Keyboard Scan Code Computation

List of Authors (David Lewis and Andrew Meesseman)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: djlewis2@oakland.edu, ameesseman@oakland.edu

***Abstract*—** The goal of this project is to create a device that will interface with an external keyboard and complete certain computer arithmetic calculations. A Nexys 4 DDR FPGA was used to interface with a usb keyboard in order to obtain two, two digit, Binary Coded Decimal (BCD) values which were then added together, with the result displayed on a few of the seven segment displays included on the board. Although this project is not all that useful in a professional engineering setting, the different components of the project are all very good for learning concepts. Knowledge from the labs were very useful when designing the architecture to convert and compute using differentiating number systems. Then, figuring out how the FPGA communicates with a keyboard and multiple seven segment displays was good for putting previously learned concepts to the test. Along the way, the group developed skills around working on an engineering project in a team as opposed to as individuals.

I. Introduction

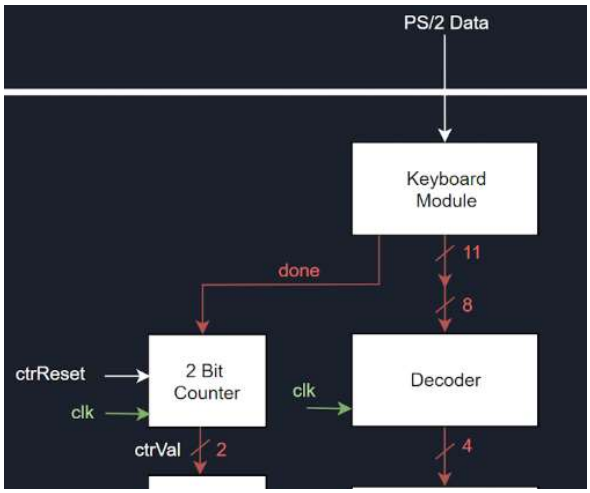
This report will cover in detail the steps involved in developing the device described above. VHDL was used to control the FPGA, and all external interfaces were controlled using VHDL in Vivado. Many logic components were used when designing the "calculator" such as registers, counters, decoders, and a finite state machine (FSM). Components were also whose sole purpose was to communicate with the initial keyboard presses and the output displays. These components were all used in order to create a few key elements. The first main element used the keyboard components, registers, and decoders to take input from the keyboard and convert it into a usable format for the calculations. The second main element used number system conversions such as BCD to Binary and Binary to BCD. These conversion are made up of many smaller logic components that do basic arithmetic. These arithmetic components were then used again to perform the desired calculation from the goal of the project. The final main element was used to take the calculation result and display it on the seven segment displays. This element made use of decoders and counter/clock processes. Lastly, a FSM was used to communicate between these elements and ensure the circuit behaved in a desired manner.

How these elements were created, simulated, and tested will be further discussed below.

II. Methodology

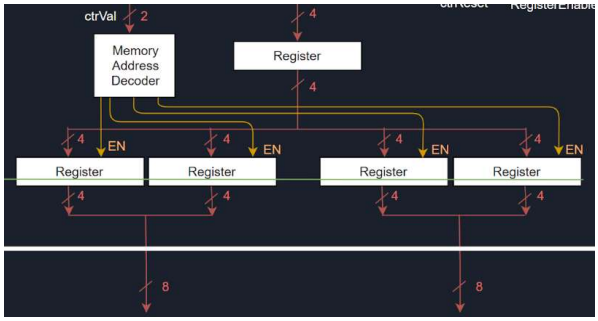
A. *Keyboard Element*

The keyboard module of the project receives a serial data input from a PS/2 keyboard. Each key press, as well as a key-up code that is sent when a key is released, consists of an 11-bit package. The package consists of a start bit of '0', an 8-bit payload (LSB first), an odd parity bit, and a stop bit of '1'. When a key is pressed and released, a key-up code package is sent which consists of a payload of 0xF0, followed by a package containing the payload for the key that is pressed. The scan-code for the numbers was found in the Nexys A7 Reference Manual [1]. In order to facilitate the data transfer, the keyboard generates 11 clock transitions when the data package is sent, with the data being valid on the falling edge. Receiving a key press from a number means that 22-bits of data will be sent, along with 22 clock transitions. In order to handle this data, a 10-bit shift register receives and shifts all incoming data. The final resulting data in the shift register is the 8-bit payload, as well as the parity bit, and stop bit. An 8-bit bus containing the payload is then sent to a decoder in order to convert the values to BCD, with the decoder created using the scan-code values found in the reference manual. When all 22-bits of data are received, a done signal is sent from the keyboard module which is sent to a counter in order to increment the count. The architecture of the previously explained can be seen below.



