Amber Open Source Project

User Guide

May 2013
Table of Contents

1 Amber Project ................................................................................................................................................. 3
  1.1 Project Directory Structure ..................................................................................................................... 3
  1.2 Amber FPGA System ................................................................................................................................. 4
2 Verilog simulations .............................................................................................................................................. 6
  2.1 Installing the Amber project ....................................................................................................................... 6
  2.2 Installing the Compiler ............................................................................................................................... 6
  2.3 Running Simulations .................................................................................................................................... 7
  2.4 Simulation output files ................................................................................................................................. 10
  2.5 Hardware Tests ............................................................................................................................................. 12
  2.6 C Programs .................................................................................................................................................. 14
  2.7 Linux ........................................................................................................................................................... 16
3 FPGA Synthesis .................................................................................................................................................. 18
4 Using Boot-Loader ............................................................................................................................................. 20
  4.1 Install and configure Minicom .................................................................................................................... 20
  4.2 Configure the FPGA .................................................................................................................................... 21
5 License .................................................................................................................................................................. 22
1 Amber Project

The Amber project is a complete embedded system implemented on the Xilinx Spartan-6 SP605 FPGA development board. The project is hosted on opencores.org. The project provides a complete hardware and software development system based around the Amber processor core. A number of applications, with C source code, are provided as examples of what the system can be used for.

The embedded system includes the Amber processor core, the Ethmac open source Ethernet MAC, a UART serial port, a timer and an interrupt controller.

The recommended development system for the project is the Xilinx SP605 development board, a PC running CentOS 6.x or 7.x, the Xilinx ISE 14.7 tool chain (free Webpack version), and the Code Sorcery GNU toolchain for ARM processors. All of these elements are free except for the actual development board which costs $495. All the information and instructions in this document are for that development system.

1.1 Project Directory Structure

The following table describes the directories and sub-directories located under $AMBER_BASE.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>Contains all project documentation.</td>
</tr>
<tr>
<td>hw</td>
<td>Contains all Verilog source files, simulations and synthesis scripts, and hardware test source files.</td>
</tr>
<tr>
<td>hw/fpga</td>
<td>Files relating to FPGA synthesis.</td>
</tr>
<tr>
<td>hw/fpga/bin</td>
<td>Contains the FPGA synthesis makefile and supporting scripts.</td>
</tr>
<tr>
<td>hw/fpga/bitfiles</td>
<td>This directory is created during the FPGA synthesis process. It is used to store the final bitfile generated at the end of the FPGA synthesis process.</td>
</tr>
<tr>
<td>hw/fpga/log</td>
<td>This directory is created during the FPGA synthesis process. It is used to store log files for each step of the FPGA synthesis process.</td>
</tr>
<tr>
<td>hw/fpga/work</td>
<td>This directory is created during the FPGA synthesis process. It is used to store temporary files created during the FPGA synthesis process. These files get erased when a new synthesis run is started.</td>
</tr>
<tr>
<td>hw/isim</td>
<td>Where tests are run from. The Xilinx iSim Verilog simulator work directory, wave dump and any other simulation output files go in here.</td>
</tr>
<tr>
<td>hw/tests</td>
<td>Holds a set of hardware tests written in assembly. These tests focus on verifying the correct operation of the instruction set. If any modifications are made to the Amber core it is important that these tests still pass.</td>
</tr>
<tr>
<td>hw/tools</td>
<td>Holds scripts used to run Verilog simulations.</td>
</tr>
<tr>
<td>hw/vlog</td>
<td>Verilog source files.</td>
</tr>
<tr>
<td>hw/vlog/amber23</td>
<td>Amber 23 core Verilog source files.</td>
</tr>
<tr>
<td>hw/vlog/amber25</td>
<td>Amber 25 core Verilog source files.</td>
</tr>
<tr>
<td>hw/vlog/ethmac</td>
<td>The Ethernet MAC Verilog source files. These files come from the Opencores Ethmac project and are reproduced here for convenience.</td>
</tr>
<tr>
<td>hw/vlog/lib</td>
<td>Hardware library Verilog files including memory models. The Amber project provides a simple generic library that is normally used for simulations. It also provides some wrappers for Xilinx library elements.</td>
</tr>
</tbody>
</table>

