Synthesis vs. Simulation: Developing a Hardware Interrupt System for the Instructional Processor

Ronald Hayne
The Citadel

Abstract

The Instructional Processor has been developed as a digital system design example. The architecture is modelled in VHDL and can be simulated and synthesized to an FPGA using Xilinx design tools. The goal of this project was to add a hardware interrupt system to the enhanced microcontroller. The interrupt system includes a hardware timer and a serial UART. The design process highlighted several important differences between what can be demonstrated via simulation and what can be synthesized to hardware. The FPGA microcontroller was tested using a time-multiplexed display and a serial RFID card reader. The expanded design example has been added to a graduate computer architecture course, which uses VHDL and FPGAs. The project continues to achieve its goal as a valuable instructional tool.

Keywords

VHDL, FPGA, UART, RFID

Introduction

Teaching digital design involves the use of many examples including counters, registers, arithmetic logic units, and memory. The design of a computer processor combines these components into an integrated digital system. The Instructional Processor has been developed as a design example in an advanced digital systems course at The Citadel[1,2]. The simple architecture provides sufficient complexity to demonstrate fundamental programming concepts. The entire system is modeled in VHDL (VHSIC Hardware Description Language) and can be simulated to demonstrate operation of the processor. Memory-mapped I/O (input/output) provides the external interfaces necessary to demonstrate example microcontroller applications, when synthesized to an FPGA (Field Programmable Gate Array).

Several VHDL models of processors exist, but some rely only on simulation to verify their operation[3]. Others include FPGA prototyping to create hardware systems[4-6], though some of these are proprietary with limited visibility of internal functions[6]. Several of these systems also include external interfacing such as UARTs (Universal Asynchronous Receiver Transmitter)[5,6]. Another limitation is that none of these systems include discussion or implementation of interrupts, which are included in most commercial processors.

A hardware interrupt is a signal sent by a device requesting attention of the processor. The current program is temporarily suspended to service the request. Responding to interrupts instead of polling status flags allows the processor to multi-task between several external devices. The goal of this project was to add a hardware interrupt system to the Instructional Processor, including an internal timer and a serial UART. The expanded design example provides an in-depth look into the implementation of more advanced capabilities. Other important teaching points, highlighted by the design process, are the differences between what can be demonstrated via simulation and what can be synthesized to actual hardware.

Interrupt System

The first challenge was how to add an interrupt system to the Instructional Processor without changing the existing instruction set architecture. The goal was to make the design modular so that sub-systems could be added as necessary to provide expanded capabilities. Maintaining a common core architecture meant that tools like the IP Assembler[1] could be used as-is without pushing changes back through the design.

The first sub-system developed was a hardware timer capable of generating interrupt signals. The registers and memory interface for Timer0 are shown in the center right of Figure 1. The timer is controlled via the T0CON register which is memory-mapped to address MEM[0x00A]. The timer can be enabled and disabled with the TMRON bit and the timing duration can be set via a 3-bit SCALE parameter. The last bit is used to read and write the interrupt flag (T0IF).

