Modular Injection System and Sampling Template (M.I.S.S.T) Design Report

Computer Engineering

California Polytechnic State University, San Luis Obispo

Froylan Aguirre

June 2018

Abstract

Digital systems are ubiquitous throughout modern life and their applications continue to grow. Thus system designers engineer and test modular systems to mitigate error rates. Smaller systems and their increasing importance in many applications demand the utmost reliability. Fault injection is the most common method used by researchers and engineers to test system reliability. However, most hardware fault injection implementations are ad hoc and only used to test a specific system or for specific tests. There is also software-implemented fault injection that adds overhead in the benchmark source code. The aim of this project is to develop a general use, fault injection hardware module that can be integrated into a digital system. This module would be easy to use and flexible for most reliability testing. This document explains the design of such a system.

Chapter 1

Overview

1.1 Introduction

The Modular Injection System and Sampling Template (M.I.S.S.T) is intended to be used as a SoC fault injector for a bus-based system architecture like most micro-processors today. MISST is able to run fault campaigns composed of a series of fault injections to a target DUT followed by sampling data from the DUT. After every sampling event, MISST resets the DUT to repeat the process with different faults. Users can configure MISST fault injection and sampling behavior via a memory mapped interface.

1.2 High Level Structural Overview

A high level view of a MISST core use case is shown in Figure 1.1. The MISST system is intended to be system independent, that is, not tied to any particular development board. However, the manner in which the MISST system communicates with a terminal device (generally a PC) will be board dependent and specific to a use case. Therefore the adapter module's responsibility is to act as an adapter between the MISST and the PC and DUT.

For most use cases, the MISST system will be connected to an adapter that handles communication with a DUT, usually an AMBA-based digital system, and a PC. A user configures the MISST system through the PC by writing to configuration registers on the MISST core. During a fault injection campaign, a user will receive sample data on the PC.

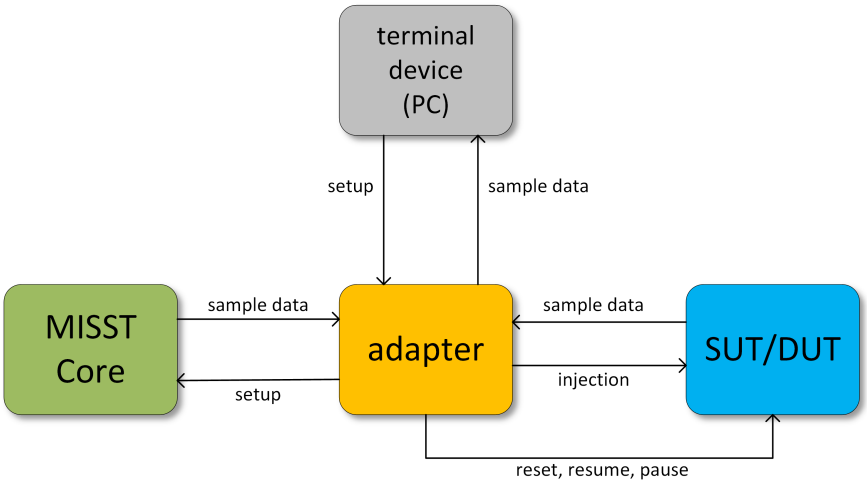

Figure 1.1: High Level View of a Use Case

1.3 Original Implementation

The MISST was originally developed on a PYNQ-Z1 Digilent development board using Vivado Xilinx tools. The PYNQ-Z1 board contains a ZYNQ 7000 family IC with a Cortex-A9 processor integrated with an Artix-7 equivalent FPGA [1]. See chapter 7 for PYNQ-Z1 board implementation details.

Chapter 2

MISST Module Overview

Overall system design schematics shown in Figure 2.1. The MISST core and supplemental modules are implemented in the FPGA fabric, and the Cortex hard processor being outside of the fabric. The AXI slave interface is not part of the core system modules, but specific to the PYNQ implementation. However, how the AXI slave connects to the system core and its required functionality are part of a standard explained in 7.3. The AXI slave interface is an example of an adapter module used to communicate with the MISST system independent of implementation.

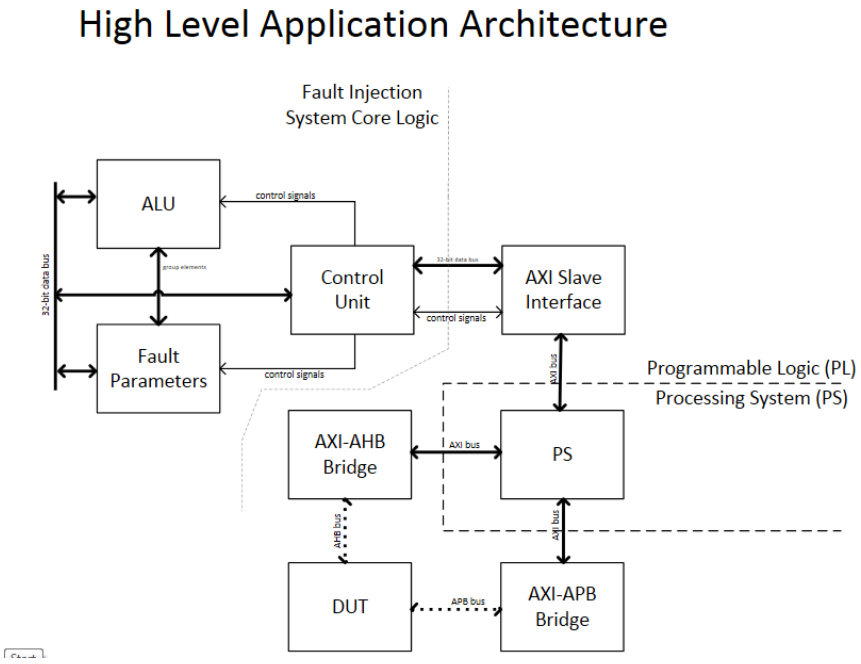

Figure 2.1: High Level Implementation Architecture for Original Implementation

The MISST core logic is responsible for timing injections and sampling. It has three main modules: the Fault Parameters, an ALU, and the Control Unit. The Fault Parameters module is essentially memory holding information used to generate faults. The ALU is directly connected to the Fault Parameters module. The ALU is in charge of random number generation and changing the values saved in Fault Parameters. The Control Unit communicates with external modules and controls the ALU and Fault Parameter modules. The Control Unit is where MISST's basic behavior is implemented. Figure 2.2 shows how the three modules are connected.

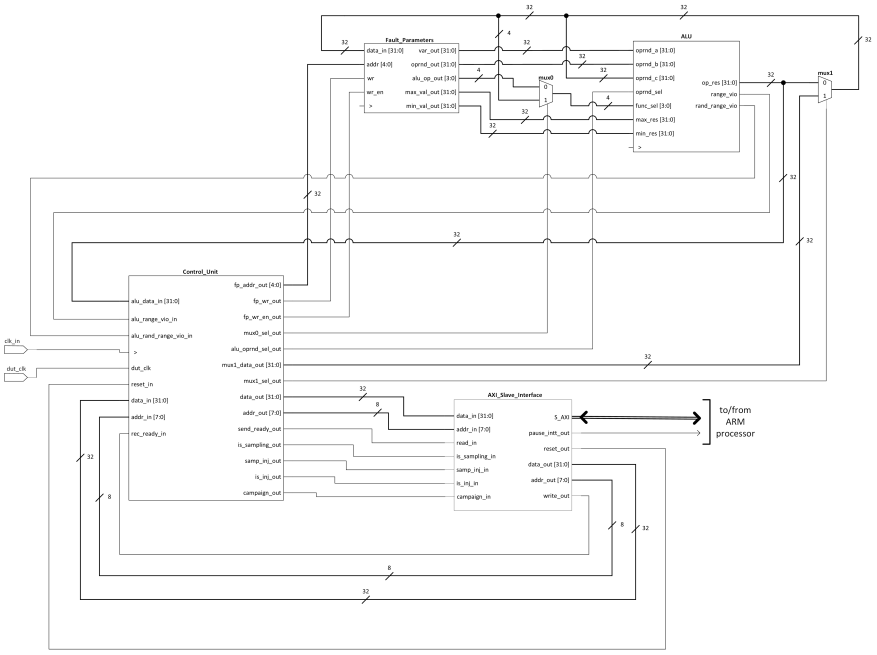

Figure 2.2: MISST Core System Interconnect. The AXI Slave is not part of the core system.