Released under the GNU Lesser General Public License (v2.1) terms
## Directory Description

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw/vlog/system</td>
<td>FPGA system Verilog source files.</td>
</tr>
<tr>
<td>hw/vlog/tb</td>
<td>Testbench Verilog files.</td>
</tr>
<tr>
<td>hw/vlog/xs6_ddr3</td>
<td>Xilinx Spartan-6 DDR3 controller Verilog files go in here. These are not provided with the project for copyright reasons. They are needed to implement the Amber system on a Spartan-6 development board and must be generated in Xilinx Coregen.</td>
</tr>
<tr>
<td>hw/vlog/xv6_ddr3</td>
<td>Xilinx Virtex-6 DDR3 controller Verilog files go in here. These are not provided with the project for copyright reasons. They are needed to implement the Amber system on a Virtex-6 development board and must be generated in Xilinx Coregen.</td>
</tr>
<tr>
<td>sw</td>
<td>Contains C source files for applications that run on the Amber system, as well as some utilities that aid in debugging the system.</td>
</tr>
<tr>
<td>sw/boot-loader-serial</td>
<td>C, assembly sources and a makefile for the serial-port boot-loader application.</td>
</tr>
<tr>
<td>sw/boot-loader-ethmac</td>
<td>C, assembly sources and a makefile for the ethernet-port boot-loader application. This application supports telnet for control and status, and tftp for uploading elf executable files.</td>
</tr>
<tr>
<td>sw/hello-world</td>
<td>C, assembly source and a makefile for a simple stand-alone application example.</td>
</tr>
<tr>
<td>sw/include</td>
<td>Common C, assembly and makefile include files.</td>
</tr>
<tr>
<td>sw/mini-libc</td>
<td>C, assembly sources and a makefile to build the object that comprise a very small and limited stand-alone replacement for the libc library.</td>
</tr>
<tr>
<td>sw/tools</td>
<td>Shell scripts and C source files for compile and debug utilities.</td>
</tr>
<tr>
<td>sw/vmlinux</td>
<td>Contains the .mem and .dis files for the vmlinux simulation.</td>
</tr>
</tbody>
</table>

### 1.2 Amber FPGA System

The FPGA system included with the Amber project is a complete embedded processor system which included all peripherals needed to run Linux, including UART, timers and an Ethernet (MII) port. The following diagram shows the entire system.
All the Verilog source code was specifically developed for this project with the exception of the following modules:

- **ddr3.v**. The Xilinx Spartan-6 DDR3 controller was generated by the Xilinx Coregen tool. The files are not included with the project for copyright reasons. It is up to the user to obtain the ISE software from Xilinx and generate the correct memory controller. Note that Wishbone bridge modules are included that support both the Xilinx Spartan-6 DDR3 controller and the Virtex-6 controller.

- **eth_top.v**. This module is from the Opencores Ethernet MAC 10/100 Mbps project. The Verilog code is included for convenience. It has not been modified, except to provide a memory module for the Spartan-6 FPGA.
2 Verilog simulations

2.1 Installing the Amber project

If you have not already done so, you need to download the Amber project from Opencores.org. The Amber project includes all the Verilog source files, tests written in assembly, a boot loader application written in C and scripts to compile, simulate and synthesize the code. You can either download a tar.gz file from the Opencores website or better still, connect to the Opencores Subversion server to download the project. This can be done on a Linux PC as follows;

$ mkdir /<your amber install path>/
$ cd /<your amber install path>/
$ svn --username <your opencores account name> --password <your opencores password> \\ co http://opencores.org/ocsvn/amber/amber/trunk

2.2 Installing the Compiler

Tests need to be compiled before you can run simulations. You need to install a GNU cross-compiler to do this. The easiest way to install the GNU tool chain is to download a ready made package. Code Sourcery provides a free one. To download the Code Sourcery package, go to this page

http://www.codesourcery.com/sgpp/lite/arm

You need to register and will be sent an email to access the download area. Select the GNU/Linux version and then the IA32 GNU/Linux Installer. Once the package is installed, add the following to your .bashrc file, where the PATH is set to where you install the Code Sourcery GNU package.

