench of TC25

The infrastructure of the top-level test bench of TC25 consists of the Opalkelly module, the memory module, the clock generator module and the TC25 block specific test bench as shown in figure 2-6. The clock generator module synthesizes the clocks to the

Memory module and the test bench using a PLL. The PLL gets the 200MHz reference clock from the crystal oscillator as input. The memory module acts as a buffer between the DUT and the test bench by making use of the block random-access memory (BRAM) blocks present on the FPGA. This module stores the clock cycle by clock cycle activity of all the signals present between the test bench and the DUT in the BRAMs. The Opalkelly module provides visibility and controllability on the FPGA for the program running on the PC. This module is used to transfer the data stored on the BRAMs to the PC and un-gate the clock signal to the DUT and the test bench. Also, this module is used to intimate the program, about the end of the test running on the FPGA.

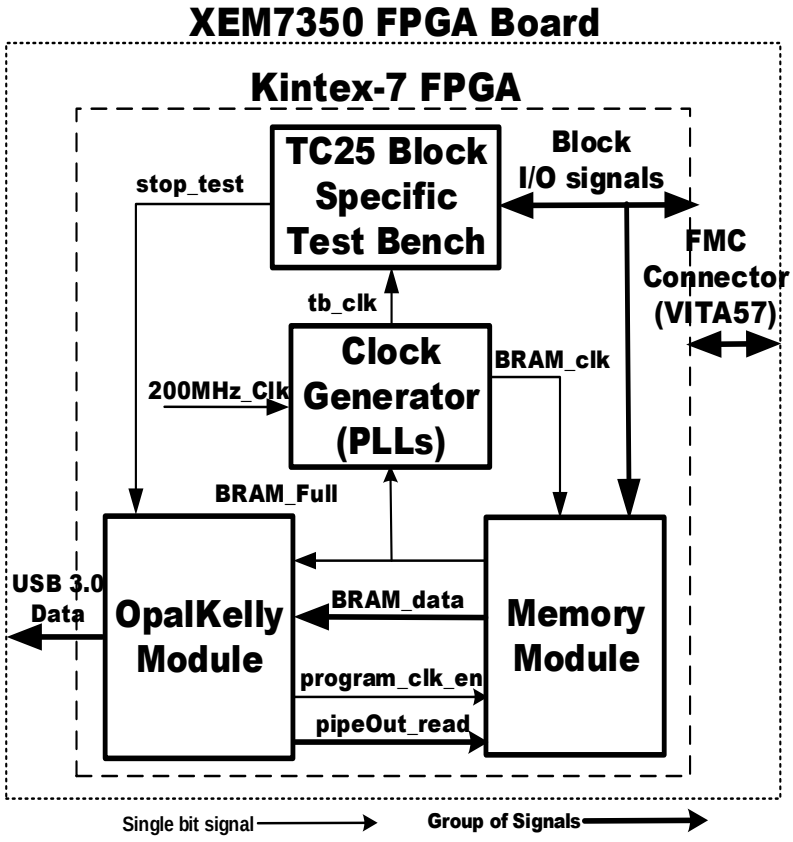

Figure 2-6 Top level test bench architecture of TC25

2.2.1. Clock Generator module

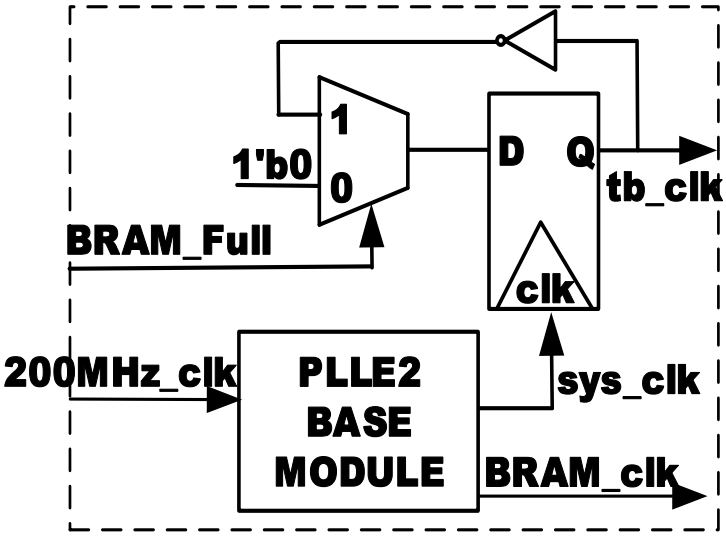

Figure 2-7 Clock generator module used in the test bench of TC25

The clock generator module consists of an instance of Kintex-7 PLL primitive PLLE2_BASE [XilClk16] and a divide-by-2 clock divider logic. The PLL module takes the 200MHz_clk as input and synthesizes two clocks of the same frequency and phase namely sys_clk and BRAM_clk as shown in figure 2-7. This module can be used to adjust the duty cycle, phase offset (relative to the input clock) and frequency of each of the output clocks individually. This module has master multiplication and division parameters whose legal ranges are 2-64 and 1-56 respectively.

The frequency range of any output clock produced using the PLL primitive is 6.25MHz – 1600MHz. This is limited by the operating frequency range of the PLL primitive's voltage controlled oscillator (VCO) (800MHz - 1600MHz), specific to the kintex-7 device [XilClk16]. The frequency value of the PLL primitive's VCO is set depending on the frequency of the input clock and the values of the master multiplication

and division parameters. In case the frequency set on the VCO is not in the range specified, design rule check (DRC) violations related to routing errors (PDRC-43) [XilClk16] are observed during the implementation phase on FPGA in vivado.

The clock divider logic generates the divide-by-2 clock using the sys_clk from the PLL primitive (figure 2-6). This logic has a 2:1 mux which gates the divide-by-2 clock depending on the value of the BRAM_Full signal. This clock divider logic is required not only to gate the divide-by-2 clock but also to generate clock frequencies less than 6.25MHz. For example, in the case of post silicon testing of the SRAM block, the frequency of the divide-by-2 clock required is 5MHz. So, a 10MHz sys_clk produced by the PLL is used along with the clock divider logic to synthesize a clock frequency of 5MHz.

2.2.2. Memory module

The memory module used in the test bench of TC25 consists of two or three BRAM instances depending on the block tested on DUT along with the read and the write logic for each of those instances as shown in figure 2-8. Each BRAM instance is configured in simple dual-port random-access memory (RAM) mode with an independent read and write port to perform simultaneous read and write operations. The read and write ports access the same address in which case the BRAM can be configured to do either the read operation or the write operation first. However, write and read operations are never performed simultaneously on any of the BRAM instances in the memory module. Therefore, they are randomly configured to do the write operation first.

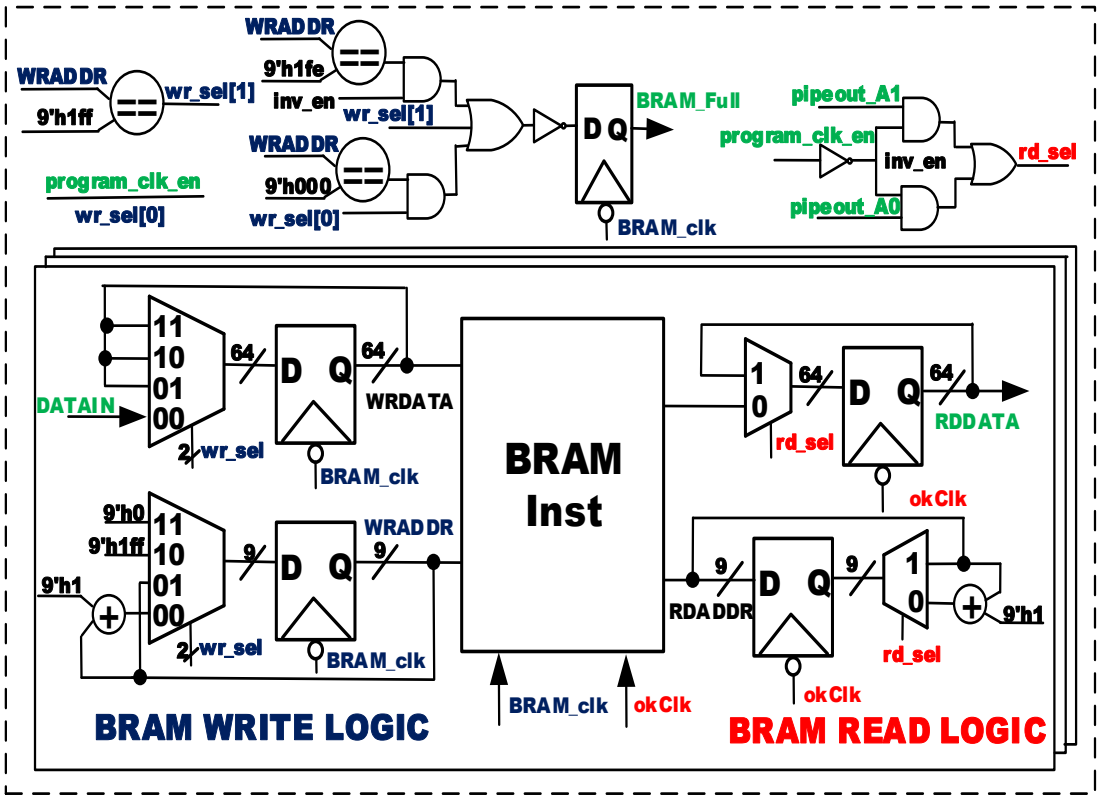

Figure 2-8 Memory module used in the TC25 test bench

Every BRAM instance can store 36 kilobits (Kb) of data [XilMem16]. They are configured with a line width of 72 bits resulting in a depth of 512 locations and an address width of 9-bits. Out of the 72 bits in a line, eight of them are parity bits and the rest of the 64 bits are data bits. Every data bit on a line can correspond to a block I/O signal of the DUT. Therefore, each BRAM instance can store the data of 64 block I/O signals of the DUT. In the case of the HERMES block, there are 145 I/O signals (table 1-1) present between the test bench and the DUT. Hence, three BRAM instances are needed for the test bench of HERMES. Also, the SRAM block and the DDR PLL have 129 and 80 I/O signals (table 1-1) respectively. Hence, two BRAM instances are needed for the test benches of the SRAM block and the DDR PLL.

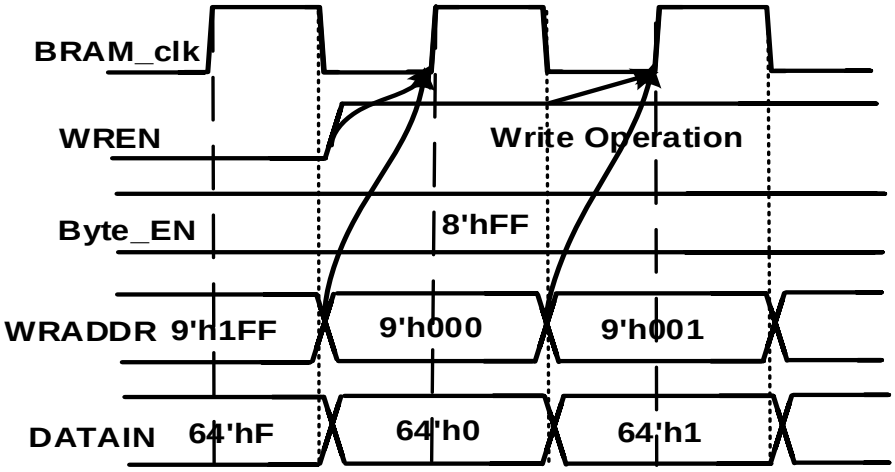

Figure 2-9 Timing diagram of the write operation on a BRAM instance [XilMem16]

The write operation on a BRAM instance is performed using BRAM_clk as the clock input. During the write operation, the address, data and write enable signals are sampled on the falling edge of the BRAM_clk (figure 2-9). This makes sure that these signals are setup to the rising edge of the BRAM_clk. Depending on the assertion of the write enable signal, the write operation is performed on the address specified at the rising edge of the BRAM_clk. The frequency of the BRAM_clk is twice the frequency of the clock to the block specific test bench and the DUT. This makes sure that every clock cycle of the block specific I/O signals is sampled twice by the BRAM write logic. Therefore, every two lines on the BRAM instance correspond to a clock cycle of the I/O signals.

The read operation on a BRAM instance is performed using okClk as the clock input. This clock is used for the read operation instead of the BRAM_clk because the logic in the Opalkelly module works only with this clock. During the read operation, the address and read enable signal is sampled on the falling edge of the okClk (figure 2-10). This makes sure that these signals are setup to the rising edge of the okClk. Based on the assertion of

the read enable signal, the read operation is performed on the address specified at the rising edge of the okClk. The data read is given to the Opalkelly module which sends it to the program running on the PC.

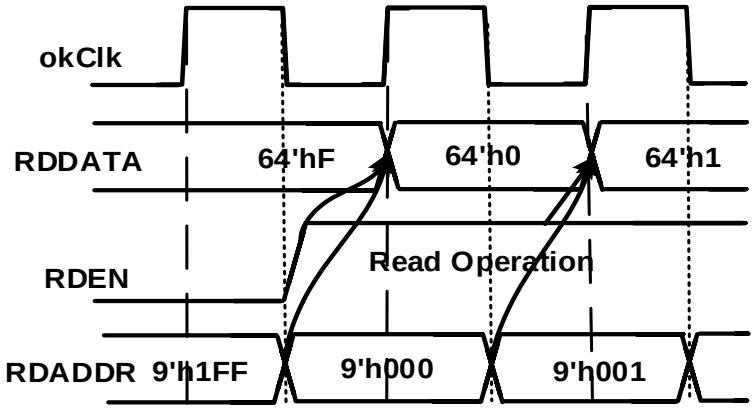

Figure 2-10 Timing diagram of read operation on a BRAM instance [XilMem16]

The write operations begin at the first location of the BRAM instance (address 9'h000) and are performed continuously till the last location of the BRAM instance (address 9'h1ff). Once the last location of the BRAM instance is reached, the signal BRAM_Full (figure 2-8) is driven low and no further write operations are performed. To make sure that no write operations are done, the write logic drives the write enable signal low and keeps the write address at the value of the last location. Also, the clock signal to the DUT and the block specific test bench is gated to avoid generating new data to the BRAM instance.

The write operations are stopped until the data stored in the BRAM instance is read by the program running on the PC. The BRAM_Full signal is used to intimate the program that the BRAM instance is fully filled. Then, the program reads the complete data stored in the BRAM instance using the Opalkelly module. Once the BRAM instance is completely read out, the read logic drives the read enable low and ties the read address to the first location until the next read operation. The program asserts the program_clk_enable signal

after finishing the read operation. This event drives the BRAM_Full signal high, resulting in un-gating of the DUT clock. Also, this is used by the write logic to resume the write operations on the BRAM instance.

