EECE573: Assignment #5

## Tejas Randeria

# CAUTION: Please use Xilinx to run the files, it’ll not work in Modelsim All the files included have to present in the same folder

# Objective:-

The assignment divided into 4 parts

* Part A: FIFO
  + To create a FIFO module (size should be parameterized and tested for more than 1 size). Make sure to find corner cases (e.g. full, empty, rollover from bottom to top). Position pointers must be self-contained in the module. Make sure to properly initialize them.
* Part B: LIFO/Stack
  + To create a LIFO module (size should be parameterized and tested for more than 1 size). Make sure to find corner cases (e.g. full, empty, rollover from bottom to top). Position pointers must be self-contained in the module. Make sure to properly initialize them.
* Part C: Synthesize each of the above queues of 4 4-bit entries. Show analysis for speed/area/power under the following constraints:
  + Minimum area
  + Minimum latency
  + Minimum power
* Part D: Synthesis
  + Create a synthesized version of the 16-bit ripple-carry adder from Assignment #2, using Design Vision
  + Repeat the process for the module “simple\_adder.v” from Blackboard.
  + Compare the final area/speed numbers for the RCA and the simple\_adder.

# Description:-

* Part A: FIFO
  + All the variables that are present in the code have been named to be self-explanatory such as
    - vector\_in: the input vector to the fifo\_top module
    - reset: to get the system to its initial state
    - clk: synchronous system
    - data\_out: the requested data
    - empty\_flag, full\_flag: denoting the current status of the fifo
  + The vector\_in has been segregated and stored in two regs
    - control\_in: the first 2 bits deciding the type of operation to be performed
    - data\_in: used only during write operation, else not used
* Part B: LIFO/Stack
  + All the variables that are present in the code have been named to be self-explanatory such as
    - vector\_in: the input vector to the lifo\_top module
    - reset: to get the system to its initial state
    - clk: synchronous system
    - data\_out: the requested data
    - empty\_flag, full\_flag: denoting the current status of the lifo
  + The vector\_in has been segregated and stored in two regs
    - control\_in: the first 2 bits deciding the type of operation to be performed
  + data\_in: used only during write operation, else not used
* Part C: Synthesized the queues and reported the results below
* Part D: Synthesized the adder modules and reported the results below

# Working:-

* Part A: FIFO
  + Being a synchronous design, we are using clock. The reset signal is used to initialize the module.
  + The status (empty and full flags) is being processed first, effecting the next process taking place i.e. whether to write or read from the fifo
  + fifo\_valid\_invalid\_bit – denotes the validity of a particular location
  + fifo\_data – contains the data present in the fifo
  + If the fifo is empty, data cannot be read and if the fifo is full, data cannot be written; the status flags notify if the above operations can take place
  + Additional conditions have to be met
    - FIFO full – denoted by the head and tail pointer pointing at the same location containing valid data
    - FIFO empty – denoted by the head and tail pointer pointing at the same location containing invalid data
  + In this case roll over can occur as the read and write pointer are moving in the same direction for a read and write
  + As we read, invalidation of the location occurs hence data can be written on it if overflow occurs
  + Write pointer always points to the next available location to be written and Read pointer always points to the last written valid location
* Part B: LIFO
  + Being a synchronous design, we are using clock. The reset signal is used to initialize the module.
  + The status (empty and full flags) is being processed first, effecting the next process taking place i.e. whether to write or read from the lifo
  + lifo\_valid\_invalid\_bit – denotes the validity of a particular location
  + lifo\_data – contains the data present in the lifo
  + If the lifo is empty, data cannot be read and if the lifo is full, data cannot be written; the status flags notify if the above operations can take place
  + Additional conditions have to be met
    - LIFO full – denoted by the head and tail pointer pointing at the same location containing valid data
    - LIFO empty – denoted by the head and tail pointer pointing at the same location containing invalid data
  + In this case roll over cannot occur as the read and write operations have a different functionality

# Conclusion:-

* Part A: FIFO
  + My test bench is extensive and covers all the cases as well as corner cases

