ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет программной инженерии и компьютерной техники
Кафедра вычислительной техники

Лабораторная работа №2

«Встроенное тестирование в рамках технологии JTAG»

Выполнил студент группы P42002:
Василенко Елена Владимировна

Санкт-Петербург
2020

Содержание

I. Архитектура

1. Основные сведения.

Функциональные возможности отладочной подсистемы расширены за счёт модификации, добавляющей совокупность блоков, направленных на обеспечение работы режима самотестирования.

Как и любая другая информационная вычислительная система, система внутреннего самотестирования включает в себя два основных информационных потока:

1. Поток Управления (Control Flow);

2. Поток Данных (Data Flow).

В рамках целевой системы работа с потоком данных будет заключаться в хранении, обработке/формировании тестовых воздействий и референсных данных. По результатам тестирования необходимо формировать некоторую итоговую тестовую сигнатуру, которая будет отражать степень успешности проведенного тестирования. Поскольку тестовых воздействий и, соответственно, референсных значений, может быть несколько, то все они объединяются в некоторый общий тестовый сценарий, который необходимо хранить в устройстве памяти, как минимум на время работы режима самотестирования, отсюда появляется необходимость в реализации как минимум двух блоков - блока памяти и блока обработки и формирования данных.

В общем случае поток управления нужен для того, чтобы корректно обработать поток данных в соответствии с поставленной задачей. В случае целевой системы необходимо, чтобы поток управления организовывал вычислительный процесс в соответствии с нижеследующим алгоритмом:

1. Подача тестовых воздействий на тестируемый блок;
2. Считывание результирующих сигналов с тестируемого блока;
3. Осуществление валидации считанного значения.

Поскольку все операции с памятью тестового сценария и с тестируемой логикой выполняются с задержкой, то при работе с этими блоками, необходимо учесть эту задержку для того, чтобы сигналы на выходе вышеописанных блоков успели установиться в нужное состояние, и не возникало никаких состояний гонок или метастабильности. Эти

задержки можно обеспечить с помощью механизма переходов в конечном автомате, который будет управлять процессом самотестирования. Таким образом было принято решение, организовать вычислительный процесс Блока Управления, занимающегося формированием потока управления, в виде конечного автомата.

С учетом вышеописанных уточнений итоговый алгоритм работы конечного автомата будет выглядеть следующим образом:

1. Запрос тестовых данных из памяти тестового сценария;
2. Считывание тестовых воздействий и референсного значения из памяти;
3. Установка тестовых воздействий на входах тестовой логики;
4. Считывание результирующих сигналов;
5. Валидация результата и формирование сигнатуры.

Для удобства дальнейшей модификации, последующего тестирования и в целях сохранения читаемости кода, было решено объединить блок управления с блоком обработки и формирования данных. Модель вычислений типа “конечный автомат” позволит это реализовать.

На достаточно большом уровне абстракции система будет выглядеть следующим образом:

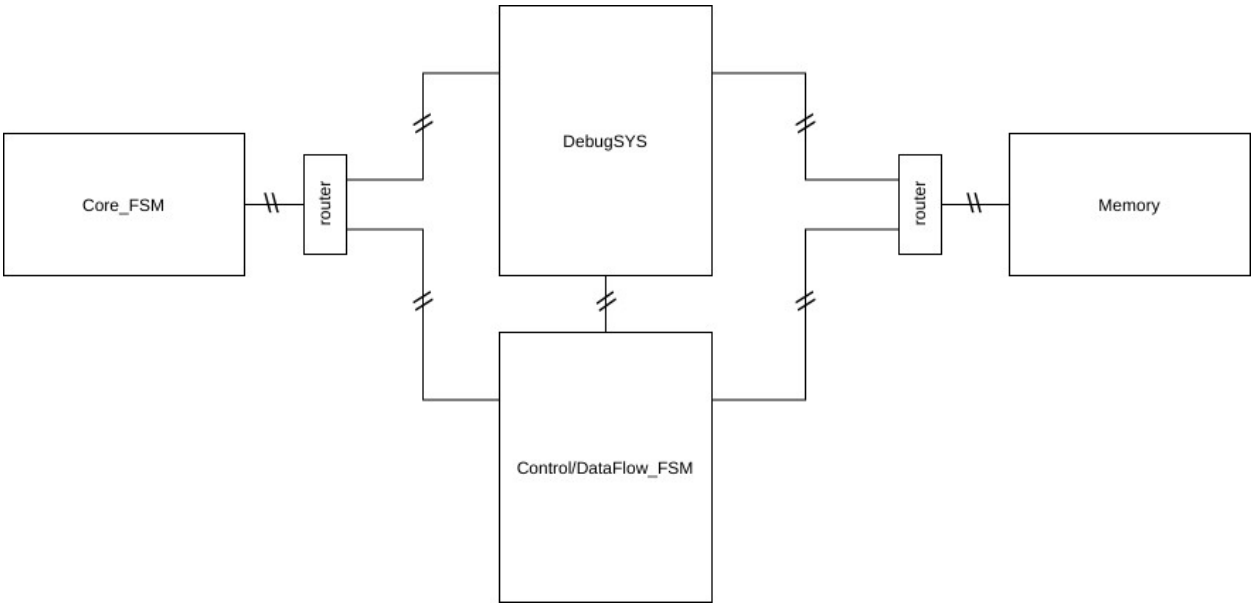


Рисунок 1. Абстрактная модель системы

1.1. Функциональные возможности системы.

Помимо стандартного требования, заключающегося в том, что система должна поддерживать переход в состояние самотестирования, было принято решение разрешить конечному пользователю запускать собственные тесты для тестируемого блока. Для этого необходимо было предоставить пользователю возможность записи в память.

В процессе тестирования блока, у пользователя может возникнуть необходимость разделить тесты на группы по некоторому признаку и запускать их в зависимости от той или иной ситуации (например, протестировать различные пути обхода графа-автомата). Удобнее всего размещать эти тесты по разным непрерывным регионам памяти, а на этапе инициализации тестирования указывать начало и конец целевого региона, из которого будут извлекаться тестовые данные. Таким образом, было решено сделать память адресуемой и добавить несколько новых команд, обеспечивающих работу требуемой функциональности:

1. Команда RUNBIST: переводит систему в режим самотестирования. По факту завершения процесса тестирования, регистр, соответствующий режиму RUNBIST, будет содержать тестовую сигнатуру, формат которой приведён ниже:

Результат тестирования	**Завершены все тесты**	**Тестовое воздействие**	**Ожидаемый результат**	**Фактический результат**
Успешно/ неуспешно (0/1)	Все/не все (0/1)	Данные поля заполняются при провале теста соответствующими значениями		

2. Команда DATALOAD: по каждому событию UPDATE_DR происходит запись содержимого сдвигового регистра в память тестовых данных. Формат данных:

Биты 15-8	**Биты 7-4**	**Биты 3-0**
Адрес в памяти	Тестовое воздействие	Эталонный результат

3. Команда ADDRLOAD: загрузка в систему начального и конечного адресов сегмента памяти, в котором хранятся интересующие пользователя тестовые воздействия. Формат данных:

