BCD to Binary Converter

List of Authors (Emma Atkinson, Kiera Woodward, Alex Rolling, Brandon Furdock)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: eatkinson@oakland.edu, kbwoodward@oakland.edu, arrolling@oakland.edu, bdfurdo2@oakland.edu

***Abstract*— A binary-coded decimal (BCD) to binary converter was designed and implemented using VHDL and an FPGA. Architecture consisted of control and datapath circuits, reproducing the reversed Double Dabble algorithm in hardware. In addition, the scalable nature of our architecture would allow modification of the converter to accommodate an increased BCD input size without a change in design philosophy.**

I. Introduction

This report will cover the algorithmic expressions, VHDL language implementation, simulation results and FPGA results of a 3 digit BCD to binary conversion architecture. The inputs will be controlled by switches on the FPGA board and the output will be displayed on built-in LEDs.

BCD is an efficient way of encoding binary numbers. Every four bits of a BCD number represents a single-digit binary number that holds the place of the decimal number it represents. The number is interpreted from left to right, with the hundreds position represented by the most-significant nibble and the ones position represented by the least-significant nibble.

BCD is commonly used as address numbers for computer memory. BCD format is also common in electronic systems where numeric digits are displayed, as well as in systems where the rounding and conversion errors introduced by binary floating point representation and arithmetic are undesirable.

II. Methodology

A. Design Overview

The project uses the reverse double-dabble algorithm to convert a number from BCD to binary. This algorithm involves a series of shifts and checks. The 12-bit input BCD is initially shifted one position to the right with the least-significant bit (LSB) becoming the most-significant bit (MSB) of the output binary number. Each of the nibbles of the newly-shifted number are then checked to determine if either is greater than seven. If the nibble in question is greater than seven, three is subtracted. The updated BCD number is then shifted, with the LSB once again becoming the MSB of the binary output. This process repeats eleven times. Figure 1 demonstrates how the process would be performed manually to convert the decimal number 175 from BCD to its value in binary.

```
        0001 0111 0101          000000000000
>>      0000 1011 1010          100000000000
           >7   >7
        0000 1000 0111
>>      0000 0100 0011          110000000000
>>      0000 0010 0001          111000000000
>>      0000 0001 0000          111100000000
>>      0000 0000 1000          011110000000
                 >7
        0000 0000 0101
>>      0000 0000 0010          101111000000
>>      0000 0000 0001          010111100000
>>      0000 0000 0000          101011110000
>>      0000 0000 0000          010101111000
>>      0000 0000 0000          001010111100
>>      0000 0000 0000          000101011110
>>      0000 0000 0000          000010101111
```

Figure 1: Example of the double dabble algorithm

This algorithm was implemented using a control circuit and a datapath circuit. The control circuit was built using a finite state machine. The outputs from the control circuit were used as signals in the datapath circuit, which included twelve input switches, two shift registers, multiplexers, a counter, subtractors and a simple-circuit comparator. The result of the conversion was displayed with the built-in LEDs on the NEXYS board. The range of valid BCD values is 0000 0000 0000 - 1001 1001 1001.This design was written in VHDL.

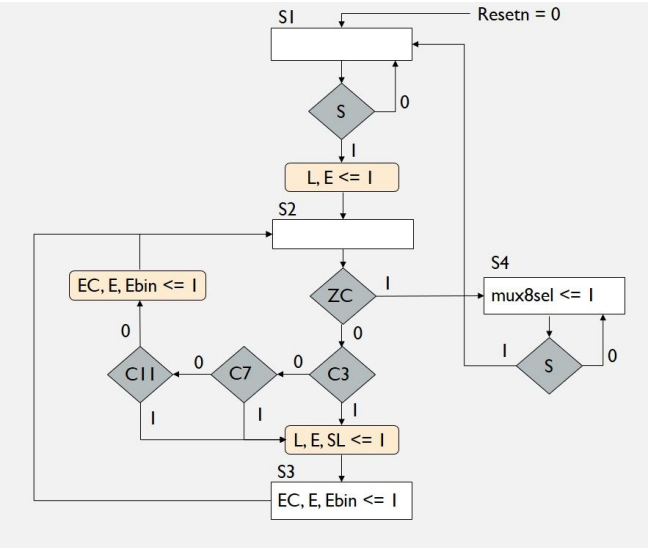

Figure 2. State machine diagram of control circuit

B. Control Circuit

Figure 2 depicts the state machine that controls the system. In State 1, the system idles until it receives notice from the input signal S, which is controlled by a switch on the board. When the switch is set to high, the load L and enable E of the BCD shift register are set to high, causing the input from the switches to load into the shift register and shift once.

