A Test Suite for Fairisle Port Controller version 3

Shaw Chuang

10th March 1994

1 Introduction

This document describes the functionality of a test suite designed for hardware testing of the Fairisle Port Controller version 3. The test suite consists of two parts. Firstly there is a ROM-based version for basic hardware and manufacturing testing. The second part is a more general purpose and flexible version for general hardware testing and debugging. This document includes a description of this test environment, the set of tests and error modes. It also provides a brief note on the implementation of the test suite.

2 ROM-based Test Suite

The purpose of the EPROMS test suite is to assist the people building the FPC3 board in fault diagnosis. The document describes version 2.2 of the test ROM. The FPC3 consists of two parts. The first is a processing unit based around an ARM RISC processor. The second section is a network unit consisting of the buffer memory and a DMA engine. Therefore there are two sets of tests in the whole test suite, each aiming to exercise one of these sections.

The basic processor section test examines the status of IOC, LEDs, serial line, ROM, DRAM. Given that the tests on this section pass, we can then assume that we basically have an functioning Archimedes and can start up the Wanda micro-kernel that will perform the second set of tests on the network section.

The second set of tests aims to exercise the network unit. It first tries to program and then re-program Xilinx, check data/irq lines from Xilinx to IOC, it then checks access to the SRAM, the state of the transmission Xilinx and also accessibility of the Podule bus (this is not really in the network section, but it is rather more convenient to do it here). It also sends cells to both the loopback FIFO and the transmission FIFO. This set of tests is started in auto-pilot mode, i.e. the all tests are performed with minimum verbosity and non-interactive. After that, we drop into interactive mode of the test suite, in which individual tests can be selected and executed in an interactive way. It also allows us to access to xi5tester, which is a beefed-up version of the xiromtester (this is what we called the network section test suite). xi5tester will be described in section 8.

Given that the hardware passes both of these tests, there should a high level of confidence that it has been constructed correctly.

3 Basic Configuration

We made assumptions on the minimal configuration that must be working before we can use software diagnostics, as shown in Table 1.

Clocks (MCLK and FCLK)
ARM 610
MEMC
ROM[0..3]
LATCH(es)
Address lines[0..??,26..31]
data[0..31]
RSTBAR, MEMREQ

Table 1: Minimal Configuration

Nothing can be done until these work to this extent (the arm is unable to read instructions from ROM).

In addition the following assumptions are made for processor unit tests:

- The board is configured with exactly 4M of dram
- The clocks are running at 25 and 8.33 MHz so that we can assume a serial line speed without reading a link on the board.

4 Hardware Setup

You should have the items in Table 2 for the test:

A FPC3 board (of course)
A crate
A clock generation board
A fabric
A Podule back plane (in the crate)
Test Podule
Transmission loopback cable
A debug LED display module
IDROM
Xilinx PROM ((marked with TAXI5)
Test ROM (Set of 4, marked FPC3T 0-3)
FPC3 PAL (marked with FPC3 PAL)

Table 2: Hardware Components

Furthermore, you need some sort of console output device. A character-based terminal set to 4800 baud will do the job. With these components, this is the setup we need for the test:

1. The transmission Xilinx PROM, IDROM, FPC3 PAL and test ROM are inserted in their corresponding sockets on the FPC3 board.
2. The debug LED module is also inserted on the FPC3 board
3. Loopback cable on the transmission interface (blue end below and red end above)
4. The test Podule is plugged into one of Podule backplane slot (any one will do as the software will find the card), and then the backplane plug into the FPC3 Podule backplane connector
5. Serial line connected
6. Setup crate with clock generation board and fabric board. Make sure that the FPC3 is in IFAB 5 position.