# Change /proj/amber to where you saved the amber package on your system
export AMBER_BASE=/<your amber install path>/trunk

# Change /opt/Sourcery to where the package is installed on your system
PATH=/<your code sourcery install path>/bin:$PATH

# AMBER_CROSSTOOL is the name added to the start of each GNU tool in
# the Code Sourcery bin directory. This variable is used in various makefiles to set
# the correct tool to compile code for the Amber core
export AMBER_CROSSTOOL=arm-none-linux-gnueabi

# Xilinx ISE installation directory
# This should be configured for you when you install ISE.
# But check that is has the correct value
# It is used in the run script to locate the Xilinx library elements.
export XILINX=/opt/Xilinx/14.5/ISE

2.2.1 GNU Tools Usage

It's important to remember to use the correct switches with the GNU tools to restrict the ISA to the set of instructions supported by the Amber 2 core. The switches are already set in the makefiles included with the Amber 2 core. Here are the switches to use with gcc (arm-none-linux-gnueabi-gcc);

-march=armv2a -mno-thumb-interwork
These switches specify the correct version of the ISA, and tell the compiler not to create bx instructions. Here is the switch to use with the GNU linker, arm-none-linux-gnueabi-ld;

```
--fix-v4bx
```

This switch converts any bx instructions (which are not supported) to 'mov pc, lr'.

Here is an example usage from the boot-loader make process;

```
arm-none-linux-gnueabi-gcc -c -Os -march=armv2a -mno-thumb-interwork -ffreestanding
-I../include  -c -o boot-loader.o boot-loader.c
arm-none-linux-gnueabi-gcc -I../include  -c -o start.o start.s
arm-none-linux-gnueabi-gcc -c -Os -march=armv2a -mno-thumb-interwork -ffreestanding
-I../include  -c -o crc16.o crc16.c
arm-none-linux-gnueabi-gcc -c -Os -march=armv2a -mno-thumb-interwork -ffreestanding
-I../include  -c -o xmodem.o xmodem.o
arm-none-linux-gnueabi-gcc -c -Os -march=armv2a -mno-thumb-interwork -ffreestanding
-I../include  -c -o elfsplitter.o elfsplitter.c
../tools/amber-memparams.sh boot-loader.mem boot-loader_memparams32.v
arm-none-linux-gnueabi-objdump -C -S -EL boot-loader.elf > boot-loader.dis
```

A full list of compile switches for gcc can be found here;

http://gcc.gnu.org/onlinedocs/gcc-4.5.2/gcc/ARM-Options.html#ARM-Options

And for ld here;

http://sourceware.org/binutils/docs-2.21/ld/ARM.html#ARM

2.3 Running Simulations

You should be able to use any Verilog-2001 compatible simulator to run simulations.

The project comes with run scripts and project files for the free Xilinx Webpack ISim 14.5 simulator.

Example usage:

```
$ cd $AMBER_BASE/hw/isim
$ .run.sh hello-world
Test hello-world, type 4
make -s -C ../lib  MIN_SIZE=1
arm-none-linux-gnueabi-gcc -c -Os -march=armv2a -mno-thumb-interwork -ffreestanding
-I../include  -c -o boot-loader-serial.o boot-loader-serial.c
../tools/amber-memparams.sh boot-loader-serial.mem boot-loader-serial_memparams32.v
../tools/amber-memparams.sh boot-loader-serial.mem boot-loader-serial_memparams128.v
arm-none-linux-gnueabi-objdump -C -S -EL boot-loader-serial.elf > boot-loader-serial.dis
../tools/check_mem_size.sh boot-loader-serial.elf  "@000020"
maker -s -C ../vlog/lib  MIN_SIZE=1
```

Released under the GNU Lesser General Public License (v2.1) terms
log file tests.log, timeout 0, test name hello-world
Load main memory from ../../sw/hello-world/hello-world.mem
Read in 9116 lines
Finished circuit initialization process.
Amber Boot Loader v20130428143120
j 0x00008000

Hello, World!