2.1 Task Flow

After the core and Adapter module have been implemented on an FPGA, the MISST system enters setup mode. In this state, the user can configure system registers listed in Table 2.1. Once the user has configured MISST registers, the fault campaign begins. The general task flow for MISST is detailed in Figure 2.3. An injection or sample occurs when certain timers (see section 5.2) timeout. If the injection and sampling timers timeout at the same time, sampling has priority.

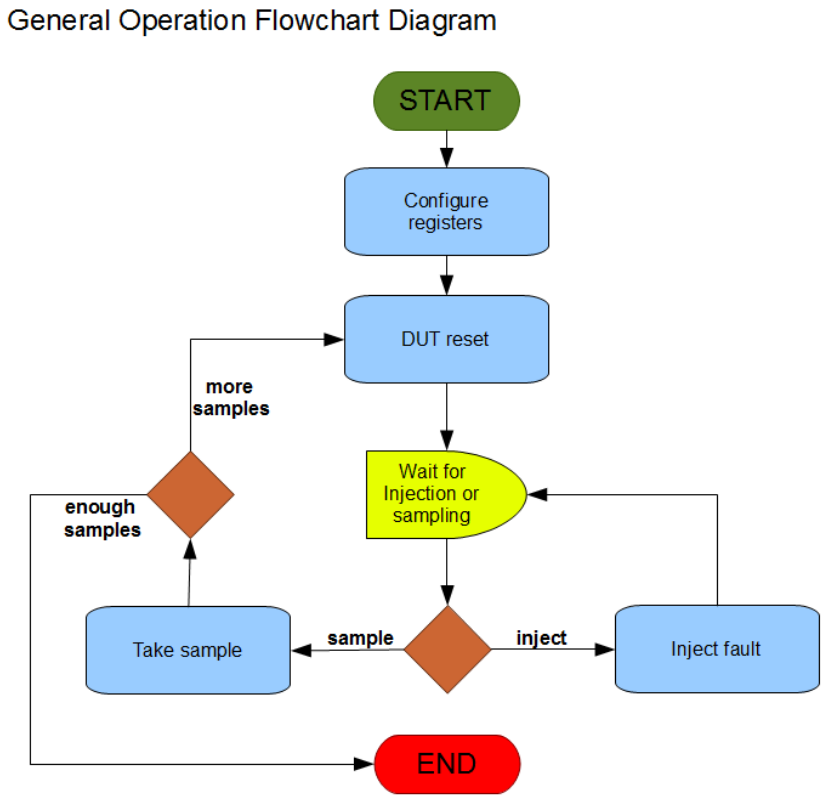

Figure 2.3: General Task Flow for MISST

2.1.1 Fault Generation

Injecting a fault involves three main steps:

1. Retrieve DUT_ADDR value from Fault Parameters module.
2. Sample data at address DUT_ADDR in the DUT memory space.
3. Create fault data to inject from sampled data.
4. Change fault parameters and save new INJ_TIME value in fault_timer register.
5. Inject fault.
6. Once acknowledgment from Adapter module has been received, resume the counters.

In this section, step 5 will be explained. Every time a fault injection occurs, new parameters for the next injection are generated by using deterministic or random operations provided by the ALU. Fault parameters can change between every injection or change between a certain number of injections. Adding another layer of complexity, multiple fault parameters can be scheduled to change in different ways. For example, all three fault parameters can be configured to change for every fault request. Another possible scenario is that the dut_addr fault parameter changes on every fault request and the flt_oprnd changes every three times dut_addr has been changed. When dut_addr has been changed three times, it will revert to an initial value or 'initialized'. When a fault parameter changes, but not initialized, it has been 'updated'. In this example, flt_oprnd would be on level 2, and dut_addr on level 1.

Each fault parameter is assigned a level that represents how it changes. Level 2 fault parameters update every time an injection occurs. Level 1 fault parameters update every time level 2 fault parameters are initialized. Level 0 fault parameters update every time level 1 parameters are initialized. It is *crucial* that if a level is meant to change throughout a fault campaign, that its lower levels (level 2 is the lowest level) are *not* constants.

A level's cycle length is the number of values in an initialize-update cycle. For example, a cycle length of 3 means that a fault parameter is initialized, updated, updated, and then the cycle is repeated again. A cycle length of two means that the fault parameter is initialized every other change. A cycle length of 1 means that the parameter does not change. A cycle length of 0 means that the parameter updates every time a fault generation is requested, like level 2. A cycle length of 0 should be used for randomly generated fault parameters.

Randomly Generated Injection Time

Extra caution should be taken when injection time is configured to be generated randomly. Since injection time can't be predicted in advance, not all faults within a set can be injected before sampling occurs. *All* faults in a set must be injected *before* sampling to avoid injection and sampling at the same time. The MISST system doesn't explicitly enforce each set to have the same number of injects before sampling so the MISST system should be configured with that in mind.

It's recommended that if injection time is randomly generated, sets be limited to only one fault. Configure the appropriate bounds for this injection time so that it occurs before sampling time. If multiple randomly generated injection time faults are desired, set the maximum value to the quotient of the sampling time and the number of injections per set. This should avoid any sampling or injection collisions.

Fault Generation Example

To illustrate fault generation, we will provide the following example.

A MISST system is configured as follows:

- Level 2 is dut_addr parameter with a cycle length of 2. It is initialized to a value A_0.
- Level 1 is flt_oprnd parameter with a cycle length of 3 also. It is initialized to a value B_0.
- Level 0 is inj_time parameter with a cycle length of 1 so it stays constant at a value C.

As the fault parameters are changed during every injection process, the pattern in Figure 2.4 emerges. In this example, the fifth fault would inject at address A_0 using value B_2 for flt_oprnd, C clock cycles after the previous injection.

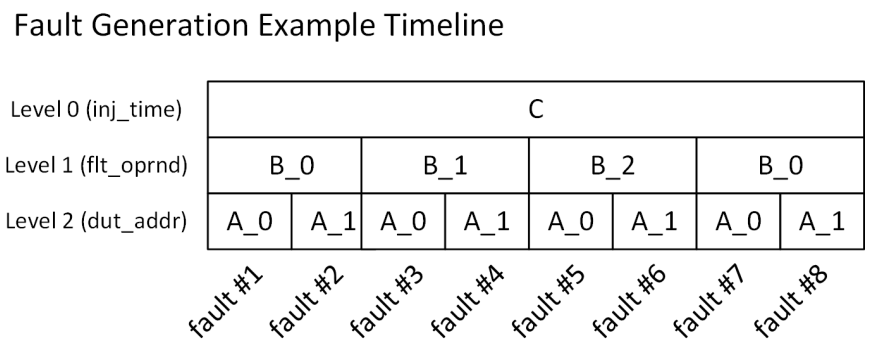

Figure 2.4: Fault Generation Example Timeline

Actual test cases won't be as repetitive, and would involve longer cycle lengths.

2.1.2 Fault Generation and Sampling Synchronization

MISST considers a set to have completed after a successful sampling process initiated by the sampling timer timeout (see section 5.2). MISST does not check if during a set any faults were injected, and does not enforce that each set have the same number of fault injections. Between DUT resets, the fault and sampling timer's count is reset to 0, but each timer's timeout value is left the same.

Generating faults in this manner allows the user flexibility to configure which faults are included within sets. Referring back to the example in section 2.1.1, if a user wants each set to have two faults so that the level 1 parameter changes after a sampling event, the user should set C (the inj_time) no less than a third the sampling time and no more than half the sampling time. Another possibility is that each set contains three faults. In that case, faults 1 to 3 would be in the first set, faults 4 to 5 in the seconds, and so on.

2.1.3 Sampling-based Shutdown

Sampling-based shutdown refers to ending a fault injection campaign based on the value of sampled data. If two locations are sampled, then the decision to shut down is based on the first sample. If MISST is configured to sample two locations, but sampling-based shutdown occurs, the second sample data will not be sent to the user. With the sampled value and the value in register sample_shutdown_value, there are three ways MISST evaluates sampling-based shutdown:

- MISST shutdowns if sampled value equals sample_shutdown_value.
- MISST shutdowns if result of bitwise AND between sampled value and sample_shutdown_value is zero.
- MISST shutdowns if result of bitwise XOR between sampled value and sample_shutdown_value is zero.