5 Basic Arm Test

As mentioned before, the basic arm test is to exercise the processor part of the port controller. Given that the board might be broken and not have the capability to output anything on the serial line, the debug LED module is used to provide some information on the state of the tests. Given the minimal working hardware it should be possible to show on the LEDs in a recognised pattern if some of the following tests fail (in order of test):

- Set debugging LEDs
- Check for IOC
- Turn on panel LED
- Send a character on the keyboard
- Do ROM Checksum test
- Change panel LEDs
- RAM test (4Mbytes)
- Move to RAM
- RAM test from RAM

If the board is working, then you should be seeing 0a, 10 and 17 on the LED display. The serial line test also send a '!' to the serial line. If all tests passed, you should then see "BASIC PROCESSOR TESTS PASSED" on the serial line. Notice that after this message, the Wanda micro-kernel would have started running and whole bunch of initialisation messages would have come up. Ignore these messages and wait until you are prompted to start the xiromtester.

If the hardware is broken, you would not drop into Wanda and the LED display will tell you the precise error (check with the table below). You might still get the '!' on the serial line depending on which stage of the tests the error crop up.

Make sure you power cycle the FPC if you want to repeat this test.

5.1 Display Code on Debugged LEDs

The LED display is used for both indicating the current state of the arm test as well as indicating the error state. The complete display code is as shown in Table 3:

Code	Test Area	Comments
00	Start	Displayed explicitly
01	IOC	Start of test
02	IOC	Didn't get back what was written
03	LED	Start of LED test
04	LED	Initial values not as expected
05	LED	Failed to program
06	KBD	Start of serial line test
07	KBD	keyboard speed links incorrect
08	KBD	kart rx data ready after read
09	KBD	kart tx data done after write
0a	ROM	Checksum tests
0b	ROM	1st ROM failed
0c	ROM	2nd ROM failed
0d	ROM	3rd ROM failed
0e	ROM	4th ROM failed
0f	RAM	Programming Memc
10	RAM	About to start first test
11	RAM	Didn't read back full word at 0
12	RAM	Didn't invert prev word at 0
13	RAM	Err in r/w single bit
14	RAM	Err in r/w all bits bar 1
15	RAM	Byte r/w error
16	RAM	Error in r/w mem block
17	RAM	All ok

Table 3: LED Display Code

5.2 Arm Test Internal

You should only read this section if you want to know exactly what the tests are doing.

1. *Start up*

   The arm test starts by getting into 32 bit data addressing mode follow by displaying '00' to indicate processor started. MEMC is then configured out of power on mode.

2. *IOC*

   There is not much of the IOC that we can test without having external hardware working, but we can write and then read back the interrupt mask registers. This test is only safe if interrupts are disabled in the PSR mask (they are when the processor is reset).

   By writing 0, 255 and 0x55 (alternate 1s and 0s) and reading each value back should reveal any dodgy data lines (data[0..7])

3. *LED*

   Check that LED lines are correct and that the transmission system is being held in reset, if OK then turn the LEDs on (but leave the transmission system in reset). The status of the LED lines and the Transmission reset line are available in the FIQ status register (though the bits are inverted from the logic level on the line)

4. *KBD*

   Set up the keyboard and try to send a character. First try and read the clock-speed links, we expect them to be in a particular state and if not we barf. Next set the kart speed and reset the receiver. After resetting the receiver there should not be a pending receive. Reset the transmitter by sending the character '!'. After sending the character the transmission status should not be empty.

5. *ROM CRC*

   CRC is performed on the whole of ROM. The CRC check routine checks individual ROM chips, and should be called 4 times, one for each chip. On return, the zero flag is set if the result was zero.

6. *Programming MEMC*

   Program MEMC for 4Mbytes of RAM. If we don't have 4M we loose. Have to do dummy writes to the address translation to set the mapping The page-address mapping is:

   $PPN[6] \rightarrow A1, PPN[5] \rightarrow A2, PPN[4] \rightarrow A0, PPN[3..0] \rightarrow A[6..3]$

   Only want 128 logical pages $LPN[7..0] \rightarrow A[22..15]$

   do the 16 (sensible) ones each time round inner loop, get low bits from table

7. *Memory Test*

   This consists of several phases.