Amber Core
> User         FIRQ         IRQ          SVC
r0       0x00000010
r1       0x00000000
r2       0x00000000
r3       0x00000000
r4       0x00000000
r5       0x0deadbeef
r6       0x0deadbeef
r7       0x0deadbeef
r8       0x0deadbeef
r9       0x0deadbeef
r10      0x00000011
r11      0x00000000
r12      0x00000000
r13      0x0deadbeef
r14      0x0deadbeef
r15      0x0deadbeef

Status Bits: N=0, Z=1, C=1, V=0, IRQ Mask 0, FIRQ Mask 0, Mode = User

+++++++++++++++++++++++++++++++++
Passed hello-world 47634 ticks
+++++++++++++++++++++++++++++++++
Stopped at time : 1191327500 ps : File "/proj/amber_trunk_working/hw/vlog/th/tb.v"
Line 503

2.4 Simulation output files

2.4.1 Disassembly Output File

The disassembly file, amber.dis, is generated by default during a simulation. It is located in the $AMBER_BASE/hw/sim directory. This file is very useful for debugging software as it shows every instruction executed by the core and the result of all load and store operations.

This file is generated by default. To turn off generation, comment the line where AMBER_DECOMPILE is defined in $AMBER_BASE/hw/vlog/amber/amber_config_defines.v.

Below is an example of the disassembly output file. The first column gives the time that the instruction was executed. The time is specified in sys_clk ticks. The second column gives the address of the instruction being executed and the next column gives the instruction. If an instruction is not executed because of a conditional execution code, this is marked with a '-' character in front of the instruction. For load and store instructions, the actual memory access is given below the instruction. This is the complete listing for the add test.

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>264</td>
<td>0: mov r1, #3</td>
</tr>
<tr>
<td>267</td>
<td>4: mov r2, #1</td>
</tr>
<tr>
<td>270</td>
<td>8: add r3, r1, r2</td>
</tr>
<tr>
<td>273</td>
<td>c: cmp r3, #4</td>
</tr>
<tr>
<td>276</td>
<td>10: movne r10, #10</td>
</tr>
<tr>
<td>279</td>
<td>14: -hne b4</td>
</tr>
</tbody>
</table>
2.4.2

Figure 2 - GTKWave waveform viewer

2.4.3 Program Trace Utility

A utility is provided that traces all function calls made during a Verilog simulation. Here is an example usage:

```
$ cd $AMBER_BASE/hw/sim
$ run ethmac-test
$ ln -s ../../sw/tools/amber-jumps.sh jumps
$ jumps ethmac-test
```

This produces the following output. The left column gives the time of the event. The next column gives the name of the calling function. The next column gives the value of the r0 register. This register holds the first parameter passed in function calls. The next column gives the name of the function called.

```
276031 u main -> ( 00008dec, ) printf u
276104 u printf -> ( 07ffff8c, ) print u
276311 u print -> ( 00000053, ) _outbyte u
```
2.5 Hardware Tests

The Amber package contains a set of tests which are used to verify the correct operation of all the instructions, interrupts, the cache and peripherals. The tests are written in assembly. Several of the tests were added when a specific bug was found while debugging the core. To run one of the tests, use run <test-name>, e.g.

```
$ cd $AMBER_BASE/hw/sim
$ run barrel_shift
```

Each test generates pass or fail when it completes, e.g.

```
# ++++++++++++++++++++
# Passed barrel_shift
# ++++++++++++++++++++
```

To run the complete test suite;

```
$ cd $AMBER_BASE/hw/sim
$ run -a
```

Once the run is complete look at the output file hw-tests.log in the $AMBER_BASE/hw/sim/ directory to check the results. All tests should pass.