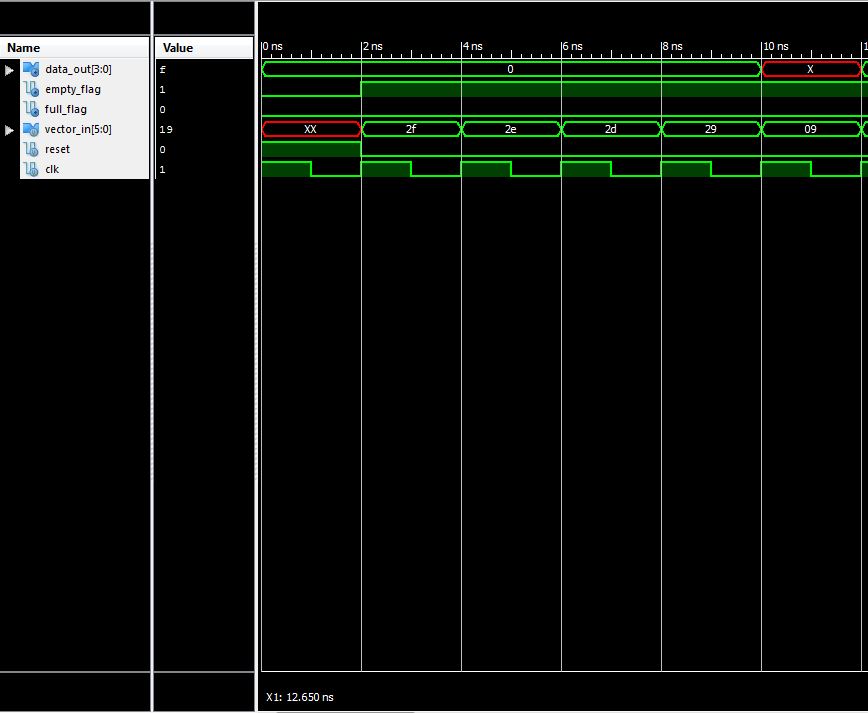


Figure FIFO Reset & Write operation

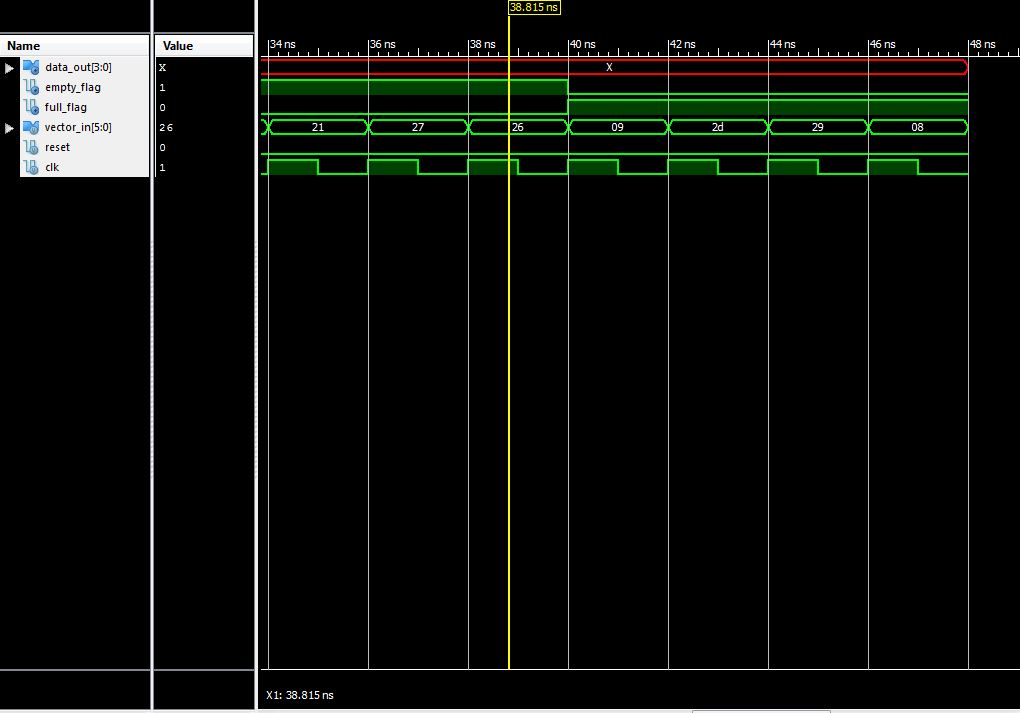


Figure FIFO Empty & Full Flags

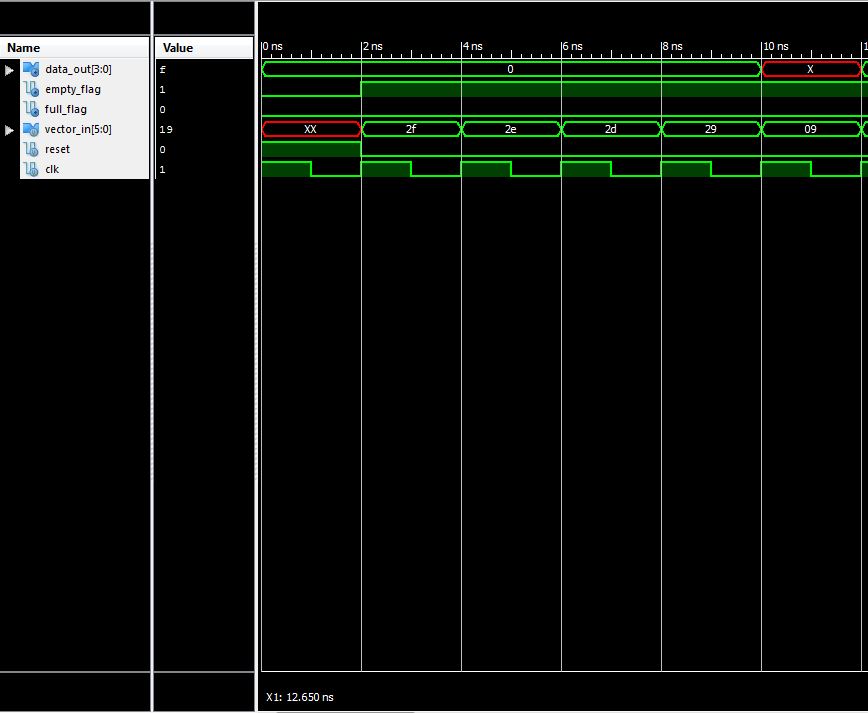


Figure FIFO Read & More Write Operations

* Part B: LIFO

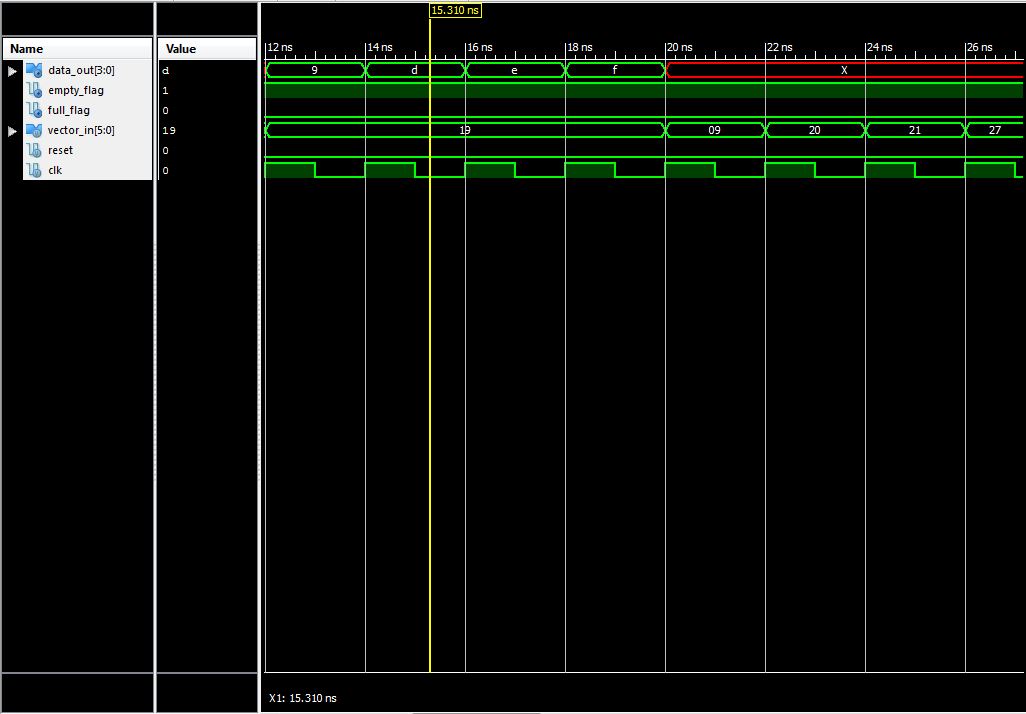


Figure LIFO Read & More Write Operations



Figure LIFO Reset & Write Operations