2.2.3. Opalkelly Frontpanel Package

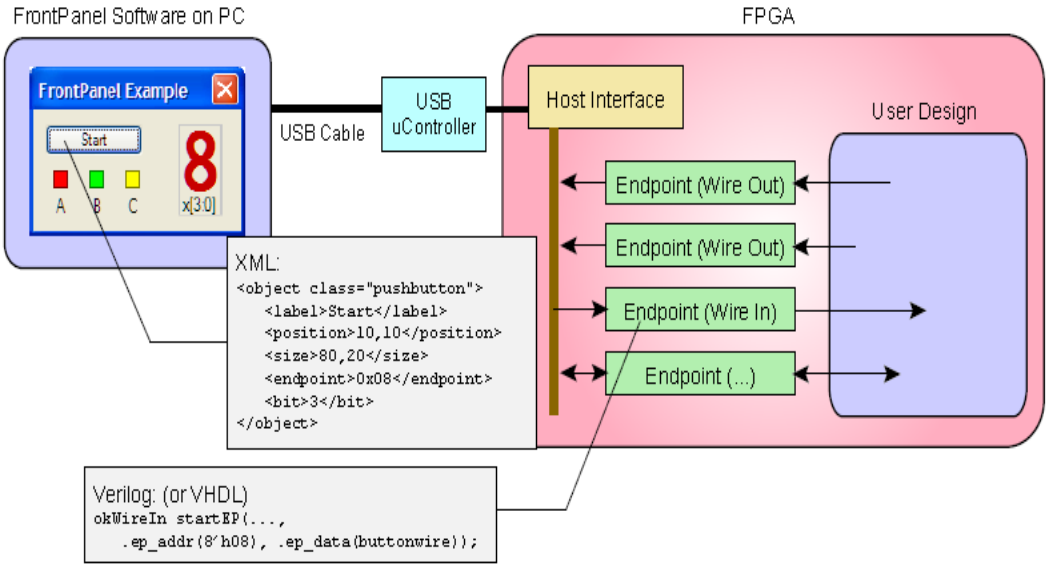

Figure 2-11 Opalkelly frontpanel enabled design on a FPGA [OpalKelly15]

A design built on the FPGA is usually debugged using oscilloscopes, LEDs and controlled using push buttons and switches. Due to the limited amount of these resources on the FPGA board, observing the signals of the design on the FPGA for debugging purposes results in longer development time and greater human effort. The frontpanel package from the Opalkelly overcomes this problem by providing visibility and controllability on the design present on the FPGA, to a program run on the PC. The package achieves it by providing frontpanel application programming software (API) and hardware description language (HDL) endpoints (Figure 2-11) to interface the design on the FPGA.

The HDL endpoint can be a wire, trigger or pipe and is directed in or out of the design on the FPGA. An in endpoint moves data into the design and an out endpoint moves data out of the design. The wire type endpoints are used to transfer signal state asynchronously into or out of the design. They can be used as virtual LEDs, switches and push buttons. The pipe type endpoints are used to perform multi-byte synchronous transfers into or out of the design. They can be used to stream data to the design, upload contents of memory and download the contents to the memory. Every endpoint for USB 3.0 interface has a bus width of 32-bits. The maximum number of endpoints that can be placed in the design is limited to 32.

The HDL endpoints are present along with the design on the FPGA. The signals in the design that need to observed and controlled are connected to these endpoints as shown in figure 2-11. These endpoint modules are placed on a shared bus along with the host interface module. The host interface module along with frontpanel drivers and API make these signals visible to the program on the PC. Also, the signals of input endpoints such as okwireIn, okPipeIn can be controlled from the program on the PC. Therefore, frontpanel's flexibility allows to display real-time information of any number of signals of the design configured on the FPGA.

2.2.4. Opalkelly module

The Opalkelly module used in the test bench of TC25 is shown in the figure 2-12. This module is made of a single instance of okHost Interface, four or six instances of okpipeOut endpoints depending on the number of BRAMs used in the memory module and two instances each of okwireIn and okwireOut endpoints. The okHost interface module

contains logic that lets the USB 3.0 microcontroller communicate with the endpoint instances. The okClk signal acts as the clock to the Opalkelly module. The frequency of the okClk is fixed at 100.8 MHz for USB 3.0 interface.

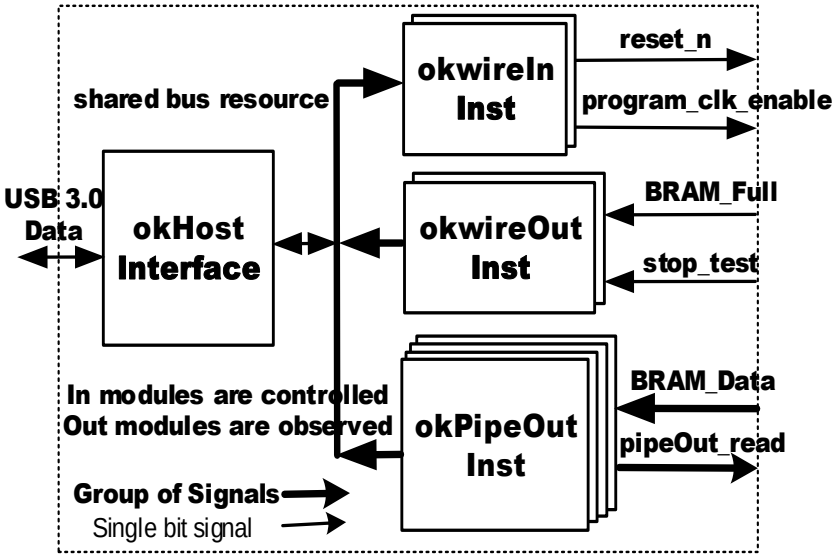

Figure 2-12 Opalkelly module used in the test bench of TC25

The C++ program running on the PC drives and samples the signals connected to the okwireIn and okwireOut instances, respectively. Out of the two okwireIn instances, one of them is connected to the active low reset signal and the other is connected to the DUT clock un-gating signal. Similarly, one of the okwireOut instances is connected to the signal which indicates the BRAMs are fully filled and the other instance is connected to a signal that indicates the end of the test run on the DUT.

The okpipeOut module has a fixed bus width of 32 bits. However, every BRAM is configured to have a data width of 64 bits per line. Also, only one line of a BRAM can be read out in a single clock cycle. Therefore, two okpipeOut instances are needed to transfer the data stored in a BRAM. Since three BRAMs are required in the test bench of the HERMES block, six okpipeOut instances are used in the Opalkelly module of the

HERMES test bench. Similarly, two BRAMs are required in the test benches of the SRAM block and DDR PLL. Therefore, their test benches have four instances of okpipeOut in the Opalkelly module.

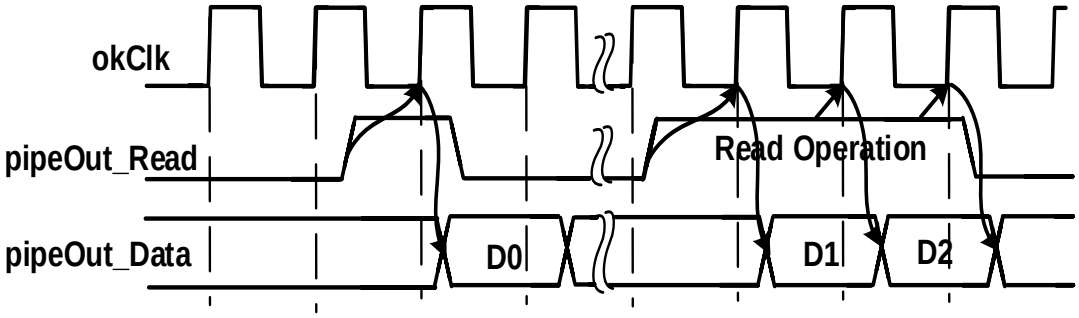

Figure 2-13 Timing diagram of the data transfer from okpipeOut module [OpalKelly15]

The C++ program running on the PC asserts the pipeOut_Read signal to read the data through the okpipeOut instance. This signal is used as the read enable signal to the BRAM instance Then, the read logic of the BRAM instance performs the read operation and places the data read on the pipeOut_Data bus in the next clock cycle as shown in figure 2-13. The data on the pipeOut_Data bus is considered as valid only in the clock cycle following the one where pipeOut_Read signal is high. The pipeOut_Read signal can be de-asserted during longer data transfers (> 256 words).

2.3. C++ Application Program

The frontpanel API is provided as a C++ library which contains methods to communicate directly with the endpoint modules on the FPGA. The classes in this dynamically-linked library are instantiated in a program and the methods are called using them. okCFrontpanel is the base class used to find, configure, and communicate with the FPGA board. The methods in this class are used to interact with the FPGA board, configure the flash memory, Kintex-7 FPGA and communicate with the FPGA.

```
// STEP 2: Interact with the XEM7350 board and open it
if (okCFrontPanel::NoError != xem->OpenBySerial())
{
    fprintf (pFile, "Device could not be opened.  Is one connected?\n");
    return(false);
}

// The okTDeviceInfo struct contains some information about our System Flash
// for USB 3.0 devices.
xem->GetDeviceInfo(&m_devInfo);
fprintf (pFile, "Found a device: %s\n", m_devInfo.productName);

//STEP 3: Load default PLL configuration
xem->LoadDefaultPLLConfiguration();

//STEP 4: Download the configuration file.
if (okCFrontPanel::NoError != xem->ConfigureFPGA(std::string(BITFILE_NAME)))
{
    fprintf(pFile, "FPGA configuration failed.\n");
    return(false);
}
else
{
    fprintf(pFile, "Device Configuration Successful \n");
    return (true);
}
```

Figure 2-14 Code snippet of the steps 2-4 of the C++ program built using frontpanel API

The application program performs the following steps:

1. Creates an instance of okCFrontpanel class.
2. Using the device interaction methods, checks for the connection with the XEM7350 board. After detecting the board, gathers information about the devices such as flash, FPGA on the FPGA board and opens the board.
3. Configures the PLL with the default configuration.
4. Downloads a configuration file to the FPGA using ConfigureFPGA method.
5. Perform TC25 specific communication with the FPGA using the FPGA communication methods.

2.3.1. Program Flow

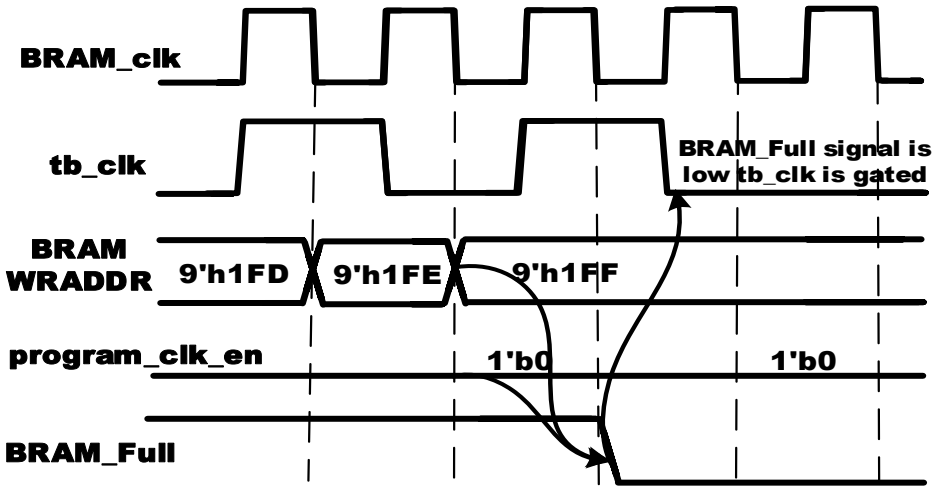

Figure 2-15 Timing diagram of the clock gating event of the DUT and the test bench clock

After configuring the FGPA, the program control flow for any test run on TC25 is as follows:

1. The configured FPGA is reset by the program. Once the reset on the FPGA is released, the test starts and the BRAMs are filled with data.
2. BRAMs reach their storage limit after recording a few clock cycles of signal activity. This is indicated by the assertion of the signal BRAM_Full (figure 2-15).
3. Then the clock to the DUT and the test bench (tb_clk) is gated in the FPGA (figure 2-15). This makes sure that there is no unrecorded signal activity between the DUT and the test bench.
4. The program detects that the BRAMs are fully filled and reads the data from them using the Opalkelly module.

5. Once the data is completely read from the BRAMs, the program un-gates the clock to the DUT and the test bench to start the signal activity and the BRAMs are filled with the new data (figure 2-16).
6. Steps 2,3 and 4 are repeated until the end of the test is reached. This is indicated by the assertion of stop_test signal by the block specific test bench.

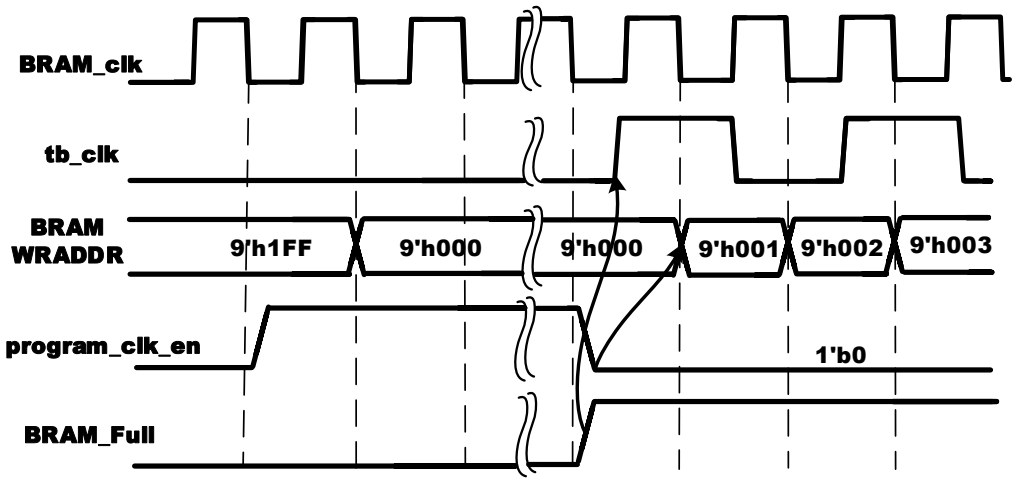

Figure 2-16 Timing diagram of the un-gating event of the DUT and the Test Bench clock

The program stores the data read from the BRAMs in character arrays. Each character can store 8-bits of data. Now, to store 32-bits of data from a single pipeOut instance across 512 clock cycles (the depth of a BRAM instance) a character array having a size of 2048 (512*32/4) is used in the program.

```
// Let's Attack PipeA0
bit_Spreg_Clk     = ( dataout_A0[k] >> 7 ) & 1;
bit_SRAM_Sel_0    = ( dataout_A0[k] >> 6 ) & 1;
bit_SRAM_Sel_1    = ( dataout_A0[k] >> 5 ) & 1;
bit_XOR_CLK0      = ( dataout_A0[k] >> 4 ) & 1;
bit_blk_sel_0     = ( dataout_A0[k] >> 3 ) & 1;
bit_blk_sel_1     = ( dataout_A0[k] >> 2 ) & 1;
bit_CLK_SEL       = ( dataout_A0[k] >> 1 ) & 1;
bit_SI_PLLReset   = ( dataout_A0[k] >> 0 ) & 1;
```

```
// Drive DI on to the BRAM
DI[71:36]  <= {  4'h0, datain };

DI[35:0]   <= {  4'h0,
                 address,  read , write,
                 1'b0, 1'b0, 1'b0,  CLK_DIV_OUT,
                 Spreg_addr,  Spreg_clk, SRAM_Sel,
                 XOR_CLK0,  BLK_SEL,  CLK_SEL,
                 SI_PLLReset
              };
```

Figure 2-17 Code snippet of the packing (right) and the unpacking (left) of the I/O signals

Packing of the data happens during the mapping of the block I/O signals to specific data bit positions of the BRAM. Hence, the data stored in the character array is unpacked and mapped to the same block I/O signal. For example, the Spreg_Clk, SRAM_Sel signals of the SRAM block are packed and unpacked as shown in figure 2-17. The unpacked data is then written to a log file which can be used for debugging or post processing purposes. The log file can be post processed using perl scripts to check for the validity of the test run or to extract specific information from the test run.