Биты 15-8	**Биты 7-0**

Начальный адрес	Конечный адрес

1.2. Описание конечного автомата

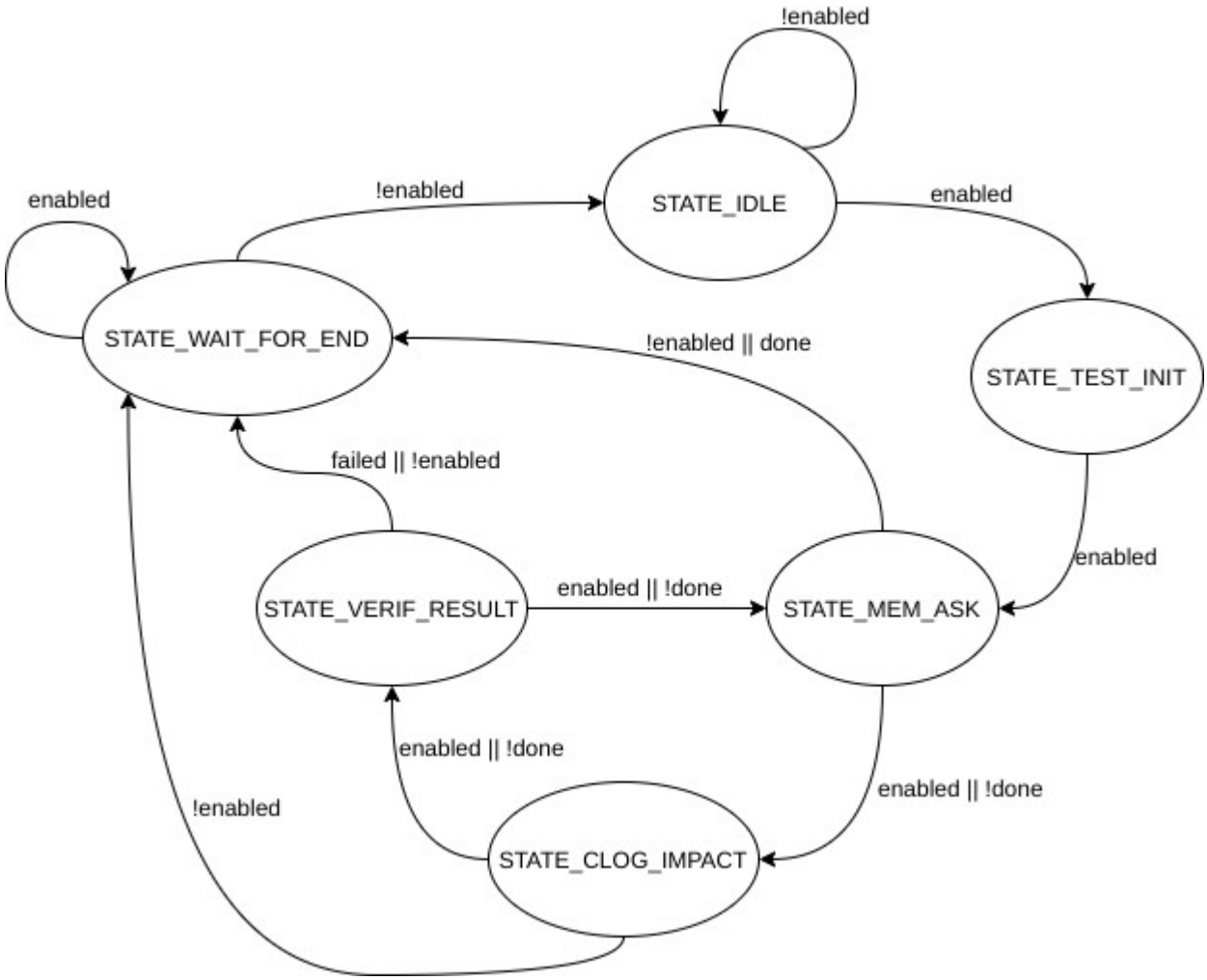


Рисунок 2. Конечный автомат BIST_FSM

Наименование состояния	**Описание состояния**
STATE_IDLE	Исходное состояние автомата. Переход из него осуществляется по факту подачи команды RUNBIST основному автомату системы. Здесь происходит обнуление флагов и тестовых воздействий.

STATE_TEST_INIT	Состояние инициализации тестирования. Здесь осуществляется обнуление сигнатуры и считывание начального адреса тестовых данных в памяти.
STATE_MEM_ASK	Состояние работы с памятью. Здесь происходит запрос нового блока тестовых данных из памяти, установка нуля на тактовом выводе тестируемой логики.
STATE_CLOG_IMPACT	Состояние работы с coreLogic. Здесь происходит получение данных из памяти и подача тестовых воздействий в блок тестируемой логики, установка единицы на тактовом выводе тестируемой логики.
STATE_VERIF_RESULT	Состояние валидации результатов. Здесь происходит получение и обработка результатов из блока тестируемой логики, а также обновление сигнатуры тестирования.
STATE_WAIT_FOR_END	Состояние ожидания. В данное состояние система попадает, если тестирование завершено раньше, чем автомат TAP-контроллера вышел из состояния Run-Test/Idle. Или во время выполнения тестового сценария один из тестов завершился с ошибкой. В этом состоянии происходит ожидание момента выхода автомата TAPC из состояния Run-Test/Idle.

