CDA 3203 – Computer Logic Design **Name____________________________**
Fall 2022 – Dr. Petrie **Lab 3** **__/100 pts**

Lab 3: The objectives of this lab are:

1) *16 pts. Truth Table, Timing Diagram and Excitation Tables that describe the behavior of D, T, S-R and J-K Flip Flops.*
2) *20 pts. Simulate in Altera Quartus a Flip Flop in VHDL using Behavioral Architecture.*
3) *20 pts. Design and simulate a 1-bit register component.*
4) *20 pts. Design and simulate a 16-bit register component.*
5) *20 pts. Design and simulate a RAM with 8 registers.*
6) *20 pts. Design and simulate a program counter.*

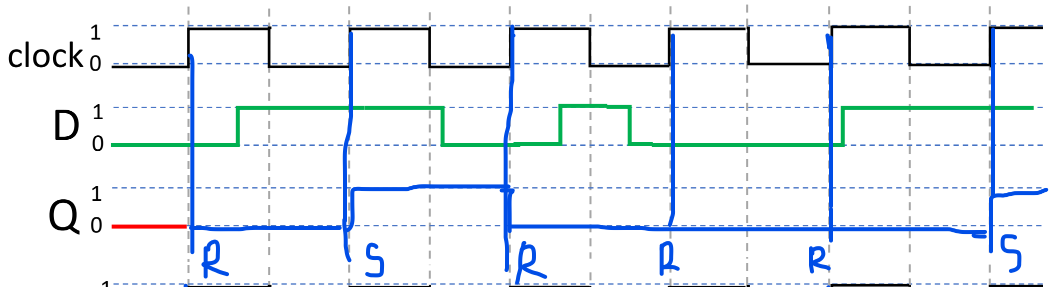
Submit a Lab Report with the required documentation.

1. 16 pts. Handwork

Complete Truth Table and Timing Diagram for each Flip Flops, **all inputs are active high**, **clock is positive-edge triggered** (rising edge)