2.4. Bit stream File Generation

2.4.1. Mapping of physical pins to logical signals

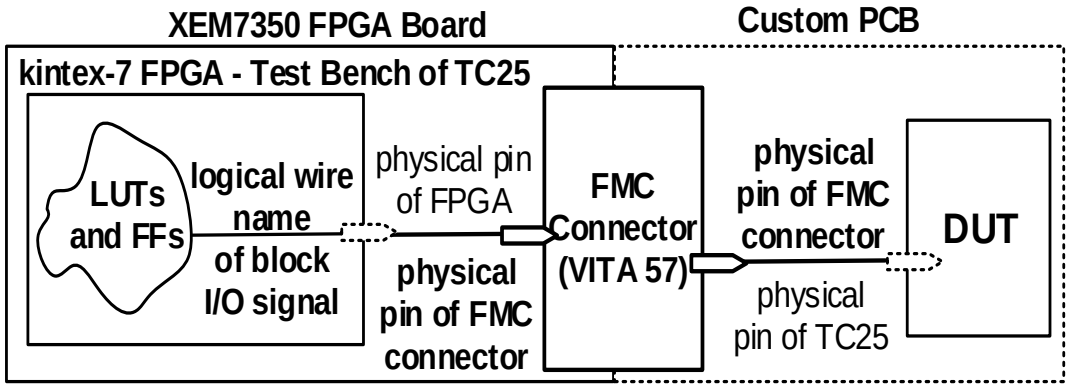

Figure 2-18 Mapping of physical pins to the I/O signals of TC25 on testing setup

The logical signals of each block on TC25 are mapped to the pads on the I/O ring of the test chip. During the packaging of the test chip, these I/O pads are connected to the physical pins on the test chip. The custom PCB has the physical mapping between the pins on the TC25 and the FMC (VITA 57) connector. The physical connections between the pins on the FPGA with those on the FMC connector are already fixed by the manufacturer of the FPGA board, Opalkelly. Therefore, the logical signals of the test bench of TC25 are

mapped to the appropriate physical pins on the FPGA. This mapping is important because any wrong connections made would result in significant debug time and human effort.

Consider an example of the signal XOR_CLK0 of the XOR Clock multiplier logic in the test chip. This signal is mapped to package pad number 107 on the test chip. This package pad is connected to pin H25 on the FMC connector of the custom PCB. This pin H25 on the FMC connector is physically connected to pin LA_21_P on the kintex-7 FPGA. When the FPGA is configured with the test bench of TC25, the logical signal used to drive the XOR_CLK0 pin of the DUT is mapped to physical pin LA_21_P of the FPGA. This mapping is stored in a file format called xilinx design constraints (XDC). The mapping information of all the block specific I/O signals is validated during the initial setup of the test bench of the block.

2.4.2. Input Files required by Vivado

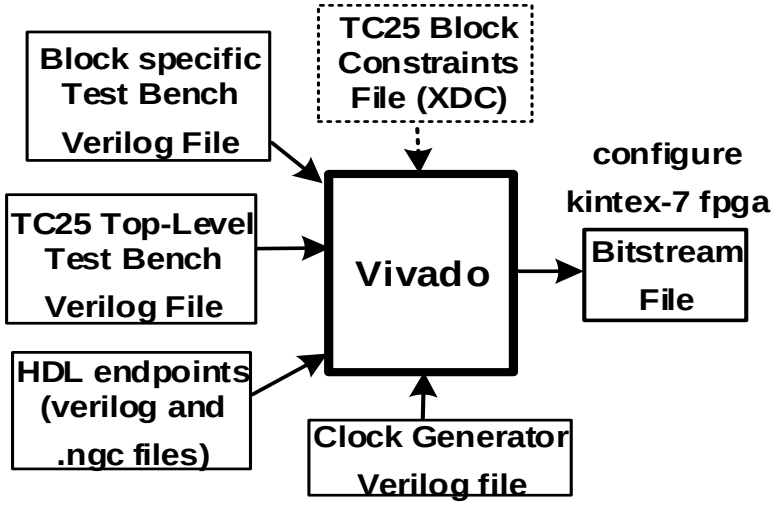

Figure 2-19 The Input files of the test bench of TC25 given to vivado

A bit stream file is needed to configure a FPGA device. This file is generated using vivado tool. The vivado tool takes the HDL files of a design and the constraints file of the

design specific to the target FPGA device as input [XilDes16]. The HDL files are simulated first to check if the design behaves as expected. Once the design works in simulation, logic synthesis step is run on the design. This step maps the logic written in HDL to LUTs, flipflops, BRAMs i.e hardware resources available on the target FPGA device. Then the implementation step is run on the design. This step does the place and route of the design, targeting the FPGA device using the timing constraints set. The design, post synthesis and implementation is again simulated to make sure that the functionality is not modified by synthesis and implementation steps. In the final step, the bit stream file is generated for the implemented design.

To generate the bit stream file of the test bench of TC25, the verilog files of block specific test bench, clock generator, TC25 top-level test bench and HDL endpoints (wire and pipe type) along with XDC constraints file are given as input to the vivado as shown in figure 2-19. The XDC constraints file contains the mapping of the logical names of the block I/O and okHost interface signals to the physical pins on the kintex-7 device. Then the synthesis and implementation steps are run targeting the kintex-7 FPGA device on the XEM7350 board. Finally, after the implementation step, the bit stream file is generated.

2.5. Validation of Post Silicon Testing Setup of TC25

The post silicon testing setup of TC25 is validated before using it to run tests on the DUT. This involves the following:

1. Checking for stuck-at 0 or 1 pins on the FPGA board and the custom PCB.
2. Checking the power connections on the custom PCB.

3. Checking the XDC file for correctness of the logical to physical pin mapping on the FPGA device.

2.5.1. Stuck-at 0 or 1 Pins

Any pin must toggle from logic “1” to “0” state and vice-versa to conclude that it is not a stuck-at 0 or 1 pin. The input pins of the DUT can be driven from the FPGA and checked for stuck-at 0 or 1 issues. However, the output pins of the DUT must be driven by a block within the DUT. Therefore, the SRAM block of TC25 is initially used to identify the stuck-at 0 or 1 pins on the setup.

Every input pin of the SRAM block is toggled from logic “1” to “0” state and vice-versa by driving it with a clock of very low frequency from the FPGA. Since the input pins are driven, the DUT is removed from the socket on the PCB. Then, all the relevant pins on the FMC connector present on the custom PCB are probed on the oscilloscope. Upon probing all the input signals, one among them namely datain[20] of the SRAM block was found to be stuck-at 0. This signal is mapped to pin F32 on the FMC connector.

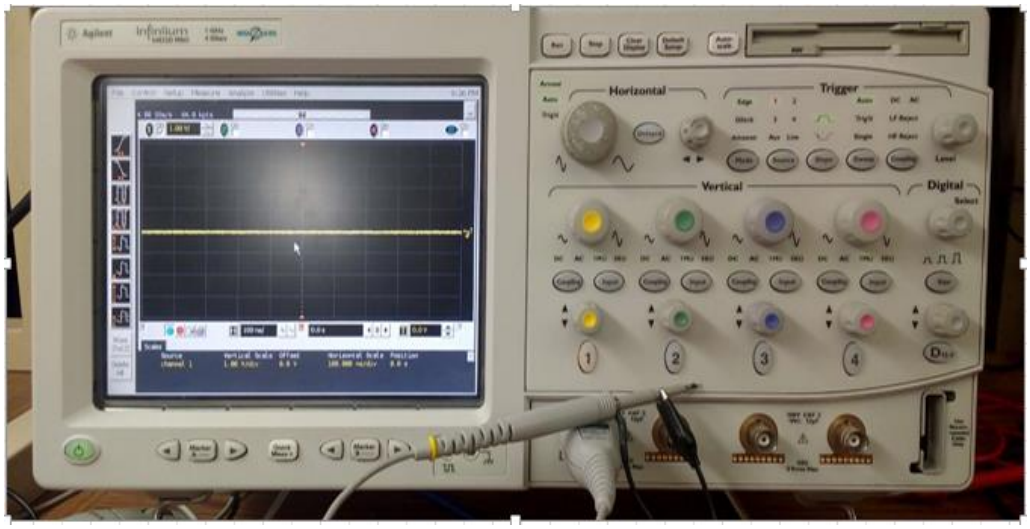

Figure 2-20 Oscilloscope used for viewing the signals of the testing setup of TC25

The stuck-at 0 issue of the F32 pin can be present on the FPGA board or the Custom PCB. Using the process of elimination, the FPGA board was checked first by connecting it to the FMC XEM105 debug board. Upon checking, the FPGA board is ruled out as the suspect because the F32 pin on the FMC connector of the board behaves as driven by the kintex-7 FPGA. Therefore, the custom PCB is identified as the culprit of this stuck-at 0 issue. Now, this could be due to an open connection between the FMC surface mount and the trace on the PCB or a short to V_{SS}. The short to V_{SS} possibility is ruled out using a multimeter. The connection is suspected as open and the FMC surface mount on the custom PCB is replaced with a new one. Upon testing the newly mounted custom PCB no stuck-at 0 issue is found with F32 pin. This confirmed that it is an open connection and solves the stuck-at 0 issue.

The output pins of the SRAM block are tested by running a simple write-read test. The test writes and reads a sequence of 0s followed by 1s and then followed by 0s from the same address location. This helps in toggling all the bits in the output dataout signal from logic “1” to “0” and vice-versa. Upon running the test, all the output signals except the dataout[45] signal toggle twice. This signal is found to be stuck-at 1. The stuck-at 1 issue is due to the signal being shorted to V_{DD} within the DUT due to an APR error and has nothing to do with the FPGA board or the custom PCB of the test setup.

2.5.2. Power Connections on PCB

The TC25 test chip has power pins for ten different voltages. These power pins are brought out from the socket into which the DUT is placed and physically connected to ten different logical voltage names on the PCB as shown in figure 2-21. The multimeter is used

to make sure that these logical voltage names are connected to the correct power pin on the test chip.

Figure 2-21 Power connections on the custom PCB

2.6. Basic Clock Divider Test

XOR_CLK[7:0]	CLK_DIV_OUT
76543210	
00000001	1
00000000	1
00000001	0
00000000	0

XOR_CLK[7:0]	CLK_DIV_OUT
76543210	
00000001	0
00000000	0
00000001	1
00000000	1

Figure 2-22 The divided clock toggling from 1 to 0 (left) and 0 to 1 (right)

This test exercises the XOR clock multiplier and clock divider logic in the test chip. This test makes sure that the chip is alive as the clock signal is the heart of this digital system. The test drives the XOR_CLK0 signal with a clock and other XOR_CLK signals with logic “0”. They are input to the test chip and the clock output from the test chip is sampled by the FPGA and written to a log file. The partial output from the log file of the test run is shown in figure 2-22. The clock output obtained, CLK_DIV_OUT has a divide-by-8 frequency relationship as expected with the clock input driven on XOR_CLK0.

CHAPTER 3. POST SILICON TESTING OF THE HERMES PROCESSOR

3.1. Architectural Overview

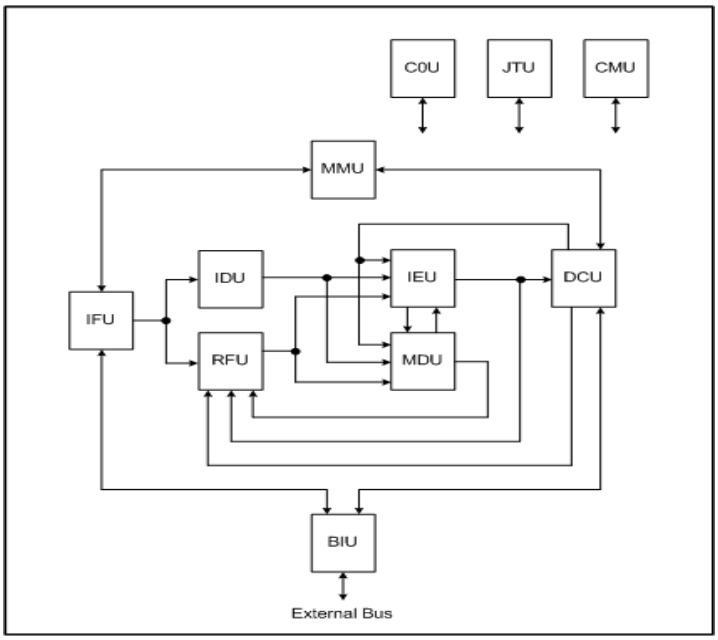

Figure 3-1 High-level block diagram of the non-radiation hardened HERMES processor

The high-level block diagram of the non-radiation hardened HERMES processor has eleven functional units as shown in Figure 3-1. The bus interface unit (BIU) acts as the interface between the external bus and the processor core. This unit receives instruction fetch requests, and load and store requests for the data memory access from the instruction fetch unit (IFU) and the data cache unit (DCU) respectively. The store requests from the DCU are stored in a write buffer before they are sent out onto the external bus. The IFU delivers instructions to the processor core pipeline. It has a 16 kB, 4-way set associative instruction cache (I-cache), one fill buffer, and a micro instruction translation look-aside buffer (ITLB) which translates virtual addresses to physical addresses. The instruction

decode unit (IDU) decodes instructions received from the IFU. The register file unit (RFU) contains the thirty-two 32-bit general-purpose registers [MIPS01].

The instruction execution unit (IEU) executes all the instructions from the instruction set except for multiply and divide instructions. The multiply/divide unit (MDU) executes all multiply and divide instructions. The DCU services load and store instructions of data memory. It has a 16 kB, 4-way set associative, write-through, a read-allocate only data cache (D-cache), one fill buffer, and a micro data TLB (DTLB) which translates virtual addresses to physical addresses. The memory management unit (MMU) contains the joint TLB (JTLB), which is a 16-dual entry TLB providing address translations for the ITLB and DTLB. The coprocessor 0 unit (C0U) contains the registers of the coprocessor 0 (system coprocessor). The JTAG unit (JTU) contains the joint test action group (JTAG) debug and testability logic. The clock management unit (CMU) provides the entire clock and power management functionality.

3.2. Test Bench Architecture

The test bench to the HERMES processor on TC25 consists of the reset generation logic module, data and instruction memory module and control logic module as shown in figure 3-2. The reset generation logic drives the PLL reset and the cold reset signals to the HERMES processor of the DUT and data and instruction memory in the test bench. The control logic provides the processor access to the instruction and data memory in the test bench. The instruction memory provides the instructions to the processor when requested by it. The data memory allows the processor to store or retrieve data from it.

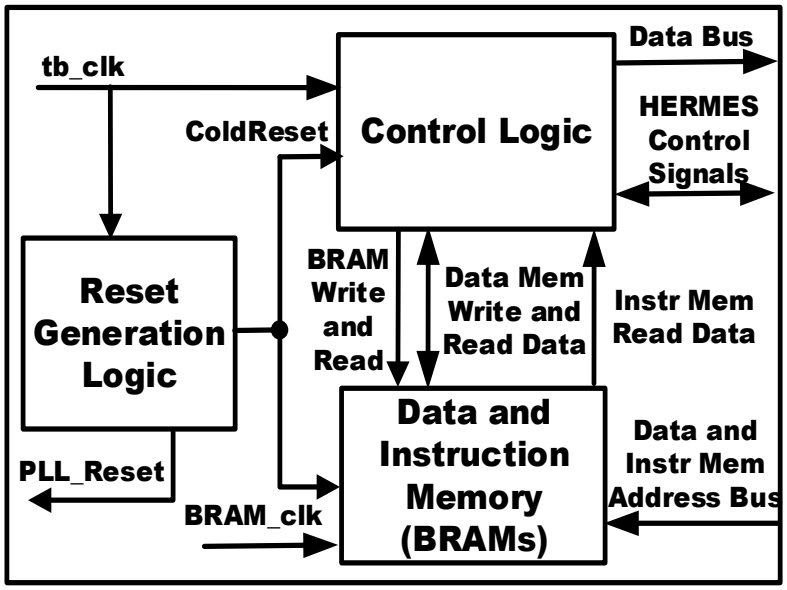

Figure 3-2 Functional block diagram of the test bench to the HERMES processor

3.2.1. Reset Generation Logic

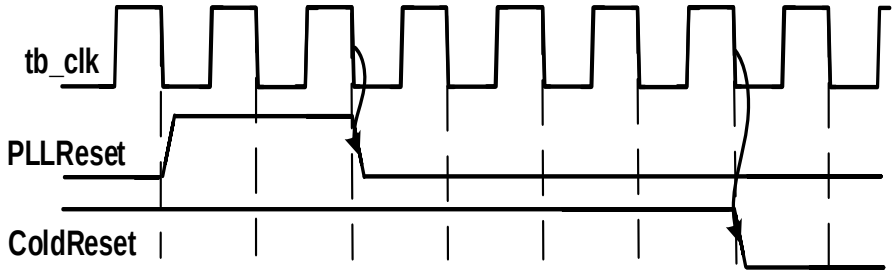

Figure 3-3 Timing diagram of the reset signals of the HERMES processor

The reset generation logic asserts and de-asserts the active high reset signals, PLLReset and ColdReset of the HERMES processor as shown in Figure 3-3. The PLL reset signal is used to load the PLL configuration and the bus-to-core clock ratio registers of the processor. The cold reset is a hard reset and causes a reset exception in the processor core. Also, this signal is required for synchronizing the external bus clock with the internal processor core clock. Both the reset signals are generated using a 4-bit counter, which is

used to count the clock cycles of the test bench clock. Initially the PLL reset is held low while the cold reset stays high. Based on the value of the counter, the PLL reset is asserted and de-asserted for two clock cycles of the test bench clock. Later-on after a few clock cycles of the test bench clock, cold reset is de-asserted.