1.3. Итоговая полная схема системы

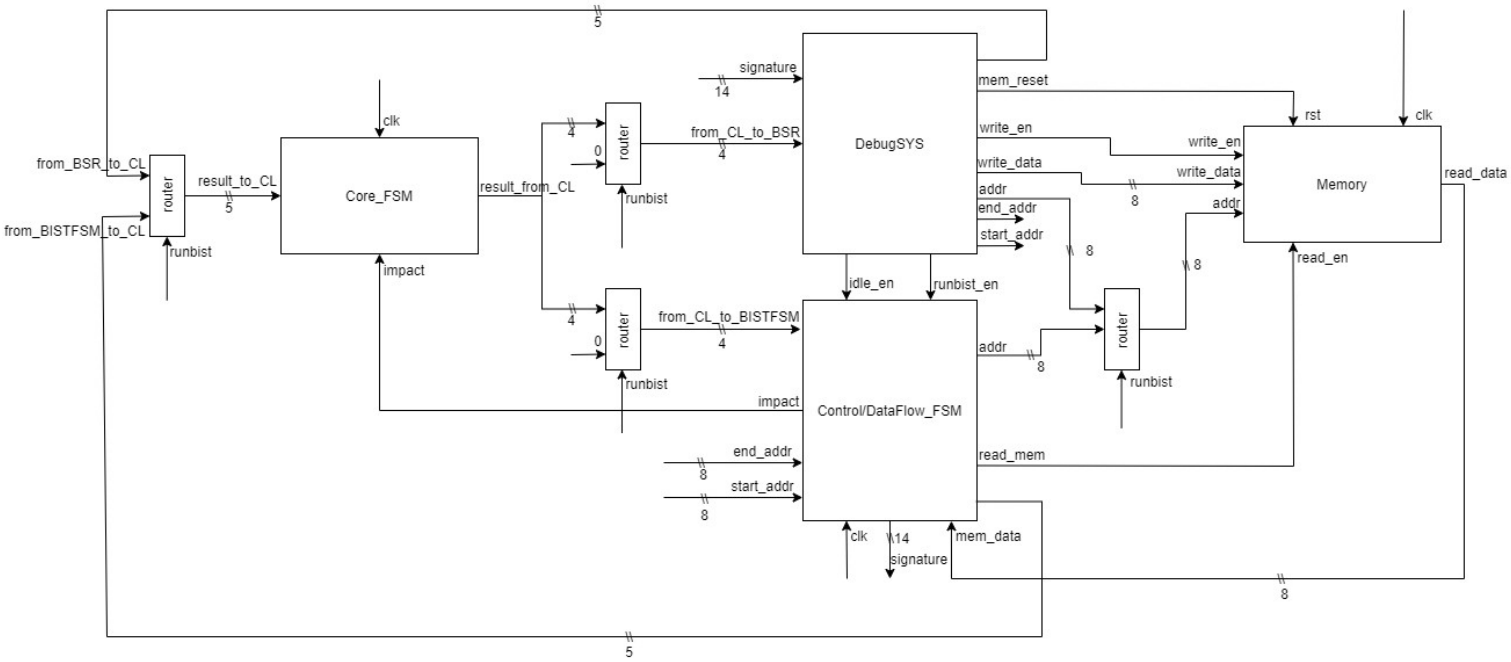


Рисунок 3. Подробная модель системы

2. Блок Tmemory

Блок адресуемой памяти. Предназначен для хранения тестовых воздействий (старшие 4 бита) и референсных значений (младшие 4 бита), может хранить в себе до 256 полей.

Наименование модуля	**Наименование порта**	**Тип порта**	**Назначение**
DebugSYS	clk	input	Сигнал синхронизации
	rst	input	Сигнал сброса
	write_en	input	Управляющий сигнал, по фронту которого осуществляется запись в память по адресу, выставленному на шине addr
	write_data	input	Шина входных данных, представляющая

			собой информацию, загружаемую в память
OnboardTop	addr	input	Адресная шина.
BIST_FSM	read_data	output	Шина входных данных, представляющая собой информацию, считываемую из памяти
	read_en	input	Управляющий сигнал, по фронту которого осуществляется чтение из памяти по адресу, выставленному на шине addr

3. Блок BIST FSM

Блок, отвечающий за организацию процесса самотестирования, в том числе за подачу тестовых воздействий на автомат тестируемой логики и за считывание данных тестового сценария, расположенных в блоке памяти.

Наименование модуля	**Наименование порта**	**Тип порта**	**Назначение**
DebugSYS	clk	input	Сигнал синхронизации
	runbist_en	input	Управляющий сигнал для перевода системы в режим самотестирования. Устанавливается и удерживается в состоянии 1 по факту выборки команды RUNBIST.
	idle_en	input	Управляющий сигнал для перевода системы в режим самотестирования.

			Устанавливается и удерживается в состоянии 1 по факту перехода в состояние Run-Test/Idle.
	signature	output	Совокупность сигналов, представляющих собой сигнатуру-результат тестирования.
	start_addr	input	Сигнал, указывающий на начальный адрес блока тестовых данных в памяти.
	end_addr	input	Сигнал, указывающий на начальный адрес блока тестовых данных в памяти.
Core_FSM	result	input	Шина входных данных, представляющая собой информацию о результате пройденного теста.
	impact	output	Шина выходных данных тестовых воздействий к автомату тестируемой логики.
Tmemory	mem_data	input	Шина входных данных, представляющая собой информацию, считываемую из памяти.
	read_mem	output	Управляющий сигнал, по фронту которого осуществляется чтение из памяти по адресу, выставленному на шине addr.
	addr	output	Адресная шина.

4. Изменения в DebugSys

Основной модуль системы, осуществляющий управление работой всей системы тестирования, в том числе декодирование инструкций и организацию работы с регистрами команд.