2.2 Memory Organization

MISST memory uses byte length addressing to write to different registers spread throughout the system core and the adapter module. There are three places where MISST addressable registers are implemented; in Fault Parameters, in Control Unit, and in the Adapter module. Table 2.1 explains each register's role in MISST. The address structure of Fault Parameters is different from the registers implemented in Control Unit and Adapter modules. See Section 3.2 for more details on the Fault Parameter addresses.

Table 2.1: Complete Register Summary.
In the Read/Write column, 0 means used internally by system.

Address: Name Read/Write Description
0x01 start_inj W If the value 0xAABBCCDD is written to this register, a fault injection campaign will start.
0x02 stop_inj W If the value 0xFFEEDDCC is written to this register, the fault injection campaign will stop.
0x03 fault_timer W The number of clock cycles until a fault is injected into the DUT.
0x04 sampling_timer W The time to sample after the last DUT reset measured in DUT clock cycles.
0x05 setNumCounter W Maximum number of sets. This register is incremented after every sampling. Mainly used as a fail safe to stop MISST system after some failure has caused it to continue execution longer than expected.
0x06 triggerPosCU W Holds trigger position inputs for cyc_cnt modules. See Figure 5.1 for reference.
0x07 dut_addr_init W Initialization DUT_ADDR value. The value that DUT_ADDR is set to after an initialization event. In the case that the DUT_ADDR fault parameter is constant, this register is unused.
0x08 inj_time_init W Initialization INJ_TIME value. The value that INJ_TIME is set to after an initialization event. In the case that the INJ_TIME fault parameter is constant, this register is unused.

Table 2.1: Complete Register Summary continued.

Address: Name Read/Write Description
0x09 flt_oprnd_init W Initialization FLT_OPRND value. The value that FLT_OPRND is set to after an initialization event. In the case that the FLT_OPRND fault parameter is constant, this register is unused.
0x0A sample_dataA 0 Sampled data is saved here.
0x0B cont_after_inj 0 When 0x1F1F1F1F is written here, MISST is ready to continue after a fault injection.
0x0C sys_status R MISST system status. This register resides in the Adapter module (see chapter 6). • bit 0: Set if system is executing an injection campaign, otherwise zero. • bit 1: Set if system is sampling data that must be sent to user. Cleared after write to sample_dataA register. • bit 2: Set if system is sampling data that will not be sent to user. Cleared after write to sample_dataA register. • bit 3: Set if system requests a fault injection. Cleared after the correct value is written to cont_after_inj. • bit 4: Set if DUT must be reset. • bit 5: Set if two samples are taken. Otherwise zero. • bit 6: Set if MISST is in setup mode. • Other bits remain unused.
0x0D dut_addr_cyc_len W The cycle length of DUT_ADDR. At the beginning of each cycle, DUT_ADDR will be set to its initial value. A value of 1 means that DUT_ADDR will never change. A value of 0 means that for every new fault generated, DUT_ADDR will always be changed to a new value (this setting mainly used with random changes).
0x0E inj_time_cyc_len W The cycle length of INJ_TIME. At the beginning of each cycle, INJ_TIME will be set to its initial value. A value of 1 means that INJ_TIME will never change. A value of 0 means that for every new fault generated, INJ_TIME will always be changed to a new value (this setting mainly used with random changes).
0x0F flt_oprnd_cyc_len W The cycle length of FLT_OPRND. At the beginning of each cycle, FLT_OPRND will be set to its initial value. A value of 1 means that FLT_OPRND will never change. A value of 0 means that for every new fault generated, FLT_OPRND will always be changed to a new value (this setting mainly used with random changes).
0x10 sampling_addrA W Address to sample in DUT memory space.

Table 2.1: Complete Register Summary continued.

Address: Name Read/Write Description
0x11 sampling_addrB W Address to sample in DUT memory space.
0x12 general_config W General system configuration. Configures MISST high level behavior. • bit[1:0] Selects fault parameter for Level 2. See (2.1.1) for level explanation. See Table 3.1 for fault parameter values (the two least significant bits). • bit[3:2] Selects fault parameter for Level 1. See (2.1.1) for level explanation. See Table 3.1 for fault parameter values (the two least significant bits). • bit[5:4] Selects fault parameter for Level 0. See (2.1.1) for level explanation. See Table 3.1 for fault parameter values (the two least significant bits). • bit[6] If set, two locations will be sampled every time the sampling timer timeouts. • bit[7] If set, MISST shutdowns if ALU experiences a non-random range violation during fault generation. • bit[9:8] Determines how MISST will shutdown based on sample taken. If two samples are taken, the decision is based on first sample. – b00 Shutdown is not based on sample value. – b01 MISST shutdowns if sample equals value stored in sample_shutdown_value register. – b10 MISST shutdowns if the resulting value of a bitwise AND between sample value and sample_shutdown_value register is 0. – b11 MISST shutdowns in a similar fashion as the previous, but with a bitwise XOR operation.
0x13 sample_shutdown_value W Value used for sample based shutdown.
0xC0 dut_sample_addr 0 Address to be sampled. This is implemented in the Adapter module. See Table 7.1.
0xC1 dut_inj_addr 0 Address of injection. This is implemented in the Adapter module. See Table 7.1.
0xC2 dut_data_out 0 Data to be written to DUT. This is implemented in the Adapter module. See Table 7.1.
0xC3 w_reg_addr W MISST register address. This is implemented in the Adapter module. See Table 7.1.
0xC4 w_data W Write data for MISST core. This is implemented in the Adapter module. See Table 7.1.

Chapter 3

Fault Parameters Module

The main role of this module is to store the fault parameters, and provide associated data to the ALU for fault generation (see section 2.1.1). There are three fault parameters and they are listed in Table 3.1. How fault parameters affect a set of faults is shown by Figure 3.1.

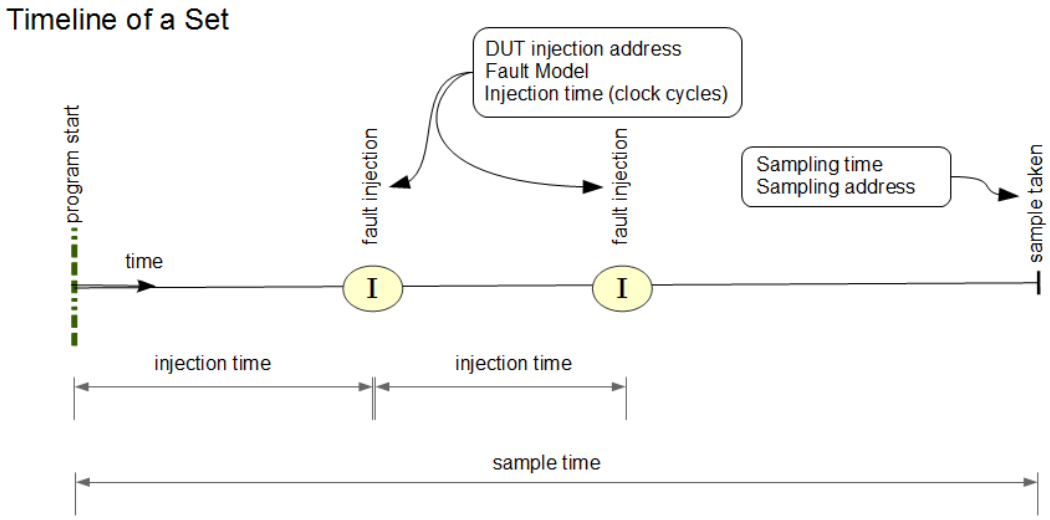

Figure 3.1: Fault Parameters on Set Timeline

3.1 Structure

The Fault Parameters module is a collection of three submodules (one for each fault parameter) called a grouping. Each grouping contains five registers related to the fault parameter referred to as an element. Table 3.1 explains each fault parameter, and Table 3.2 describes each element in a grouping. Figure 3.2 details how groupings are connected.