3.2.2. Data and Instruction Memory

The data and instruction memory of the test bench of HERMES is implemented using BRAMs on the FPGA as shown in figure 3-4. The instruction memory provides 32-bit instructions to the processor. The BRAM of the instruction memory is loaded with the instructions that need to be tested on the processor. Then, the processor reads instructions and executes them one after another. Since the processor solely reads from the instruction memory, only the read operations are performed on the memory. The processor reads from and writes 32-bit data to the data memory and hence, both read and write operations are performed on the memory.

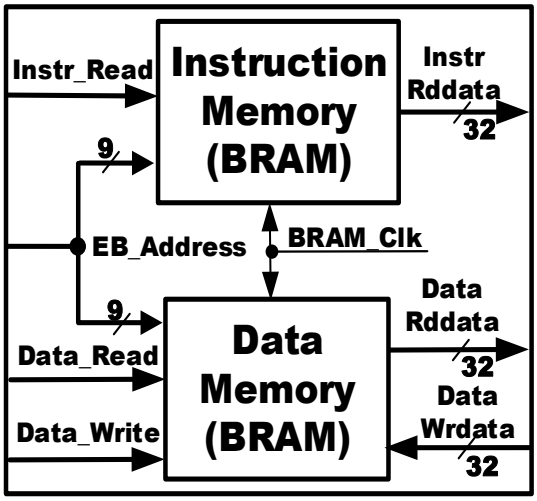

Figure 3-4 Block Diagram of the data and instruction memory of the test bench

The address sent out by the processor to read an instruction is 32-bits wide. This translates to an address space of 4 GB of byte addressable memory. This address space is divided into four regions namely kuseg, kseg0, kseg1 and kseg2[MIPS01]. The kuseg and kseg2 regions are only accessed in user and kernel modes after the MMU is setup. The addresses in the kseg0 (0x8000.0000 – 0x9FFF.FFFF) and kseg1 (0xA000.0000 – 0xBFFF.FFFF) [MIPS01] regions are stripped of the top one and three bits respectively when translated to physical addresses.

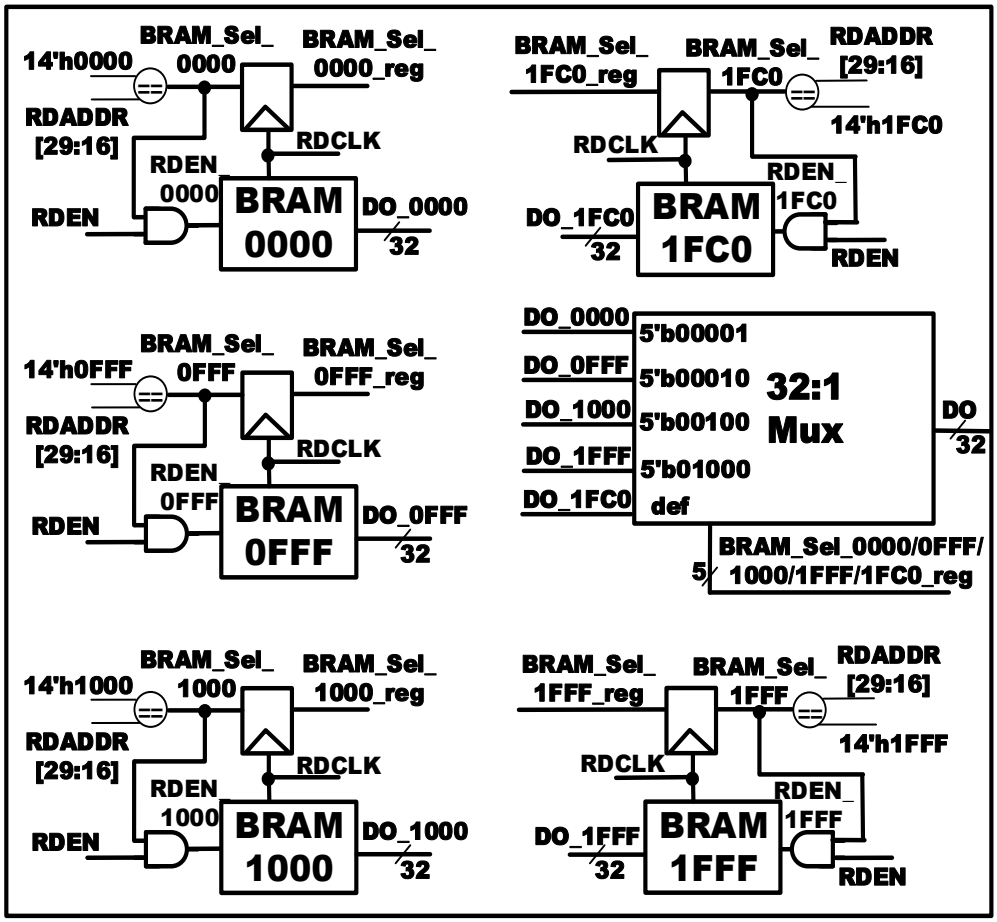

Figure 3-5 Functional block diagram of the instruction memory using BRAM

The post silicon testing of the HERMES processor on TC25 makes use of only kseg0 and kseg1 segments of the instruction memory. The instruction memory used in the test

bench of the processor makes use of 5 instances of BRAM to access these segments. The BRAM instances access different address ranges based on the top 16-bits of the address bus. The address ranges accessed are 0x1FC0, 0x0000, 0x0FFF, 0x1000, 0x1FFF. Depending on the instruction read address from the processor, the select signal of the corresponding BRAM is asserted. This signal is flopped with positive-edge triggered flipflop using BRAM read clock as the clock signal. The flopped signal makes sure that the data output from the correct BRAM is selected and placed for one clock cycle on the output data bus of the instruction memory.

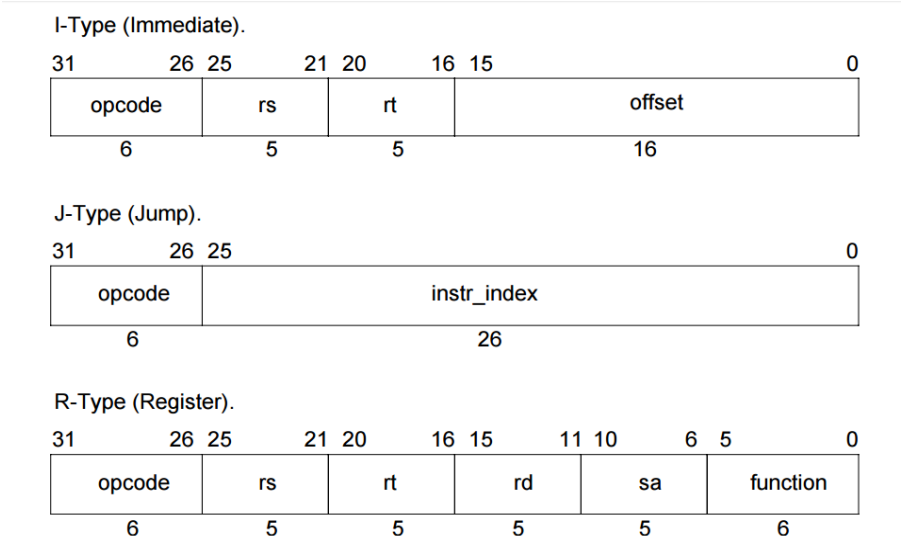

Figure 3-6 Type of instructions in MIPS instruction set architecture(ISA)

The type of instructions that can be run on the HERMES processor are of the register or jump or immediate types as shown in figure 3-6. The opcodes and function codes used in the instructions, initialization data of the BRAMs are parameterized. Instructions are built using the parameters of opcodes and function codes and loaded into the BRAMs using initialize data parameters of the BRAM as shown in figure 3-7.

```
// Parameters used to initialize the BRAM with instructions                    INSTRUCTION ASSEMBLY CODE
parameter INIT_BRAM_1FC0_000 = { LUI_OP   , 5'h00  , 5'h01 , 16'h0    };   // -- LUI   $1,  0x0000
parameter INIT_BRAM_1FC0_001 = { LUI_OP   , 5'h00  , 5'h02 , 16'h0    };   // -- LUI   $2,  0x0000
parameter INIT_BRAM_1FC0_002 = { ORI_OP   , 5'h01  , 5'h01 , 16'hFF21 };   // -- ORI $1,$1,  0x0003
parameter INIT_BRAM_1FC0_003 = { ORI_OP   , 5'h02  , 5'h02 , 16'h0    };   // -- ORI $2,$2,  0x0000
```

Figure 3-7 Code snippet on how instructions are loaded into BRAMs

The data memory used in the test bench of the processor makes use of 2 instances of BRAM to access the address ranges 0x0000 and 0x1000 (figure 3-5). Similar BRAM select and data output mux logic is used in the data memory to read the data from the appropriate BRAM.

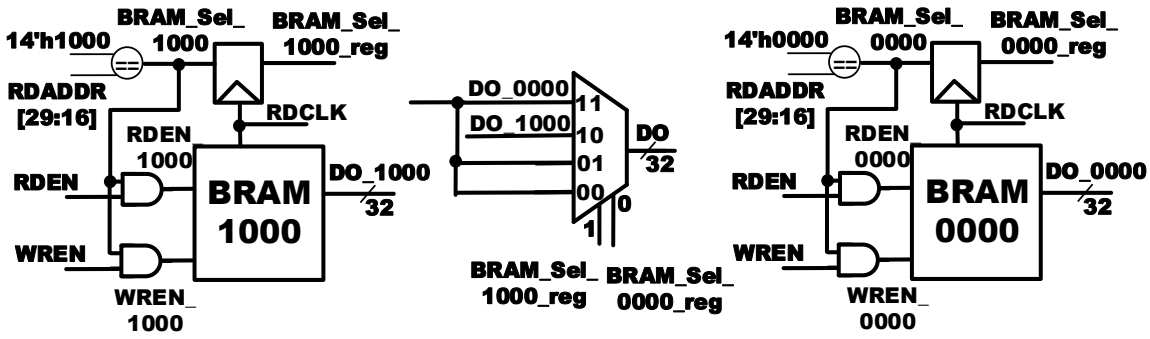

Figure 3-8 Functional block diagram of the data memory using BRAM

The sequence of read operation on instruction and data memory is shown in figure 3-9. The BRAMs in the instruction and data memory use a clock signal which has the same frequency but 180 degrees out of phase with respect to the bus clock signal of the HERMES. Initially, the processor places the instruction or data address on the address bus and asserts the instruction or data read signal. Then, instruction or data is read on the rising edge of the BRAM clock. The read data is placed on the read data bus of the processor for one clock cycle so that the processor can capture the data on the rising edge of the bus clock of HERMES.

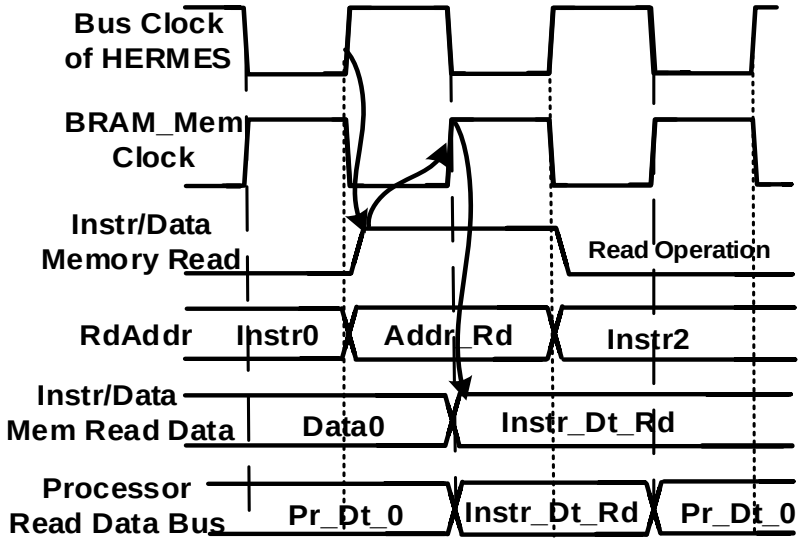

Figure 3-9 Timing diagram of the read operation on the instruction and data memory

The sequence of the write operation on the data memory is shown in figure 3-10. The processor places the write address and data on the respective buses and asserts the write signal for a clock cycle of the bus clock of HERMES. Then, the write operation is performed on the following rising edge of the BRAM clock.

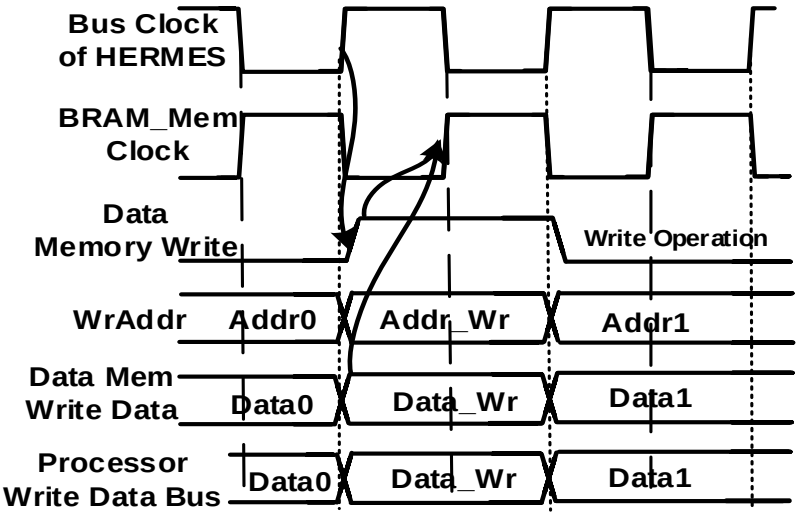

Figure 3-10 Timing diagram of the write operation on the data memory

3.2.3. Control Logic

The control logic generates the read, write signals to the BRAMs in the data and instruction memory depending on the assertion of EB_AValid, EB_Instr and EB_Write signals from the processor as shown in figure 3-11. This block contains a 4:1 mux which decides whether data read from instruction memory or data memory be placed on the HERMES read data bus, for one clock cycle of bus clock of the processor. The bus-core-clock ration is also set by this mux at the start of any test, when the processor and test bench are reset by SI_ColdReset signal. The write data bus from the processor is connected to the write bus of the BRAM in the data memory.