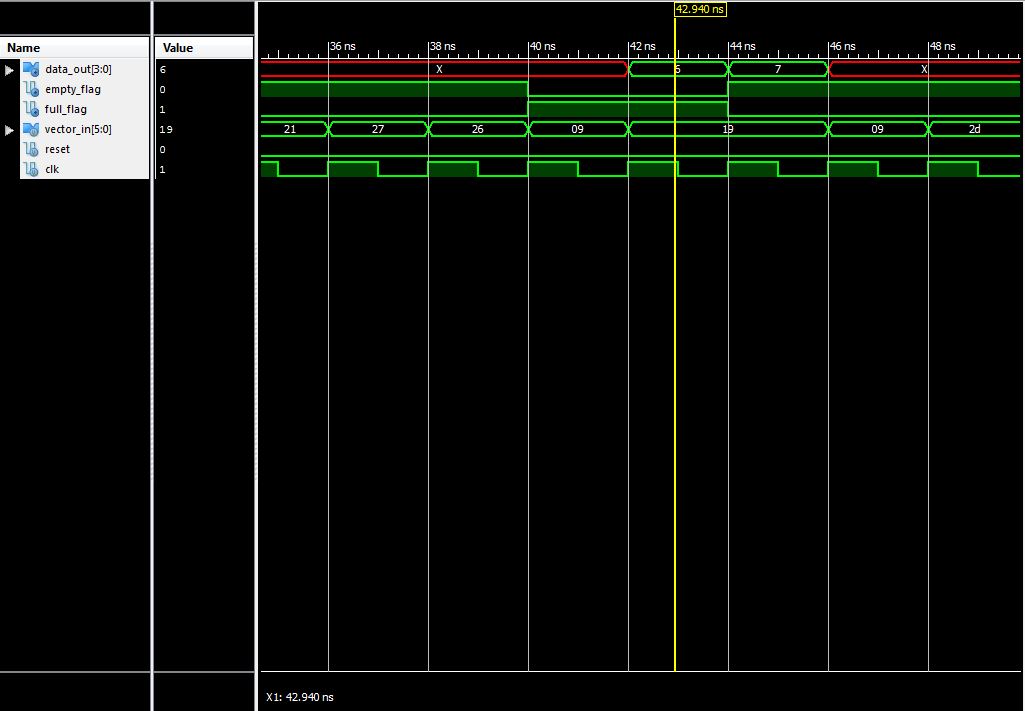


Figure LIFO Empty & Full Flags

* Part C: Synthesis Analysis (Queues)
  + After each analysis removed the design and loaded it again
  + FIFO
    - Considering no optimization is applied while compiling the design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis (No optimization) | Timing (Arrival) | Total Area | Dynamic power | Leakage power |
| FIFO | 0.86 ns | 1559 | 18.09 uW | 7 uW |

* + - Optimizing for Timing
      * The minimum I could get with the optimizations was 0.86ns
    - Optimizing for Area
      * Settings the max area - 0
      * Compiling with high map effort and area effort
      * Decreases the area by a small amount while increasing the dynamic power