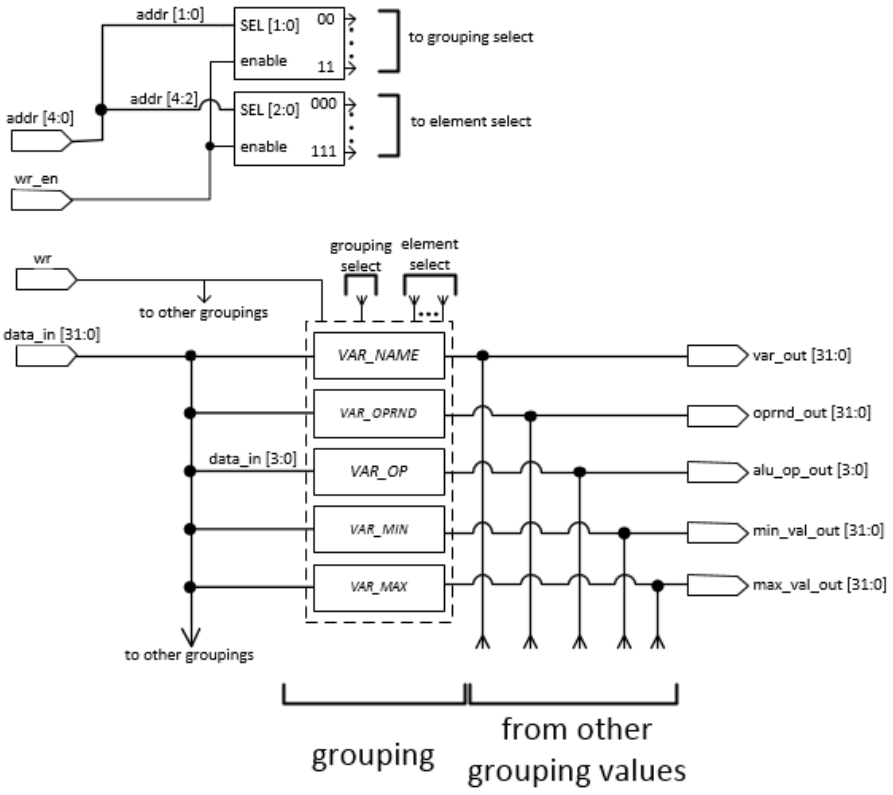

Figure 3.2: Fault Parameters Structure

3.2 Address Scheme

When the three most significant bits are b100, the address refers to memory in the Fault Parameters. The address has two parts: the two least significant bits refer to a fault parameter, and the next three bits refer to a fault parameter element. Tables 3.1 and 3.2 explain the possible addresses.

Table 3.1: Fault Characteristic Registers. The 'X' denote "don't cares".

Address	Name	Description
0b100X-XX00	DUT_ADDR	Current value of injection address in DUT memory space.
0b100X-XX01	INJ_TIME	Current value of injection time. Injection time is measured as the number of DUT clock cycles since the previous fault injection or DUT reset if this will be first fault in a set.
0b100X-XX10	FLT_OPRND	Short for fault operand. Depends on type of fault. For a SEU, it's the bit mask of bits to invert. For additive error model, it's the additive error.

Table 3.2: Fault Parameter Element Summary. The 'X' denote "don't cares".

Address	Suffix	Description
0b1000-00XX	_NAME	Value of associated fault parameter.
0b1000-01XX	_OPRND	Operand for arithmetic operation used when changing the value of the fault parameter.
0b1000-10XX	_OP	Arithmetic operation to change fault parameter. Unlike the other elements which are 4 bytes, this element is a nibble. NOP (no operation) function denotes a constant fault parameter.
0b1000-11XX	_MIN	Minimum possible value of fault parameter, inclusive.
0b1001-00XX	_MAX	Maximum possible value of fault parameter, inclusive.

3.3 Port Descriptions

Table 3.3: Fault Parameters Port Descriptions

Port Name	Description
data_in [31:0]	Write input data bus.
addr [4:0]	Address input bus. Addresses explained in section 3.2.
wr_en	A chip enable. Must be high for element values to output.
wr	Write enable on rising edge.
var_out [31:0]	NAME element for currently selected grouping.
oprnd_out [31:0]	OPRND element for currently selected grouping.
alu_op_out [3:0]	OP element for currently selected grouping.
min_val_out [31:0]	MIN element for currently selected grouping.
max_val_out [31:0]	MAX element for currently selected grouping.

Chapter 4

The ALU

4.1 Mathematical Operations

The ALU changes fault parameters based on a fault parameter's elements. Other than providing common arithmetic operations, the ALU provides random number generation. The operation is chosen through the func_sel input port as a nibble. These values are shown in Table 4.1.

Table 4.1: ALU Op-Codes

Code	Operation	Result
0x0	no operation	Outputs input to oprnd_a port.
0x1	addition	The sum of oprnd_a and selected second operand.
0x2	increment	The value of oprnd_a incremented by one.
0x3	decrement	The value of oprnd_a decremented by one.
0x4	left shift	Single left shift of oprnd_a.
0x5	right shift	Single right shift of oprnd_a.
0x6	OR	Bitwise OR of oprnd_a and selected second operand.
0x7	AND	Bitwise AND of oprnd_a and selected second operand.
0x8	subtraction	Difference between oprnd_a and selected second operand.
0x9	unimodal Gaussian	A random number with a unimodal, Gaussian distribution.
0xA	uniform uniform	A random number with a uniform distribution with no variance.
0xB	uniform average	A random number with a uniform distribution with some variance.
0xC	bimodal Gaussian	A random number with a bimodal, Gaussian distribution.
0xD	random bit flip	Inverts a single, randomly selected bit in the least significant byte of oprnd_a.

4.2 Random Number Generation

The ALU contains a module called noise_gen to generate random numbers with specified distributions. The VHDL source code for this module is originally from [2]. The noise_gen module generates random values using linear feedback shift registers. The four possible distributions generated are shown in Figure 4.1 for 16-bit random numbers.

The noise_gen module has two outputs, Gaussian and uniform, that produce random numbers. Each of these outputs have two subtypes. To produce all four possible distributions, the ALU has two noise_gen modules. However, at the moment they produce 16-bit numbers. Unfortunately, noise_gen is not scalable because of the properties of linear feedback shift registers.