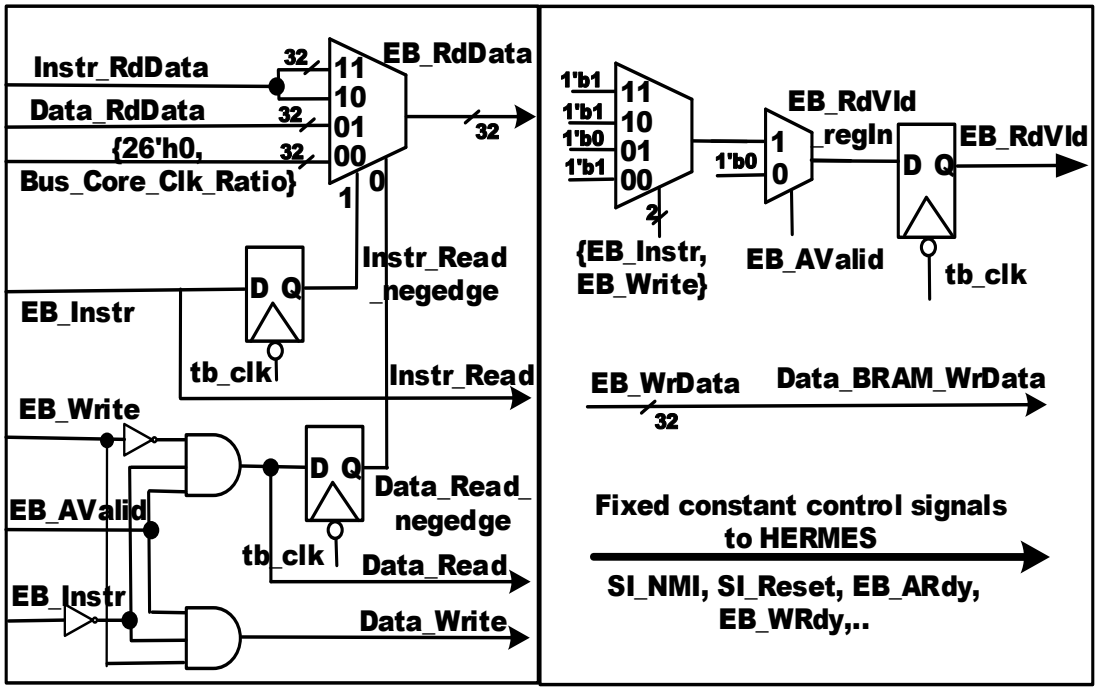

Figure 3-11 Functional block diagram of the control logic of the test bench

3.3. Testing of the HERMES processor

3.3.1. Trace sample of a test run

The trace sample of a test run on HERMES processor in big endian configuration is shown in figure 3-12. The multibit and bus signals such as EB_RData, EB_WData, EB_A and EB_BE are displayed in hexadecimal format whereas the single bit signals are displayed in binary format. The signal, Fixed_Value in the trace corresponds to the grouping of the static input signals to the processor. The left most signal in the trace is the clock signal to the XOR clock multiplier which produces clock to the processor. The right most signal of the trace corresponds to the write data bus displayed in character format to help in the "Hello World" test run on the processor.

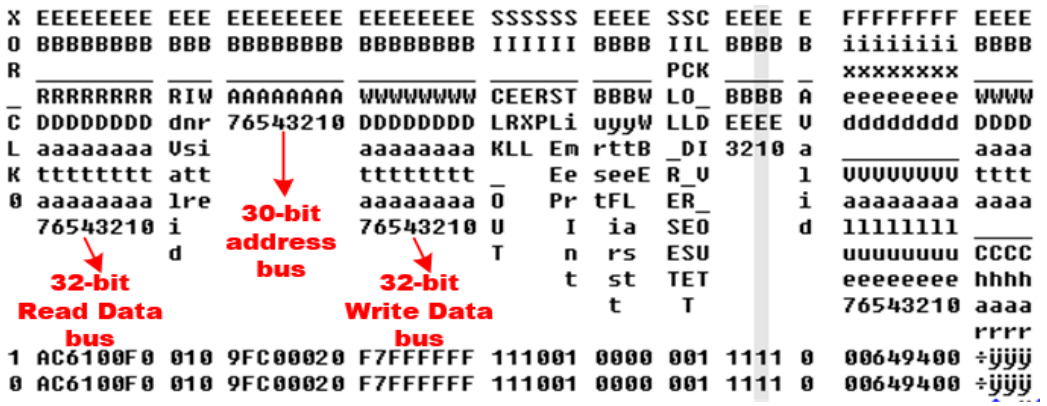

Figure 3-12 Trace sample of a test run on the HERMES processor

3.3.2. Hello World Test

The objective of the hello world test is to make the processor display "Hello World" continuously on its write data bus. The test makes the processor read the preloaded "Hello World" from the data memory and store it back in an infinite loop. The "Hello World" is

split as “Hell”, “o Wo”, “rld “and preloaded at three different memory locations of the data memory since each location can only store 32-bits. The processor reads these three locations one after another into local registers and then writes them back in the same order to the data memory. The sequence of steps followed in this test is given below

1. Initialize all the registers ($1 - $31) with value 0.
2. Have two registers ($9, $10 in this case) store two values with a difference of 12. This difference determines the number of times the inner loop is executed.
3. Have the start location of “Hello World “in the data memory stored in another register ($12 in this case).
4. An Inner loop:
    a. Load the data from the location specified by the contents of $12.
    b. Store the same data to the same location.
    c. Increment the value stored in $12 by 4 to make it point to the next location.
    d. Increment value of $9 by 4 to complete one iteration of inner loop.
    e. Compare the contents of $9 with $10 to check if the inner loop needs to be reiterated or not.
    f. In case the comparison fails, jump to the start address of outer loop.
5. Execute Steps 2 and 3 indefinitely.

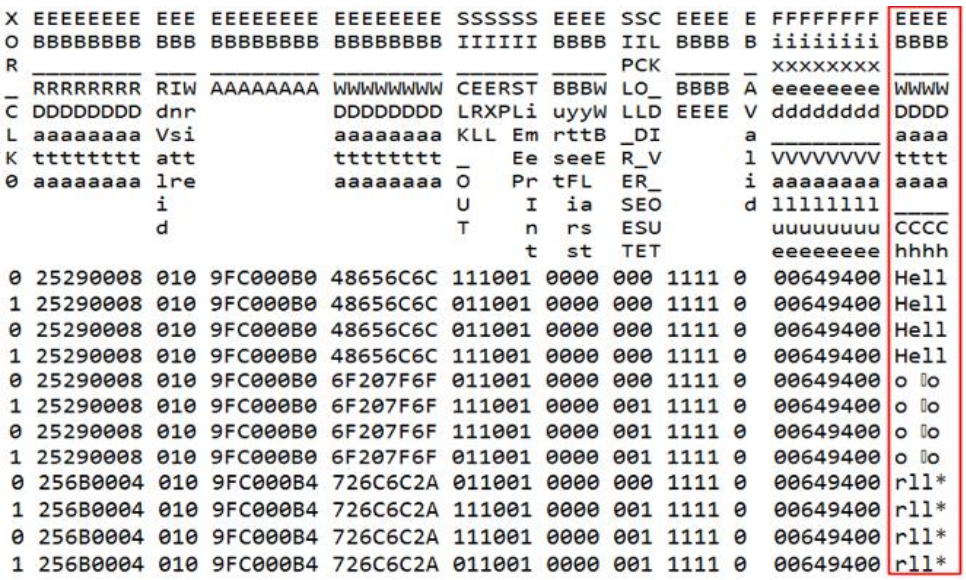

Figure 3-13 The output trace of the HERMES displaying "Hello World"

The processor successfully displayed "Hello World" onto the write data bus as shown in figure 3-13. Even though "Hello World" is correctly mapped to its binary value and placed on the read data bus, the processor can't display "W" and "d "on its write data bus. This is due to the shorted net on the 6th bit of the read data bus.

3.3.3. Data and Instruction Cache test

The I-cache and D-cache of the HERMES processor are validated by running a test with the following sequence of steps:

1. After initial reset sequence, the processor starts at address (32'hBFC0_0000 or 30'h9FC0_0000) that lies in the kseg1 segment of the memory.
2. Then the test turns on the data and instruction caches by making the kseg0 segment of memory cacheable. This is achieved by clearing the kseg0 bits in the configuration register as shown in figure 3-14.

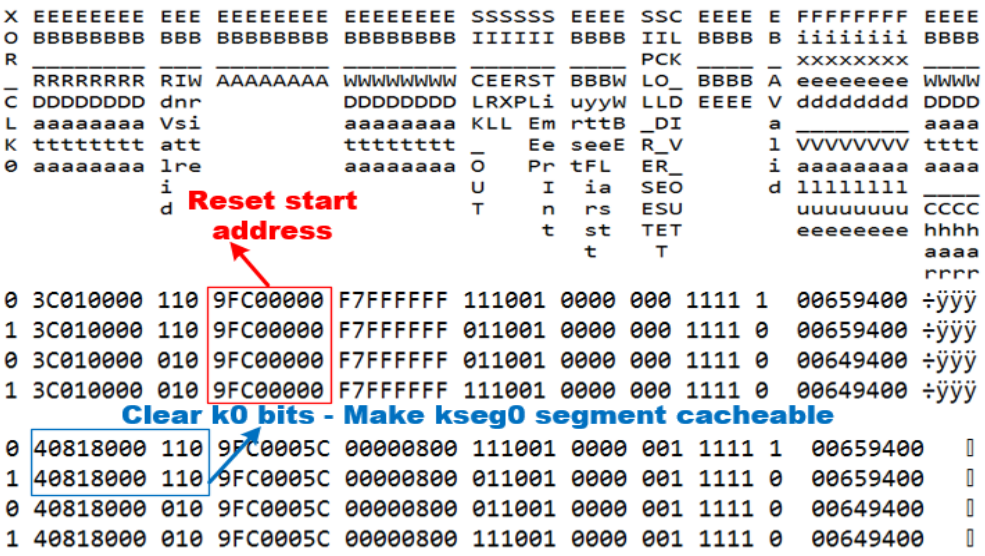

Figure 3-14 Output trace of steps 1 and 2 of I-cache and D-cache test

3. The program control is shifted from kseg1 to kseg0 segment of instruction memory, since instructions from kseg0 segment are cacheable and are stored in I-cache. To achieve this a mix of jump and branch not equal instructions are used, since jump register instruction can't be identified by the processor due to the shorted net on the read data bus.
4. All the registers ($1 - $31) are initialized with value zero. Then, read and write operations are performed on an address location in the data memory as follows:
    a. Load the address location, in kseg0 segment (32'h9000_0000) of data memory into $1 register.
    b. Write any 32-bit data (32'hF33F_FB3F) to that memory location.
    c. Read the data back from the same memory location.
    d. Modify the data read from that memory location (by adding 16'h0030).

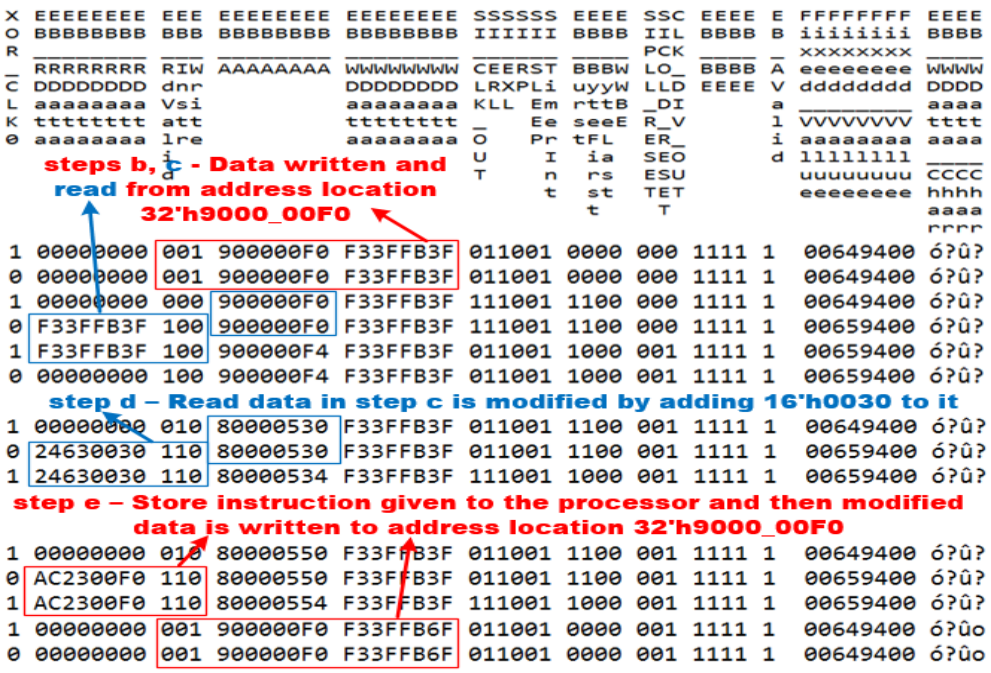

Figure 3-15 Output trace of steps b - e of I-cache and D-cache test

e. Write the modified data back to the same memory location.

f. Repeat steps c – e infinitely.

During the first iteration of the above procedure, both data and instructions are accessed from the data and instruction memory respectively. In the subsequent iterations, both the instructions and read data are accessed from the I-cache and D-cache respectively as shown in figure 3-16. The caches are write through and hence, the modified data when written back is observed on the write data bus. The read data is modified and written back to check whether the D-cache stores the updated value or not. The read data bus and address bus of the processor doesn't change value while the write data bus has updated value of the data written in every iteration. This confirms the access of I-cache and D-cache in the processor.

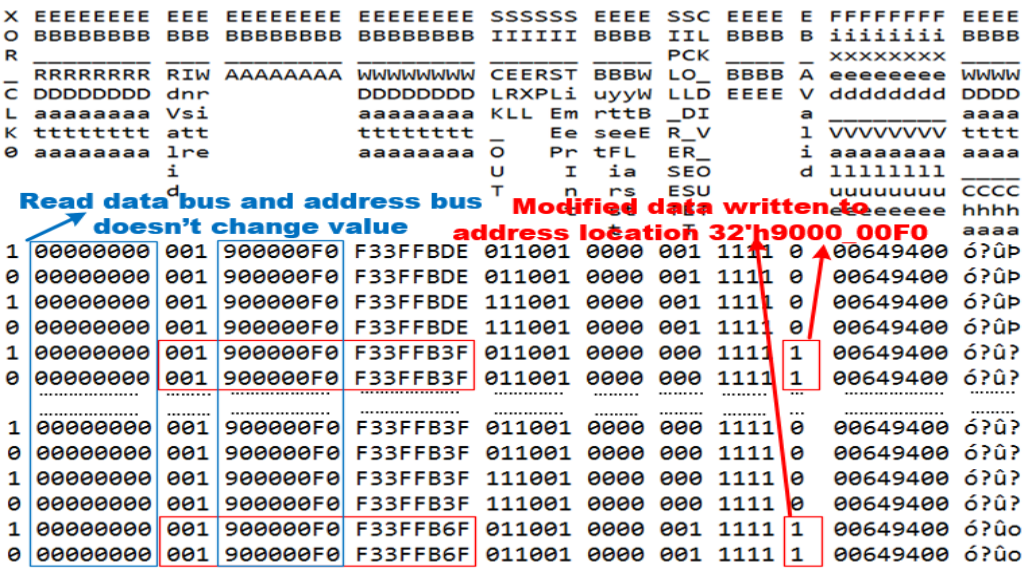

Figure 3-16 Output trace indicating no change in value of address and read data bus

3.3.4. Speed Test

The HERMES processor works using two clocks, the external bus clock and the core clock. The core clock is usually run at higher frequency compared to the bus clock. Hence, the core clock signal is generated by using the XOR clock multiplier, while the bus clock signal is driven directly from the I/O pads. The HERMES processor is found to be working at a core clock frequency of 200MHz but not functional at 400MHz. This is due to the alteration of the phase relationship between the XOR clocks and the bus clock due to different wire lengths on the PCB.