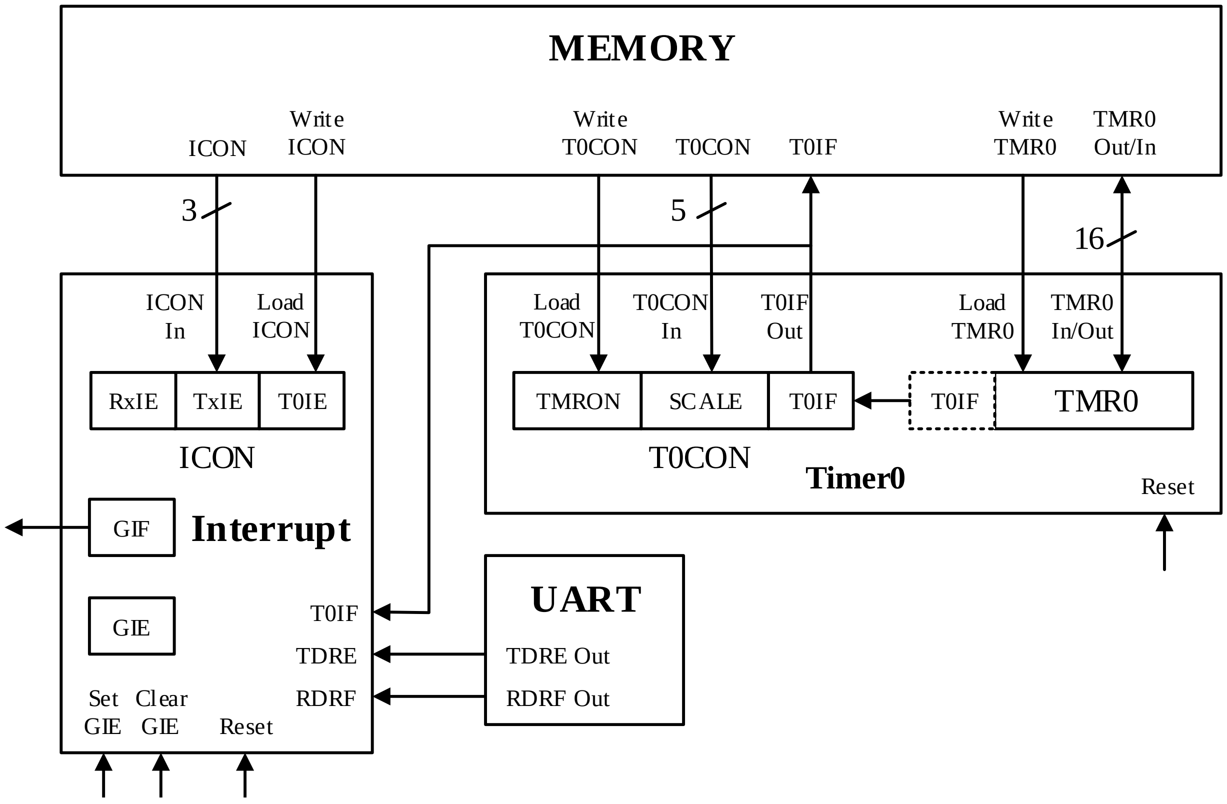


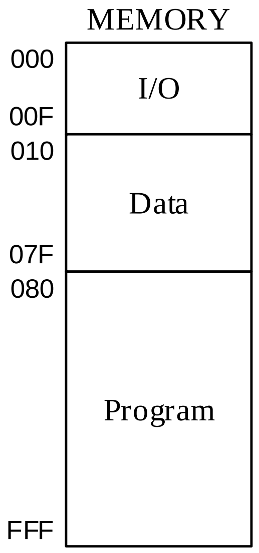
Figure 1. Interrupt System with Timer0, UART, and Memory Interface

Timer0 is a 16-bit counter register (TMR0) which is also memory-mapped. The register can be read and written to determine the timing interval. When TMR0 reaches maximum, it resets back to zero and sets the T0IF. When implemented on a BASYS3 FPGA Board, the system clock is 100 MHz[7]. Combined with the SCALE parameter, the interrupt time can be varied from milliseconds to seconds.

The second sub-system is a UART that was previously adapted to the Instructional Processor[8]. The UART enables serial communication with the FPGA microcontroller at baud rates configured from 300 to 38,400 bits per second. The system is monitored via two status flags, RDRF (receiver data register full) and TDRE (transmit data register empty). These flags can now be integrated into the new interrupt system.

The interface for the new interrupt system is shown in the lower left of Figure 1. The ICON (interrupt control) register is used to enable the three possible interrupts from the receiver (RxIE), transmitter (TxIE), and timer (T0IE). There is also a global interrupt enable (GIE), which is cleared during an interrupt and set again by the return from subroutine (RTN) instruction. The individual enable signals are combined with the interrupt flags to trigger the global interrupt flag (GIF), which alerts the processor that there is a valid interrupt request.