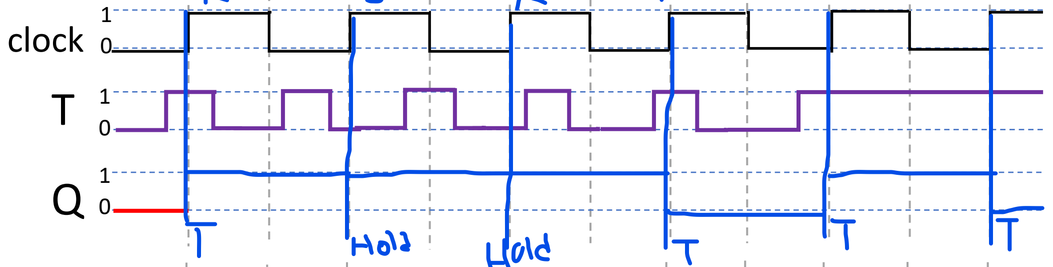
1.1 D Flip Flop

D	Q	Command	Q+
0	0	Make Q=0	0
	1		0
1	0	Make Q =1	1
	1		1



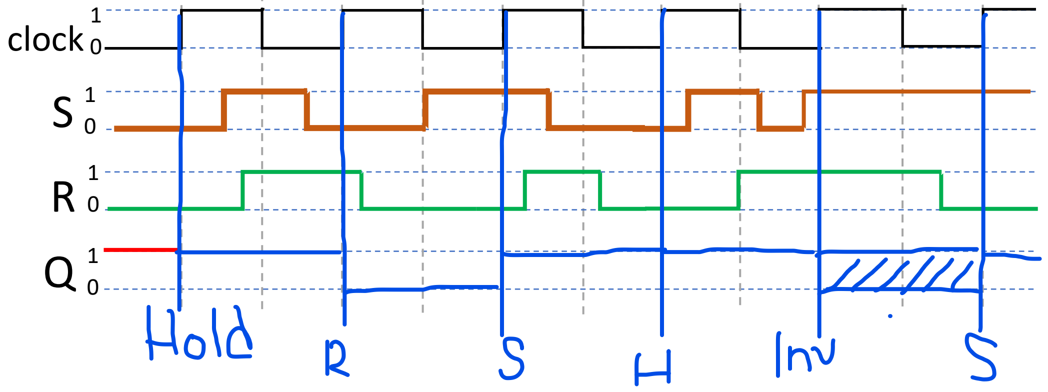
1.2 T Flip Flop

T	Q	Command	Q+
0	0	Hold	0
	1		1
1	0	toggle	1
	1		0



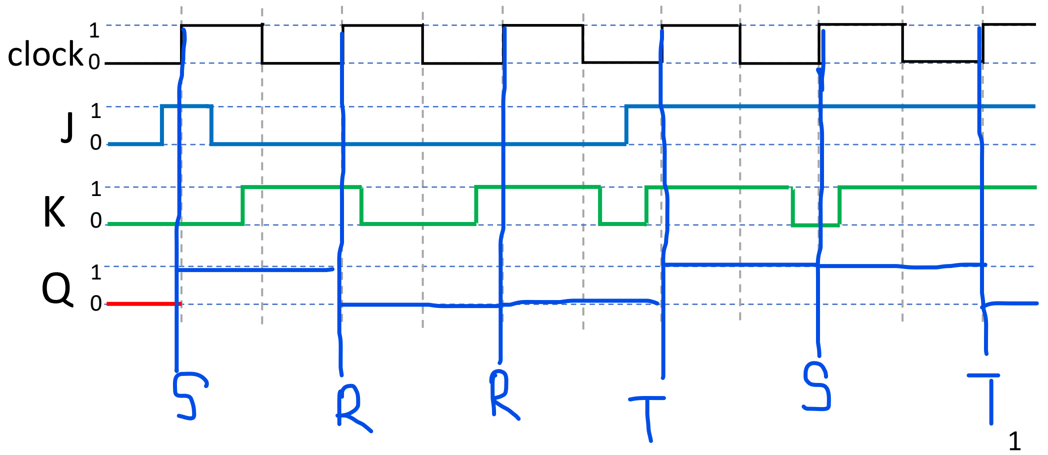
1.3 S-R Flip Flop

S	R	Q	Command	Q+
0	0	0	HOLD	0
		1		1
0	1	0	RESET	0
		1		0
1	0	0	SET	1
		1		1
1	1	0	INVALID	?
		1		?



1.4 J-K Flip Flop

J	K	Q	Command	Q+
0	0	0	HOLD	0
		1		1
0	1	0	RESET	0
		1		0
1	0	0	SET	1
		1		1
1	1	0	Toggle	1
		1		0



1.5 Excitation Tables are used to figure out what you need to place on the input of a flip flop to force Q to the desired transition. Look at Truth Tables in the previous page to find what commands and flip flop input values are needed to force next transition.
Note : all inputs of the flip-flopss are **active high**

Q→Q⁺	Command	D
0→0	Make Q=0	0
0→1	Make Q=1	1
1→0	Make Q=0	0
1→1	Make Q=1	1

Q→Q⁺	Command	T
0→0	Hold	0
0→1	toggle	1
1→0	toggle	1
1→1	Hold	0

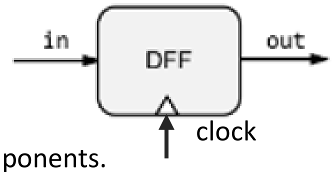
Q→Q⁺	Commands	S R
0→0	Hold reset	0X
0→1	set	10
1→0	reset	01
1→1	Hold set	X0

Q→Q⁺	Commands	J K
0→0	Hold reset	0X
0→1	toggle Set	1X
1→0	toggle reset	X1
1→1	Hold set	X0

2. Design and Simulation of Sequential Components in Altera Quartus using VHDL

VIEW **LAB 3 EXPLAINED VIDEO** FOR MORE DETAILS, CODE AND DIAGRAMS

2.1 Data Flip Flop (DFF)



Below is the schematic for a Data Flip Flop, which part of the design of the other components.
Up to now we have used a *Structural* VHDL description of how to connect the hardware components that make up the circuit. To the right is the *Behavioral* VHDL description of the circuit, which looks more like a software program and describes how it behaves instead of how it is built.
Use this code to Simulate the DFF

The VHDL statement **process** is used to execute the body of the process (enclosed between the **begin** and **end process**) each time a signal in the **sensitivity list** (in parenthesis) changes. You also see the conditional statement **if, then**, **end if** which works exactly as you experience in programming. In the condition, you see the **rising_edge** built-in VHDL clock function, which is true when a signal transitions from low to high. There is also a **falling_edge** function available in VHDL for negative edge triggered flip flops. Use the code at right and set the clock input to Value -> Clock In the Timing Diagram, and D values to mimic Ex. 1.1.