Test run	Test Result	Bus Clock Frequency	Core Clock Frequency
hello world test	Pass	10MHz	40MHz
hello world test	Pass	25MHz	100MHz
hello world test	Pass	50MHz	200MHz
hello world test	Fail	100MHz	400MHz

Table 3-1 The various frequencies of operation of HERMES on TT09 at VDDH = 0.9V

CHAPTER 4. POST SILICON TESTING OF THE SRAM BLOCK

4.1. Test Setup of SRAM Block

4.1.1. Test Bench

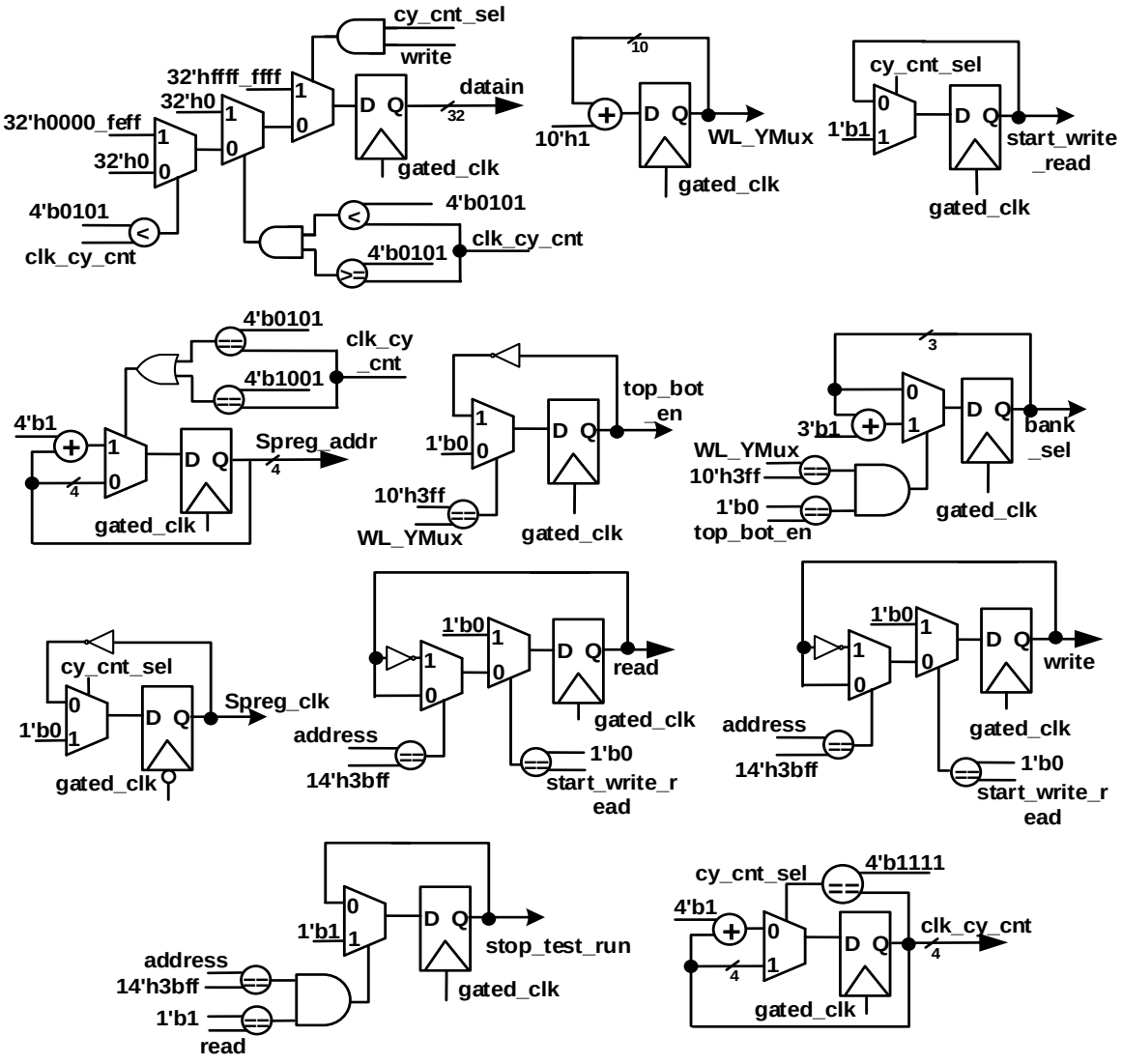

Figure 4-1 Functional block diagram of the test bench of the 1Mb SRAM 6T array

The functional block diagram of the test bench of the SRAM block is shown in figure 4-1. The SRAM block is configured using three special registers. Out of the three

special registers present, the first register is used to enable the test modes and program the sense amplifier delay, the second and third registers are used only in the DAT mode. The test bench initially configures the special registers to either do normal read and write operations or to enable the test modes. The special registers have a separate clock and address signal. The 32-bit data input to the special registers block is shared with the data input to the SRAM arrays.

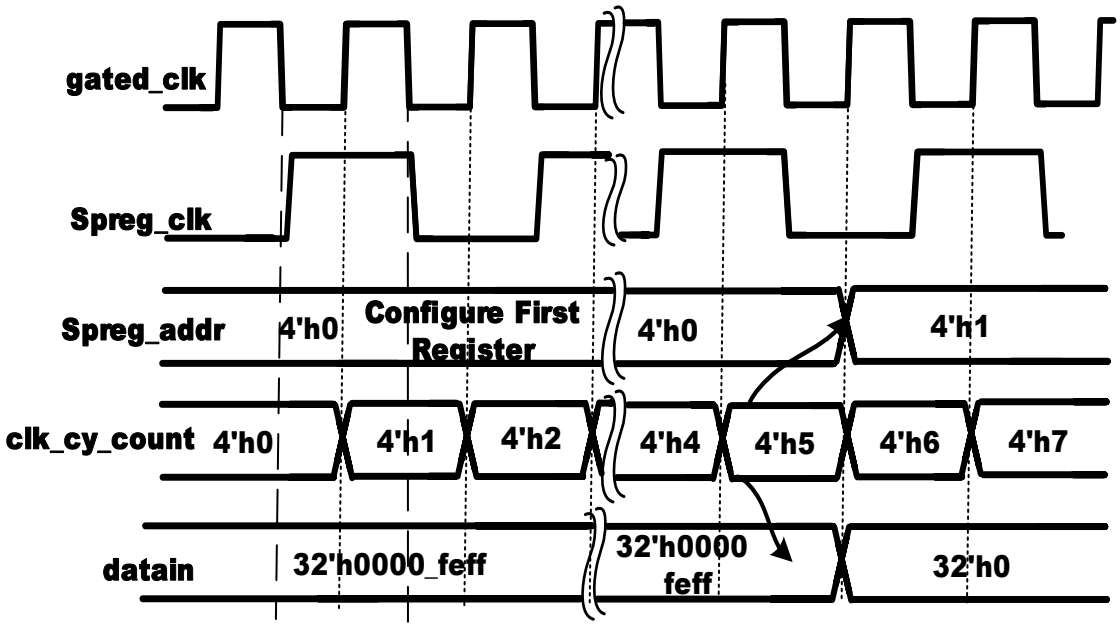

Figure 4-2 Timing diagram of the configuration of the first and second special registers

Once the special registers are configured, then the start_write_read signal is asserted when the clock cycle counter reaches value 15. This is used by the address generation, write and read logic to perform write and read operations on either one of the SRAM 6T arrays depending on the value of the SRAM_Sel signal as shown in figure 4-3.

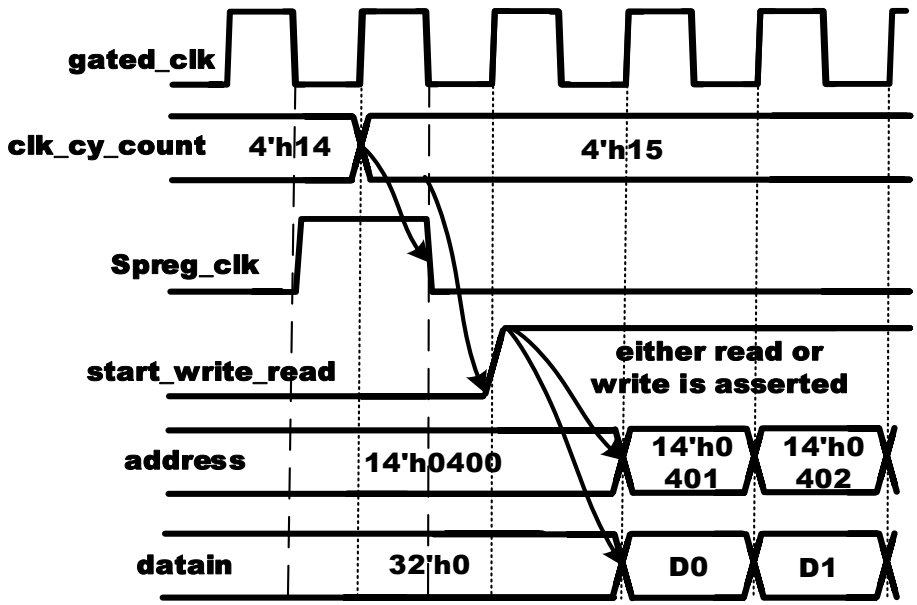

Figure 4-3 Timing diagram depicting the write and read operations on SRAM 6T array

The address generation logic starts with first bank and word line 0 at address 14'h0400 and ends with the last bank and word line 127 at address 14'h3bff. This covers all the 16384 addresses of 1Mb SRAM array. For either the read or write operation performed, the whole address range of a 1Mb SRAM array is covered. Depending on the type of the test needed the data input is driven with the appropriate value. For example, in the case of write all 1s test, input data is driven with all 1s for the whole address range. The stop_the_test_run signal is used to stop the test run. This signal is asserted when the test finishes its operations. This is communicated to the program running on the PC using an instance of okwireOut module which then stops the test.

Depending on the type of test required, either write signal or read signal is asserted but not both at the same time. In the case of write all 1s then read test, first the write signal is asserted and then the read signal is asserted for the whole address range. After the read

operation is performed on the last address, the stop_the_test_run signal is asserted as shown in figure 4-4.

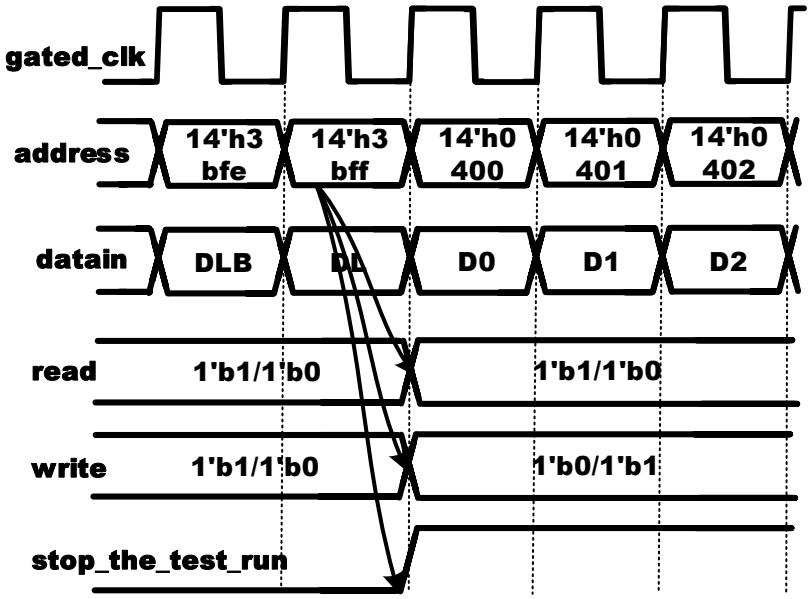

Figure 4-4 Timing diagram depicting the end of the test of SRAM 6T array

4.1.2. Automatic Control of Agilent E3646A DC power supply

As mentioned, TC25 has ten distinct power connections controlled by five agilent E3646A power supplies. Out of these ten, five of them need to be changed to run different tests on the SRAM block. Depending on the type of test and the corner part used, different combinations of these five voltages are required. However, few of the tests like the PUF mode test or the read minimum voltage test must be run for many number of iterations (in the range of 5-50). Most importantly, these voltages must be changed in between the iterations of these tests. Changing the voltages manually is a painful task, given that the tests are run again and again for many iterations. Hence, the power supplies are automatically controlled from the PC.

The agilent E3646A power supply has a recommended standard number 232 (RS-232) interface through which it can be controlled. The PC has universal serial bus (USB) ports to talk to the power supply. The PC communicates with the power supply using the USB-female serial connector and the 9-pin (DB-9) male-male serial connector and as shown in figure 4-5. To achieve this two handshake signals, data terminal ready (DTR) and data set ready (DSR) on the 9-pin connector are used. The RS-232 interface on the power supply as well as the appropriate USB port on the PC is configured with the parameters related to data frame and transfer rate as shown in the table 4-1.

Parameter	**Value set**
Baud Rate	9600
Parity bits	None
Data bits	8
Number of start bits	1 bit
Number of stop bits	2 bits

Table 4-1 RS-232 configuration settings of the power supply

The power supply is controlled using standard commands for programmable instruments (SCPI) commands sent over the serial interface. All these commands are provided to the power supply through an API for windows written in perl. This perl API module (Win32::SerialPort) is available on the comprehensive perl archive network (CPAN) database [Cpan10]. Using this API, a new perl module "control_the_power_supply" is written stitching the commands needed to control the voltages of the power supply into sub routines [agilent13]. This module instantiates and configures the serial port along with subroutines to reset the power supply and write user defined values to both the voltages of the power supply as shown in figure 4-5. This module is included in the perl scripts used to run different types of tests on the SRAM block.

```
sub out1_out2_power_supplies_voltage_level_control
{
        # These input arguments specify the voltage values on the OUT1 and OUT2
        # nodes of the power supply
        $config_volt_out1 = shift;
        $config_volt_out2 = shift;

        # Now select out1 explicitly
        print "Selecting OUT1 node on the power supply \n";
        write_power_supply ( "INSTrument:NSELect 1\n");
        print "Selecting OUT1 node to $config_volt_out1 on the power supply \n";
        write_power_supply ( "VOLT $config_volt_out1\n");

        # Now select out2 explicitly
        print "Selecting OUT2 node on the power supply \n";
        write_power_supply ( "INSTrument:NSELect 2\n");
        print "Selecting OUT2 node to $config_volt_out2 on the power supply \n";
        write_power_supply ( "VOLT $config_volt_out2\n");
}
```

Figure 4-5 subroutine to write user defined values to both the voltages of the power supply

4.1.3. Voltages of the SRAM 6T array

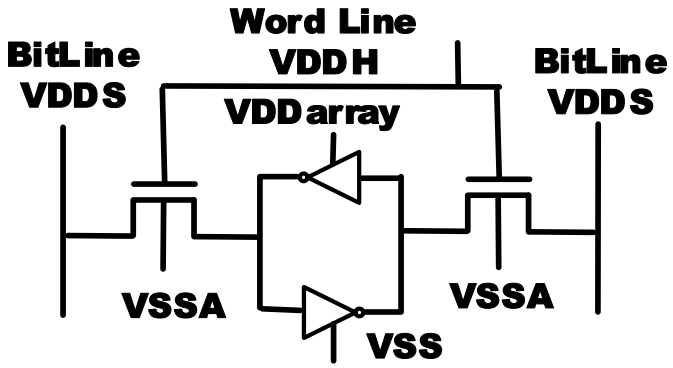

Figure 4-6 Voltages used in the bit cell of the SRAM 6T array

The bit cell of the SRAM 6T array requires six different voltages as shown in figure 4-6. Out of the six voltages, two of them are the body voltages of the PMOS transistors (VDDAarray) and the NMOS transistors (VSSA) used in the array. The body voltages are set different for different corners. They help in adjusting the threshold voltage (V_t) of the transistors. The remaining voltages are the supply voltage to the back to back inverters (VDDarray), supply voltage used to charge the PMOS transistors of the bit lines

(VDDS), word line voltage (VDDH) and ground supply voltage(VSS). The VDDarray, VDDH and VDDS voltages are altered to de-stabilize the bit cell in the PUF mode. The typical values of these voltages are given in table 4-2.

