Summary

This document describes my implementation of a 6502 microprocessor into a Lattice LCMXO2280C FPGA. The hardware is based on an existing PCB leftover from a controller for a soft X-ray generator. It is anticipated that the design can be ported to other FPGA devices. It is written in VHDL and used GNU ghdl for initial development and Lattice ispLeaver to implement the design into the FPGA. The GNU software was found to be significantly faster than Lattice. Files and information is provided to implement the design into both Lattice parts and hopefully other parts.

The 65C02 instructions have yet to be implemented. The timing bears no relationship with that of a genuine 6502; it is intended to execute the instruction set as fast as possible. This will make it very difficult to use on an Apple 2 or other 6502 based PC.

Also included are a UART TX and RX and sufficient assembly language firmware to communicate to a dumb terminal (IE Linux gtkterm). This provides a simple OS, Kernel or monitor that is able to display and modify memory and load Motorola S code. This monitor is burnt into the FPGAs ROM. The capability to load and execute a program via the terminal is much faster than re-generating the FPGA each time a program change is required to validate an instruction.

The status is such that code can be run on a programmed FPGA and instructions verified one by one. I intend to provide two zip files, one with minimum necessary to implement a FPGA and one with all my modules.

Table of Contents

Release notes.

Release	Comment	Author	Date
0	First Release	Ian C	October 23, 2010

License

This is released under the terms of the The GNU Lesser General Public License. This can be obtained from "www.gnu.org".

Legal Stuff

Two files are included in the package. Copying has the details of the GNU General Public License and Copying.Lesser provides the details on the GNU Lesser General Public License.

Details

The iss_Stuf.zip has the minimum files to program a FPGA and contains four VHDL files that define the Processor, UART-TX, UART-RX and 6502. The processor is the top level file and is essentially address decode and cobbles the rest together into a system. The necessary are ROM and RAM files that are generated automatically by the Lattice IP (tools/IP/memory/EBA/RAM or ROM. To generate these components a dot.mem file is required (see section Assembler code generation tools). These RAM/ROM files are the IP of Lattice and I am not in a position to publish them. They are simple to generate using the Lattice tool set ispLeaver and I have included the dot.mem file. The dot.mem file is the simple OS monitor previously mentioned.

VHDL Tools

I use Ubuntu Linux as my OS with XP under VirtualBox running the Lattice ispLeaver starter. The ispLeaver is used to generate the dot.jed file that is programmed into the FPGA. On the Linux side I use the GNU ghdl and gtkwave to develop and debug most of the hdl code. The XP Lattice side is documented by Lattice. The isp_Stuff.zip and FPGA software should be all that is needed to implement the 6502 into a FPGA.

Generating the dot.jed file.

I used ispLeaver 8.0.01.28.12.10 which was the latest version a few months before Lattice Diamond was released. This runs under XP and I have it under VirtualBox under Ubuntu. Diamond runs on both windows and Linux Redhat. There is a write up on the web to get Diamond running on Debian versions of Linux.

In ispLeaver Project Navigator start a new project. Copy the files Processor.vhd, 65C02.vhd, UART_TX.vhd and UART_RX.vhd into the top project directory. Right click LXCMO2280 import then pull in the files one after the other.

To generate the Lattice_ROM file (RAM too) from Project Navigator "Tools/IP express" memory/EBI/ then select 5.0 ROM or 7.1 RAM-DQ, set the size to 1024, un-check the "Enable output register box" check the "hex" box and browse to the dot.mem file then generate the RAM/ROM.vhd files. Then

import them as above. I named them Lattice_RAM and Lattice_ROM; if these names are changed the processor.vhd file needs the corresponding changes.

The next step is to assign pin numbers. This is not the best way. Generate the dot.jed file then (even if it fails) “Tools/design planner” then in design planner “file/open design” and open the top ncd file. Select spread sheet view, then Port Attributes tab and edit pin numbers to what you use. Then select the Period Frequency tab, click add preference, highlight the frequency and clock net buttons and enter 10 into the frequency field, highlight clk_pin_c and the select add. File save it and exit design planner.

This clk_pin_c stuff is important to avoid hold time violations. I did not do this on my X-ray project but Lattice support had me do that to fix my hold time problems with this design. With all the above in place a dot.jed file can be generated and dumped into a FPGA.

Linux Tools

On the Linux side I use a slightly different processor.hdl and my own design for ROM and RAM that are intended to have the same timing as the Lattice IP. A few microseconds of simulation is all that is needed to validate a new or suspect instruction however it takes a minute or so to generate 15 m seconds of UART activity. This activity can be displayed using gtkwave. This was used for detailed debug of instructions and timing.

The Lattice IP can not be run by ghdl and my ROM/RAMs are impractical to be synthesize by ispLeaver. It should be noted that ghdl is a simulator and the simulation may or may not be practical to synthesize into a FPGA or CPLD. IE a tri-stated internal nodes inside the FPGA like in a genuine 6502 can not be synthesized.