The next challenge was how to implement an interrupt vector within the constraints of the existing assembler. The memory map for the Instructional Processor, on the left of Figure 2, shows designated sectors for I/O, Data, and Programs. The sample assembly language code on the right shows how assembler directives (.define, .data, and .program) are used to map programs and data to the appropriate memory locations. The interrupt vector (INTV), pointing to the interrupt service routine (ISR), has been fixed at MEM[0x081] and the MAIN program is accessed via the unconditional branch (BRA) at the START of program memory.



```
.define  LED    [1]      ;LEDs
.define  ICON   [9]      ;Interrupt Control
.define  T0CON  [0xA]    ;Timer0 Control

.data                    ;0x010
INTV     0x81            ;Address of ISR

.program                 ;0x080
START:   BRA    MAIN     ;Goto Main
ISR:     ADD    1, R2    ;Inc LEDs
         MOVE   R2, LED


MAIN:    MOVE   1, R0    ;T0IE
         MOVE   R0, ICON
```

Figure 2. Memory Map and Assembly Language Program

VHDL Simulation and Synthesis

The new sub-systems were next modelled in VHDL and integrated into the existing Instructional Processor. Xilinx[9] design tools were used to simulate the models for functional verification before they were synthesized to FPGA hardware.

When designing Timer0 there were several options to be considered. The timer needs to be able to respond to both the system clock to load the registers and a separate timing clock determined by the SCALE parameter. Taking a software-minded approach implies simply making the timer register conditional on both clocks, which certain simulation tools will allow and seem to verify the desired behavior. However, an important distinction is that VHDL is a hardware description language and not all simulated behavior can be synthesized to hardware.

The target FPGA on the BASYS3[7] does not support multiple clock drivers and the Timer0 system needed to be redesigned. This iterative design process is an important teaching point for the students and emphasizes the need to understand the CAD (computer aided design) tools and hardware system constraints. An updated model was developed using an enable (EN) driven by the SCALE parameter and a separate counter (CTR). A small sample of the VHDL model, which was successfully simulated and synthesized, is show in Figure 3.

```
if rising_edge(CLK) then
  if RESET = '1' then
    CTR <= "000000000";
  elsif TMRON = '1' and T0IF = '0' then -- Disable counter on interrupt
    CTR <= CTR + 1;
  end if;
  if RESET = '1' then
    TMR0 <= "00000000000000000";
    SCALE <= "000";
    TMRON <= '0';
  elsif Load_TMR0 = '1' then           -- Load Timer0
    TMR0(15 downto 0) <= TMR0_In;
  elsif Load_T0CON = '1' then          -- Timer0 Control
    TMR0(16) <= T0CON_In(0);           -- Interrupt flag
    SCALE <= T0CON_In(3 downto 1);     -- Prescale
    TMRON <= T0CON_In(4);
  elsif EN = '1' then                  -- Enable: increment Timer0
    TMR0 <= TMR0 + 1;
  end if;
end if;
```

Figure 3. VHDL Model for Timer0

Another example of the differences between simulation and synthesis is highlighted in the modification of the instruction fetch cycle. If a valid interrupt request is indicated by the global interrupt flag, then the processor fetches the instruction indicated by the interrupt vector, rather than the current program counter. Tracking the separate fetch sequences required use of an interrupt in-progress flag during the three clock steps. Incorrect modelling of this flag using variables vs. signals or combinational vs. sequential processes produced simulation results that did not exhibit correct timing behavior in synthesized hardware.