(FIGURE 1)

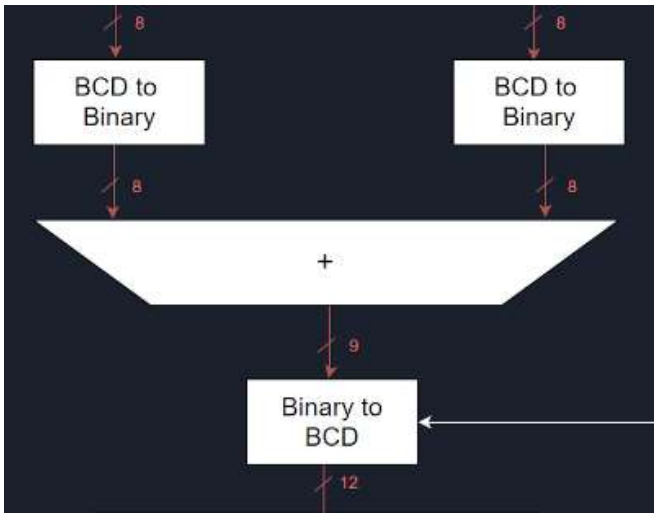
The circuit as a whole stores the values from four key presses into four 4-bit registers. As each key press is registered, the done signal increments the 2-bit counter. The 2-bit output from the counter is fed through a decoder which enables the respective register to be written to. The decoded BCD keyboard data, which is stored in a buffer, is constantly fed into the four 4-bit registers and whichever register is enabled is written to. Following four key presses, the state machine (explained in Section F) disables the registers from being written to and resets the counter. No more key presses will be read after this point until a CPU reset is enacted. As there are four registers and four 4-bit signals, the first two and last two buses are combined into two 8-bit buses. These two 8-bit signals are then sent to the computation module. The architecture of the previously explained can be seen below.



(FIGURE 2)

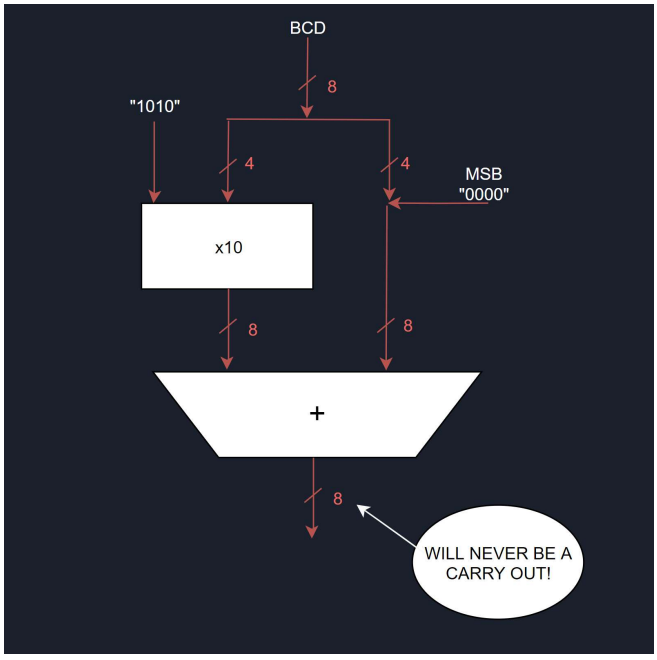
B. Computation Element

The computation element of the project was entirely done using VHDL, there was no interfacing with external devices. The architecture of this element can be seen below.



(FIGURE 3)

Each of the two 8 bit outputs from the keyboard element of the circuit are received by their own BCD to Binary conversion module. The 8 bits of data are assumed to be in BCD format and the resulting signal from this component would be inaccurate otherwise. The architecture of this component can be seen below.



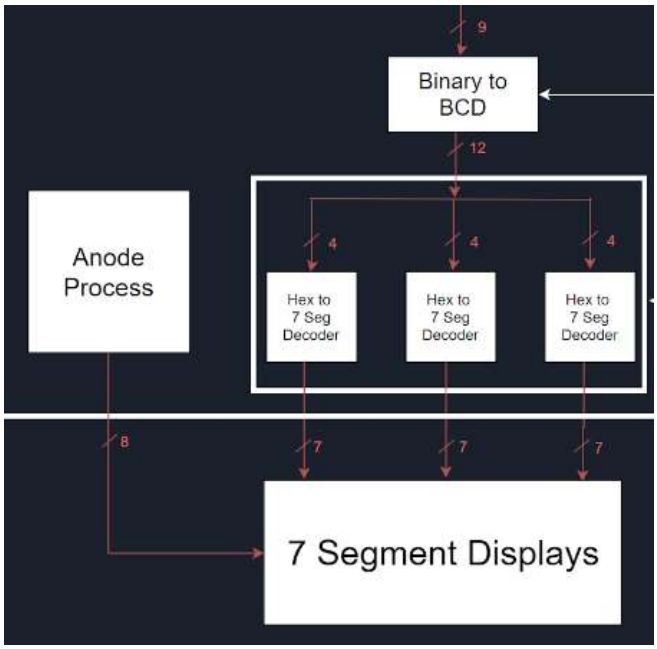
(FIGURE 4)