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis (Area Optimized) | Timing (Arrival) | Total Area | Dynamic power | Leakage power |
| FIFO | 0.86 ns | 1552.7 | 19.12 uW | 7.03 uW |

* + - Optimizing for Power
      * Setting the dynamic to 10uW
      * Compiling with high power effort, low area effort and high map effort
      * Increases the longest path timing by a small amount and decreases the dynamic power by a small amount

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis (Power Optimized) | Tming (Arrival) | Total Area | Dynamic power | Leakage power |
| FIFO | 0.90 ns | 1572.13 | 17.94 uW | 7 uW |

* + LIFO
    - Considering no optimization is applied while compiling the design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis (No Optimization) | Timing (Arrival) | Total Area | Dynamic power | Leakage power |
| LIFO | 0.88 ns | 1618.15 | 19.87 uW | 7 uW |

* + - Optimizing for Timing
      * The minimum I could get with the optimizations was 0.88ns
    - Optimizing for Area
      * Settings the max area - 0
      * Compiling with high map effort and area effort

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis (Area Optimized) | Timing (Arrival) | Total Area | Dynamic power | Leakage power |
| LIFO | 0.91 ns | 1626 | 21.04 uW | 7.2 uW |

* + - Optimizing for Power
      * Setting the dynamic to 10uW
      * Compiling with high power effort, low area effort and high map effort
      * Increases the longest path timing by a small amount and decreases the dynamic power by a small amount

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis (Power Optimized) | Timing (Arrival) | Total Area | Dynamic power | Leakage power |
| LIFO | 0.90 ns | 1642.12 | 19.21 uW | 7 uW |

* Part D: Synthesis Analysis (RCA and Simple adder)
  + Ripple Carry Adder (RCA)
    - Optimizing for Area and Timing
      * Settings the max area - 0
      * Compiling with high map effort and area effort

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis | Timing (Arrival) | Total Area | Dynamic power | Leakage power |
| RCA | 2.14 ns | 645.446 | 206.0775 uW | 3.1385 uW |

* + Simple Adder
    - Optimizing for Area and Timing
      * Settings the max area - 0
      * Compiling with high map effort and area effort

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Analysis | Timing (Arrival) | Total Area | Dynamic power | Leakage power |
| Simple Adder | 4.10 ns | 472.84 | 133.42 uW | 2.27 uW |

* + The clock frequency for the RCA = (1/Tming) = (1/2.14ns) = 467 Mhz
  + The clock frequency for the Simple Adder = 243 Mhz