Multi-Tasking Example

After independent testing of the interrupt sub-systems, an example microcontroller application was chosen to test the hardware interfaces and demonstrate the new multi-tasking capabilities. A Parallax RFID (Radio Frequency IDentification) Card Reader[10] senses passive transponder tags and transmits serial data to the UART. The ASCII tag ID can be stored in memory and then displayed on the four seven-segment displays on the BASYS3 board. The displays are tied together into one common anode circuit node, but the LED cathodes provide four separate enables. The displays must be time-multiplex scanned at a frequency above 60 Hz[7]. The hardware set-up is shown in Figure 4. Interconnects are power (red), ground (black), enable (green), and serial data (yellow).

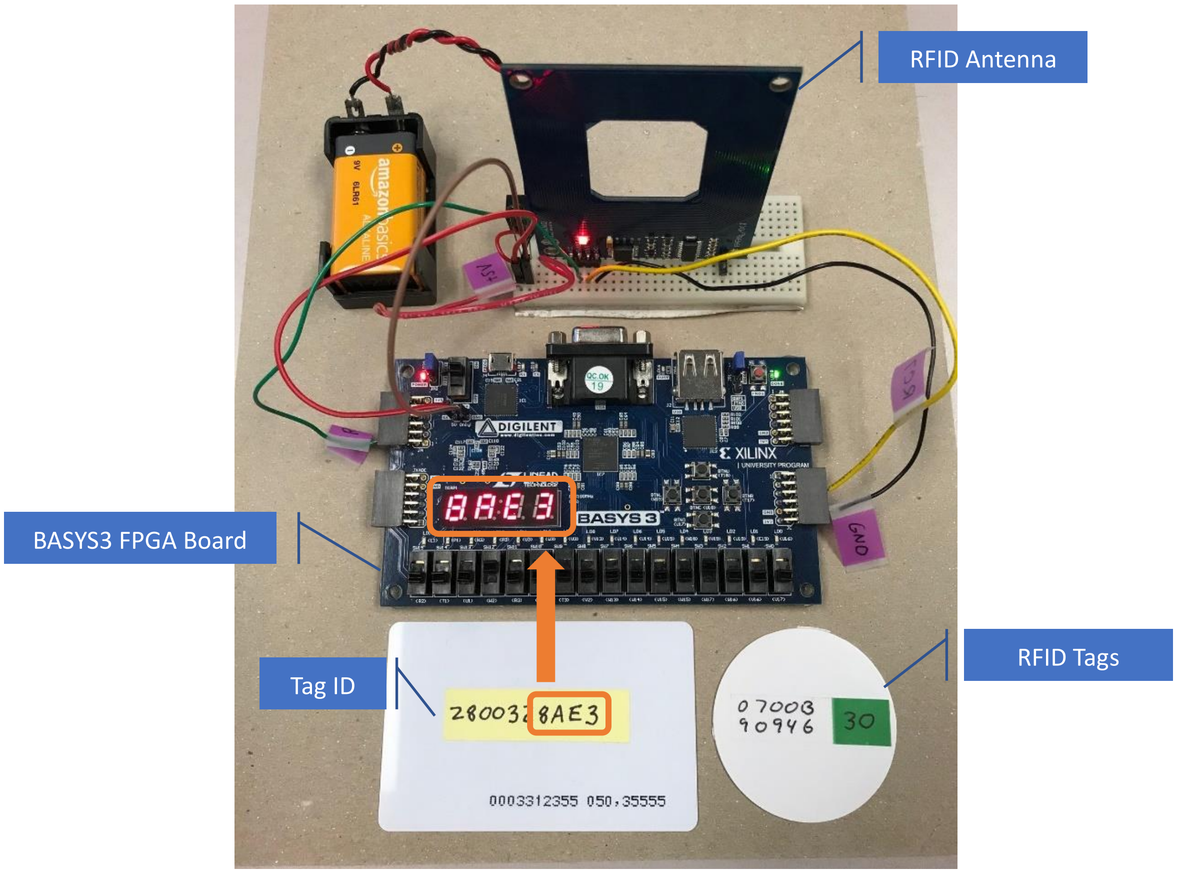


Figure 4. RFID Card Reader and BASYS3 FPGA Board

The application example requires students to develop interrupt service routines for the UART and Timer0. The UART will generate an interrupt each time a byte of serial data is received from the RFID antenna. Timer0 must be configured to generate interrupts for the time-multiplex scanning of the seven-segment displays. The systems operate independently and require careful deconfliction of resources, since one can't predict where in a program an interrupt will occur. This hardware example provides hands-on experience that can't be replicated via simulation alone.