This module was not given to the group and needed to be designed. The easiest way to do this conversion was to split the 8 bit BCD signal into its two separate 4 bit digits, and then treated as if it was a decimal number of sort. The 4 bits that represents the tens place in a decimal number is multiplied by a constant value of 10 (“1010” in binary),

which results in an 8 bit binary number. This number is then added to the 4 bits (8 bits once alignment zeros are added) that represent the ones column. The output from the following adder will be binary representation of the original BCD number. The adder does not require the carry out bit. Due to the parameters of the arithmetic done in this component, no overflow will ever occur. Once both BCD inputs are converted, they are added together using a basic adder. The sum from this adder will require 9 bits because overflow is possible at this point in the circuit. This 9 bit value is the result of the desired calculation of the project, but needs to be converted back into the original BCD format so that it can be easily displayed. Using components provided by Professor Llamocca this was completed [2]. Professor Llamocca's binary to BCD converter uses an algorithm known as the "double dabble", and was modified to work with a 9 bit input as the original was set to work with 8. The conversion component notably requires a clock and start signal, and outputs a done signal. These will be used with the FSM to control the circuit. The resulting signal from the conversion is a 12 bit BCD value. 12 bits in BCD represents a 3 digit decimal value, which is what is required to represent a 9 bit binary value.

E. Display Element

This element of the circuit consists of a few hex to seven segment decoders, a few processes, and signal manipulation. A simplified layout of how this element works can be seen below.



(FIGURE 5)

The 12 bit number from the binary to BCD converter is first split into three 4 bit signals. Each of the 4 bit signals represents the three digits of a decimal number (hundreds, tens, and ones). Each one of these is sent through a hex to seven segment decoder. This decoder takes a hex value and converts it into a value that is useful with the seven segment display. BCD digits in this case are interchangeable with hex.

Displaying the a multiple digit number is a little bit tricky on the FPGA because each separate display cannot show separate values simultaneously. This is resolved by rapidly cycling through each display and changing the value to be displayed each cycle. The process used to accomplish this is shown below.

```
displays: process (display_counter)
begin
    if displayPOWER = '0' then
        display <= "11111000"; sevenseg <= "1111110";
    else
    case display_counter is
    when "000" => display <= "11111011"; sevenseg <= seven(20 downto 14);
    when "001" => display <= "11111101"; sevenseg <= seven(13 downto 7);
    when "010" => display <= "11111110"; sevenseg <= seven(6 downto 0);
    when "011" => display <= "11101111"; sevenseg <= kbInputs(6 downto 0);
    when "100" => display <= "11011111"; sevenseg <= kbInputs(13 downto 7);
    when "101" => display <= "10111111"; sevenseg <= kbInputs(20 downto 14);
    when "110" => display <= "01111111"; sevenseg <= kbInputs(27 downto 21);
    when others => display <= "11111111"; sevenseg <= "1111111";
    end case;
    end if;
end process;
end Behavioral;
```

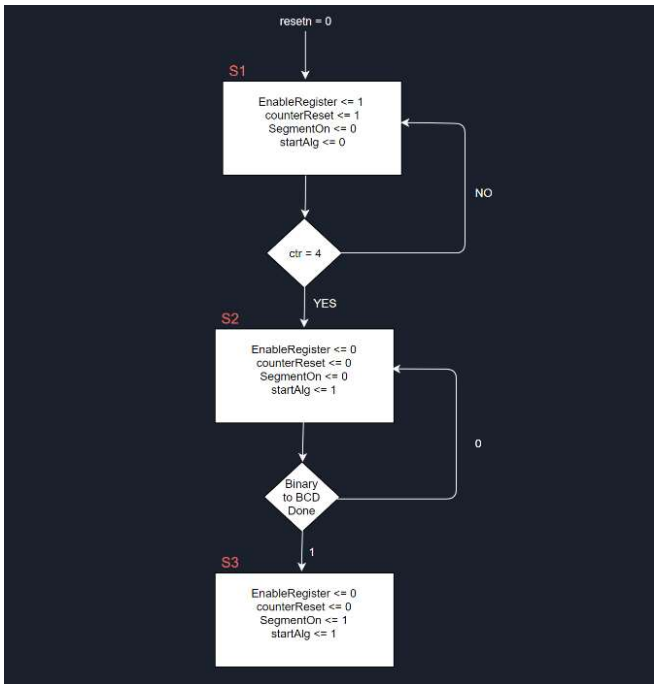
(FIGURE 6)

