**ESET 219 Digital Electronics**

Laboratory Report

Lab 4

Decoder and Encoder Design

David Ezuma

10/28/2023

All of the information contained in this report is my own work that I completed as part of this lab assignment. I have not used results or content from any unauthorized sources or fellow students.

Ogochukwu David Ezuma Date: 10/28/2023

**Introduction**

Task 1 of this lab required the creation of a full adder using a 3:8 decoder and any other necessary logic. Then, cascading the full adder created, the task required that a 4-bit ripple adder/subtractor be created, such that it can reproduce the binary results of A + B or A + (-B). Math operations were requested to be computed with the adder, on the FPGA board (switches as the inputs, and LEDs as the output). Task 2 required the use of encoders to indicate which given switch on the FPGA board is active high. Priority is assigned to the higher number switch. For example, if switches 6, 3, 2, and 1 are active, switch 6 will take priority and is indicated on the output. The switch number was indicated on a 7-segment display.

**Background**

A logic device input/output can be active HIGH or LOW. Active HIGH is when a 1 activates a given input, or output. Active LOW is when a 0 activates a given input, or output. A negation bubble denotes active LOW input or output. If no bubble is present, input or output is active HIGH.

A diagram of a circuit

Description automatically generated

Figure : Negation bubbles indicate that inputs A and D are active LOW.

Decoding is the action of taking an input combination, or code, and translating it to one or more active outputs. Only one output is active at a time. On a standard decoder, the number of outputs is 2n, where n is the number of inputs. Their typical nomenclature is <number of inputs> to <number of outputs> (Not including enable input). If the enable is not active, all outputs are inactive.

A diagram of a circuit

Description automatically generated

Figure : 2x4 decoder

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **G** | **B** | **A** | **Y0** | **Y1** | **Y2** | **Y3** |
| 1 | X | X | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |

Table : truth table for the 2x4 decoder seen in figure 2. Since the enable is active HIGH, all outputs are inactive when it is 1. The enable is also the MSB.

An encoder is the reverse of a decoder. Accepts one active input and encodes it to a multiple binary value. They identified by the same nomenclature as decoders <number inputs> x <number outputs>.

A white rectangular object with black text

Description automatically generated

Figure : layout of a 8x3 encoder.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **D0** | **D­1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7** | **x** | **y** | **z** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Table : Truth table of 8x3 encoder illustrated in figure 3 above. D7 is the MSB input, D0 is the lSB input, x is the MSB output, and z is the LSB output.

Enables tend to use priority encoding, which assigns activation priority to the MSB on truth table.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **D0** | **D­1** | **D2** | **D3** | **D4** | **D5** | **D6** | **D7** | **x** | **y** | **z** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| x | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| X | X | X | X | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| X | X | X | X | X | 1 | 0 | 0 | 1 | 0 | 1 |
| X | X | X | X | X | X | 1 | 0 | 1 | 1 | 0 |
| x | x | x | x | x | X | x | 1 | 1 | 1 | 1 |

Table : priority encoding nullifies the activation of less significant bits.

E0 (enable output), and GS (group select) are used to cascade for expanding input and output. If no input is active on the encoder, E0 is active, while if any input is active, GS is active. Most encoders are active low and follow the configuration to be cascaded as shown below.

A diagram of a circuit board

Description automatically generated

Figure : EN0 (LSB) will serve inputs 0 - 7, while EN1 (MSB) will serve inputs 8 – 15.

An adder is the logical unit of digital processors. Their ability to perform basic binary addition designates them as the building block of ALUs (Arithmetic Logic Units). A half adder can be represented as either a schematic or a truth table, its inputs being the bits to be added and its outputs being their sum and their carryout as shown in the table and figure below.

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **S** | **C** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Table A and B are the input bits, S is the sum, and C is the carryoutA drawing of a couple of pipes

Description automatically generated with medium confidence

Figure The schematic of a half adder: sum follows a xor gate logic while the carryout follows an and gate logic

The reason this is called a half adder is that it can only produce a carryout bit, void of the ability to accept carry in bits. Full adders have the capability of doing so, and subsequently cascaded to include more bits.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **Cin** | **S** | **Cout** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Table Truth table for a full adder. Cin is the input carry and Cout is the output carry.

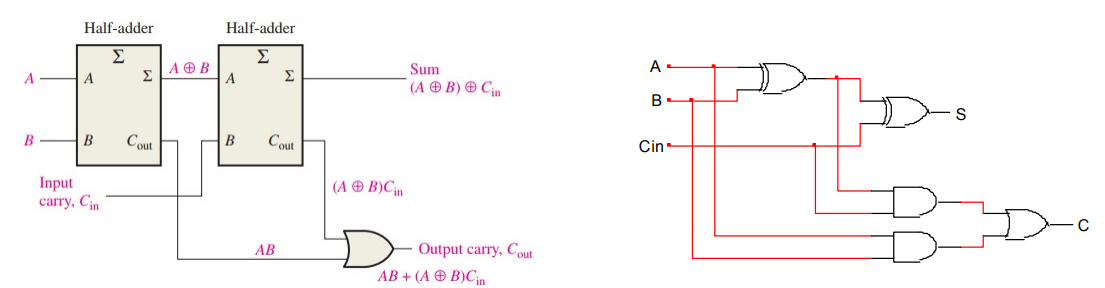
The schematic of a half adder can be produced by cascading 2 half adders together, as shown in the figure below.