   (a) Test one word thoroughly to check each bit. Store 0x55555555 at offset 0, and read it back. Then try with the inverse.
   (b) Store bytes in a word to check independence of bytes. Store 0 in the word and then set individual bytes to 0xff
   (c) Check that writing to one word does not affect others. If the previous test passes, it indicate that an individual word seems to work ok. Now we need to find out if all memory locations are independent. This one uses all words in RAM. Write the address of each memory location into itself on the first pass, then read back and verify on the second.

8. *That's all folk's*

   Things seems to be alright if reach here. Now boot the test software that is linked with a Wanda kernel: it is stored in the ROM immediately after this code. The first three words of a normal boot ROM are not executable at this offset, so we must skip over them.

6 Basic Xilinx Test - Xiromtester

Xiromtester, the network part test is based on xi5tester. Xiromtester performs the following basic tests:

- Program Xilinx
- Re-program Xilinx
- Check data/irq lines from Xilinx to IOC
- SRAM test
- transmission Xilinx 3030 programming test.
- Podule test
- Loopback FIFO cell
- Transmission FIFO cell

All these tests are performed in the Wanda environment. After the processor tests have completed, Wanda is started. Xiromtester will automatically be invoked from Wanda kernel during boot. Xiromtester at this stage is configured in silent and auto-pilot mode. Tests are then carried out with minimum verbosity. During the silent mode test, only failed tests will be indicated on the console line. At the end of the test, a summary report will be shown to indicated the no. of tests failed or that all passed. After this, we enter an interactive mode xiromtester.

Do expect a small delay between finishing the basic test and any error/summary reports appears. An example of the test output looks like:

```
!BASIC PROCESSOR TESTS PASSED


Podule test                                        FAIL

        SUMMARY: 1 tests FAIL
```

6.1 Further Test and Debugging

If an error occurs, you will get error message indicating which test has failed. We would then be required to run the test in a more interactive way to find out about the precise error.

We can get into the interactive-mode by just pressing RETURN at where the tests stopped (as shown above). You will see this

```
TOP MENU

Choose tests from the following options:

1.      Xilinx basic Tests
2.      Xilinx xi5 bits Tests
x.      Exit Xilinx ROM Test
```

Choose 1 and you will get

```
XiRomTester Alive

xiromtester: MAIN MENU

Choose tests from the following options:

0.      Program Xi5 bits
1.      Program test1 bits
2.      Data/IRQ lines from Xilinx to IOC
3.      Test SRAM with fabric clock
4.      Transmission Xilinx Test
5.      Podule test
6.      Send Loopback Cells
7.      Send Tx Cells
t.      Interactive toggle
a.      All tests
e.      Execute selection
l.      Repeat Last (`` '')
x.      Exit xiromtester
```

To repeat all previous tests, choose option a and hit RETURN. You will see a whole bunch of messages indicating the setup of the tests. Ignore them for now. You then choose option e and hit RETURN. This will execute the tests and you should see something like:

```
Xi5 bits programming test               PASS
Test1 bits programming test             PASS
IRQ/IOC test                            PASS
SRAM test 30 iteration                  PASS
TXRX Test                               PASS
```

```
       ERROR: test podule: No test podule found
Podule test                                          FAIL
Fabric loop back test                        PASS
Transmission loop back test                  PASS
```

6.2 Tests and Error Modes

6.2.1 Xilinx Programming

The first 2 tests in the list programmes the Xilinx with the two designs that we are going to use in our tests. If the Xilinx manage to program for the first time, there is a high likelihood that it will succeed the second time. Possible error conditions are:

1. Panic with "Xilinx INIT signal did not go inactive"
2. Panic with "HDC set, so re-program asserted → INIT stays active"
3. Panic with "HDC signal was not asserted after re-program"
4. Panic with "Busy prior to configuration"

6.2.2 IRQ/IOC Check

This test aims to examine the integrity of the lines between Xilinx and IOC. If test 1 and 2 passed, the Xilinx programme data lines, INIT, HDC and BUSY should be intact. This test thus looks at the 3 FIQ and 2 IRQ conditions. The test1 xbits is used so that data written to Xilinx is made to appear on the IOC lines. By controlling the data written, different FIQ and IRQ conditions can be generated.