A counter was used to count from 0 to 6, one for each of the 7 displays used. Count 0-3 accounts for the 3 digits of the BCD result from the circuit. The other 4 cases will be discussed later in this report. A second counter was used as a clock divider for the counter used in the process above. The clock of said counter runs at the 100 MHz rate of the FPGA and counts from 0 to 250,000. Every time the counter reaches 250,000, it outputs a '1' for one clock cycle. This signal is used as the clock for the counter used in the display process. This was done because the rate of 100 MHz is far too fast of a cycle speed to view the seven segment displays. Then in each case of the process, the signals "display" and "sevenseg" are used. The display signal controls which display will be powered on each cycle. The sevenseg signal controls what that particular display will show each cycle. The values of sevenseg are mapped to the three 7 bit outputs from the hex to seven segment decoders, and were combined into one signal "seven" for organization.

F. FSM

A FSM was used to control the time at which each portion of the circuit would function. Three main states

were used. S1 occurs during the time in which keyboard inputs are being accepted by the circuit. S2 is the time in which calculations are being done by the circuit. This happens extremely quickly and does not require user input. S3 is the time at which the results of computation are completed and are shown on the displays. The transitions and outputs of the FSM can be seen below.



(FIGURE 7)

While in S1, the registers are enabled to accept the BCD values that result from keyboard input. Two signals called "SegmentOn" and "startAlg" are set to '0'. SegmentOn controls whether the seven segment displays are powered on. startAlg is mapped to the start signal in the algorithm used for converting binary to BCD. Once the counter used to determine how many keyboard inputs have been entered by the user reaches a value of 4, the FSM enters S2. In S2 the registers are disabled so that further keyboard inputs will not impact the calculations. The start signal for the binary to BCD conversion is sent, and the displays remain off. Once the algorithm is completed, a done signal is sent from the FSM used in the algorithm back to this FSM which lets the circuit know that calculations are done. This puts the FSM in S3, where SegmentOn is now '1' and mapped to displayPower seen in FIGURE 4. The displays will remain on until the reset button is pressed on the FPGA. This will clear the counter and return the FSM to S1.

III. Experimental Setup

In order to confirm the correct functioning of the project, the circuit was separated into two large components to be tested. The portion of the circuit which included the keyboard and registers was created separately and tested using simulations, and then using a keyboard and seven segment displays to confirm values were being input correctly. The other portion of the circuit including the calculations and displayed results was tested using simulations, and then tested using switches on the FPGA and the seven segment displays. Once both portions were confirmed to be working properly they were connected in VHDL in one overarching top file. This final circuit was tested again using both simulations in Vivado, and then using a keyboard and the displays.

IV. Results

For the most part, the circuit behaved in a way that was expected and desired. Getting the keyboard portion of the circuit to work in simulation proved more difficult than on the FPGA. The calculation portion of the circuit worked as expected right away. This could be due to the group spending a large amount of time conceptualizing the circuit before coding. The circuit was not completely without glitches though. Originally, the counter would not reset when the FSM went from S3 back to S1, or when the reset button was pressed before calculations were completed. For example, if 2 numbers were entered by the keyboard and then the circuit was reset, the counter would remain at 2. This meant that the next time keys were pressed, the values would be entered into the third and fourth registers immediately, and then the FSM would enter S2 before the first two registers had values. Another issue that occured is when the user accidentally pressed a key that was not a number 0-9. The keyboard decoder still assigns a value to the key given by the "others" case. A value would still display on the displays as a result, but the value would not make sense in context. This is why there are 7 cases in the display process. The final four cases are used to display the original 4 key presses that were used in calculations. Any key press that was represented by the "others" value in the decoder was set to be shown as the letter E. This was to let the user known there was an *Error* in the inputs and to disregard the result. Some other minor tweaking was done to the FSM to create a smoother experience while using the circuit, but overall the results were expected due to the extensive testing of each component during the project process.

Conclusions

In conclusion, the group has gained useful knowledge regarding VHDL, computer logic, and teamwork. Creating a project requires greater problem solving and debugging skills compared to replicating a pre-designed circuit from laboratory. Many concepts ranging from keyboard interfacing to computer arithmetic were learned and this knowledge will be useful in future endeavors. The project

results were better than originally expected, but there were still a few things that could be added to the project later on to improve quality of use.

References

[1] https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual

[2] Llamocca, Daniel. "VHDL Coding for FPGA's." Reconfigureable Computing Research Laboratory. N.p., n.d. Web. 5 Apr 2019.