The following table describes each test. The source files for these tests are in the directory $AMBER_BASE/hw/tests.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adc</td>
<td>Tests the adc instruction. Adds 3 32-bit numbers using adc and checks the result.</td>
</tr>
<tr>
<td>addr_ex</td>
<td>Tests an address exception interrupt. Sets the pc to 0x3ffffffc and executes a nop. The pc then increments to 0x4000000 triggering an address exception.</td>
</tr>
<tr>
<td>add</td>
<td>Tests the add instruction. Runs through a set of additions of positive and negative numbers, checking that the results are correct. Also tests that the ‘s’ flag on the instruction correctly sets the condition flags.</td>
</tr>
<tr>
<td>barrel_shift_rs</td>
<td>Tests the barrel shift operation with a mov instruction, when the shift amount is a register value. Test that shift of 0 leaves Rm unchanged. Tests that a shift of &gt; 32 sets Rm and carry out to 0.</td>
</tr>
<tr>
<td>barrel_shift</td>
<td>Tests the barrel shift operation with a mov instruction when the shift amount is an immediate value. Tests lsl, lsr and ror.</td>
</tr>
<tr>
<td>bcc</td>
<td>Tests branch on carry clear.</td>
</tr>
<tr>
<td>bic_bug</td>
<td>Test added to catch specific bug with the bic instruction. The following instruction stored the result in r3, instead of r2. lst r2, r0, lsl r3 bicne r2, r2, r0, lsl r3</td>
</tr>
<tr>
<td>bl</td>
<td>Test Branch and Link instruction. Checks that the correct return address is stored in the link register (r14).</td>
</tr>
<tr>
<td>cache1</td>
<td>Contains a long but simple code sequence. The entire sequence can fit in the cache. This sequence is executed 4 times, so three times it will execute from the cache. Test passes if sequence executes correctly.</td>
</tr>
<tr>
<td>cache2</td>
<td>Tests simple interaction between cached data and uncached instruction accesses.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cache3</td>
<td>Tests that the cache can write to and read back multiple times from 2k words in sequence in memory - the size of the cache.</td>
</tr>
<tr>
<td>cacheable_area</td>
<td>Tests the cacheable area co-processor function.</td>
</tr>
<tr>
<td>cache_flush</td>
<td>Tests the cache flush function. Does a flush in the middle of a sequence of data reads. Checks that all the data reads are correct.</td>
</tr>
<tr>
<td>cache_swap_bug</td>
<td>Tests the interaction between a swap instruction and the cache. Runs through a main loop multiple times with different numbers of nop instructions before the swap instruction to test a range of timing interactions between the cache state machine and the swap instruction.</td>
</tr>
<tr>
<td>cache_swap</td>
<td>Fills up the cache and then does a swap access to data in the cache. That data should be invalidated. Check by reading it again.</td>
</tr>
<tr>
<td>change_mode</td>
<td>Tests teq, tst, cmp and cmn with the p flag set. Starts in supervisor mode, changes to Interrupt mode then Fast Interrupt mode, then supervisor mode again and finally User mode.</td>
</tr>
<tr>
<td>change_sbits</td>
<td>Change status bits. Tests moves where the destination register is r15, the pc. Depending on the processor mode and whether the s bit is set or not, some or none of the status bits will change.</td>
</tr>
<tr>
<td>ddr31</td>
<td>Word accesses to random addresses in DDR3 memory. The test creates a list of addresses in an area of boot_mem. It then writes to all addresses with data value equal to address. Finally it reads back all locations checking that the read value is correct.</td>
</tr>
<tr>
<td>ddr32</td>
<td>Tests byte read and write accesses to DDR3 memory.</td>
</tr>
<tr>
<td>ddr33</td>
<td>Test back to back write-read accesses to DDR3 memory.</td>
</tr>
<tr>
<td>ethmac_mem</td>
<td>Tests wishbone access to the internal memory in the Ethernet MAC module.</td>
</tr>
<tr>
<td>ethmac_reg</td>
<td>Tests wishbone access to registers in the Ethernet MAC module.</td>
</tr>
<tr>
<td>ethmac_tx</td>
<td>Tests ethernet MAC frame transmit and receive functions and Ethmac DMA access to hiboot mem. Ethmac is put in loopback mode and a packet is transmitted and received.</td>
</tr>
<tr>
<td>firq</td>
<td>Executes 20 FIRQs at random times while executing a small loop of code. The interrupts are triggered using a ransom timer. Test checks the full set of FIRQ registers (r8 to r14) and will only pass if all interrupts are handled correctly.</td>
</tr>
<tr>
<td>flow_bug</td>
<td>The core was illegally skipping an instruction after a sequence of 3 conditional not-execute instructions and 1 conditional execute instruction.</td>
</tr>
<tr>
<td>flow1</td>
<td>Tests instruction and data flow. Specifically tests that a stm writing to cached memory also writes all data through to main memory.</td>
</tr>
<tr>
<td>flow2</td>
<td>Tests that a stream of str instructions writing to cached memory works correctly.</td>
</tr>
<tr>
<td>flow3</td>
<td>Tests ldm where the pc is loaded which causes a jump. At the same time the mode is changed. This is repeated with the cache enabled.</td>
</tr>
<tr>
<td>inflate_bug</td>
<td>A load store sequence was found to not execute correctly.</td>
</tr>
<tr>
<td>irq</td>
<td>Tests running a simple algorithm to add a bunch of numbers and check that the result is correct. This algorithm runs 80 times. During this, a whole bunch of IRQ interrupts are triggered using the random timer.</td>
</tr>
<tr>
<td>ldm_stm_onetwo</td>
<td>Tests ldm and stm of single registers with cache enabled. Tests ldm and stm of 2 registers with cache enabled.</td>
</tr>
<tr>
<td>ldm1</td>
<td>Tests the standard form of ldm.</td>
</tr>
<tr>
<td>ldm2</td>
<td>Tests ldm where the user mode registers are loaded whilst in a privileged mode.</td>
</tr>
<tr>
<td>ldm3</td>
<td>Tests ldm where the status bits are also loaded.</td>
</tr>
<tr>
<td>ldm4</td>
<td>Tests the usage of ldm in User Mode where the status bits are loaded. The s bit should be ignored in User Mode.</td>
</tr>
<tr>
<td>ldr</td>
<td>Tests ldr and ldrb with all the different addressing modes.</td>
</tr>
<tr>
<td>ldr_atr_pc</td>
<td>Tests lrd and str of r15.</td>
</tr>
<tr>
<td>mla</td>
<td>Tests the mla (multiply and accumulate) instruction.</td>
</tr>
<tr>
<td>mlas_bug</td>
<td>Bug with Multiply Accumulate. The flags were getting set 1 cycle early.</td>
</tr>
<tr>
<td>movs_bug</td>
<td>Tests a movs followed by a sequence of ldr and str instructions with different condition fields.</td>
</tr>
<tr>
<td>mul</td>
<td>Tests the mul (multiply) instruction.</td>
</tr>
<tr>
<td>sbc</td>
<td>Tests the 'subtract with carry' instruction by doing 3 64-bit subtractions.</td>
</tr>
<tr>
<td>stm_stream</td>
<td>Generates as dense a stream of writes as possible to check that the memory subsystem can cope.</td>
</tr>
</tbody>
</table>
## Name | Description
--- | ---
STM1 | Tests the normal operation of the STM instruction.
STM2 | Test jumps into user mode, loads some values into registers r8 - r14, then jumps to FIRQ and saves the user mode registers to memory.
STRB | Tests STR and STRB with different indexing modes.
SUB | Tests SUB and SUBS.
SWI | Tests the software interrupt – SWI.
SWP_LOCK_BUG | Bug broke an instruction read immediately after a SWP instruction.
SWP | Tests SWP and SWPB.
UART_REG | Tests Wishbone read and write access to the Amber UART registers.
UART_RX | Tests the UART receive function. Some text is sent from the test_uart to the uart and an interrupt generated.
UART_TX | Uses the tb_uart in loopback mode to verify the transmitted data.
UNDEFINED_INS | Tests Undefined Instruction Interrupt. Fires a few unsupported floating point unit (FPU) instructions into the core. These cause undefined instruction interrupts when executed.

