Assignment 3: Mano Machine Synthesis

Class	ENGI5611FD
Professor	Dr. K. Natarajan
Student	Greg Toombs
ID	0331894
Date due	2007-12-10

Table of Contents

Abstract

Verilog is a hardware description language useful in pre-fabrication stages of engineering digital electronics, and is particularly useful for modelling the behaviour of large, complex systems with significantly less investment in time and effort.

This project is a Verilog implementation of the “Mano machine”, a theoretical computer processor described by M. Morris Mano. Details on the machine's characteristics can be found in reference [1]. The Verilog code targets a Xilinx® Virtex™-II Pro FPGA.

The synthesis code has been developed in the Xilinx® ISE™, and simulated and tested in ModelSim® from Mentor Graphics®. ModelSim is a robust, full-featured simulator more suited to detailed analysis than the simulator built into the Xilinx ISE. ModelSim and the ISE are both available as free downloads on the Xilinx website.

The test programs have been written in assembly, and assembled by a version of Mark Roth's Mano machine assembler modified for this project. The programs are assembled into a core generator coefficient file containing block RAM pre-initialization values.

Hardware Synthesis

The Mano machine is synthesized in Verilog with the Xilinx® ISE™ Webpack™ 9.2.03i into a Virtex™-II Pro XC2VP2 FPGA, package FF672, speed grade -7. The machine includes a set of 25 instructions matching the format of those described in Mano's reference material. However, the implementation has been modified in several respects.

In the reference material, Mano describes memory as part of the computer system in general, but makes no specific comments as to whether the memory is on-die or off-die. In the problem parameters, it is specified that there be memory synthesized onto the die, but also that instructions be clocked in from an external source. If the problem parameters are to be followed strictly, the most reasonable way to implement external instruction control would be to create a parallel external read-only memory segment in the test fixture. The external memory would share the same instruction space as the internal memory. Advantages of this approach would include an effective doubling of the system memory, as well as the safety of program/data isolation. However, this approach increases the complexity of the system, as memory control needs to be implemented in the test fixture, and extra ports need to be added to the processor module.

Due to the fact that data memory initialization needs to be done in the internal primitive instantiation regardless of how the program memory is implemented, it is more practical to implement program memory and data memory in the same shared internal segment, so that memory initialization accommodates both programs and data. This simplifies the design, as well as better adhering to Mano's original specifications. The instructions can still be monitored from the test fixture by adding a hierarchical reference to the data bus register in the wave window. There is no need for handshaking, and the only ports needed in the processor module are the input clock and the character bus lines.

There is a full 4096x16 memory segment embedded in the processor that contains both program and data memory. The memory is implemented with the Xilinx Core Generator and initialized with a COE file. A Mano machine assembler has been included in the project tree. It was originally written by Mark Roth and released under an open-source license, but has been modified for the purposes of this project. It parses simplistic Mano assembly files and generates the COE file.

The only primitives instantiated in the module are the RAM blocks used by the generated core. All other primitives are instantiated in the generated post-place-and-route simulation model.

The time required for a given instruction varies based on the instruction. The implemented state machine is very different from the canonical implementation of the Mano machine. To make the most use of the ISE state machine optimization, every datum defining the state has been integrated into the state counter. To this end, the *I*, *R*, *IR*, and *TR* registers have been obviated, and the *SC* register has been expanded to 28 states. The state counter includes all information about the current instruction, indirect operation mode, interrupt operation mode, and fetch operations. The ISE re-encodes the state counter as a 28-bit one-hot register for speed optimization.

For performance, the state machine has been implemented such that there will always be a memory operation underway. This is why the memory core's enable flag has not been generated.

Another variation from Mano's specifications is the explicit inclusion of external *fgiset* and *fgoset* input lines for character devices to assert the input and output flags. Without these lines there would be no way for character devices to set the interrupt flag latches.

It should be noted that Mano does not specify a program start vector. It is assumed that the start vector is 000. Care must be taken to jump from address zero over the interrupt space (001) to the main program.

Of interest are the reports from the Xilinx ISE that display the use of Virtex device resources after synthesis:

```
Logic Utilization:
  Number of Slice Flip Flops:          85 out of   2,816    3%
  Number of 4 input LUTs:             569 out of   2,816   20%
Logic Distribution:
  Number of occupied Slices:          346 out of   1,408   24%
Total Number of 4 input LUTs:            628 out of   2,816   22%
  Number used as logic:               569
  Number used as a route-thru:         52
  Number used as Shift registers:       7

  Number of bonded IOBs:               21 out of     204   10%
    IOB Flip Flops:                     8
  Number of PPC405s:                   0 out of       0    0%
  Number of Block RAMs:                 4 out of      12   33%
  Number of GCLKs:                      1 out of      16    6%
  Number of GTs:                        0 out of       4    0%
  Number of GT10s:                      0 out of       0    0%

Total equivalent gate count for design:  267,185
Additional JTAG gate count for IOBs:  1,008

[...]

Device Utilization Summary:

   Number of BUFGMUXs                       1 out of 16      6%
   Number of External IOBs                 21 out of 204    10%
      Number of LOCed IOBs                  0 out of 21      0%

   Number of RAMB16s                        4 out of 12     33%
   Number of SLICEs                       346 out of 1408   24%
```

Simulation

Simulation is performed with ModelSim® XE III 6.2g. Three different test fixtures have been written. The first test fixture, *sim.v*, is for behavioural modelling. *sim_ppar.v* and *sim_speed.v* are for post-place-and-route modelling.