Voltages	Typical values
VDDH	0.8V
VDDarray	0.9V
VDDS	0.9V
VDDAarray	1.3V
VSSA	-0.4V

Table 4-2 Typical values of the voltages used in the bit cell of the SRAM 6T array

4.1.4. Test Data Acquisition

The manufactured test chips received from Fujitsu are 250 in number. There are distributed as 50 each across five process corners namely slow-slow (SS), fast-fast (FF), fast-slow (FS), slow-fast (SF) and typical-typical (TT). The chips of the TT corner are marked as TT01, TT02…TT50 and the other corner parts are also marked similarly. There are different tests like PUF mode test, read minimum voltage test, write all 1s then read test that need to be run on these parts. Also, these tests are run across different combinations of the five different voltages used in the bit cell. Moreover, the date and time of any test run on these parts, needs to be noted in the log file. Therefore, to uniquely identify any test run on any part and on any date and time, perl scripts are setup to acquire the test data.

The perl script needs the input arguments corner part number, type of test, voltages set and frequency of the SRAM clock before the test is run. The script needs only those voltages that are altered from their typical values. Each type of test has a specific string attributed to it and has a separate run directory. In each run directory, bit stream files

corresponding to different frequencies of the SRAM clock are present. The generic template of any test run perl script used for data acquisition contains four steps described as follows:

1. The first step involves checking the input arguments such as corner part number, type of test, voltages set and frequency of the SRAM clock entered by the user of the perl script. The string of the process corner entered needs to be either SS, FF, TT, SF or FS and the part number entered can't exceed 50 since only 50 parts are available for each corner. Similarly, the voltages entered can't exceed or go below certain values. For example, in the case of VDDarray the voltage entered can't exceed 1.3V which is the burn-in voltage. Similarly, the string entered for the type of test needs to be among the valid set of strings specified for different types of tests. Also, the frequency of SRAM clock entered must be either 5MHz or 10MHz or 20MHz as the bit stream files only for these frequencies are made available in the test run directory.
2. The second step involves extracting the information from the input arguments of the above step into global variables which are used later. For example, the voltages entered by the user are updated in the hash table of the voltages. The date and time of the test run is captured into their respective global variables.
3. The third step is used to run the test in the specific directory depending on the input argument, type of test. The information about the test run, obtained and stored in the global variables, is used to construct the name of the log file. This log file stores the data of the test run.

4. The fourth step is used to copy the log file to a location and delete the log file in the run area.

The perl module “data_acq_base_lib” is the base library which contains the global variables date_format_used, time_format_used, corner_part_number, type_of_test_run, PortObj_Com, hash table of voltages and hard coded path of test run directories. This library also has subroutines related to steps 1 and 2 which are common to any test run perl script and steps 3 and 4 which vary depending on the test run. The wrapper scripts for any test are built when the test needs to be iterated multiple time.

4.1.5. Test Data Processing

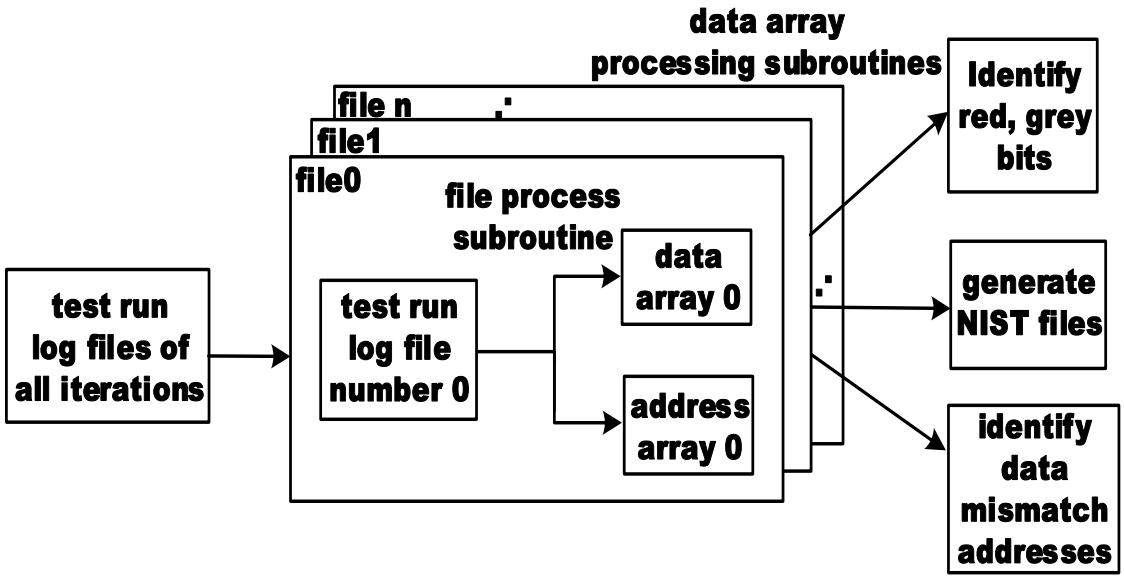

Figure 4-7 Block diagram depicting how the test run log files are processed

The log files of the tests run to read the SRAM array need to processed to capture the results of the test. This processing is achieved easily using perl scripts. All the data processing perl scripts have a common subroutine to capture the addresses and the corresponding data into two different arrays as shown in figure 4-7. This is repeated

depending on the number of the iterations of the same test run. Then the data stored in the arrays is processed as per the need and the output is written out to different files

Depending on the type of data written to the SRAM array, the data read should match it. The data patterns written to all the addresses of the SRAM array are, all the data input bits are 1s or 0s and the data input bits are same or inverted as the corresponding address bits. Therefore, when the data read is processed, the actual data should match the expected data. This comparison helps in identification of the bad addresses in a chip and is one type of processing done by the perl scripts. Similarly, the data read from multiple log files is compared to identify the grey bits, bits that are inconsistent across multiple runs of the same test.

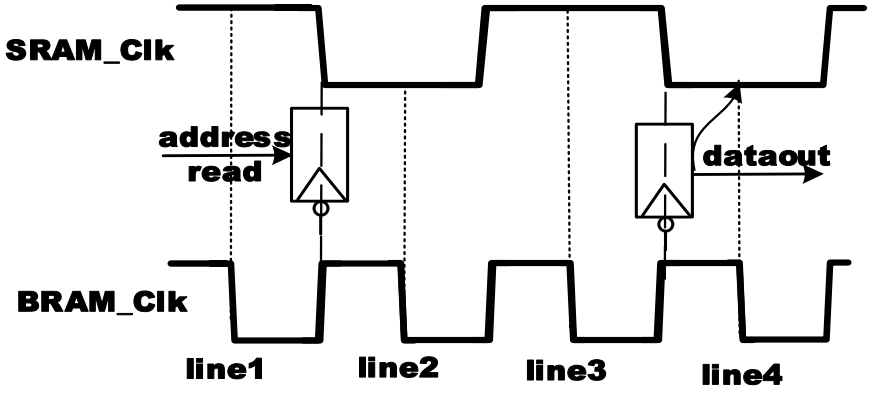

Figure 4-8 The timing relationship between the address read and data received

The log file of the read test, has all the addresses read and the corresponding data received from those addresses. For any address stored in a line in the log file, the corresponding data output received is stored four or more lines later depending on the frequency of the SRAM clock. The data output sent by the SRAM array must travel through the PCB trace before it is captured by the BRAMs on the FPGA. This PCB trace delay is observed to be around 20ns. Therefore, if the frequency of the SRAM clock is low (5MHz)

then the PCB trace delay is masked and the data output is stored four lines later in the log file as shown in figure 4-9.

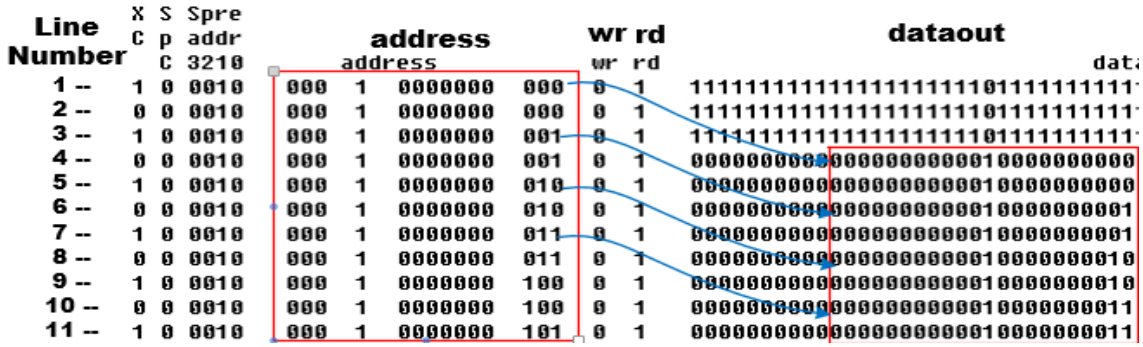

Figure 4-9 Trace sample of the test run at 5MHz with data out same as the address read

4.2. Tests run on SRAM 6T array

The tests run on the SRAM 6T arrays include the write read test, the minimum read voltage test, the data retention voltage test, the power up read test and the PUF mode test. These tests are run at a clock frequency of 5MHz and the longest setting of the sense amplifier delay (16'h00ff) to make sure that most of the bit cells don't fail during a read operation. However, different combinations of the voltages are used in all the tests.

The write read tests are run to validate the part based on which further testing is done on the part. Depending on the type of the corner, the reverse body bias (RBB) voltage is adjusted to make sure that normal write and read operations are performed with minimal failures. This voltage is adjusted to bring the V_t of FF and SS corners towards the V_t of the TT corner. Therefore, the RBB for normal write and read operations is set to 0.7V for the FF corner parts and 0.1V for the SS corner parts.

4.2.1. Write Read Test

Type of data pattern	Data input to the SRAM array
all 1s	32'hFFFF_FFFF
all 0s	32'h0
data same as address	{18'h0, 14-bit address}
data inverted as address	{18'hF_FFFF, inverted 14-bit address}

Table 4-3 Different types of data written to the SRAM 6T array

The write-read test is the first test run on any new part to validate the SRAM 6T arrays in it. Initially this test is run at nominal voltages to check the part. This test writes a specific data pattern to all the addresses of the 1Mb SRAM 6T array. The data patterns written consist of two complementary pairs as shown in table 4-3. Then, the read operation is performed on all the addresses one after another and the data read is compared with the expected data.

The write-read test validates the SRAM array on the part by writing separately both 1 and 0 on each bit cell of the array and reading the bit cell back to confirm the value written. To achieve this one of the complementary pairs described in table 4-3 are used. Initially, during the first iteration of the write read test, all 1s data pattern is used to perform the write operations. Then during the second iteration of the write read test, the complementary data pattern all 0s is used. The read data is checked in both cases to confirm the working of the SRAM array on the part.

All those addresses whose expected data doesn't match the actual data are treated as bad addresses, since they can't be used to reliably store the data. The write read test helps in identifying the bad addresses of an array on a part. This test run with complementary data patterns is iterated multiple times (50) to identify the bad addresses.

The number of iterations of the test confirms the consistency of the bad addresses on a part. Out of the all the identified bad addresses at the minimum read voltage of various parts, two of them per part are shown in table 4-4.

Part Number	Two bad addresses
FF06	14'b 000_1_0000000_001, 14'b 000_0_0000000_010
SS09	14'b 000_1_0000000_001, 14'b 000_1_0100000_001
TT09	14'b 000_1_0000000_001, 14'b 000_0_0000000_001

Table 4-4 Two bad addresses at the minimum read voltage of various parts

4.2.2. Minimum Read Voltage Test

The minimum read voltage test is used to identify the voltage operating point of the SRAM 6T array below which the data can't be read from the array reliably. The sequence of steps followed to run this test is as follows:

1. Write operation using either all 1s or all 0s data pattern is performed on the SRAM 6T array at nominal operating voltages of VDDarray, VDDH and VDDS. The RBB voltage is set to 0.2V, 0.4V and 0.7V for the SS, TT and FF corners respectively.
2. Then, the read operation is performed on the array and the number of bit failures is noted at the voltage combination specified in the above step.
3. The voltage value of VDDarray and VDDS is decreased by 1mV and the value of VDDH is adjusted as per the new value of VDDarray. (0.889 * voltage value of VDDarray). This adjustment is based on the nominal voltage relationship between VDDarray (0.9V) and VDDH (0.889 * 0.9V = 0.8V).

4. Then read operation is again performed on the array and the number of bit failures is noted at the new voltage combination.

5. Steps 3 and 4 are repeated continuously until VDDarray reaches 0.4V.

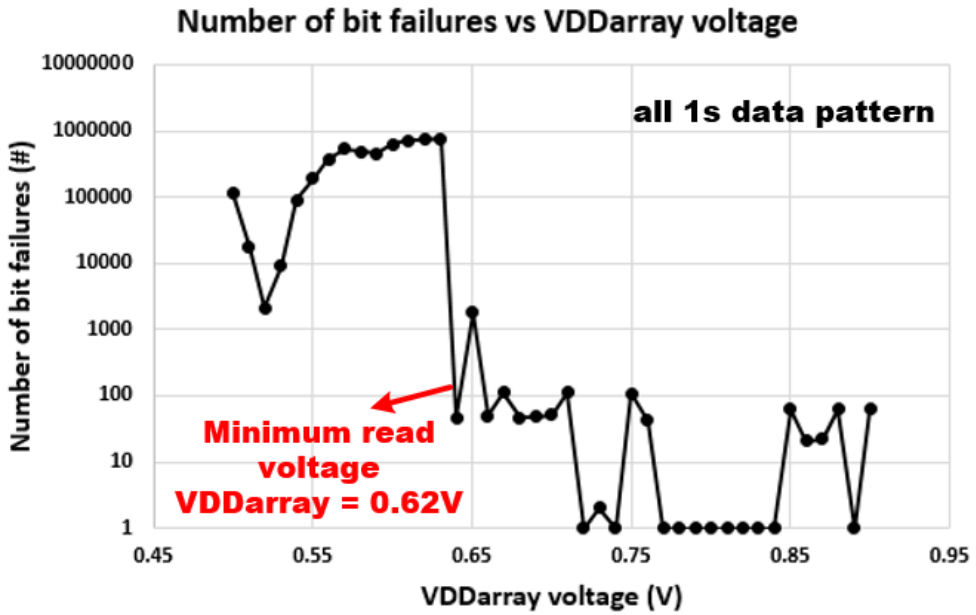

Figure 4-10 The plot of number of bit failures vs VDDarray voltage of SS09

The voltage combination below which bit failures in the range of thousands and above which bit failures in the range of tens or hundreds is identified for both all 1s and all 0s data patterns. Then, the largest of these voltage combinations is identified as the minimum read voltage for the part. The bit failures are observed to be cumulative in nature. The minimum read voltages for FF06, SS09 and TT09 parts are shown in table 4-5.

Part Number	VDDH	VDDarray	VDDS	RBB
FF06	0.54V	0.61V	0.61V	0.7V
SS09	0.55V	0.62V	0.62V	0.4V
TT09	0.58V	0.66V	0.66V	0.2V

Table 4-5 Minimum read voltages of FF06, SS09 and TT09 parts

4.2.3. PUF Mode Test

The SRAM bit cells have a built-in mismatch due to the variations of the fabrication process [Chellappa11]. This is manifested as transistor threshold voltage mismatch. The bit cells are also subjected to random telegraphic noise (RTN). The impact of RTN is especially predominant in the lower technology nodes due to aggressive scaling of supply voltages and transistor sizes [Mao16]. Both the RTN and the process mismatch are important factors that help in using SRAM to assign a unique finger print to the manufactured IC.

The bit cell of the SRAM has two back to back inverters and can take two possible stable states (10 and 01). The bit cell is in an unstable metastable state right after power-up or during the PUF mode of the SRAM. When the bit cell is in the metastable state, both the RTN and the process mismatch decide the tilt towards one of the stable states of the bit cell [Chellappa11]. Similar tilt towards a stable state happens on all the bit cells of the SRAM. Few of the bit cells are well matched and therefore have their stable states determined primarily by RTN. These bit cells contribute to grey bits and can be used to generate random numbers. Few of the other bit cells with large process mismatch favor a stable state not affected mostly by RTN. These bit cells can be used to generate a number unique to the IC. Hence, the state of the SRAM right after power-up or PUF mode of operation can be used to assign a unique finger print to the IC [Chellappa11].