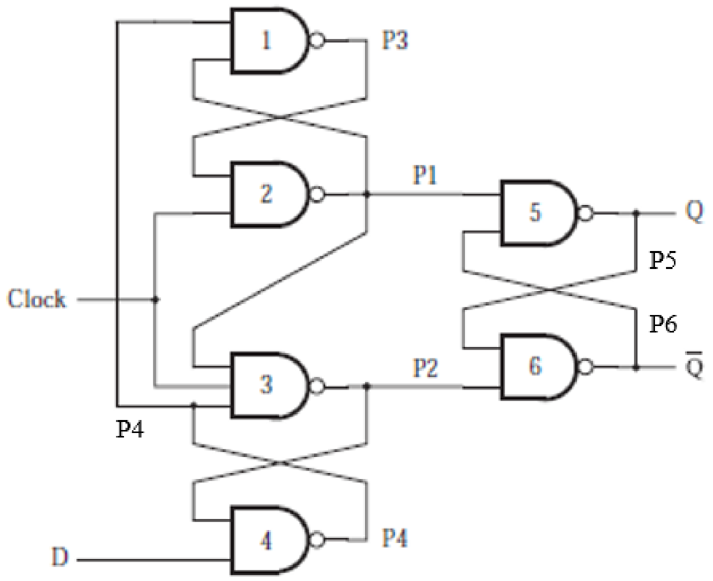
```
-- Data Flip Flop positive-edge triggered
-- Behavioral VHDL model
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY DFF_PetrieMaria IS
   PORT (D, clock: IN STD_LOGIC;
         Q       : INOUT STD_LOGIC );
END DFF_PetrieMaria;

ARCHITECTURE Behavior OF DFF_PetrieMaria IS
BEGIN
-- Update on the rising edge of the clock
   PROCESS (clock)
   BEGIN
      IF (RISING_EDGE(clock)) THEN
         Q <= D;
      END IF;
   END PROCESS;
END Behavior;
```

-- In .vwf Timing Diagram set values of inputs to mimic 1.1:
-- clock input to Value -> Clock
-- and adjust the parameters to set the Duty Cycle, length
-- of simulation, and Period; and
-- D input to same values as in Timing Diagram in 1.1

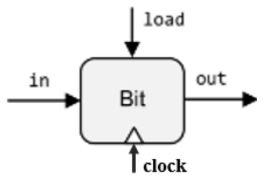
Turn in the snips of Project Setting, VHDL, Successful Compilation, and Timing Diagram in the portfolio



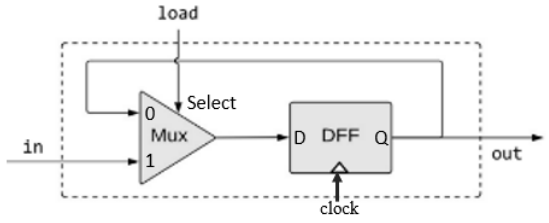
2.2 1-bit Register

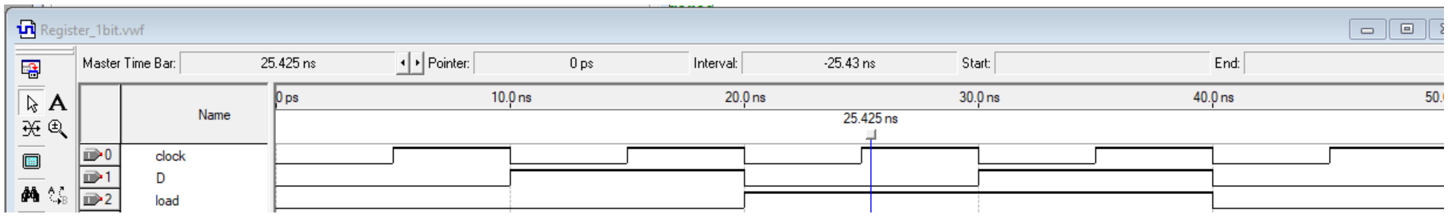
At right are 3 descriptions of a 1-bit register. Design and simulate in VHDL. You can use either the Structural or the Behavioral approach to specify it. Set the inputs of the timing diagram to match the one at right.