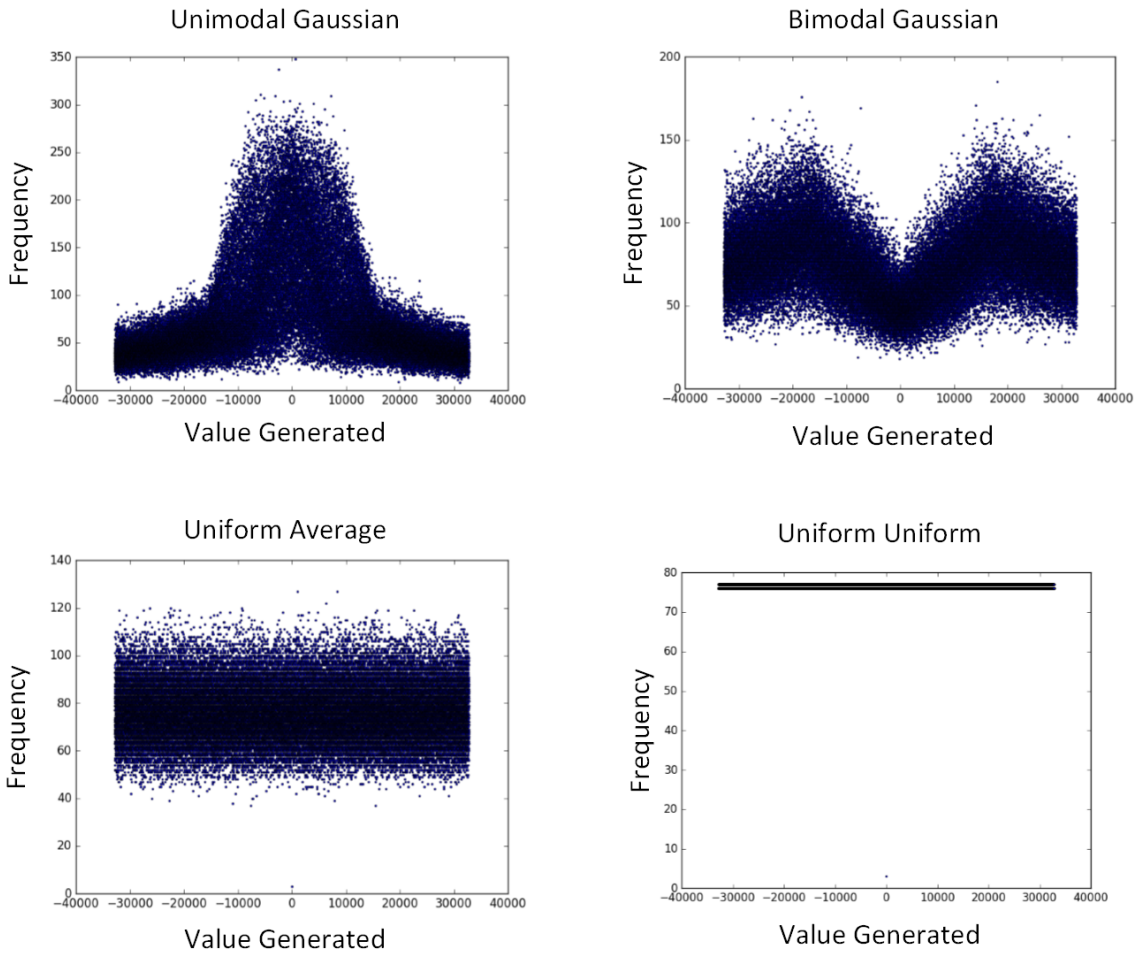

Figure 4.1: Distribution Tests for 16-bit Random Number Generation

4.3 Port Descriptions

Table 4.2: ALU Port Descriptions

Port Name	Description
oprnd_a_in [31:0]	Operand A. This operand is used in single operand operations.
oprnd_b_in [31:0]	Operand B.
oprnd_c_in [31:0]	Operand C. The optional operand selected by setting oprnd_sel_in.
min_res_in [31:0]	Lower bound (inclusive) for ALU result.
max_res_in [31:0]	Upper bound (inclusive) for ALU result.
func_sel_in [3:0]	Selects ALU operation. See Table 4.1 for op-codes.
oprnd_sel_in	If set, oprnd_c_in is the second operand, otherwise the second operand is oprnd_b_in.
clk_in	Clock input.
range_vio_out	Set if ALU result for a non-random operation is not between the values of min_res_in and max_res_in.
rand_range_vio_out	Set if ALU result for a random operation is not between the values of min_res_in and max_res_in.
op_res_out [31:0]	Result of selected operation.

Chapter 5

The Control Unit

The Control Unit is responsible for MISST high level behavior. This includes:

- Fault parameter generation.
- Sampling scheduling.
- Injection scheduling.
- External communication via Adapter module.
- Writing to registers.

5.1 Structure

To carry out the aforementioned tasks, the Control Unit is implemented using three modules; the register control, the memory interconnect, and the injection campaign finite state machine (ICF). These modules and their interconnections are detailed in Figure 5.1. Note that registers are represented as rectangles labeled with the register's name. All these registers are implemented in the memory interconnect submodule. Figure 5.1 also shows three cycle counter modules (labeled cyc_cnt#). Cycle counter modules count cycles of a signal and are further explained in section 5.1.1. Cycle counter 0 counts DUT clock cycles until a fault injection occurs. Cycle counter 1 counts DUT clock cycles until a sampling event occurs. Cycle counter 2 counts the number of sets (essentially the number of sample events) until the maximum number of sets to shut down.

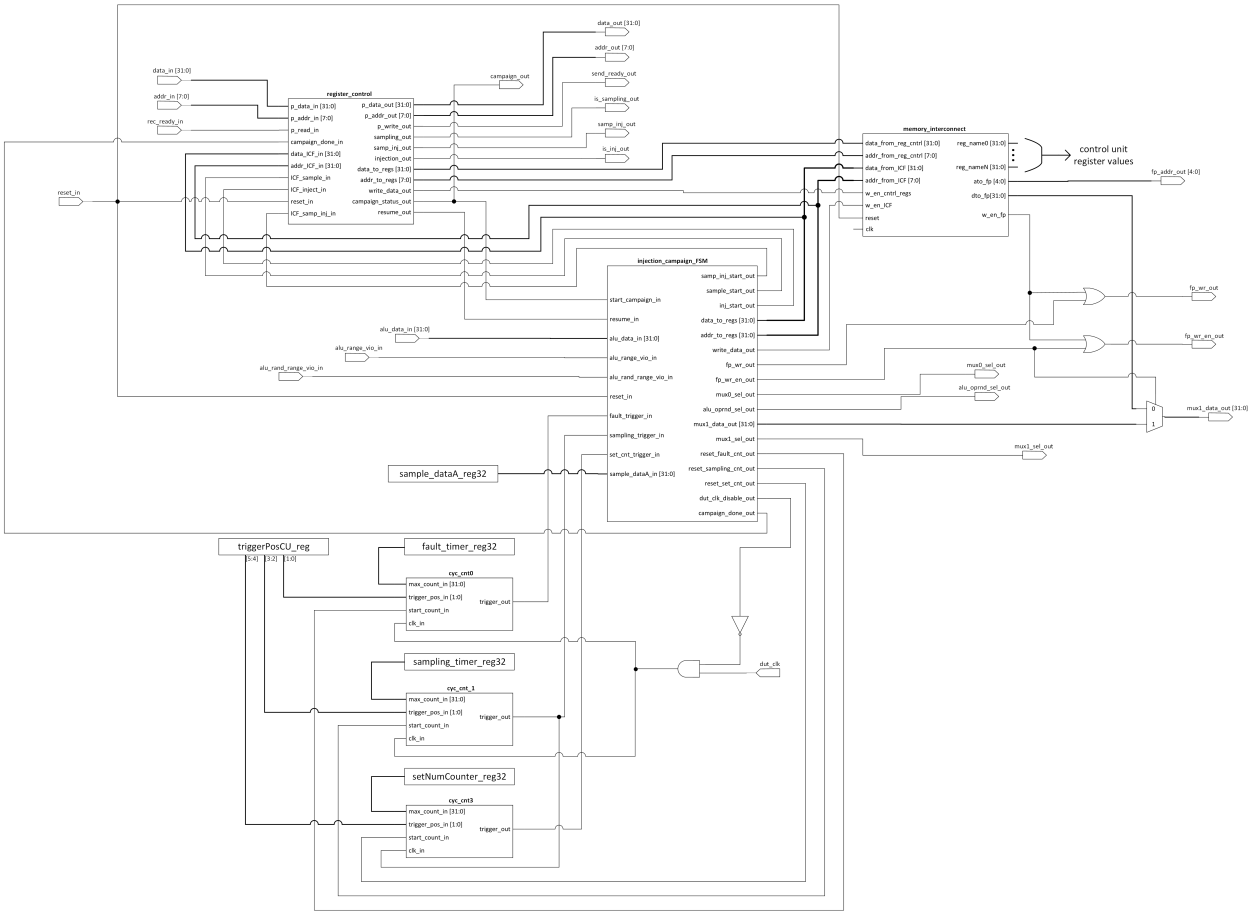

Figure 5.1: Control Unit Implementation

5.1.1 Cycle Counter Module

The cycle counter module is especially designed for use in MISST. The cycle counter module simply counts rising edges of the input signal on clk_in port until the number of rising edges hits a given value. Table 5.1 explains the role of each port.

Table 5.1: Cycle Counter Ports

Port	Polarity	Description
max_count_in [31:0]	input	Maximum number of cycles to count.
trigger_pos_in [1:0]	input	When the counter hits the maximum cycle count, this value determines on which edge of clk_in the port trigger_out asserts. See Figure 5.2 for a timing diagram. • b00 trigger_out asserts on first rising edge (aka front edge) of clk_in. • b01 trigger_out asserts on first falling edge (aka middle edge) of clk_in. • b10 trigger_out asserts on last rising edge (aka back edge) of clk_in. This option essentially delays trigger_out by one clock cycle. • b11 trigger_out never asserts
start_count_in	input	Resets count to zero.
clk_in	input	A rising edge increments count. Named clk_in because this module is generally used to count clock cycles.
trigger_out	output	A rising edge indicates that the number of counts exceeds the value of man_count_in.