To run this set of tools make a new Linux directory and copy the 6 files ghdl_processor.vhd, 65C02.vhd, ghdl_rom.vhd, ghdl_rom.vhd UART_TX.vhd and UART_RX.vhd plus the Makefile. Then run “make” and an executable file is generated and run producing a wform.vcd file that can be examined with the command “gtkwave wform.vcd wave_setup.sav”.

Assembler code generation tools

This is how I did it on the Linux side. There may be better tools on the windows side. I use crasm to generate a listing and Motorola S code. I have also used sim65 from 6502.org to check out the assembly code before running it on the processor both ispLever and Linux side. In addition I have two pearl scripts. The sim65 software has at least one bug 9-3=5 and based on my limited math it should be 6. Other than that it's quite good at stepping through code.

- asm2rom.pl. Running “./asm2rom.pl filename” runs crasm generating filename.o, filename.lst. The script generates filename.mem from filename.o, this mem file can be used by Lattice IP to produce executable code in the ROM.
- Asm2bin.pl Running “./asm2bin.pl filename” generates filename.o, filename.lst. The script generates the filename.bin from filename.o, this bin file is coped and pasted into the ghdl_rom.vhd file.

- Please note that filename.o is the Motorola S code.

GNU software from the Ubuntu repository.

- Ghdl
- gtkwave
- crasm
- gtkterm
- pearl should be pre-installed
- make should be pre-installed
- sim65 from 6502.org

Overview

Why a 6502 over any other micro? I've used a Z80 and of course and 8080, which are good devices. The 6800 with and 8 bit accumulator and 16 bit index register to me was impractical. The 68000 is my preferred processor from the days when I was serious about that. I have looked at some of the more modern 8 bit micros and often wonder why so many? I do not like paging schemes to increase memory space as these need a software overhead or result in impossible to follow code. I guess what I know and like is the main reason.

The 8 bit microprocessor should be limited to a simple controller applications, closely coupled to hardware where speed is important. If speed is not a primary consideration a PLC should be considered. The PLCs use higher levels of language and tend to be slow. Maybe up to a few seconds to complete a cycle. The PLC is good at putting coke in a six pack and six packs into a shipping box, opening gates on parking lots etc. Sophisticated applications need to pull together gobs of existing software. Thus needing large memories and powerful processors.

Open cores already contains two 6502 projects. One in veralog and another VHDL generated from a c language program. I could not tell if these projects had advanced to running code in a FPGA or CPLD. I will give them a try once I have finalized this version. I hope to steal ideas to improve speed and reduce the number of LUTs used. In any event when I started I wanted to learn how to implement and optimize. I still need to learn about optimizing the design for both speed and size.

Status

A Lattice LCMXO2280C has been programmed with the code in the tarball. A simple monitor, OS or Kernel is implemented in the ROM (kernel.mem) and is able to communicate with my PC running gtkterm. A few basic commands can be issued from the terminal to the monitor.

Command	Response
“r FC00”	FC00 A2
“R FC00”	FC00 A2 00 9A 8E 00 02 8E 01 02 8E 02 02 8E 03 02 8E
“w 0300 00”	write 0 into 0300
“F 300 00 10”	write 0 into 0300 to 030F

“g”	goto 290
“G 323”	goto 0323

In addition a Motorola S code file usrcode.o can be loaded into the FPGA. The file usrcode is used to check the performance of an instruction and flags without the bother of re-generating the ROM flies. Just type in the suspect instruction, ./asm2rom usrcode then run gtkterm, then file/send raw file “usrcode.o” and simply g or G 290 to run the test. All flags and the registers are displayed in gtkterm.

I have all flip flops clocked from the same 10MHz clock. IspLeaver predicts no problems at 25MHz but fails at 50MHz. I need to so some work there. I'm not able to use the internal PLL as it needs a higher input frequency clock. Maybe I'll get round to fitting a 100MHz oscillator.

Odds and Ends

- When generating the Lattice_ROM file (RAM too) from Project Navigator “Tools/IPexpress” set the size to 1024, uncheck the “Enable output register box” check the “hex” box and browse to the dot.mem file.
- When starting ispLeaver the syntax checker finds some errors with typecasting inside the ROM RAM port mapping. Lattice support quote “*It looks like the parser that builds the hierarchy is using an older version of VHDL syntax, or just isn't as robust as the actual synthesis tools*”.
- I've also found that to have consistent results each time I generate a new ROM file in ispLeaver I need to “file clear all” then exit, then restart ispLeaver and generate the dot.jed file. I also add or delete a space in a comment line and re-save the processor.vhd, rom.vhd and 65c02.vhd files so as to get a new date stamp before generating the dot.jed file.
- One minor addition to gedit for Linux (and maybe others) is the file crasm.lang. I derived this from vhdl.lang by modifying it to highlight the crasm source file. It's not necessary but it may be useful, just copy it into “/usr/share/gtksourceview-2.0/language-spec/”.

Cost consideration.

The Lattice parts at $19.00/ea Mouser to $40.00/ea Newerk and it's 80% full so that may not be too promising for an application. Xilinx have some cheaper parts and maybe I'll download 3G of software and give it a try.