BCD to Binary Converter

List of Authors (Robert D'Angelo, Colton Palmer, Joseph Jarbo)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: robertdangelo@oakland.edu, cpalmer@oakland.edu, josephjarbo@oakland.edu

***Abstract*—A binary-coded decimal (BCD) to binary converter was utilized to add two numbers input from a keyboard. This task was accomplished using VHDL and a Nexys 4 DDR FPGA. Data was input in groups of three decimal digits and was read using a ps2 interface. The two three digit numbers were appropriately converted and added together so that the result could be displayed on the 7-segment displays of the FPGA in hexadecimal.**

I. INTRODUCTION

This report will discuss the individual elements of our circuit and how they interact to accomplish the end goal. A BCD to binary converter is very useful since data from a keyboard can only feature the numbers 0 through 9. As a user is actively entering data, we need a way to store that data and then convert to binary once the user is finished entering the entire number. We cannot accomplish this by converting each decimal digit to binary and combining them together. For this reason, we convert each digit to its BCD equivalent and convert to binary using a multiplicative algorithm. In short, the motivation for this project is to make it easier to take data from a keyboard and perform operations on said data.

To construct the circuit for this project, we utilized control circuits (finite state machines) as well as a datapath circuit. These circuits consist of both combinational and sequential components. Making this circuit exposed us to the ps2 interface as well as the seven segment serializer, both being components discussed in class but not utilized until now.

II. METHODOLOGY

A. Design Overview

The most essential part of the design is the multiplicative algorithm that converts the 12 digit BCD numbers into binary numbers. This is accomplished first by splitting the number up into groups of four (nibbles). Decimal numbers 0 through 9 have a BCD value that is equivalent to their binary value (shown in Figure 1).

Decimal #	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Figure 1

However, for numbers larger than 9, the BCD equivalent is not equal to the binary value and is instead equal to the BCD value of the individual digits combined in order. Each nibble effectively represents a number in a decimal place value (ones, tens, hundreds, etc.). An example for the decimal number 999 is shown in Figure 2.

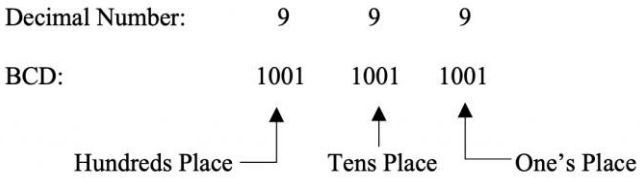

Figure 2

Fortunately, VHDL features a variety of libraries. One of these libraries (ieee.std_logic_unsigned.all) allows us to perform unsigned operations on binary numbers. Therefore, to convert from BCD to binary, we simply multiply the ones place nibble by binary number 1 (01), the tens place nibble by binary number 10 (1010), and the hundreds place nibble by binary number 100 (1100100) and add the products together. The sum we obtain is the binary equivalent of the three-digit decimal number originally entered on the keyboard [1].

B. Datapath Circuit

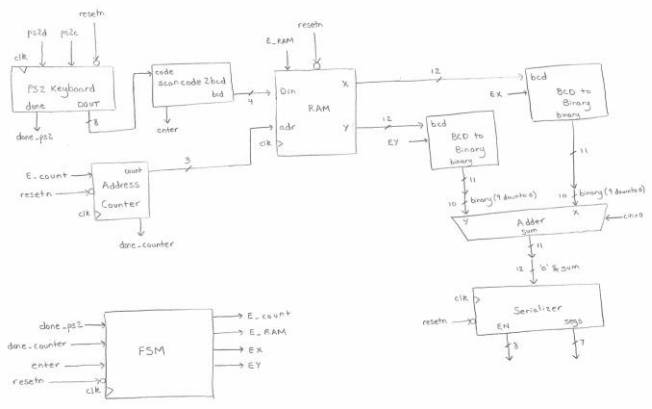

Figure 3

The datapath circuit is depicted in Figure 3 and features a ps2keyboard, a scan code to BCD decoder, an address counter, random access memory (RAM) that contains 6 registers and an address decoder, two BCD to binary converters, a ten-bit adder, and a seven segment serializer.

A standard keyboard will send a scan code for the particular character being pressed every 100 ms [2]. Once the key is released, the keyboard will send a keyup code (F0) followed by the scan code once again. The ps2keyboard detects this keyup code and outputs the following scan code. The ps2keyboard's done bit will go high when the scan code is output [3].

The scan code to BCD decoder takes the output from the ps2keyboard and converts it into the appropriate BCD value. If the scan code is for a character other than the numbers 0 through 9, the decoder will generate "0000," preventing user error from affecting our circuit.

The address counter features enable, clock, and resetn as inputs. Provided that enable and resetn are high, the counter will count from "000" to "101," incrementing by one at each clock tick. This count serves as the address for the RAM. Once the count reaches its maximum value of "101," the counter will set its done bit to 1 and reset the count.

The RAM is made up of six four bit registers and one decoder that converts the address input to the RAM into enable bits for the registers. The decoder is controlled by the enable for the RAM (controlled by the FSM). If the enable is '0,' the decoder will disable all registers to retain the data currently stored. Otherwise, it will ensure that only one of the registers is enabled at any given time. The RAM's other input, Din, will be stored in the register that is enabled at the clock tick. At this point, each decimal digit that was originally input on the keyboard has been converted to BCD and stored in its own register. Before outputting the data, the RAM splits the data in half (groups of 12 bits) and outputs it on two buses that will feed into the BCD to binary converters. The digits are in the same order as the user input them.