The PUF mode involves de-stabilizing the bit cells of the SRAM. During this mode, the supply voltage of the bit cell is reduced, the NMOS access transistors of the bit cell and the pre-charge PMOS transistors of the bit lines are turned on and ymux is turned off. Now,

by decreasing the supply voltage, the static noise margin of the bit cell is degraded and this de-stabilizes the bit cell. The number of the bit cells de-stabilized is determined by the amount of the supply voltage reduction. However, the bit cells which are still stable contribute to the number of red bits. The de-stabilized bit cells tilt towards a stable state depending on RTN and process mismatch variations. These states of the bit cells can be used to generate random numbers and provide a unique number to the IC.

The PUF mode test run on the SRAM 6T array involves the following sequence of steps

1. Write all 0s or all 1s data pattern to all the address locations of the array at the nominal voltage combination and RBB depending on the corner.
2. Once the write operation has finished, voltages VDDH, VDDarray, VDDS are set as needed by the PUF mode of operation of the SRAM array. The RBB voltage is set to the typical value of 0.4V for all corners to keep the corner as is.
3. Then, turn on the PUF mode of the SRAM array by setting the bit number 31 of the first special register to logic “1”.
4. Now, read operation is performed on all the addresses during PUF mode. This operation turns on the word line of the bit cell and destabilizes it.
5. Then, voltages VDDH, VDDarray, VDDS and RBB are changed to minimize read failures during normal read operation on the SRAM array.
6. Normal read operation is performed on all the addresses of the SRAM array and the read data is recorded for analysis.

The difference between the voltages VDDH and VDDarray is defined as V_{diff1}. Similarly, the difference between the voltages VDDS and VDDarray is defined as V_{diff2}. Both V_{diff1} and V_{diff2} are used to measure the amount of de-stability imparted on the bit cell during PUF mode. The various steps of the PUF mode test are short hand denoted as follows

1. step 1 involving write all 0s or write all 1s with "w0" or "w1" respectively.
2. step 4 describing the read operation during PUF mode with "p".
3. step 6 performing the normal read operation with "r".
4. The number of iterations of the test is indicated using x followed by the number. For example, "x50" in the case of 50 iterations.

The name of any variation of the PUF mode test is built using these short hand notations and underscores. For example, when the PUF mode test described above is run 50 times, it is denoted as "(w0_p_r)x50" or "(w1_p_r)x50".

Write value	Read value on every iteration	Type of the bit
0 or 1	0	White
0 or 1	1	Black
0 or 1	0 or 1	Grey
0 and 1	0 and 1 respectively	Red

Table 4-6 The type of bit cells determined after multiple runs of the PUF mode test

The first goal of the PUF mode test run is to de-stabilize all the bit cells of the 1Mb SRAM array. This translates to an ideal target of zero red bits because a bit cell once de-stabilized favors either stable logic "1" or logic "0" state and hence can't remain a red bit. This test is run 50 times to average out the effect of noise and then red bits are calculated

based on the data read from all the iterations. However, the number of red bits are observed to be in the order of thousands.

To identify the cause of the red bits issue, circuit simulations on a single bit cell with voltage conditions resembling those observed during the PUF mode are performed. The issue is identified to be the amount of time each bit cell stays in PUF mode. The tests which are initially run at the SRAM clock frequency of 50MHz are now run at 5MHz. This makes sure that the bit cells stay in PUF mode for a long duration of time (~200ns). Then, the PUF mode test is again run 50 times and read data is analyzed. Significant decrease in the number of red bits is observed with the frequency change. However, the number of red bits are still present in the order of hundreds. Later, the same test is run with different values of V_{diff1} and V_{diff2} and the number of red bits are observed to decrease continuously as VDDS is increased as shown in table 4-7. Also, the red bit count of zero is observed at VDDS = 1.2V.

Test Run	V_{diff1}	V_{diff2}	RBB	Number of red bits
(w1_p_r)x50, (w0_p_r)x50	0.3V	0.3V	0.4V	5549
(w1_p_r)x50, (w0_p_r)x50	0.3V	0.4V	0.4V	129
(w1_p_r)x50, (w0_p_r)x50	0.3V	0.5V	0.4V	67
(w1_p_r)x50, (w0_p_r)x50	0.3V	0.6V	0.4V	0

Table 4-7 The number of red bits of the PUF mode test run on TT10 at different VDDS

The second goal of the PUF mode test is to generate good random numbers qualified by passing the statistical test suite developed by national institute of standards and technology (NIST). These tests determine the amount of non-randomness in the binary sequences constructed using the random number generators such as the PUF mode of SRAM. The number generated using PUF mode of SRAM should have an equal number

of 1s and 0s for it to be random. The previous runs of "(w0_p_r)x50" and "(w1_p_r)x50" tests have an unequal number of 1s and 0s. The number of white or black bits is observed to be a function of data pattern written to the SRAM array as shown in table 4-8. This is attributed to the data remanence of the SRAM bit cells.

Test Run	White bits (%)	Black bits (%)	Grey bits (%)
(w0_p_r)x50	43	33	24
(w1_p_r)x50	35	41	24

Table 4-8 The distribution of read bits of PUF mode test run on TT06

The single iteration of the PUF mode test involves taking the bit cells to a known state using write operation. Then, the bit cells are taken into PUF mode where the initial values written are altered. Now after the read operation, when the power is removed from the SRAM array, data stored in the bit cells is not lost completely. This data remembered by the bit cells affects the next iteration of the PUF mode test resulting in large number of grey bits. Therefore, to remove the data remanence in the read data, the "(w1_p_r)x50" and "(w0_p_r)x50" tests are modified by performing the write operation only once in the first iteration and then the PUF mode of operation followed by the normal read operation is repeated 50 times. Also, power is not turned off in between these iterations. The modified PUF mode tests run are "w1_(p_r)x50" and "w0_(p_r)x50".

The modified PUF mode tests are run on TT10 part and the number of 1s and 0s read after every iteration are analyzed. The percentage of white and black bits oscillate after every normal read iteration as shown in the figure 4-11. This oscillation is due to the well-matched bit cells, switching from one state to another for every iteration of PUF mode followed by normal read operation. This oscillation can be due to the PUF mode of

operation or the normal read operation of every iteration.

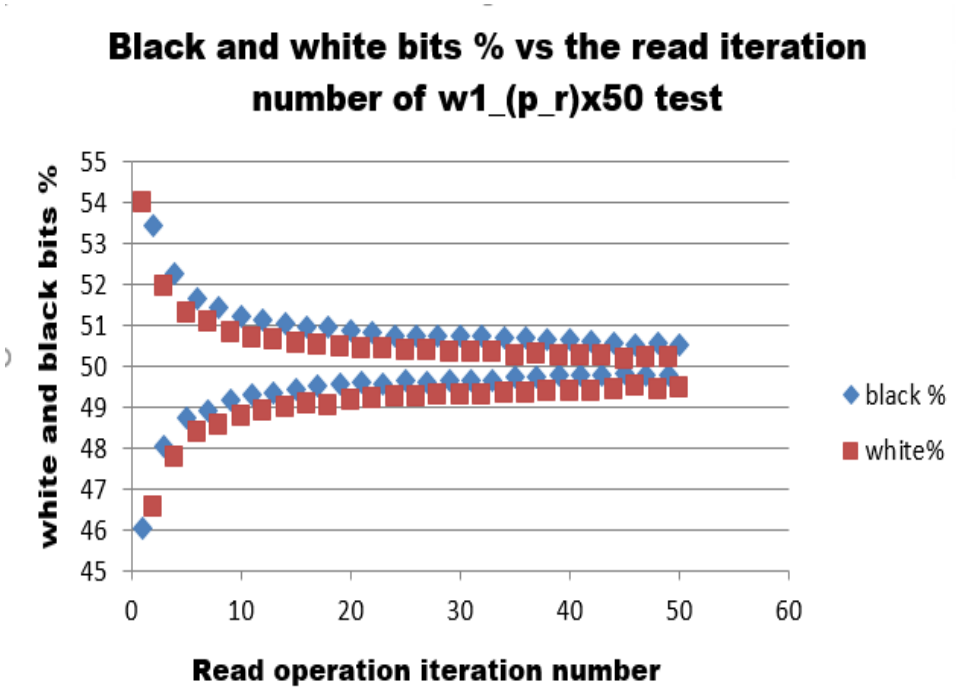

Figure 4-11 The percentage of black and white bits for each read iteration of w1_(p_r)x50

During the normal read operation, RTN could fill a transistor trap in the bit cell, changing their threshold voltages (V_t). This causes the bit cell to behave as a grey bit. The RTN is a function of the read voltages VDDH, VDDarray and VDDS of the bit cell. Therefore, the normal read operation is performed at minimum read voltage of the bit cell to potentially reduce its effect. This makes sure that the grey bits generated are more likely due to the PUF mode of operation. Hence, the latest PUF mode tests are run by performing normal read operation at minimum read voltages.

The latest PUF mode tests are run on the parts TT06, FF09 and SS07 at VDDarray voltages of 0.3V, 0.35V, 0.4V and 0.45V during the PUF mode of operation. The results show the decreasing order of the number of well-matched cells as FF followed by TT and

then SS as shown in table 4-9. Therefore, the bit cells of the FF part can be used to generate a good random number and vice-versa the bit cells of the SS part can be used to generate a unique finger print to the part.

Test Run	Corner Part Number	Grey bits (%)
w0_(p_r)x50, w1_(p_r)x50	SS09	14.6
w0_(p_r)x50, w1_(p_r)x50	TT09	26.3
w0_(p_r)x50, w1_(p_r)x50	FF06	50.2

Table 4-9 Percentage of grey bits at VDDarray = 0.35V and read at minimum read voltage

CHAPTER 5. CONCLUSIONS

The HERMES processor was tested using hello world and cache functionality tests. The processor successfully passed the hello world test which confirms that it can understand and execute instructions. The data and instruction caches of the processor are confirmed as functional using the cache functionality test. Both these tests are run on the processor at a frequency of 100MHz. Finally, the processor is found to be operational at a core clock frequency of 200MHz when run at nominal voltages, while it can run at a maximum core clock frequency of 450MHz confirmed by post-layout circuit simulations. Operation at this speed has not been confirmed on silicon as of this writing

The SRAM 6T arrays of the test chip 25 are tested using write read, minimum read voltage and PUF mode tests. The write read tests used different data patterns to confirm the storage functionality of all the bit cells of the SRAM array at nominal voltages. Also, these tests confirmed that the location at bank 0, word line 0 with ymux value 1 failed to store the data irrespective of the corner part and sense amplifier delay settings at nominal voltages. The minimum read voltage test confirmed the array read voltages as 0.61V, 0.66V and 0.62V for FF, TT and SS parts with body bias voltages of 0.7V, 0.4V and 0.2V respectively.

The PUF mode test confirmed the PUF test mode of operation of the SRAM 6T array by altering the known state of the bit cells of the array. The number of red bits are observed to be zero for TT10 part at VDDS = 1.2V and VDDarray = 0.6V. This confirms that all the bit cells are de-stabilized by the PUF mode. The SRAM array is read at minimum read voltages of respective corner parts to eliminate the grey bits generated due to RTN of the read operation. The number of grey bits are observed to be around 50%,

26% and 14% for FF, TT and SS parts respectively at $V_{diff1} = V_{diff2} = 0.55V$. These grey bits from the FF part are used to generate random bit sequences. These sequences pass the NIST tests with the assistance of helper functions, which are commonly used.

The automation using perl to control the power supplies, capture and process the test data of the SRAM arrays saved a lot of time and manual effort. However, more could be done. Separate bit stream files were used based on each test, frequency of the SRAM clock, sense amplifier delay setting. This could have been avoided and a single bit stream file can be used to perform all the tests on the SRAM array. The trace file of the test run had lot of information about the state of all the signals between the DUT and test bench. Once the test is confirmed as working all the signals which don't convey any information could have been taken out of the trace.

REFERENCES

[agilent13] Agilent, “Agilent E364xA Dual Output DC Power Supplies”, User Manual, 10th edition, August 2013.

[Barn06] Barnaby, H. J., "Total-Ionizing-Dose Effects in Modern CMOS Technologies," Nuclear Science, IEEE Transactions on, vol.53, no.6, pp.3103-3121, Dec. 2006.

[Chellappa11] S. Chellappa, A. Dey and L. T. Clark, "Improved circuits for microchip identification using SRAM mismatch," *2011 IEEE Custom Integrated Circuits Conference (CICC)*, San Jose, CA, 2011, pp. 1-4.

[Chellappa16] S. Chellappa and L. T. Clark, "SRAM-Based Unique Chip Identifier Techniques," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 4, pp. 1213-1222, April 2016.

[Cpan10] CPAN Perl module, “Win32::SerialPort”, ver 0.22, April 2010.

[Edward07] G. E. Suh, C. W. O'Donnell and S. Devadas, "Aegis: A Single-Chip Secure Processor," in *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 570-580, Nov.-Dec. 2007.

[Gogula15] A. R. Gogulamudi, L. T. Clark, C. Farnsworth, S. Chellappa and V. Vashishtha, "Architectural and Micro-Architectural Techniques for Software Controlled Microprocessor Soft-Error Mitigation," *2015 15th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Moscow, 2015, pp. 1-6.

[Gogula16] Anudeep R. Gogulamudi, “Post-silicon Validation of Radiation Hardened Microprocessor, Embedded Flash and Test Structures,” Master’s thesis, Arizona State University, 2016.

[Hind09] Hindman, N.D.; Pettit, D.E.; Patterson, D.W.; Nielsen, K.E.; Xiaoyin Yao; Holbert, K.E.; Clark, L.T.; , "High speed redundant self-correcting circuits for radiation hardened by design logic," Radiation and Its Effects on Components and Systems (RADECS), 2009 European Conference on , vol., no., pp.465-472, 14-18 Sept. 2009.

[Hind11] Hindman, N.D.; Clark, L.T.; Patterson, D.W.; Holbert, K.E.;, "Fully Automated, Testable Design of Fine-Grained Triple Mode Redundant Logic," Nuclear Science, IEEE Transactions on, vol.58, no.6, pp.3046-3052, Dec. 2011.

[Mao16] D. Mao, S. Guo, R. Wang, M. Luo and R. Huang, "Deep understanding of random telegraph noise (RTN) effects on SRAM stability," *2016 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, Hsinchu, 2016, pp. 1-2.

[Mavis02] Mavis, D.G.; Eaton, P.H.; , "Soft error rate mitigation techniques for modern microcircuits," Reliability Physics Symposium Proceedings, 2002. 40th Annual, vol., no., pp. 216- 225, 2002.

[Mentor04] Mentor Graphics, “ModelSim: Advanced Verification and Debugging”, v 6.0b, Nov 2004.

[MIPS00] MIPS, “MIPS32 4Kc Processor Core Datasheet,” pp. 1–30, 2000.

[MIPS01] MIPS, “MIPS32 4K Processor Core Family Software User’s Manual,” 2001.

[Opalkelly15] Opal Kelly Incorporated, “FrontPanel User Manual”, rev, Mar.2015.

[Rama13] C. Ramamurthy, “Chip Level Implementation Techniques for Radiation Hardened Microprocessors,” Master’s thesis, Arizona State University, 2013.

[Vash15] V. Vashishtha, L. T. Clark, S. Chellappa, A. R. Gogulamudi, A. Gujja and C. Farnsworth, "A soft-error hardened process portable embedded microprocessor," *2015 IEEE Custom Integrated Circuits Conference (CICC)*, San Jose, CA, 2015, pp. 1-4.

[XilClk16] Xilinx, “7 Series FPGAs Clocking Resources”, User Guide, UG472, Sept. 2016.

[XilDes16] Xilinx, “Vivado Design Suite User Guide”, Implementation, UG904, Dec.2016.

[XilMem16] Xilinx, “7 Series FPGAs Memory Resources”, User Guide, UG473, Sept. 2016.