The purpose of this first homework assignment is to design different register-level solutions for specific hardware implementations and problems.

**PART 1**

**Subpart A – 8-to-1 Multiplexer**

Using a standard 2-to-1 MUX as a primitive gate, the following tree-structure can be built in order to form an 8-to-1 MUX:



This concludes the analysis for Homework 1, Part 1, Subpart A.

**Subpart B – Quad 8-to-1 Multiplexer**

In order to form the Quad 8-to-1 Multiplexer, combine a total of 4 of the previous multiplexers from Subpart A, and put them next to each other. To aid in visual simplicity, the following picture depicts the Quad 8-to-1 Multiplexer as one joint image, where each dotted rectangle indicates a separate 8-to-1 multiplexer. Note that all multiplexers use the same select signals:



This concludes the analysis for Homework 1, Part 1, Subpart B.**Subpart C – 2-bit Magnitude Comparator**

In order to build a 2-bit Magnitude Comparator, we utilize the Quad 8-to-1 Multiplexer from Subpart B. Each of the individual 8-to-1 multiplexers will be used in order to implement the following functions: EQ, LT, GT, and LE. The select inputs are {A[0], B[1:0]}, with A[0] being the MSB. Available to the inputs for each of the functions are 0, 1, A[1], and ~A[1].

Given that there are 16 situations that will govern the individual 8 inputs for all 4 multiplexers, it is important to analyze each function separately. In the following diagrams, all functions are analyzed separately, with a sequence of bits highlighted in **RED** indicating the function is **NOT** met, and **GREEN** indicating it **IS** met.









Given the inputs to the 2-bit Magnitude Comparator of A[1:0] and B[1:0], the last two columns in each of the tables above depict the required connections to the inputs of that specific multiplexer. If all four multiplexers of the Quad 8-to-1 Multiplexer are wired as described above, then the 2-bit Magnitude Comparator will function as intended.

The image below is a pictorial representation of what the 2-bit Magnitude Comparator would look like:



This concludes the analysis for Homework 1, Part 1, Subpart C.**Subpart D – 8-bit Magnitude Comparator**

In order to build an 8-bit Magnitude Comparator, we can take the 2-bit Magnitude Comparator created in Subpart C and cascade the outputs. Since the 2-bit Magnitude Comparator was not built with cascading EQ, LT, GT, and LE inputs, glue logic will needed in order to cascade the different outputs. Below is a diagram of the circuit layout generated:



The following diagrams explicitly show the glue logic being used:



Glue logic, cascaded downwards starting from the MSB and going to the LSB, must be used in order to maintain proper functionality and order between the different 2-bit comparator outputs. Some discrete gates are saved by realizing that the outputs of the **GT** and **LE** functions, as seen in the tables in Subpart C, are complements.

This concludes the analysis for Homework 1, Part 1, Subpart D.**PART 2 – 1011 Moore Machine Sequence Detector**

Start by designing the State Diagram for the 1011 Detector:

1

0

1

1

0

1

0

0

1

0

Utilizing this diagram as a basis, develop the State Table of the 1011 Detector:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| State | 0 |  | 1 | Output |
| A | A |  | B | 0 |
| B | C |  | B | 0 |
| C | A |  | D | 0 |
| D | A |  | E | 0 |
| E | A |  | B | 1 |

Now, create the state assignment matrix:

|  |  |  |  |
| --- | --- | --- | --- |
| State | y2 | y1 | y0 |
| A | 0 | 0 | 0 |
| B | 0 | 0 | 1 |
| C | 0 | 1 | 0 |
| D | 0 | 1 | 1 |
| E | 1 | 0 | 0 |