Possible error modes:

1. ERROR: Xilinx signal XXX does not happen

   Follow by information about the particular signal. This basically means no IRQ/FIQ has been seen.

2. ERROR: Xilinx signal XX causes wrong interrupt

   An interrupt has happened, but is not the one we expected. This is follow by information about the particular signal and a dump of what we think the current IOC status is.

6.2.3 SRAM Test

This test serves two purposes:

1. check address to Xilinx is intact for SRAM access
2. Given that Xilinx is the only device which drives the SRAM access, being able to read/write the SRAM correctly should imply the "slowarm" interface (using fabric clock) is OK. Xi5 bits is used here.

The test mainly does 30 iterations of write and read back to SRAM. In each iteration, the whole SRAM is initialised to a test pattern. The data is then read back and compare.

Possible error modes:

1. ERROR: at XXX (got YYY not ZZZ) again AAA

   Where XXX is the memory location, YYY is what we read back and ZZZ what we expected. AAA is another read retry.

2. ERRORS: XXX to YYY

   There are error from memory location XXX to YYY.

6.2.4 Transmission Xilinx Test

We want to verify that the transmission Xilinx has been programmed correctly. There is no easy way to deduce this. This test and the transmission FIFO loopback test should implicitly tell us the state of the programming.

We know that the transmission Xilinx generates a violation if we have transmission cable connected incorrectly or disconnected. Therefore if we have the loopback cable plug in, we should not see any violation.

Possible error mode:

1. ERROR: TXRX Test failed : Violations

   We still see violation when loopback cable is in. Something must be wrong. Check your connection and cable.

6.2.5 Podule Test

The Podule test tries to examine the integrity of lines to the Podule backplane. A simple Podule was constructed that allows us to first write the Podule and read it back for comparison. The test will search for the right Podule ID before the write/read. Therefore you can plug in any Podule slot on the backplane.

1. ERROR: test podule: error: XXX != YYY err=ZZZ

   The read back data YYY does not agree with data written XXX. err is the cumulative bit error (done using ORing all XOR of XXX and YYY). Using err we can then deduce what has gone wrong with the Table 4.

6.2.6 Cell loop back Test

The last two tests try to send a single cell to itself via the loop back FIFO and the transmission FIFO. This will exercise almost all parts in the network section of the FPC3 and therefore if the tests pass, we can declare the FPC3 to be in a sane state.

Possible error states are:

Code	Address/Data line
0x01	D0
0x02	A2/D1
0x04	A3/D2
0x08	A4/D3
0x10	A5/D4
0x20	A6/D5
0x40	A7/D6
0x80	A8/D7
0x0100	D8
0x0200	A9/D9
0x0400	A10/D10
0x0800	A11/D11
0x1000	A12/D12
0x2000	A13/D13
0x4000	A14/D14
0x8000	A15/D15

Table 4: Podule Error Code

1. Cell Was NACKed This is the most likely error. One possible cause is when FPC3 is in the wrong slot (in technical term, we are using IFAB 5).
2. ERROR: Lost Cell
3. Free Q is empty
4. REMAP Error
5. Recv cell had XX errors:
6. after errors
7. Left slip
8. Right slip
9. Sent from word

Normally when error occurs, beside the basic message, a whole bunch of status information is also dumped on the screen. If error other than 1 occurs, something serious has happen. Check FPC3 has been plug in correctly in the right slot, loopback cable is in, fabric is in.

7 Interactive Mode

Once the auto test completed, we are drop into interactive mode. The user interface is rather rudimentary but does allow character input and backspace (CTRL H). You can choose option 1 to repeat the xiromtester tests.