The trigger_pos_in port controls on which edge of the clk_in signal the trigger_out port will assert. Figure 5.2 shows the timing diagram of a counter with a max count value of three. Note that the start_cnt_in input doesn't have to be kept high, but in this instance it is. The cycle counter multiplexes between the three internal signals front_edge_s, back_edge_s, and middle_edge_s for the trigger_out output. All three signals assert during the third cycle of clk_in, but they differ during which edge of that cycle they assert. For example, middle_edge_s asserts on the middle edge (the falling edge) of the third cycle of clk_in.

5.1.2 Register Control

The register control module is responsible for processing incoming and outgoing data between the MISST core and the Adapter module. Table 5.2 describes the ports of the register control module.

The register control module receives data from the Adapter module and forwards it to the memory interconnect. Register control also receives data from the ICF module (see section 5.1.4) for fault injection and sampling. Note than in Table 5.2 the module distinguishes between two types of sampling: sampling data to corrupt and use in upcoming fault injection (sample inject), and sampling data to send to user (simply referred to as sampling).

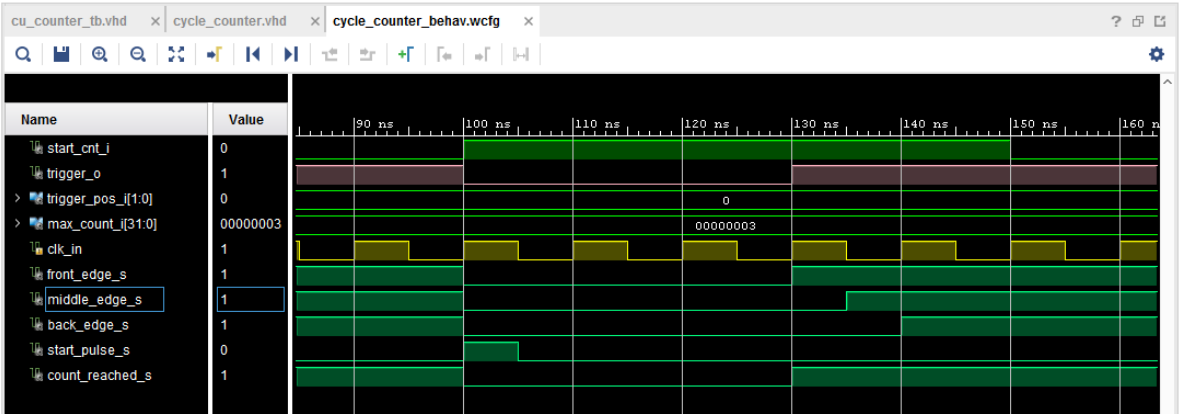

Figure 5.2: Different Types of Trigger Position

Table 5.2: Register Control Port Descriptions

Port Name	Description
p_data_in [31:0]	Data bus for incoming data to MISST core. Part of the Adapter-Core Interface (see section 7.3).
p_addr_in [7:0]	Address bus for incoming data to MISST core. Part of the Adapter-Core Interface (see section 7.3).
p_read_in	A logic high during a rising edge of clk_in initiates a read of address and data buses from Adapter module (see section 7.3).
campaign_done_in	Rising edge signals end of fault campaign.
data_ICF_in [31:0]	Data from the ICF module (see section 5.1.4) to be sent to Adapter module.
addr_ICF_in [31:0]	Address in DUT address space for injection or sampling from the ICF module (see section 5.1.4) to be sent to Adapter module.
ICF_sample_in	Initiates a sampling operation on a rising edge. Sampled data is sent to the user via the Adapter module.
ICF_samp_inj_in	Initiates a sampling operation on a rising edge. Sampled data is not sent to the user. This port used when retrieved data for a fault injection.
ICF_inject_in	Initiates a fault injection on the rising edge.
clk_in	Clock input.
reset_in	Initializes registers to zero on rising edge.
p_data_out [31:0]	Data bus for outgoing data to MISST core. Part of the Adapter-Core Interface (see section 7.3).
p_addr_out [7:0]	Address bus for outgoing data to MISST core. Part of the Adapter-Core Interface (see section 7.3).
p_write_out	A logic high during a rising edge of clk_in initiates a write of address and data buses to Adapter module (see section 7.3).
data_to_regs_out [31:0]	Data bus to memory interconnect module.
addr_to_regs_out [7:0]	Address bus to memory interconnect module.

Table 5.2: Register Control Port Descriptions Continued

Port Name	Description
write_data_out	Rising edge enables write to MISST registers.
campaign_status_out	High during a fault injection campaign, and low when MISST is idle or during setup.
sampling_out	A high value signals that sampling is in progress and that MISST is waiting for incoming data. Part of the Adapter-Core Interface (see section 6). Cleared when sampling data received.
samp_inj_out	A high value signals that sampling (specifically for retrieving data in preparation for a fault injection) is in progress and that MISST is waiting for incoming data. Part of the Adapter-Core Interface (see section 6). Cleared when sampling data received.
injection_out	A high value signals that MISST is in the process of injecting a fault and that MISST is waiting for incoming data. Part of the Adapter-Core Interface (see section 6). Cleared when the cont_after_inj register is written with proper value signaling injection process completion and DUT execution resume (see section 2.1).
resume_out	A rising edge signals that MISST can resume execution after sampling (for user data or for injection data), DUT reset, or fault injection.

5.1.3 Memory Interconnect

The memory interconnect module has two roles: implement Control Unit registers, and pass on Fault Parameter data. The memory interconnect module receives data from the register control and ICF modules. If a write to the same address at the same time occurs, the data from the ICF takes precedence. Table 5.3 describes the ports of the memory interconnect module, but the register outputs are excluded. The only registers not implemented in the memory interconnect module are listed below:

- Fault Parameter registers (see section 3.2).
- start_inj implemented in register control module.
- stop_inj implemeneted in register control module.
- cont_after_inj implemeneted in register control module.
- sys_status implemented in Adapter module.

The registers implemented in the register control module do not store data to be used at a later time, but are used to signal specific events. For example, the start_inj register is used to signal the start of a fault campaign.

Table 5.3: Memory Interconnect Port Descriptions

Port Name	Description
data_from_reg_cntrl_in [31:0]	Data bus from register control module.
addr_from_reg_cntrl_in [7:0]	Address bus from register control module.
data_from_ICF_in [31:0]	Data bus from ICF module.
addr_from_ICF_in [7:0]	Address bus from ICF module.
w_en_ICF_in	Write enable on rising edge for ICF data.
w_en_cntrl_regs_in	Write enable on rising edge for control register data.
reset_in	Resets all registers to 0.
clk_in	Clock input.
ato_fp_out [4:0]	Fault Parameter address bus. Connected to the Fault Parameter's input address bus (see chapter 3).
dto_fp_out [31:0]	Fault Parameter data bus. Connected to the Fault Parameter's input data bus (see chapter 3).
w_en_fp_out	Write enable in rising edge for data bound to Fault Parameter module.

5.1.4 Injection Campaign FSM (ICF)

The Injection Campaign FSM (ICF) is responsible driving other modules to implement high level logic. It implements the logic for the following processes:

- Fault injection.
- Data sampling for user.
- Data sampling in preparation for fault injection.
- Timer reset after timeout (for both sampling and injection timers).
- Fault generation using fault_gen module. Refer to section 2.1.1 for high level description of fault generation. Refer to section 5.1.5 for information on fault_gen module.

Essentially, this module is a finite state machine that transitions between states when sampling timer timeouts, injection timer timeouts, or on rising edge of resume_in port. The FSM that controls a fault campaign is shown in Figure 5.3. Table 5.4 describes ICF IO ports.