Now in State 2, the output of the counter ZC is checked. If the output is low, the conversion is not complete and the MSB of each nibble is checked to determine if it is greater than seven. These outcomes are represented by the signals C3, C7, and C11. A signal of low sets enable, load, and the binary shift register enable Ebin to high to shift from the BCD register into the binary register. The counter enable EC is set to high with each shift.

When a comparator signal is high, State 3 is entered. Load and enable for the BCD shift register are set to high as is the selector for the multiplexer that controls whether the original or subtracted bits are used as the load. With this, the updated BCD number is loaded into the shift register and shifted. The Ebin is set to high after the subtraction, allowing for the LSB of the updated BCD number to shift into the MSB position of the binary number. Each time a shift occurs, the counter is enabled as EC is set to high.

Otherwise, if the output of the counter is high, the system transitions to State 4. When ZC is high, the conversion is complete and the selector to the output multiplexer is set to high, causing the converted number to be displayed with the on-board LEDs. The system then returns to State 1, idling until S transitions from low to high.

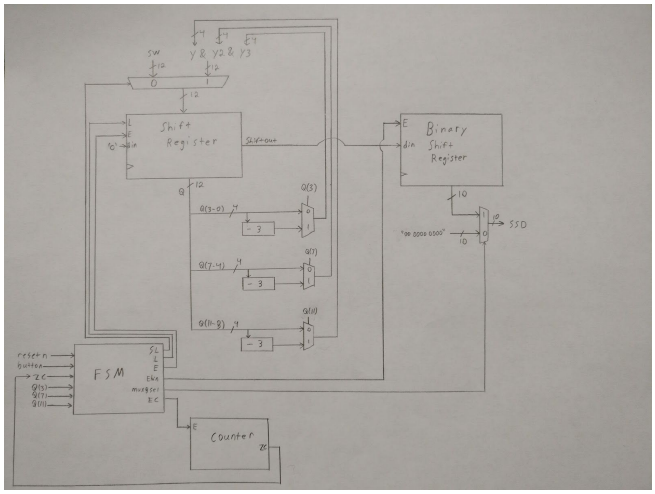

Figure 3: BCD to binary datapath schematic

C. Datapath Circuit

Figure 3 depicts the datapath circuit used in the design, which consisted of twelve switches, two shift registers, multiplexers, a counter, subtractors and a simple-circuit comparator. The input from the switches serves as one of the inputs to a multiplexer that determines what the parallel load into the BCD shift register will be. The other input to the multiplexer is the result of subtracting three from the BCD nibbles. When the selector sL is high, the altered BCD becomes the parallel load when both load L and enable E of the shift register are high.

To determine if the nibbles of the shifting BCD are less than seven, each nibble's MSB is compared: if it is one, the nibble is greater than seven. The MSBs serve as selectors to two multiplexers, each with two inputs: the unaltered nibbles and the subtracted nibbles. When the MSB is one, the subtracted value becomes the output. These nibbles get concatenated to create the adjusted BCD. This is the adjusted BCD number that becomes the parallel load to the BCD multiplexer when sL is high.

With each clock tick, one bit is shifted out of the BCD shift register and into the binary shift register. This value becomes the MSB of the converted binary number.

D. Project Revision Process

As experiments and tests were conducted on the various components, we learned more about the inner-workings of the BCD to Binary converter. Each component was debugged individually before being incorporated in the top file. After debugging, we were able to run the project simulation and check for design errors. One notable change to the design was when expanding the algorithm to work for a 12-bit BCD input. The original double dabble algorithm, which accounts for a an 8-bit BCD input, requires that the

least-significant nibble is checked for values greater than four. In our model, each of the nibbles are checked for values greater than seven. This change occurred because of the addition of the four bits that represent the hundreds place in decimal. The revision process also allowed us to identify an error in the subtractor circuit. This was corrected by revising and checking the Karnaugh maps used to create the circuit.