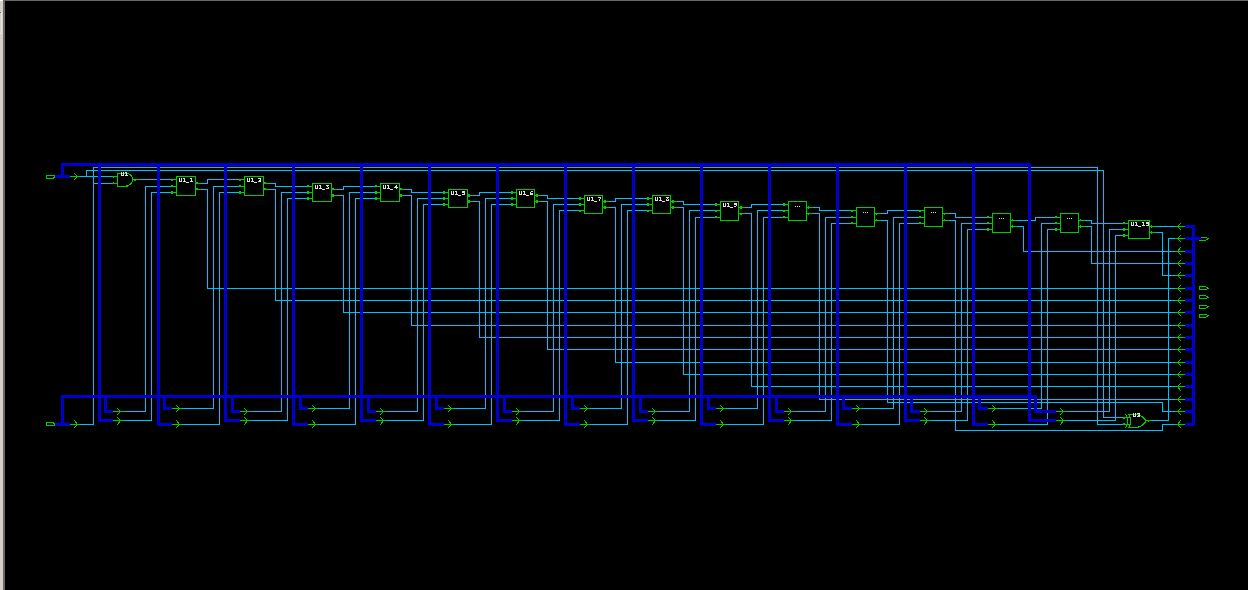


Figure Simple Adder Schematic

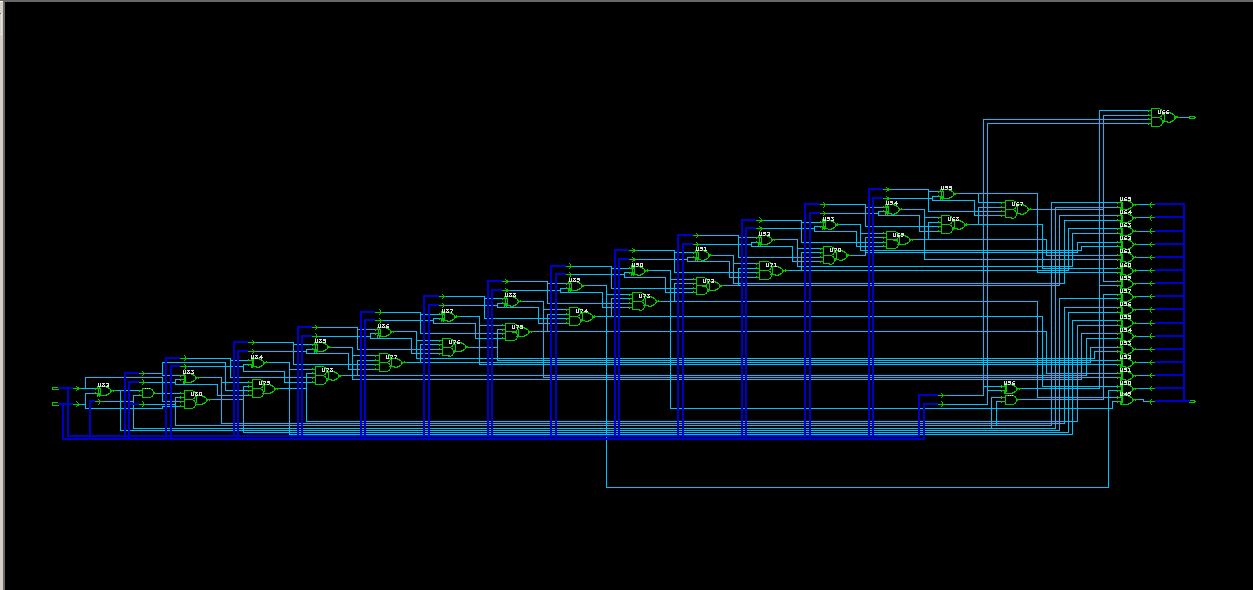


Figure RCA Schematic

* + As we can see from the schematic, the gate implementation is very different
  + Due to this varied implementation the longest timing path of the adders is different. Now considering this particular technology, the simple adder implementation introduces a lot of the delay in the circuit. Whereas the RCA circuit has a smaller longest path delay compared to the simple adder
  + Even though the area requirements of the simple adder is less it is severely affected by the longest path delay and dynamic power

# Feedback:-

* Tejas Randeria
  + Implemented the lifo and fifo queues in Verilog
  + More understanding of the synopsys tool
  + Understood the various tools in optimizing the circuit for various parameters