Наименование модуля	**Наименование порта**	**Тип порта**	**Назначение**
Tmemory	clk	input	Сигнал синхронизации
	mem_reset	output	Сигнал сброса
	write_en	output	Управляющий сигнал, по фронту которого осуществляется запись в память по адресу, выставленному на шине addr
	write_data	output	Шина входных данных, представляющая собой информацию, загружаемую в память
BIST_FSM	clk	input	Сигнал синхронизации
	runbist	output	Управляющий сигнал для перевода системы в режим самотестирования. Устанавливается и удерживается в состоянии 1 по факту выборки команды RUNBIST
	state_idle	output	Управляющий сигнал для перевода системы в режим самотестирования. Устанавливается и удерживается в состоянии 1 по факту перехода в

			состояние Run-Test/Idle
	signature	output	Совокупность сигналов, представляющих собой сигнатуру-результат тестирования
	start_addr	output	Сигнал, указывающий на начальный адрес блока тестовых данных в памяти
	end_addr	output	Сигнал, указывающий на начальный адрес блока тестовых данных в памяти

5. Изменения в OnboardTOP

На этом уровне осуществляется маршрутизация сигналов, относящихся к процессам тестирования. Сигналы, изменение которых влияет на режимы работы системы в целом:

addr_result – совокупность сигналов, представляющих собой адрес, с которым будет работать память тестовых данных. Этот адрес будет подаваться из блоков, соответствующих текущему режиму системы. Если система находится в режиме самотестирования (runbist = 1), то адрес подаётся из блока BIST_FSM, в противном случае адрес подаётся из блока TAPC (DebugSYS);

result_to_CL – входной сигнал в coreLogic. В зависимости от режима работы системы изменяется подключение входных сигналов тестируемой логики: если сигнал runbist выставлен в 1, то coreLogic принимает входные воздействия из блока BIST_FSM, иначе принимает входные воздействия из блока TAPC.

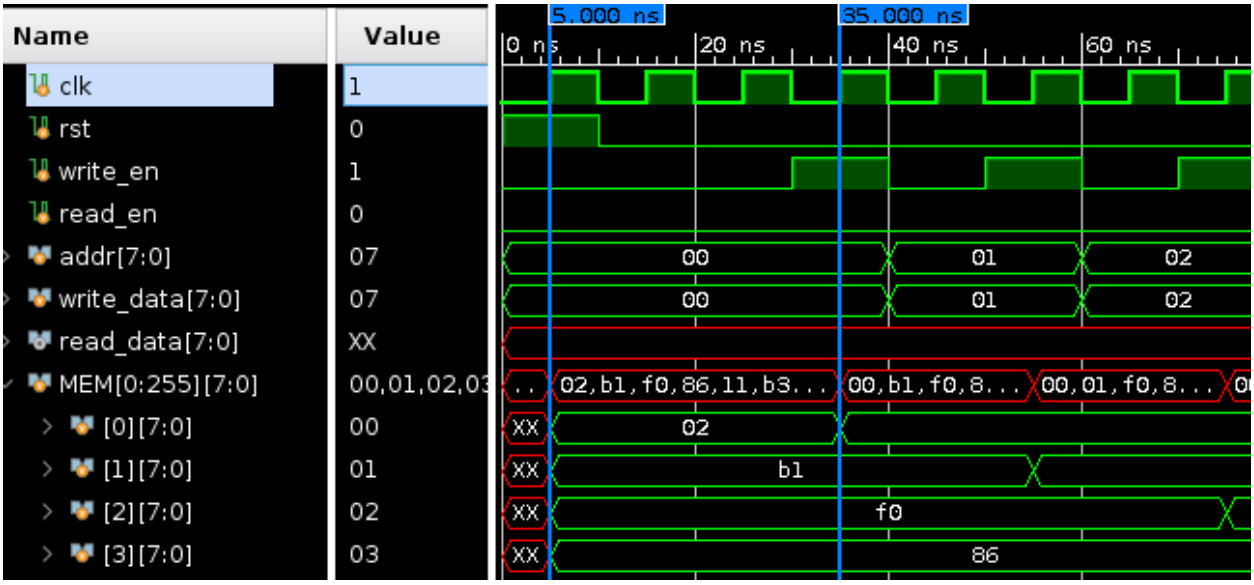
Когда система находится в режиме самотестирования, то выходные сигналы тестовой логики должны поступать в блок самотестирования для дальнейшей валидации. Если же система находится в своём обычном режиме работы, то выходные сигналы тестовой логики должны быть подключены к регистру BSR. Для этого описаны два сигнала:

from_CL_to_BISTFSM и from_CL_to_BSR, которые принимают соответствующие режиму работы системы значения и 0 в случае несоответствия.