Revising the project inspired new ideas about controlling the system more efficiently, one such idea being to use a switch to control when the BCD input is loaded into the system. The position of the Load switch serves as the signal S in the finite state machine in Figure 1. Manually controlling when the system should begin the conversion assures that the process will begin correctly.

A simulation followed each alteration to ensure that the outputs were being computed as expected.

III. Experimental Setup

The setup we utilized first to verify the functionality of our project is a testbench in Vivado. The timing diagram produced from the Behavioral Simulation of the architecture served as a tool to guide us through the process of debugging. By analyzing each component's contribution to the outcome, we were able to identify when and where problems were occurring.

We then implemented our design on the NEXYS 4 DDR board, as seen in Figure 4, using twelve switches for the BCD input and eight LEDs for the binary output, starting with the least significant bit on the right for both the input and the output. The two leftmost switches were designated for Resetn and Load.

We verified that the conversion is performed correctly with experimental values on the test bench and then we input our own values with the implemented design on the NEXYS board. The expected result was a fully functioning BCD to binary converter.

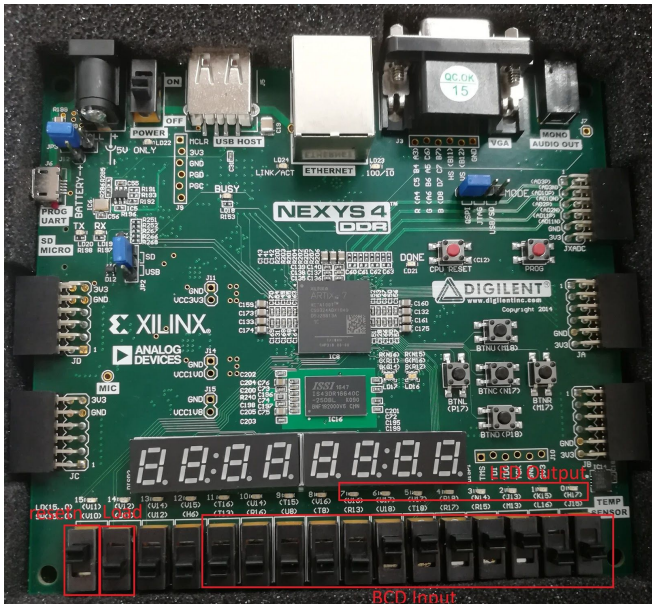

Figure 4: NEXYS 4 DDR board experimental setup

IV. Results

We were ultimately able to successfully convert a given BCD number to binary using the code we developed in VHDL. Consistently, the expected binary number was displayed on the built-in LEDs given any BCD input from 000-999, indicating that we had translated the algorithm into digital logic correctly.

Our greatest discovery from designing and implementing BCD to binary conversion architecture was the hidden complexity of a seemingly simple algorithm. One such area of unexpected complexity was in replacing the BCD input with the subtracted BCD when a given nibble is greater than seven. The initial design made the modification bit by bit but was not yielding correct results. This issue was resolved by incorporating a multiplexer with two inputs: the unaltered BCD and the altered BCD. Making this modification broadened our understanding of shift registers and how their load inputs can be manipulated to yield the desired results.

Designing architecture yields concrete results due to only using digital signals. Though the results can deviate from expected, there are never inconsistencies. This can be contrasted to developing a system that requires analog input. Analog data requires quantization, adding steps into the process that could produce errors. Working with a purely digital system avoids these issues, allowing for a deeper exploration into what can be created with digital logic.

Conclusions

A BCD to binary converter was successfully implemented on an FPGA. An input of 3 BCD digits was efficiently converted into a 10 bit binary output. In addition, the scalable nature of our architecture would allow modification of the converter to accommodate an increased BCD input size without a change in design philosophy.

Through completing this project, the group gained substantial experience in the design of digital systems, VHDL coding, and use of the NEXYS-4 board.

References

[1] Convert Binary numbers to BCD in VHDL and Verilog, nandland.com

[2] BCD to Binary Conversion on an FPGA, https://embeddedthoughts.com