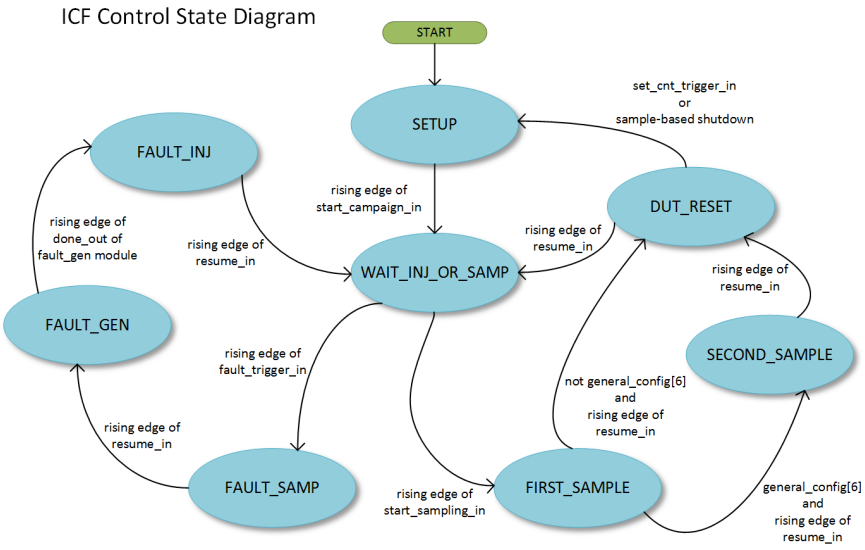

Figure 5.3: ICF Control State Diagram

A description of each state are listed below.

- SETUP: MISST registers are configured with desired values by user.
- WAIT_INJ_OR_SAMP: DUT runs while MISST waits for the injection and the sampling timers to timeout.
- FIRST_SAMPLE: Retrieves the first sample. Value based shutdown is also evaluated here (see section 2.1.3 for more information).

- SECOND_SAMPLE: Retrieves the second sample if two samples will be taken.
- DUT_RESET: DUT is reseted in preparation for a new run with a different group of faults.
- FAULT_SAMP: Sample the fault location.
- FAULT_GEN: Generate a new fault and calculate corrupted data to inject. This process mainly takes place in the fault_gen module.Refer to section 2.1.1 for high level description of fault generation. Refer to section 5.1.5 for information on fault_gen module.
- FAULT_INJ: Injecting fault into target DUT.

Table 5.4: ICF Port Descriptions

Port Name	Description
start_campaign_in	A rising edge starts a fault injection campaign.
resume_in	A rising edge signals state transition after an injection or sampling.
alu_data_in [31:0]	Output from ALU module.
alu_range_vio_in	High if ALU experiences a range violation on a non-random operation (see Table 4.2).
alu_rand_range_vio_in	High if ALU experiences a range violation on a random operation (see Table 4.2).
reset_in	Resets registers and ICF state.
fault_trigger_in	A rising edge initiates a fault injection.
sampling_trigger_in	A rising edge initiates sampling a single address on DUT. The data sampled is sent to user.
set_cnt_trigger_in	A rising edge signals that the maximum number of sets has been met and to end the current fault injection campaign.
clk_in	Clock input.
samp_inj_start_out	Rising edge initiates data sampling without sending data sampled to user. Kept high during sampling and cleared on rising edge of resume_in. This is used for reading data at injection location.
sample_start_out	Rising edge initiates data sampling to send data to user. Kept high during sampling and cleared on rising edge of resume_in.
inj_start_out	Rising edge initiates data injection using current fault parameters. Kept high during sampling and cleared on rising edge of resume_in. This is used for reading data at injection location.
data_to_regs_out [31:0]	Data bus to register control and memory interconnect. Data headed to register control module will be injected into DUT during fault injection. Data headed to memory_interconnect will be written to a Control Unit register.
addr_to_regs_out [31:0]	Address bus to register control and memory interconnect. Addresses sent to memory interconnect only use least significant byte and represents address of register. Addresses sent to register control represent an address in the DUT for injection or sampling.
write_data_out	A write enable on rising edge for memory interconnect.

Table 5.4: ICF Port Description Continued.

Port Name	Description
fp_wr_out	A write enable on rising edge for Fault Parameter module.
fp_wr_en_out	An active high chip enable for Fault Parameter module.
mux0_sel_out	Output to mutliplexer. If low, alu_op_out of Fault Parameter module outputs to func_sel of the ALU. If set, func_sel is connected to ALU-Fault Parameter bus. This bus can be driven by mux1_data_out if mux1_sel_out is set.
alu_oprnd_sel_out	Drives oprnd_sel input of the ALU (see Table 4.2).
mux1_data_out [31:0]	Data input for mux1(see Figure 2.2).
mux1_sel_out	Drives selection input of mux1 (see Figure 2.2). When low, the ALU output op_res drives the ALU-Fault Parameters bus. When high, mux1_sel_out drives the ALU-Fault Parameters bus.
reset_fault_cnt_out	Resets fault timer after a successful fault injection.
reset_sampling_cnt_out	Resets sampling timer after a successful sampling.
reset_set_cnt_out	Resets set counter after a timeout, but before MISST enters setup mode.
dut_clk_disable_out	If high, disables DUT clock input to fault and sampling timers.
campaign_done_out	Rising edge signals end of injection campaign.

5.1.5 The fault_gen Module

This module implements the logic described in section 2.1.1. The module is implemented as three cycle counters in series and the first cycle counter in the chain being incremented for every generation request. Initializations are started when a level's associated timer timeouts. Updates are started when a level's associated cycle counter increments.

Table 5.5: fault_gen Port Descriptions

Port Name	Description
gen_request_in	Initiates fault generation on rising edge.
reset_in	Resets counters and registers on rising edge.
clk_in	Clock input.
rand_range_vio_in	Connected from rand_range_vio output of ALU (see Table 4.2).
lvl2_cyc_len_in [31:0]	Cycle length of level 2.
lvl1_cyc_len_in [31:0]	Cycle length of level 1.
lvl0_cyc_len_in [31:0]	Cycle length of level 0.
lvl2_addr_in [7:0]	Level 2 fault parameter name address (see section 3.2).
lvl1_addr_in [7:0]	Level 1 fault parameter name address (see section 3.2).
lvl0_addr_in [7:0]	Level 0 fault parameter name address (see section 3.2).
done_out	Rising edge signals to ICF that fault generation is complete.
addr_out [7:0]	Current fault parameter being updated.
w_fp_out	Connected to write enable for Fault Parameter module.
wen_fp_out	Connected to chip enable for Fault Parameter module.
init_on_out	Set during an initialization operation. Cleared when operation is complete.
mux_sel_out [1:0]	Level being initialized. Outputs 0b00 (level 1), 0b01 (level 2), 0b10 (level 3), and 0b11 (nothing selected) sequentially as each level is updated.

5.2 Fault Timer, Sampling Timer, and Set Counter

During a fault campaign, the Control Unit responds to the timeouts of the three timers shown in Figure 5.1. Module cyc_cnt0 counts the number of DUT clock cycles until a fault injection, and is reset by the ICF after fault injection is completed. Module cyc_cnt1 counts the number of DUT clock cycles until sampling, and is reset by the ICF after sampling is completed. Module cyc_cnt3 counts the number of sets completed so far. When this timer reaches its maximum value, the fault campaign is over. Whenever any of these counters timeout, the ICF takes the appropriate actions to execute the timer's associated action.

All three modules are instances of the cycle counter module. Refer to section 5.1.1 for more information on the cycle counter.

Chapter 6

Adapter Module

The main purpose of the Adapter module is to provide an interface for the MISST core to communicate externally with a DUT or an intermediary module. In the case of the Original Implementation, for example, the Adapter module consists of an AXI slave module controlled by a Cortex processor. It is the user's responsibility to implement this module for their specific DUT.

For any implementation, the Adapter module is required to have certain ports and functional requirements. The required ports are listed in Table 6.1. Using these ports, the adapter module can fill in the required bit-fields for the required register sys_status. The bit-fields of sys_status are shown in Table 6.2. The reasoning for requiring sys_status is that the adapter class needs to be aware of MISST high level behavior to properly coordinate traffic between the PC, MISST, and target DUT. For instance, the Adapter module needs to be aware if MISST is configured to send one or two samples to the user.

Table 6.1: Required Adapter Ports

Port Name	I/O	Description
data_in	input	Data from MISST core.
addr_in	input	Destination address of incoming data.
read_in	input	Writes incoming data at incoming address on rising edge.
campaign_in	input	A high value signals system executing fault campaign.
is_sampling_in	input	A high value indicates system during sampling process.
is_inj_in	input	A high value indicates system during injection process.
samp_inj_in	input	A high value indicates that the value at the injection location is being read for corruption and injection.
data_out	output	Data to MISST core.
addr_out	output	Destination address of outgoing data to MISST.
write_out	output	Rising edge signals write enable for outgoing data at outgoing address to the Control Unit.