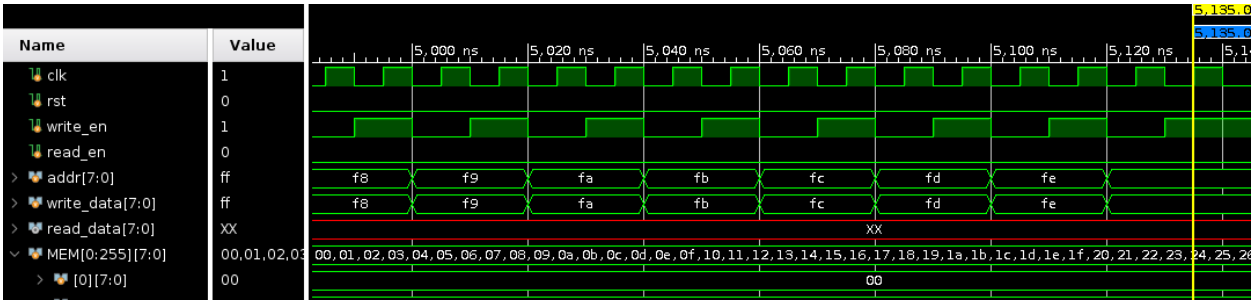
II. Симуляционное тестирование

Блок Tmemory

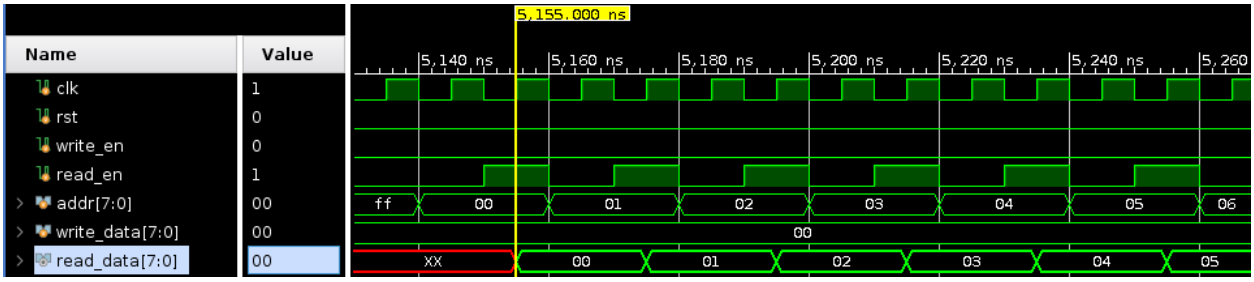
Идея тестирования блока памяти заключается в том, чтобы проверить возможность чтения/записи из каждой ячейки памяти. Для этого в тестовом сценарии проводится последовательная загрузка значений в память, а затем - последовательное считывание значений из памяти.



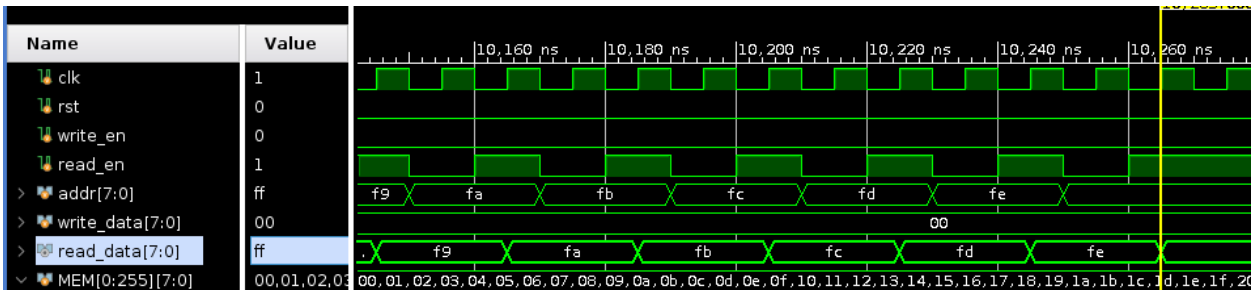
Первый маркер - “сброс” памяти. Второй маркер - первая запись