### 2.6 C Programs

In addition to the short assembly language tests, some longer programs written in C are included with the Amber system. These can be used to further test and verify the system, or as a basis to develop your own applications.

The source code for these programs is in $AMBER_BASE/sw.

#### 2.6.1 Serial Boot Loader

This is located in $AMBER_BASE/sw/boot-loader-serial. It can be run in simulation as follows;

```bash
$ cd $AMBER_BASE/sw/ism
$ ./run.sh boot-loader-serial
```

The simulation output looks like the following:

```plaintext
# Test boot-loader, log file boot-loader.log
# Load boot memory from ../../sw/boot-loader/boot-loader.mem
# Read in 1928 lines
# Amber Boot Loader v20110202130047
# Commands
# l : Load elf file
# b <address> : Load binary file to <address>
# d <start address> <num bytes> : Dump mem
# h : Print help message
# j : Execute loaded elf, jumping to 0x00080000
# p <address> : Print ascii mem until first 0
# r <address> : Read mem
# s : Core status
# w <address> <value> : Write mem
# r 0 0000000c
# r 1 00001b76
# r 2 00000000
# r 3 00000000
# r 4 deadbeef
# r 5 deadbeef
# r 6 deadbeef
# r 7 deadbeef
# r 8 deadbeef
```
The boot loader is used to download longer applications onto the FPGA development board via the UART port and using Hyper Terminal on a host Windows PC.