In xiromtester, you can select tests by specifying the test numbers in sequence follow by return. This will only construct a task list. xiromtester will only start executing the test if you say 'e'xecute. Choosing the 'a'll option will be equivalent to the auto test (if further interactive mode is not turned on).

The tests have been constructed to be configurable. The auto mode has an option to turn off interactive setting of parameter. You can use the 't' option to toggle the mode and input a different parameter when you select a test.

8 Xi5tester

The xi5tester (`master/kernel/kernel/machdep/arm/xi5tester.c`) is a test suite for both Xi5 and Xi6 bits. Xi5tester is expected to be a continuing evolving program. As of the time of writing this document, xi5tester performs the following basic tests:

- Test Reset State
- Reset FIFOs
- goXilinx
- goTxRx
- Test SRAM
- Test Swap
- Test LDM STM
- Test ArmP Write
- Test HeadP write
- Dump Status
- Configure TxRx
- TC FRC check
- Get cells for Wanda
- Send Cells (MDH)
- Send Cells (NEW)
- Send Short Cells
- Back-to-Back cells
- DelayStamp Test
- Cell Generator
- Cell Sink

Most of these tests are self-explanatory. "`Send Cells (MDH)`" and "`Send Cells (NEW)`" are both working version of single cell test which expects the cell to be loop back through the loopback FIFO or the transmission FIFO. However, the `Send Cells (NEW)` has correct handling of the situation with lost cells and replicated cells (when doing serial multicast).

The xi5tester uses a greatly simplified version of FIQ code where the only action to take when a FIQ happens is to post an always IRQ. Thus when a FPC is booted with a xi5tester kernel, it is not able to perform normal communications on the ATM interfaces. If the FPC also has Ethernet interfaces, you should still be able to have communications with it and perform operations like remote reset (wreset).

8.1 Adding Tests to Xi5tester

Xi5tester has been designed to be modular and allows easy addition of further tests. Each test is executed independently and has been written with few side-effects.

The structure that requires particular attention is xi5test, defined in xi5tester.h, whose fields are as shown in Table 5:

Field	Comments
test	routine that actually execute the test
init	routine that init reqd for test to work
getparms	routine that does I/O to get params for test
exit	routine that clean up any mess
irq	test specified irq handler
parms	extra params for this test
enabled	execute or not
name	name of test (use in menu)
count	number of times to execute
interactive	solicit params or not
pr_level	level of debugging info
num_subtests	the number of subtests
subtest	any subtests for this test
next	we keep a list of tests

Table 5: xi5test

For each new test added, you would need to increment the `NUM_TESTS` in `xi5tester.c` and initialised a new entry in `tblock`. Five routines are needed for each test. For most tests, default routines can be use for `exit (xi5_exit_null)` and `irq (xi5_irq_null)`. Some of the useful functions are listed in Table 6.

It is recommended that before any new test is written, the existing functions in xi5tester be explored and that one tries to construct the new test from some of the existing functions where possible.

The order of execution of the routines in the xi5test structure is as follows: getparms is performed for every test selected, with the parameters kept in parms. The xi5tester forms a linked-list of test selected. When (e)xecute selection is chosen, xi5tester goes through the

Routine	Functionality
xi5_init_null	Null initialisation
xi5_getp_null	Don't need to get parameters
xi5_exit_null	Null exit routine
xi5_irq_null	Just kick the semaphore irq_sem
xi5_exit_free	Just free the structure holding the parameters
xi5_dump_status	Dump Xilinx status
xi5_test_reset	reset Xilinx
xi5_getp_cells	Get parameters for a normal cell test
resetFifos	reset FIFO
Wanda_Get_Cell	Request a Wanda cell

Table 6: Some Useful Routines

linked-list and executes each test in sequence. For each test `init` is first done, follows by the execute of the actual `test` and finally `exit` is called to clean up the state.

It should be noted that for each of the test selected, it inherits the `enabled` and `count` field from the previous test.