Table 6.2: Register sys_status Bit Fields

Bit Position	Description
0	Set if campaign_in port is high, low otherwise.
1	Set if is_sampling_in port is high, low otherwise.
2	Set if samp_inj_in port is high, low otherwise.
3	Set if is_inj_in port is high, low otherwise.
4	Set if DUT must be reset. Clear after DUT operation has been performed and MISST is signaled to continue.
5	Set if MISST is configured to sample two locations, clear otherwise. This value should be assigned when the Adapter module receives a write to general_config[6] (see Table 2.1).
6	Set if MISST is in setup mode, clear otherwise.

Chapter 7

Original Implementation Details

In this chapter we will primarily focus on the Adapter module implementation on the PYNQ board because the MISST system core does not change between implementations. For the original implementation, the adapter module includes the AXI slave interface, the Cortex hard processor, and a bridge from Xilinx IP library. The relevant ones are AXI AHBLite Bridge and the AXI APB Bridge. The IP library doesn't have an AXI to JTAG Bridge, but it does include a JTAG to AXI Master Bridge.

7.1 Serial Communication

The only way to access the UART port for PC serial communication is through the Cortex-A9 processor. The UART from the PYNQ board arrives at 115200 baud. If the MISST is implemented on a development board whose FPGA was direct access to a serial port, two design approaches could be:

- Use adapter module as an interface between MISST system core and the PC.
- Include a soft processor in the adapter module to process communication between board elements and a PC.

7.2 Role of Cortex Processor

The Cortex processor receives user input and sends data to MISST via AXI-Lite protocol to the AXI slave interface. The processor can read and write from/to the AXI slave interface, but the MISST system can't send data directly to the Cortex. The only way MISST can notify the processor of anything is via a single wire connection that triggers an interrupt on the processor (this is implemented as an output pin on the AXI slave).

MISST triggers an interrupt whenever it has to sample data from the DUT or inject a fault into the DUT. The processor evaluates what to do based on the value of the sys_status register at the time of the interrupt. Figure 7.1 outlines how the processor keeps track of tasks.

- WAITING_FOR_INTT: Processor is waiting for interrupt.
- SAMPLE_N_SEND: Processor reads from the DUT's memory and sends data to MISST and user through UART.

- DUT_RESET: Processor resets and then restarts DUT. Shortly after restarting DUT, processor writes data to sampling register to resume fault campaign.

- SAMPLE_INJ: Processor reads from DUT's memory and only sends sample data to MISST.

- INJECT: Processor reads dut_inj_addr and dut_data_out registers, and writes dut_data_out data at address dut_inj_addr in DUT memory.

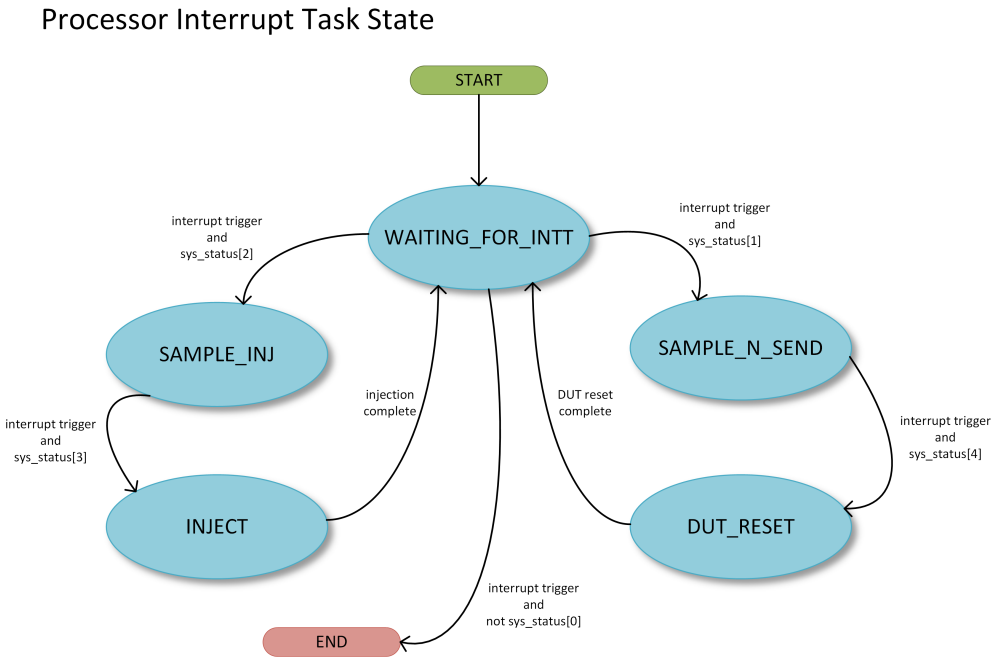

Figure 7.1: Processor Interrupt Task States

7.3 AXI Slave Interface

During setup, the user is simply writing data to registers in MISST core. In order to write to MISST registers, the processor first writes the data to the AXI slave's w_data register. Then the processor writes the write address to w_reg_addr register. The write is automatically executed upon a write to the w_reg_addr register. Table 7.1 lists registers implemented on the AXI slave interface. All AXI slave interface registers are four bytes wide, but not all registers use four bytes.

Table 7.1: AXI Slave Interface Registers

Address	Register Name	Description
0x0C	sys_status	Required register. Used by processor to choose correct course of action. See Table 2.1.
0xC0	dut_sample_addr	Sampling address.
0xC1	dut_inj_addr	Injection address.
0xC2	dut_data_out	Data to write to DUT. Only used during fault injection.
0xC3	w_reg_addr	Write address of register in MISST. Writing to this register starts writing transaction to selected MISST register. Only uses least significant byte since register addresses are a byte long.
0xC4	w_data	Write data.

The AXI slave interface is also responsible for asserting the interrupt input of the processor. The AXI slave sets the interrupt pin if is_sampling_in, samp_inj_in, or is_inj_in are high. Most importantly, the AXI slave interface must conform to the Adapter-Core interface. Table 7.2 describes the AXI slave's ports and points out which ones are required by the Adapter-Core interface.

Table 7.2: AXI Slave Interface Port Descriptions

Port Name	Description
data_in [31:0]	Input data bus. Part of the Adapter-Core interface.
addr_in [7:0]	Input address bus. Part of the Adapter-Core interface.
read_in	Reads address and bus lines on rising edge.
is_sampling_in	A high value represents that MISST is in the process of sampling. The data sampled will be sent to the user. Part of the Adapter-Core interface.
samp_inj_in	A high value represents that MISST is in the process of sampling. The data sampled will not be sent to the user. Part of the Adapter-Core interface.
is_inj_in	A high value represents that MISST is in the process of fault injection. Part of the Adapter-Core interface.
campaign_in	A high value represents that MISST is running a fault campaign. Part of the Adapter-Core interface.
S_AXI	Name for all signals of the AXI-LITE protocol. The processor sends data and reads data to/from here.
pause_intt_out	Connected to the processor's interrupt inputs. Referred to as the interrupt pin.
reset_out	Resets MISST core.
data_out [31:0]	Data bus to Control Unit. Part of the Adapter-Core interface.
addr_out [7:0]	Address bus to Control Unit. Part of the Adapter-Core interface.
write_out	Write enable for data sent to Control Unit. Part of the Adapter-Core interface.

The Adapter-Core interface provides rules on how the MISST core and the Adapter module exchange data. The Adapter-Core interface requires the ports with the following functionality:

- A pair of data bus, address bus, and enable signals for read and write with the Control Unit. The read or write operation is executed on a rising edge of the enable signal.
- A signal that goes high during the sampling process and clears when sampling data has been received. Sampled data is sent to the user.
- A signal that goes high during the sampling process and clears when sampling data has been received. Sampled data is not sent to the user.
- A signal that goes high during the injection process and clears when fault injection process is complete.

Bibliography

[1] PYNQ-Z1 Board Reference Manual. (2018). [ebook] Pullman. Available at: https://reference.digilentinc.com/_media/reference/programmable-logic/pynq-z1/pynq-rm.pdf [Accessed 24 Feb. 2018].

[2] T. Storey. "Pseudo Random Number Generator with Linear Feedback Shift Registers (VHDL)." Internet: https://eewiki.net/pages/viewpage.action?pageId=10125438, Sept. 8, 2017 [March 1, 2018].