### 2.6.2 Hello World

This is located in `/AMBER_BASE/sw/hello-world`. It can be run in simulation as follows;

```
$ cd $AMBER_BASE/hw/isim
$ ./run.sh hello-world
```

This is a very simple example of a stand alone C program. The printf function it uses is contained in `/AMBER_BASE/sw/mini-libc`, so that it can run on an FPGA without access to a real libc library file.

### 2.6.3 Ethmac Boot Loader

This is located in `/AMBER_BASE/sw/boot-loader-ethmac`. This is an 'over the network' boot loader. It supports telnet for command and status, and tftp for uploading executable programs (as elf files) to run on the FPGA.

The IP address is hard-coded in `/AMBER_BASE/sw/boot-loader-ethmac/packet.c`, line 56. To change it, edit that file and rebuild the FPGA, creating a new bitfile.

Here's an example usage of the boot-loader:

```
$ telnet 192.168.0.17
Trying 192.168.0.17...
Connected to 192.168.0.17.
Escape character is '^]'.
Amber Processor Boot Loader
```
2.7 Linux

A memory file is provided to run a simulation of Linux booting. The main reason for providing this file is to have a long test to further validate the correct operation of the core. This file was created from a modified version of the 2.4.27 kernel with the patch-2.4.27-vrs1.bz2 patch file applied and then some modifications made to source files to support the specific hardware in the Amber 2 FPGA.

The vmlinux.mem memory file contains an embedded ext2 format ramdisk image which contains the hello-world program, but renamed as /sbin/init. The kernel mounts the ramdisk as /dev/root and runs init. This program prints "Hello, World" and writes the test pass value to the simulation control register. To run this simulation:

```
$ cd $AMBER_BASE/hw/isim
$ ./run.sh vmlinux
```

This simulation takes about 6 million ticks to run to completion, or between 5 minutes and an hour of wall time depending on your simulator and PC. The following is the output from this simulation:

```
# Amber Boot Loader v20110117211518
# J 0x2080000
# Linux version 2.4.27-vrs1 (conor@server) (gcc version 4.5.1 (Sourcery G++ Lite 2010.09-50) ) #354 Tue Feb 1 17:56:00 GMT 2011
# CPU: Amber 2 revision 0
# Machine: Amber-FPGA-System
# On node 0 totalpages: 1024
# zone(0): 1024 pages.
# zone(1): 0 pages.
# zone(2): 0 pages.
# Kernel command line: console=ttyAMO mem=32M root=/dev/ram
# Calibrating delay loop... 19.91 BogoMIPS
# Memory: 32MB = 32MB total
# Memory: 31136KB available (493K code, 195K data, 32K init)
# Dentry cache hash table entries: 4096 (order: 0, 32768 bytes)
# Inode cache hash table entries: 4096 (order: 0, 32768 bytes)
# Mount cache hash table entries: 4096 (order: 0, 32768 bytes)
# Buffer cache hash table entries: 8192 (order: 0, 32768 bytes)
# Page-cache hash table entries: 8192 (order: 0, 32768 bytes)
# POSIX conformance testing by UNIFIX
# Linux NET4.0 for Linux 2.4
# Based upon Swansea University Computer Society NET3.039
# Starting kswapd
# ttyAMO at MMIO 0x16000000 (irq = 1) is a WSBN
# pty: 256 Unix98 pty configured
# RAMDISK: driver initialized: 16 RAM disks of 208K size 1024 blocksize
# NetMinder Floating Point Emulator V0.97 (double precision)
# RAMDISK: ext2 filesystem found at block 8388608
# RAMDISK: Loading 200 blocks [1 disk] into ram disk... done.
# Freeing initrd memory: 208K
```
The program trace utility can be used to trace the Linux execution, as follows:

```
$ cd $AMBER_BASE/hw/sim
$ ln -s ../../sw/tools/amber-jumps.sh jumps
$ jumps vmlinux
```
3 FPGA Synthesis

