FPGA Integrated Co-Design

Richard E. Haskell and Darrin M. Hanna
Computer Science and Engineering Department
Oakland University
Rochester, MI 48309
haskell@oakland.edu

Abstract

The main problem in hardware/software co-design is how to design an embedded system that contains both hardware in the form of FPGAs or ASICs and a microprocessor for which software must be written. A critical decision that has a profound effect on overall system cost is how to partition the system into its hardware and software components. A mistake made in this decision, which must be corrected by reworking the entire design, can add significant delay and cost to the design process. The longer the irrevocable decision of how to partition the hardware and software can be delayed, the better is the chance to keep overall system cost to a minimum. This paper describes an approach that has been tested in a graduate course on FPGA design that will allow the hardware/software partition decision to be delayed to the very end of the design process.

1. Introduction

A typical embedded system contains a micro-processor and possibly an FPGA. Microcontrollers such as the Motorola 68HC12 contain integrated modules for implementing parallel and serial I/O, A/D conversion, and various timing functions [1]. While these functions are convenient to have available, they may not provide all the hardware functions needed for a particular design. Such a microcontroller will certainly include some functions that are not actually used in a particular design.

As the size of an FPGA (in terms of the number of equivalent gates) has increased while its cost has decreased, it is becoming feasible to consider putting all functions, including a microprocessor core, into the same FPGA forming a true System-on-a-Chip (SOC). The software running on the microprocessor core would also be stored in the form of instructions in the same FPGA.

VHDL is used to design all the hardware, including the microprocessor core, that is synthesized to the FPGA. The microprocessor core is a stack-based computer that will efficiently execute Forth code. A program written in Forth can be translated to a VHDL program that will synthesize only the hardware necessary to implement the particular Forth program. Because the Forth software program ends up as just more VHDL code that can be simulated and synthesized with the rest of the hardware design, the boundary between hardware and software has become almost entirely obliterated. This has the advantage of delaying (or avoiding) the hardware/software partition decision. This is possible because the same person is designing both the hardware and software as a unified whole. Changes can be made at any point in the design process as simulations and synthesis tests provide information about speed and area tradeoffs.

2. An Elastic Microprocessor Core

Students in a junior-level course at Oakland University have designed an 8-bit microcontroller, called W8X, using VHDL and implemented it in a XILINX FPGA [2]. This microcontroller contains a 4-element register array data stack in which all Forth stack manipulation words can be executed in a single clock cycle. It also contains a 16-word return stack, an ALU, a program counter, a program ROM, an instruction register, and a controller. RAM and I/O registers can be added to this basic design.

This microprocessor core has been enhanced in a graduate course in FPGA design. The WnZ microprocessor core can have an arbitrary bus width. The ALU has been replaced with a Function Unit that implements all Forth arithmetic, logical, relational, and shifting operations. However, for a given Forth program only those operations that are actually used will be synthesized from the VHDL code. This will reduce the size of both the Function Unit and the controller that implements the instructions.

Table 1 shows the Forth words that are implemented in the WnZ. Each of the instructions in this table, except for UM* and UM/MOD are implemented in a single clock cycle. For the W8Z the instruction UM* will

multiply two 8-bit unsigned numbers and produce a 16-bit result in 10 clock cycles using the single-clock-cycle instruction *mpp* (multiply partial product). The instruction UM/MOD will divide a 16-bit unsigned number by an 8-bit unsigned number and produce an 8-bit quotient and an 8-bit remainder in 10 clock cycles using the single-clock-cycle instruction *shldc* (shift left division conditional).

Table 1 Forth words implemented in the W8Z

-	2*	INVERT	SWAP
+	2/	LIT	TRUE
<	2DROP	LSHIFT	TUCK
<=	AGAIN	NEXT	U<
<>	AND	NIP	U<=
=	C!	OR	U>
>	C@	OVER	U>=
>=	DROP	R@	U2/
>R	DUP	R>	UM*
0<	ELSE	R>DROP	UM/MOD
0=	FALSE	REPEAT	WHILE
1-	FOR	ROT	XOR
1+	IF	RSHIFT	

Special purpose I/O registers can be implemented to meet specific needs. For example, the top of the data stack has been implemented as an SPI register that allows the microcontroller to communicate with devices with a SPI serial interface [2].

3. FPGA Co-Design Example

As a simple co-design example consider the problem of making the eight LEDs on the Digilent Digilab XL board [4] cycle through the 16 states shown in Figure 1 with about a half-second delay between states. With the elastic microprocessor core in place the Forth program shown in Figure 2 will produce this pattern. The synthesized design used 212 CLBs, about 53% of the CLBs in a Xilinx 4010E FPGA.

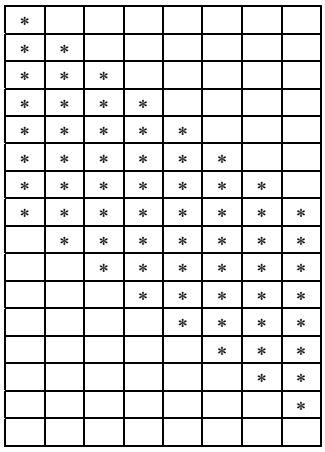

Figure 1 LED Display Sequence

```
HEX
: MAIN ( -- )
  BEGIN
     80
     8 FOR
       DUP LD! DELAY 2/
     NEXT
     8 FOR
       U2/ DUP LD! DELAY
     NEXT
     DROP
   AGAIN ;

: DELAY  ( -- )
   F0 FOR NEXT ;
```

Figure 2 Forth program for LED display

After writing and testing the program in Figure 2 a pure hardware solution to the problem was implemented using a single Johnson counter. This solution took only 6 CLBs or 1% of the CLBs in the XC4010E FPGA.

The point of this simple example is that the same person implemented both solutions in a matter of minutes, not hours or days, and the same VHDL simulation tools were used to debug both the "software" and "hardware" solutions.

4. Conclusions

Students in a graduate course on FPGA design at Oakland University have designed a microprocessor core that will execute most Forth instructions in a single clock cycle. Forth programs written for this microprocessor core are translated to VHDL code that will synthesize only those instructions required for a particular program. Students are able to design specific hardware components needed for a particular design and then define new instructions to interface the microprocessor core with the new hardware component. This flexibility defines a new hardware/software co-design paradigm.

References

[1] Haskell, R. E., *Design of Embedded Systems Using 68HC12/11 Microcontrollers*, Prentice Hall, Upper Saddle River, NJ, 2000.

[2] Haskell, R. E., and D. M. Hanna, "Implementing a Forth Engine Microcontroller on a Xilinx FPGA," Looking Forward – The IEEE Computer Society's Student Newsletter (A Supplement to Computer), Vol. 8, No. 1, Spring 2000.

[3] Clarke, Peter, "Xilinx Launches Support for Internet Reconfigurable Logic," EE Times, May 21, 1999.

[4] http://www.digilent.cc/, Visited 3/8/20001.