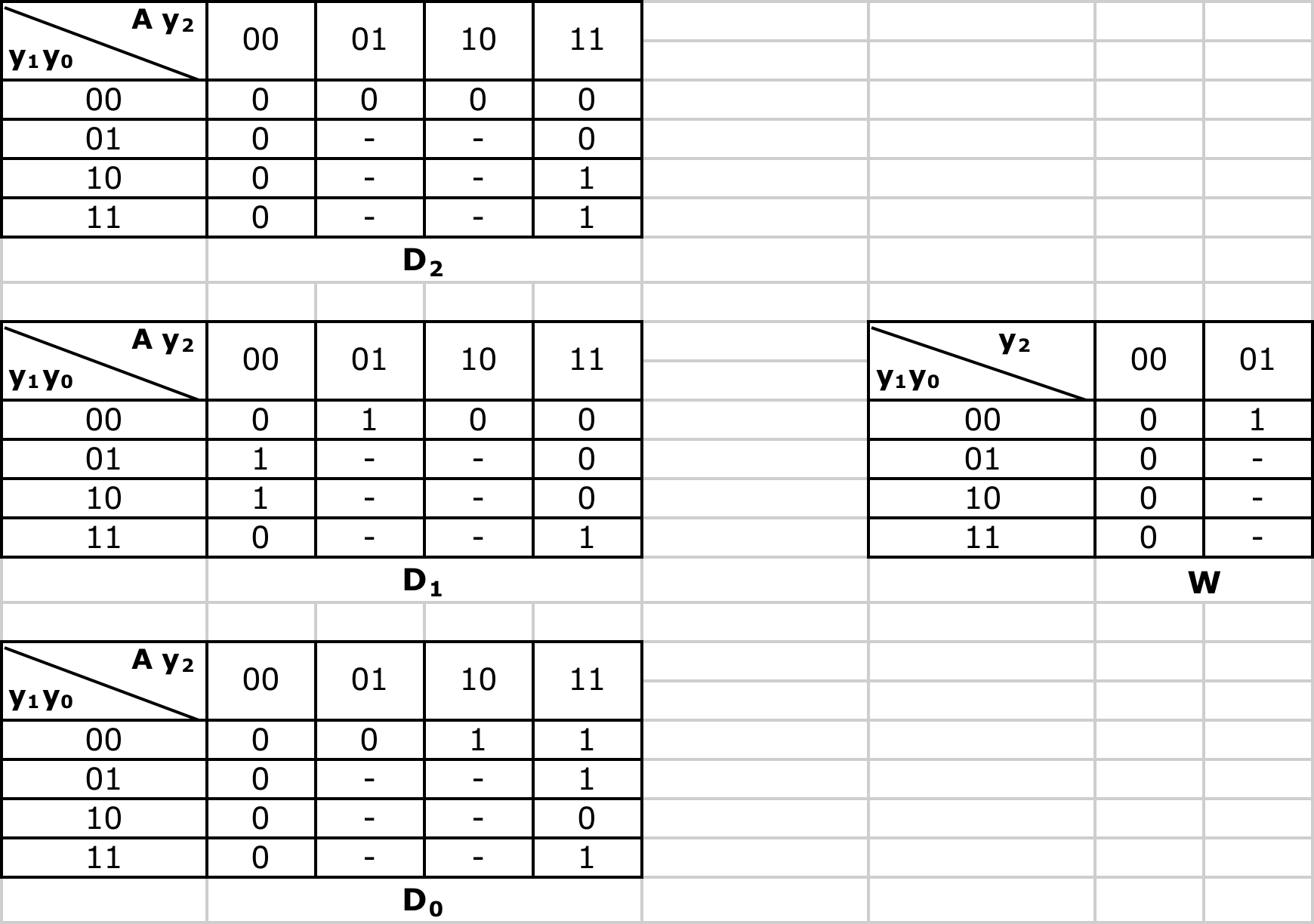
With the state assignments, create the Transition Table for the 1011 Detector:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | A | |  |  |
| State | **y2** | **y1** | **y0** |  | **0** | **1** |  | **Output** |
| A | 0 | 0 | 0 |  | 0 0 0 | 0 0 1 |  | 0 |
| B | 0 | 0 | 1 |  | 0 1 0 | 0 0 1 |  | 0 |
| C | 0 | 1 | 0 |  | 0 0 0 | 0 1 1 |  | 0 |
| D | 0 | 1 | 1 |  | 0 1 0 | 1 0 0 |  | 0 |
| E | 1 | 0 | 0 |  | 0 1 0 | 0 0 1 |  | 1 |
|  |  |  |  |  | **y2+ y1+ y0+** | |  | **W** |

Using this transition table, derive the Flip Flop Excitation Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | A | |  |  |
| y2 | **y1** | **y0** |  | **0** | **1** |  | **Output** |
| 0 | 0 | 0 |  | 0 0 0 | 0 0 1 |  | 0 |
| 0 | 0 | 1 |  | 0 1 0 | 0 0 1 |  | 0 |
| 0 | 1 | 0 |  | 0 0 0 | 0 1 1 |  | 0 |
| 0 | 1 | 1 |  | 0 1 0 | 1 0 0 |  | 0 |
| 1 | 0 | 0 |  | 0 1 0 | 0 0 1 |  | 1 |
| 1 | 0 | 1 |  | - - - | - - - |  | - |
| 1 | 1 | 0 |  | - - - | - - - |  | - |
| 1 | 1 | 1 |  | - - - | - - - |  | - |
|  |  |  |  | **D2 D1 D0** | |  | **W** |

With the Flip Flop Excitation Table, develop the Karnaugh Maps for each of the flip-flops and the output. Then, use the Karnaugh Maps to determine the logic needed to model the flip-flops and outputs in terms of the state variables:



Using the Karnaugh maps, the following minimizations are realized.

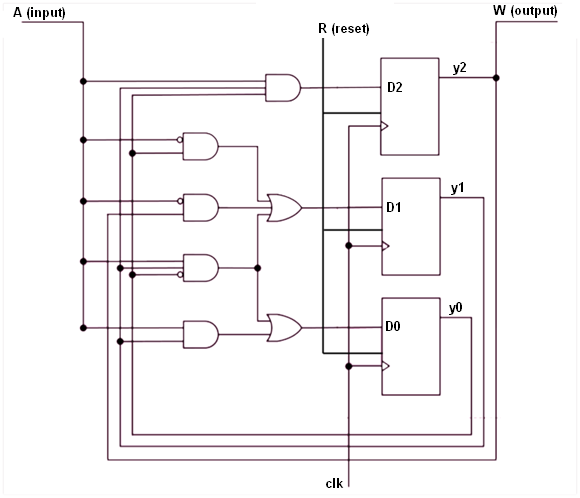
**D2= A y1 y0**

**D1 = A y0 + A y2 + A y1 y0**

**D0 = A y1 + A y0**

**W = y2**

With all of the flip-flop models established, the following circuit diagram is drawn:



Each of the D flip-flops contains the asynchronous reset variable R, which when modeled in the Verilog HDL, is added to the sensitivity list to allow for independent behavior from the clock. This will essentially force all y outputs (Q) to be 0.