All simulation modes use custom TCL scripts in additional to the automatic scripts to direct ModelSim to correctly compile modules, reference libraries, and display wave signals and memory.

The *sim*.v and *sim_ppar.v* test fixtures utilize a clock period of 10 ns. This is to promote simple and "safe" simulation. The synthesized machine is capable of running faster, but in simulation the behaviour of the processor is more important than its speed.

The *sim_speed.v* test fixture is a special implementation that gradually increases the frequency to see how long a UUT can maintain expected output before the approximate maximal frequency is attained.

Depending on the program's instruction usage, the effective execution rate will vary, as some instructions require more state machine transitions than others. Due to the modified state machine, there are typically more computations per clock cycle than in the canonical implementation, requiring more logic and increasing the synthesis timing delays. Despite these delays, the processor should still execute faster than one implemented canonically when the maximal frequency is used.

Sample Programs

The simulated program is provided as an assembly file to the Mano assembler written by Mark Roth. The assembler is free and open-source, and has been used with express permission. The core generator loads the .coe produced by the assembler when the core is regenerated. The Verilog test fixtures (for the most part) does not need to be modified on a per-program basis and simply runs the clock indefinitely. No handshaking is needed.

This generalized program model allows for an easy, modular way of switching between the different sample programs provided.

Booth Multiplier (booth.asm)

This program first accepts two seven-bit signed values from the character bus, as if a user were entering them from a modem or keyboard. It then multiplies them with Booth's algorithm, outputs the product on the character bus, and halts.

Fibonacci Sequence Generator (fibb.asm)

This program generates the recursive Fibonacci sequence and outputs on the character bus all values lesser than 2^{16}, then halts.

"Speed test" Fibonacci Sequence Generator (speed.asm)

This program is similar to the previous program, but stores generated values to memory instead of sending them to the character bus. It tests the indirect instructions more thoroughly.

Procedure

To generate all files and inspect simulation results, follow these steps.

1. Verify that the Xilinx ISE, the Xilinx ISE IP Update, the Xilinx ISE Service Pack, ModelSim III XE, and TortoiseSVN (or another Subversion client) are properly installed. These applications are all free for download.
2. Visit the project site, http://code.google.com/p/manomachine/. Perform an anonymous check out of the repository to a local directory. Google Code contains tutorials on the site for Subversion operations.
3. Open an operating system command window in /synth/programs. Select the program to be synthesized. Invoke the COE generation batch file with the assembly file as an argument:

   ```
   assemble myprogram.asm
   ```
4. Open /synth/mano.ise in the Xilinx ISE.
5. Open sim_ppar.v. If running the Booth multiplier example, set the *par_input* parameter to 1; otherwise, set it to 0.
6. Select "Sources For: Post-Route Simulation". Select the *sim_ppar* module. Invoke "Simulation Post-Place & Route Model".

Current Issues

- When generating the post-place-and-route simulation model, many of the internal wires and registers are dropped or replaced with optimized logic that is named differently. As a result, it is sometimes difficult to determine behaviour of certain internal processing aspects.
- There is a bug (that I reported) with the ModelSim TCL generator when mem.do is saved from the memory pane format. The TCL generator should save primitive array names with escaped '[' and ']' characters, but does not, resulting in broken .do files that have to be edited manually. A workaround is to disable the ISE from escaping characters in element names, replacing them with underscores instead.
- There is a bug in ModelSim where *`define* fails to substitute macros for the names of instantiated UUT models.
- The post-place-and-route simulation model does not allow ModelSim to monitor internal memory contents unless character escaping is disabled in the model generation options. This is another ModelSim bug.
- Trying to invoke ModelSim from the ISE Behavioural Simulation section does not start ModelSim and fails silently.
- In ModelSim, if there is an error in a sourced (included) TCL file, execution is cancelled even if *onerror {resume}* is specified.
- In the ISE, separate settings cannot be made for separate test fixtures in the simulation source lists (for example, setting the use of custom .do files).
- The processor could have been more optimized, but the time constraints of the project prevented this from being accomplished.

Conclusions

Unfortunately, the Mano machine lacks many important features that virtually all processors provide. Due to its limited instruction set, it is missing valuable instructions such as the loading of an immediate constant into the accumulator. It has no multiplication, division or subtraction instructions, and is further limited in its capabilities by the restriction that all operands exist within the instruction word. It has an address space insufficient for any useful amount of dynamic memory. It would be trivial to expand the instruction set by replacing the current one-hot nature of the register-reference and I/O instructions with normal 12-bit values, increasing the maximum register-reference and I/O instruction count from 24 to a possible 2^{13}.

Albeit these disadvantages, HDL synthesis of the Mano machine still presents an interesting and educational case for the practical design and implementation of processors. It illustrates the process of implementing register transfer logic into a state machine, managing memory, registers, interrupts, input and output, and designing an ALU.

References

[1]: Mano, M. Morris. (1993). "Computer System Architecture", 3rd ed. Prentice-Hall, Inc..