Figure Full adder schematic developed by cascading two half adders. The carry input is added with the sum of the previous adder, and the carry out is any given carry output from either adder.

Given that full adders handle one sum and one carryout at a time, cascading them to handle respective digits allows for the arithmetic operations of multi-bit numbers.

A diagram of a circuit

Description automatically generated

Figure A 2 bit adder: the input carry on the LSB bit is "grounded" to 0, as there is no summation before it. the input carry of the MSB is the output carry of the LSB

Ripple adding is the process as shown above, but can also be expanded, such that each high order bit adder receives the carry output of the adder before it as its carry input.

A diagram of a circuit

Description automatically generated

Figure : 4-bit ripple adder

Binary subtraction can be evaluated by the addition of a signed negative integer. Producing a negative integer follows the process of taking the 2s complement of the integer’s positive value, which is to reverse each bit from a 1 to a 0 or a 0 to a 1, in other words NOTing each bit. Then, the numbers 1s compliment is taken, which is simply adding 1 to the number, giving a negatively signed binary number using 2s compliment. Luckily anything Xor 1 will toggle a switch of the binary value, essentially a logical 2s complement of any bit if the other is HIGH, as shown in the table below.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Table : Xor truth table.

Taking the 1s compliment can be done by using the initial carry in on LSB adder to add 1. This process effectively results in the creation of an adder, which allows for an input to toggle it into subtraction mode.

A diagram of a block diagram

Description automatically generated

Figure : Adder/subtractor (M=0: addition | M=1: subtraction). Xor gates attached to be variables indicate the B is the number be subtracted.

**Implementation**

For task 1, First the inputs for the full adder were established as shown in the figure below.

A line of black and white lines with text

Description automatically generated with medium confidence

Figure : Full adder input pins. A and B are the bits to be added while Cin is the carry in bit

A decoder has the ability to take any code, for example a truth table, and translate it to an output. Thus, running the POS circuit of a truth table through a decoder would yield outputs that correspond to the original output. Logically, this would mean determining when an output has minterms, and connecting them together with an OR gate, yielding the original output. This was done for both the sum output and the Cout output for the full adder, using the full adder truth table. G1, G2AN, and G2BN were negligible inputs, thus were indefinitely activated.

A diagram of a computer program

Description automatically generated

Figure : Full adder outputs created using decoder and the full adder truth table SOPs.

These were all done in a single file, then replicated into a .bsf file. Now representing the full adder as a high-level block diagram in a separate file, a 4-bit adder/subtractor was created using the process presented in the introduction. 4 adders were cascaded together, each carry input being the carry output of the lower adder, and the ‘m’ subtraction activator was connected logically to the entire schematic. The inputs, outputs, and adders can be seen in the figure below. Inputs were pin planned as switches while outputs were pin planned as LEDs on the FPGA board.

A diagram of a circuit

Description automatically generated

Figure : Full schematic of 4-bit ripple adder subtractor. B bits are XORed to m, so B is the number being subtracted.

For task 2, the inputs, being the switches, were established and accordingly pin planned as shown in the schematic below.

A diagram of a circuit

Description automatically generated

Figure : input switches. The numbers in their names correspond to the number they display on the 7-segment display. They are NOTed for the encoders used are active LOW.

The Encoder part provided was an 8x3 encoder, meaning that two encoders will have to be cascaded to account for all 9 possible inputs. This was done, implementing the logic design for cascading active low encoders, as shown previously. The last inputs of the MSB encoder were left perpetually inactive, as they went out of the range of a single 7-segment display. A decoder for binary to 7-segment was provided to make the task of displaying the number more feasible. The decoders inputs were attached to the cascaded encoders’ outputs, and its output was presented as individual segments as shown in the figure below.

A diagram of a circuit

Description automatically generated

Figure : schematic of cascaded encoders, which are outputted to the BCD to 7SEG decoder

**Results**

When displaying task 1, the 4-bit adder worked as intended, as a number was inputted in binary along A and Bs respective switches, it returned the binary sum, of A switches and B switches, through LEDs. For example, 12 +10 returned binary 22. When the M switch was activated, ensuring B is the negative integer, it returned their subtraction. For example 8{A} + (-4){B} returned binary 4. When running task 2, as each switch was flipped up in ascending order, the number representing the highest number switch flipped was displayed on the 7-segment display.

**Conclusion**

The tasks displayed how decoders are devices, useful in efficiently implementing tedious logic into outputs. As seen in task one, a decoder was used to implement the SOP circuits of a full adder into two outputs, which then afforded the ability to cascade the new device into a more complex system. Encoders also proved to be lucrative devices, in their capacity to transfer data from one state to another.