This concludes the analysis for Homework 1, Part 2.

**PART 3 – Counter Circuit**

The counter circuit for Part 3 of this homework is to be developed with the

following sequence progression in mind:

It is important to note that this counter is being developed with a possibility of a direction change in counting. If the signal *direction* = 1, then the circuit will count as intended based on the figure above. If the signal *direction* = 0, then the circuit will count in reverse order based on the figure above. For the sake of visual simplicity, the variable *direction* has been renamed *d*. The following State Transition Table is then formed:



In order to develop the Karnaugh maps for the individual D inputs, it is important to remember the Transition Table for the D Flip-Flop:



Utilizing both State Transition Tables, develop the Karnaugh maps for all D Flip-Flop inputs for the counting portion of the circuit:



**D2 = ~d~y1~y0 + dy1~y0 +dy2~y1 + ~dy2y1**

**D1 = y1~y0 + ~dy1y0 + ~d~y1~y0 + d~y2y0 + dy2~y0**

**D0 = ~y1~y0  + y1~y0**

Connecting the D Flip-Flop inputs as indicated based on the Karnaugh maps above will result in the desired counting sequence, based on the *direction* signal *d*.

Implementing the negating and resetting operations can be done with multiplexers, taking advantage of the ~Q output that comes standard with most D Flip-Flops. Since the D Flip-Flops are inherently synchronous (clock dependant), the output of the multiplexers will only “matter” when we hit a pulse on the clock. The following diagram below depicts the overall logic of the counter circuit, given the Flip-Flop logic derived from the Karnaugh maps above:



This concludes the analysis for Homework 1, Part 3.

**PART 4 – Serial Odd-Parity Generator**

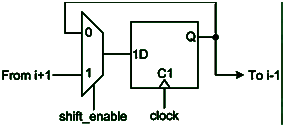
For this last part of the assignment, we are asked to take in 7 bits of data, considered an input stream, on *x* once the *start* signal is issued. The bits will be placed one at a time on output *z*. On the 8th clock pulse, the output *z* will generate an odd-parity bit from the 7 bits of input data it just received. At this point, it will assert the *ready* signal, indicating that the system is ready to process another stream of input. A *reset* input was added for the Controller. The high level system diagram is as follows:



In order to correctly detect odd parity with a 7-bit input stream, the resultant output of a 7-bit XOR operation will result in the correct parity bit output. As for the Datapath of the system, this entire process can be implemented with a 2-bit shift register, and 2 multiplexers. The 1st multiplexer will cause input to be first placed into the shift register once the first bit is received, and will then serve as feedback in order to keep performing the serial XOR operation. The 2nd multiplexer is used to determine whether the output will be the input, or in the case of the 8th clock pulse, the parity bit output. A design of the Datapath is seen below:



The two bit shift-register is easily implemented by cascading two of the following individual shift-register blocks:

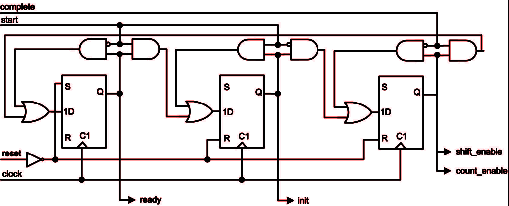


Based on the modular design of the system, and given the fact that it processes in a total of eight clock pulses, the controller layout from the Serial Adder Controller can be used as is, ensuring to send the complete signal up to the Datapath:

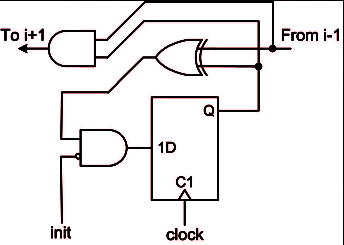
**Serial Parity Generator – Controller Design**

The state-machine is realized as a one-hot implementation using R-S Flip\_Flops, and again, is leveraged from the Serial Adder using all mentioned control signals:

**Serial Parity Generator – State-Machine Design**



The last portion to implement is the counter. Since we are interested in a total of 8 clock pulses, we can utilize a modulo-8 counter, which can be implemented by cascading together three of the following counter blocks:



Once the counter finishes counting and *complete* gets set to 1, complete the final odd-parity of the input stream being collected and place it on *z*. Based on the one-hot implementation, a pulse on *start* in the middle of the sequence will reset the circuit and input will begin to be gathered once start is unasserted.

This concludes the analysis for Homework 1, Part 4.