A makefile is provided that performs synthesis of the system to a Xilinx Spartan-6 FPGA. To use this makefile you must have Xilinx ISE installed. I have tested it with ISE v14.5. The makefile is quite flexible. To see all its options, type;

```
$ cd $AMBER_BASE/hw/fpga/bin
$ make help
```

To use the script to perform a complete synthesis run from start to finish and generate a bitfile;

```
$ cd $AMBER_BASE/hw/fpga/bin
$ chmod +x *.sh
$ make new
```

The script performs the following steps

1. Compiles the boot loader program in $AMBER_BASE/sw/boot-loader, to ensure the latest version goes into the boot_mem ram blocks.

2. Runs xst to synthesize the top-level Verilog file $AMBER_BASE/hw/vlog/system/system.v and everything inside it.

3. Runs ngbbuild to create the initial FPGA netlist.

4. Runs map to do placement.

5. Runs par to do routing.

6. Runs bitgen to create an FPGA bitfile in the bitfile directory.

7. Runs trce to do timing analysis on the finished FPGA.

The Spartan-6 FPGA target device is the default. To compile for the Virtex-6 FPGA, set VIRTEX6=1 on the command line, e.g.

```
$ cd $AMBER_BASE/hw/fpga/bin
$ make new VIRTEX6=1
```

The Amber 23 core is the default. To synthesize the Amber 25 core instead, set A25=1 on the command line, e.g.

```
$ cd $AMBER_BASE/hw/fpga/bin
$ make new A25=1
```

If the par step fails (timing or area constrains not met), you can rerun map and par with a different seed. Simply call the makefile again without the new switch. The makefile will automatically increment the seed, e.g.

```
$ cd $AMBER_BASE/hw/fpga/bin
$ make new
```
$ make

The system clock speed is configured within the FPGA makefile, $AMBER_BASE/hw/fpga/bin/Makefile. To change it, change the value of AMBER_CLK_DIVIDER in that file. The system clock frequency is equal to the PLL's VCO clock frequency divided by AMBER_CLK_DIVIDER. By default it is set to 40MHz for Spartan-6 and 80MHz for Virtex-6.
### 4 Using Boot-Loader

If you have a development board with a UART connection to a PC you can use boot-loader to download and run applications on the board. I have tested this with the Xilinx SP605 development board. It provides a UART connection via a USB port on the board.

#### 4.1 Install and configure Minicom

The following commands installs the lsusb and minicom utilities;

```
$ sudo yum install usbutils
$ sudo yum install minicom
```

Connect the SP605 serial port USB to the PC and check that the port is visible;

```
ls -l /dev/ttyUSB0
```

```
crw-rw---- 1 root dialout 188, 0 May  4 11:02 /dev/ttyUSB0
```

```
lsusb
```

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 002: ID 05e3:0608 Genesys Logic, Inc. USB-2.0 4-Port HUB
Bus 005 Device 003: ID 04f3:0212 Elan Microelectronics Corp. Laser Mouse
Bus 002 Device 013: ID 03fd:0008 Xilinx, Inc.
Bus 008 Device 006: ID 10c4:ea60 Cygnal Integrated Products, Inc. CP210x UART Bridge / myAVR mySmartUSB light
```

Configure minicom

```
sudo minicom -s
```

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 921600 8N1                                |
| F - Hardware Flow Control : Yes                                       |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

Save setup as dfl. Then to run minicom,

```
> sudo minicom
```

Released under the GNU Lesser General Public License (v2.1) terms
4.2 Configure the FPGA

Load the bitfile into the FPGA on the development board. This can be done using Xilinx iMPACT. Once the FPGA is configured the boot loader will print some messages via the UART interface onto the minicom screen, as follows;
5 License

All source code provided in the Amber package is release under the following license terms;

---

Copyright (C) 2010 Authors and OPENCORES.ORG

This source file may be used and distributed without restriction provided that this copyright statement is not removed from the file and that any derivative work contains the original copyright notice and the associated disclaimer.

This source file is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This source is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this source; if not, download it from http://www.opencores.org/lgpl.shtml

Author(s):
- Conor Santifort, csantifort.amber@gmail.com

---