The BCD to binary converters take in the 12 bit BCD numbers and convert them to 11 bit binary numbers. The eleventh bit is always zero and will be removed when the data is passed to the ten-bit adder. Since BCD is being used, the highest number that can be obtained is 999 (a ten-bit binary number). However, VHDL does not know we're using BCD and believes higher numbers can be obtained (such as FFF, which will be 11 bits after the conversion). The conversion is achieved using the algorithm discussed in Part A. Each converter also has an enable controlled by the FSM.

The ten-bit adder simply takes in the outputs from the BCD to binary converters (minus the eleventh bit) and adds them together. The carry in bit will be zero in all cases. This is accomplished by utilizing ten full adders. The output will be 11 bits (binary).

The goal of the serializer is to take the binary sum from the adder and display it in hexadecimal on three seven segment displays. Before inputting the sum to the serializer, a zero is added in front of the MSB so that the sum can be equally divided into groups of four for easy conversion to hexadecimal. Since only one seven segment display can be active at once, the serializer must continuously cycle through the three displays (enabling one at a time), showing the proper number on each display. Since the cycling process is so fast, the user cannot tell that only one display is active at a time and all three appear to be on at once. The serializer features two outputs, one that is 8 bits controlling the active display, and one that is 7 bits controlling the individual segments of a particular display [3].

C. Control Circuit

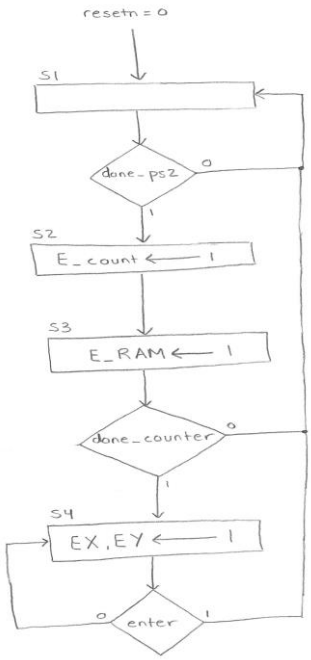

Figure 4

The block diagram (featuring the inputs and outputs) and state diagram (showing the transitions) for the control circuit

are displayed in Figures 3 & 4 above. The finite state machine features four states total (S1, S2, S3, and S4).

Pressing the resetn button will take the user back to S1, but should only be used at the very beginning of operation. In order to advance to S2, the done bit from the ps2keyboard must go high, indicating that the key stroke has been read. Otherwise, the circuit will remain in S1, waiting for a key stroke.

In S2, the address counter will be enabled so it can increment accordingly. S2 represents the point when a key has been pressed and the data is being converted in preparation for storage in memory. On the following clock tick, the circuit will advance to S3 automatically. S2 exists in order to give the circuit enough time to perform the conversion before storage.

In S3, the RAM will be enabled, allowing the current data to be stored in the appropriate register based on the address generated by the address counter. From S3, the circuit will either go back to S1 and read the next input, or advance to S4 if the done bit from the counter is high (indicating that the six digits have been read and are currently stored in memory).

S4 will enable the two BCD to binary converters that will convert each 12 bit BCD number into its binary equivalent. From this point on, the remainder of the circuit is combinational and is not affected by the state machine. To repeat the process and input new values, the user must push the enter key (taking them back to S1). Otherwise, the user will remain in S4 and the result will continue to be displayed on the seven segment displays.

III. Experimental Setup

In order to debug our project, we went through each block individually to verify functionality. This allowed us to make small changes and retest to determine the root cause of our issue. This is an effective method because it provided us with a step by step procedure to find our issues. Along the way, we were able to rule out various components as being the cause of our issue.

IV. Results

We successfully achieved our goal of adding two decimal numbers input from a keyboard and displaying the result on seven segment displays. In addition, we were able to add functionality to the enter key, allowing the user to reset the circuit and type in new values to add. In completing this project, we were able to apply a variety of topics learned in class. These topics include finite state machines, ps2 interface, serializer, decoders, and memory. In the end, we achieved the results we expected, but not without obstacles along the way. After a long debugging process, we were to determine that the decoder in our RAM contained a latch that would sometimes write unintended data to the memory. In addition, we had a misunderstanding of the ps2keyboard code that prevented us from properly obtaining our input data.

Figure 5 displays the simulation results of our BCD to binary converter, the main component of our design. The conversion of three different decimal numbers is shown. The numbers are 321, 364, and 539.

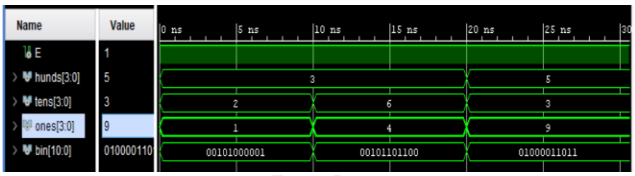

Figure 5

Conclusions

While the objective of our project sounds simple in theory, it required much more time and effort than we expected. All of our effort paid off in the end since we achieved what we set out to do. Our project is valuable since it demonstrates how data taken from a keyboard can be stored and utilized in further operations.

There a few potential improvements that could be made. One improvement would be to use sequential logic to implement the BCD to binary converter so that it takes a set amount of time to complete the conversion. Another improvement would be to allow the user to enter larger numbers via the keyboard. Both would improve the versatility of the code as a whole. Lastly, it may be useful to allow the user to toggle the output between decimal and hexadecimal depending on their needs.

References

[1] VHDL coding tips and tricks. In: VHDL code for BCD to Binary conversion. http://vhdlguru.blogspot.com/2015/04/vhdl-code-for-bcd-to-binary-conversion.html. Accessed 5 Dec 2018

[2] Nexys 4 DDR Reference Manual. In: Using the Oscilloscope [Reference.Digilentinc]. https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual. Accessed 5 Dec 2018

[3] Llamoocca, Daniel. "VHDL Coding for FPGAs." VHDL Codng for FPGAs. http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html