Первый маркер - последняя запись



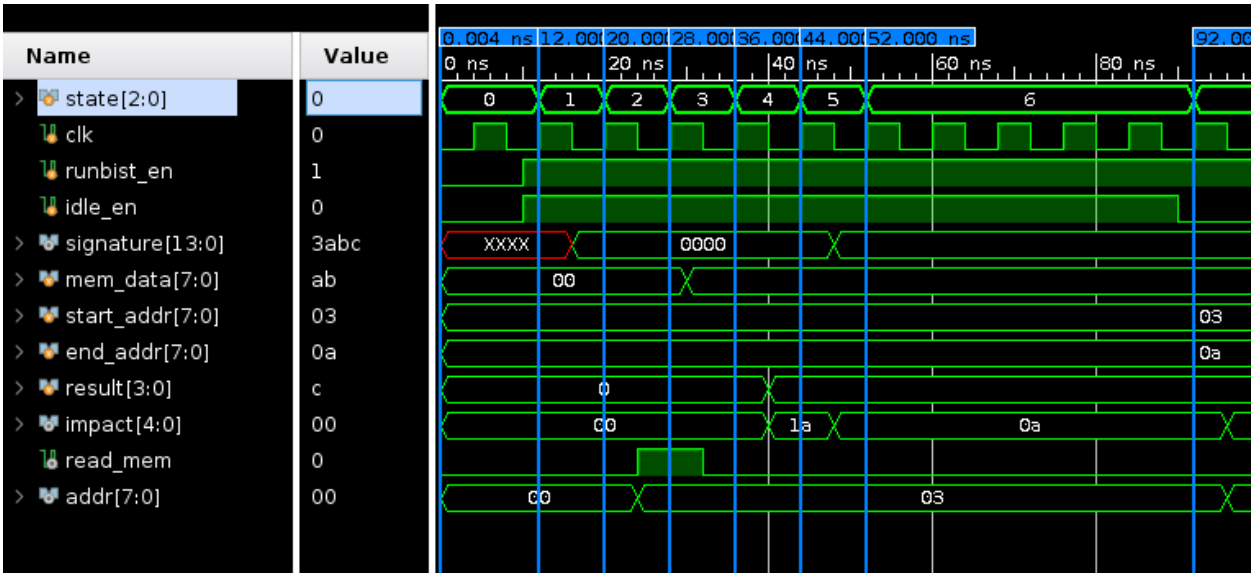
Первый маркер - первое считывание



Первый маркер - последнее считывание

Блок BIST FSM

Идея тестирования блока, представляющего собой управляющий автомат для режима RUNBIST, заключается в том, чтобы совершить все возможные переходы и убедиться в корректности алгоритма подачи управляющих сигналов. Алгоритм тестирования заключается в симуляции теста, завершившегося с ошибкой, то есть автомат пройдёт полный цикл проверки, а на этапе валидации завершится с ошибкой и будет ожидать ситуации выхода TAPC из состояния Run-Test/Idle.



Маркер 1 - автомат находится в состоянии IDLE;

Маркер 2 - автомат переходит в состояние INIT, так как выполнились условия для начала работы режима самотестирования - выборка команды RUNBIST и переход TAPC в состояние TEST_RUN;

Маркер 3 - Переход в состояние MEM_ASK. В этом состоянии, по спаду сигнала CLK автомат выставляет текущий адрес на шину адреса и взводит управляющий сигнал, отвечающий за чтение из памяти;

Маркеры 4-5 - Переход в состояние CLOG_IMPACT, в котором считывается запрошенное из памяти значение и выставляются управляющие воздействия на автомат тестируемой логики;

Маркер 6 - Переход в состояние VERIF_RESULT, в котором происходит формирование сигнатуры;

Маркеры 7-8 - Ожидание завершения тестирования, в состоянии WAIT_FOR_END.

III. Ход работы

Тестирование автомата в режиме INTEST выполнялось с использованием приведённого ниже SVF-файла (приведена только часть).

```
SDR 5 TDI (0);
SDR 5 TDI (1) TDO (0);
SDR 5 TDI (0B);
```

```
SDR 5 TDI (1B) TDO (2);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (1);
SDR 5 TDI (08);
SDR 5 TDI (18) TDO (0);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (6);
SDR 5 TDI (0B);
SDR 5 TDI (1B) TDO (1);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (3);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (4);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (1);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (8);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (1);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (B);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (1);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (0);
SDR 5 TDI (03);
SDR 5 TDI (13) TDO (A);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (2);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (5);
```

```
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (0);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (D);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (0);
SDR 5 TDI (06);
SDR 5 TDI (16) TDO (2);
SDR 5 TDI (00);
SDR 5 TDI (10) TDO (7);
SDR 5 TDI (00);
SDR 5 TDI (10) TDO (0);
SDR 5 TDI (00);
SDR 5 TDI (10) TDO (2);
SDR 5 TDI (00);
SDR 5 TDI (10) TDO (9);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (4);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (7);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (2);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (E);
SDR 5 TDI (0B);
SDR 5 TDI (1B) TDO (1);
SDR 5 TDI (06);
SDR 5 TDI (16) TDO (3);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (F);
SDR 5 TDI (0A);
```

```
SDR 5 TDI (1A) TDO (3);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (4);
SDR 5 TDI (0E);
SDR 5 TDI (1E) TDO (C);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (3);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (4);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (7);
SDR 5 TDI (03);
SDR 5 TDI (13) TDO (5);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (2);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (5);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (4);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (7);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (5);
SDR 5 TDI (0D);
SDR 5 TDI (1D) TDO (8);
SDR 5 TDI (06);
SDR 5 TDI (16) TDO (3);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (F);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (6);
```

SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (5);
SDR 5 TDI (08);
SDR 5 TDI (18) TDO (0);
SDR 5 TDI (03);
SDR 5 TDI (13) TDO (6);
SDR 5 TDI (03);
SDR 5 TDI (13) TDO (8);
SDR 5 TDI (03);
SDR 5 TDI (13) TDO (7);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (A);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (5);
SDR 5 TDI (08);
SDR 5 TDI (18) TDO (0);
SDR 5 TDI (09);
SDR 5 TDI (19) TDO (6);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (B);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (4);
SDR 5 TDI (09);
SDR 5 TDI (19) TDO (C);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (6);
SDR 5 TDI (0D);
SDR 5 TDI (1D) TDO (E);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (4);
SDR 5 TDI (09);

```
SDR 5 TDI (19) TDO (C);
SDR 5 TDI (0E);
SDR 5 TDI (1E) TDO (6);
SDR 5 TDI (00);
SDR 5 TDI (10) TDO (F);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (A);
SDR 5 TDI (0B);
SDR 5 TDI (1B) TDO (8);
SDR 5 TDI (0D);
SDR 5 TDI (1D) TDO (B);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (8);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (D);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (2);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (9);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (6);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (5);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (0);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (2);
SDR 5 TDI (0E);
SDR 5 TDI (1E) TDO (9);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (C);
```

SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (9);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (E);
SDR 5 TDI (03);
SDR 5 TDI (13) TDO (7);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (A);
SDR 5 TDI (09);
SDR 5 TDI (19) TDO (D);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (3);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (4);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (1);
SDR 5 TDI (09);
SDR 5 TDI (19) TDO (B);
SDR 5 TDI (0D);
SDR 5 TDI (1D) TDO (E);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (4);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (C);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (B);
SDR 5 TDI (05);
SDR 5 TDI (15) TDO (4);
SDR 5 TDI (00);
SDR 5 TDI (10) TDO (C);
SDR 5 TDI (0D);

```
SDR 5 TDI (1D) TDO (E);
SDR 5 TDI (01);
SDR 5 TDI (11) TDO (4);
SDR 5 TDI (00);
SDR 5 TDI (10) TDO (7);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (0);
SDR 5 TDI (0E);
SDR 5 TDI (1E) TDO (D);
SDR 5 TDI (0C);
SDR 5 TDI (1C) TDO (5);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (0);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (D);
SDR 5 TDI (03);
SDR 5 TDI (13) TDO (A);
SDR 5 TDI (0F);
SDR 5 TDI (1F) TDO (2);
SDR 5 TDI (02);
SDR 5 TDI (12) TDO (5);
SDR 5 TDI (0A);
SDR 5 TDI (1A) TDO (8);
SDR 5 TDI (0B);
SDR 5 TDI (1B) TDO (1);
SDR 5 TDI (06);
SDR 5 TDI (16) TDO (3);
SDR 5 TDI (06);
SDR 5 TDI (16) TDO (F);
```

Режим RUNBIST был протестирован следующим SVF-скриптом:

```
ENDIR IDLE;
RUNTEST 3000;
```

Оба SVF-сценария были выполнены успешно.

IV. Выводы

В ходе выполнения данной лабораторной работы был реализован метод тестирования целевого блока с использованием режима RUNBIST. Режим RUNBIST позволил в два раза сократить набор тестовых воздействий за счет управления сигналом тактирования тестовой логики. То есть, при использовании режима INTEST пользователю необходимо “вручную” задавать фронт тактового сигнала путём последовательной установки соответствующего бита в BSR сначала в “0”, а потом в “1”. В случае с режимом RUNBIST изменение тактового сигнала выполняется в момент перехода из состояния STATE_MEM_GET в состояние STATE_MEM_IMPACT. Таким образом за счёт сокращения набора тестовых воздействий уменьшается фактическое время тестирования. Также при использовании RUNBIST, в силу использования предопределенного набора тестовых данных, затрачивается меньше времени на переходы между состояниями загрузки команд и данных.