Results and Conclusions

This project successfully added a hardware interrupt system to the Instructional Processor. Timer0 was developed to provide periodic interrupt signals, which can be used for applications such as time-multiplex scanning of displays. The previously developed UART was adapted to the new interrupt sub-system, providing asynchronous serial communication with external devices. The interrupt system was developed with independent enables to control three different interrupt sources. The interrupt vector was added to the existing memory map without modification of the instruction set architecture.

The iterative design process allows students to verify functional behavior via VHDL simulation with the additional teaching points of the differences from hardware synthesis. A key insight is the need to understand these CAD tools and constraints of the target hardware system. A digital system engineer needs to be able to go beyond classroom models and gain experience with implementation on hardware platforms like FPGAs. Students surveys acknowledge benefits from this expanded design experience.

An RFID microcontroller application was chosen to demonstrate the multi-tasking capabilities of the new interrupt system. Students experience additional design challenges in the development of multiple interrupt service routines. Simulation alone is not sufficient to provide insights into complexities such as deconfliction of resources for independent hardware systems.

The design and testing of the interrupt system for the Instructional Processor has been added to a new graduate computer architecture course at The Citadel. The course uses VHDL modelling and simulation for the design of a simple computer processor with new capabilities. Synthesis to FPGA hardware provides hands-on experience with a functional microcontroller that can be interfaced with external devices. The end result of this project is an expanded processor design example that continues to achieve its goal as a valuable instructional tool.

References

1 Hayne, Ronald and John Moore, Jr., "Evolution of the Instructional Processor," *Computers in Education Journal, ASEE*, Vol. 6 No. 4, October - December 2015.
2 Hayne, Ronald, "Design of an Instructional Processor," Supplement to: Roth, Charles and Lizy John, *Digital Systems Design Using VHDL*, Third Edition, Cengage Learning, Boston, MA, 2018. [Online]. Available: http://academic.cengage.com/resource_uploads/downloads/1305635140_559956.pdf.
3 Hwang, Enoch, *Digital Logic and Microprocessor Design with VHDL*, Thompson, Toronto, Canada, 2006.
4 Harris, David and Sarah Harris, *Digital Design and Computer Architecture*, Morgan Kaufmann, San Francisco, CA, 2007.
5 Unsalan, Cem and Bora Tar, *Digital System Design with FPGA*, McGraw Hill, New York, NY, 2017.
6 Chu, Pong, *FPGA Prototyping by VHDL Examples*, Wiley, Hoboken, NJ, 2017.
7 BASYS 3 FPGA Board Reference Manual, Digilent, Inc., April 2016.
8 Hayne, Ronald "Implementing Serial Communication for the Instructional Processor," *Proceedings ASEE Virtual Conference*, June 2020.
9 Vivado Design Suite User Guide, UG892, v2019.1, Xilinx, Inc., 2019.
10 RFID Card Reader, Serial (#28140), v2.2, Parallax, Inc., March 2010.

Ronald Hayne

Ron Hayne is a Professor in the Department of Electrical and Computer Engineering at The Citadel. He received his B.S. in Computer Science from the United States Military Academy, his M.S. in Electrical Engineering from the University of Arizona, and his Ph.D. in Electrical Engineering from the University of Virginia. Dr. Hayne's professional areas of interest include digital systems design, computer architecture, and hardware description languages. He is a retired Army Colonel with experience in academics and Defense laboratories.