Turn in the snips of Project Setting, VHDL, Successful Compilation, and Timing Diagram in the portfolio.



```
-- 1-bit register:
-- if load(t – 1) then
--         out(t) =    in(t – 1)
-- else  out(t) = out(t – 1))
```



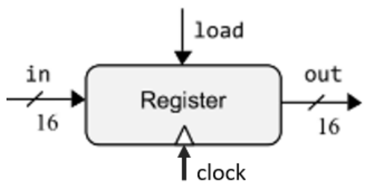


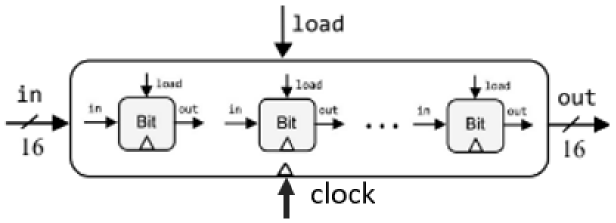
Register_1bit.vwf

Master Time Bar: 25.425 ns Pointer: 0 ps Interval: -25.43 ns Start: End:

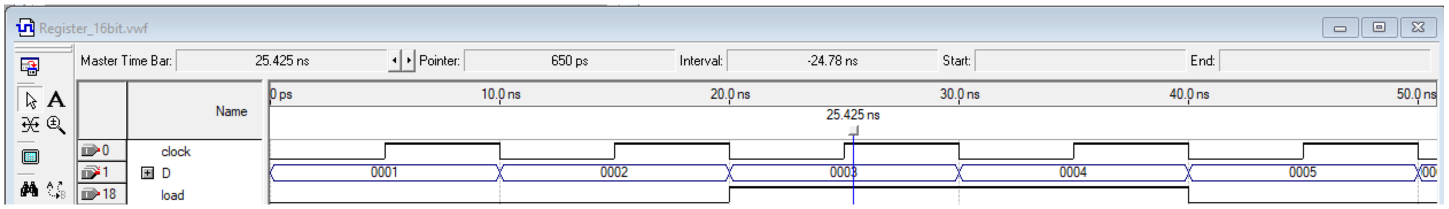
	Name
0	clock
1	D
2	load

2.3 16-bit Register





```
-- 1-bit register:
-- if load(t – 1) then
--         out(t) =    in(t – 1)
-- else
        out(t) = out(t – 1))
```

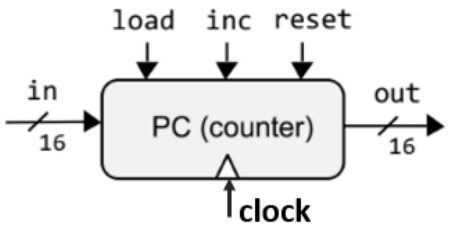


Register_16bit.vwf

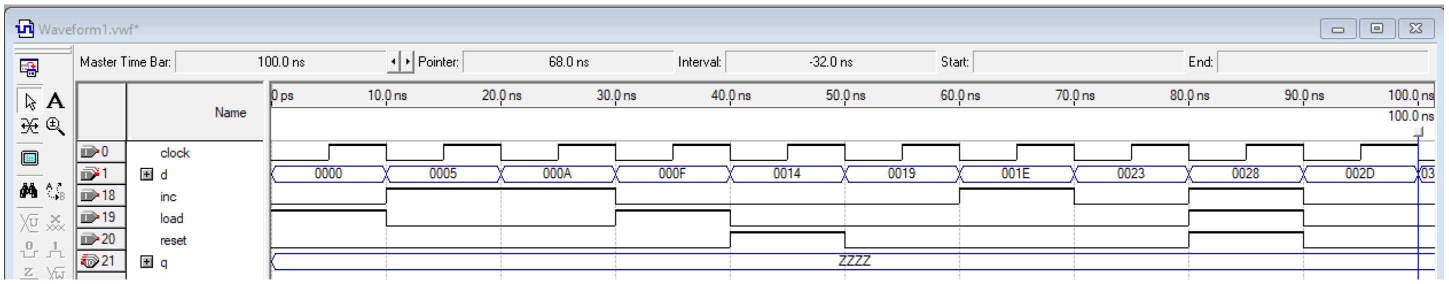
Master Time Bar: 25.425 ns Pointer: 650 ps Interval: -24.78 ns Start: End:

	Name
0	clock
1	D
18	load

2.4 Program Counter Register



```
-- A 16-bit counter with control bits.
-- if reset(t – 1) out(t) = 0                    // resetting
-- else if load(t – 1) out(t) = in(t – 1)        // setting
-- else if inc(t – 1) out(t) = out(t – 1) + 1 // incrementing
-- else out(t) = out(t – 1)                      // maintaining
```

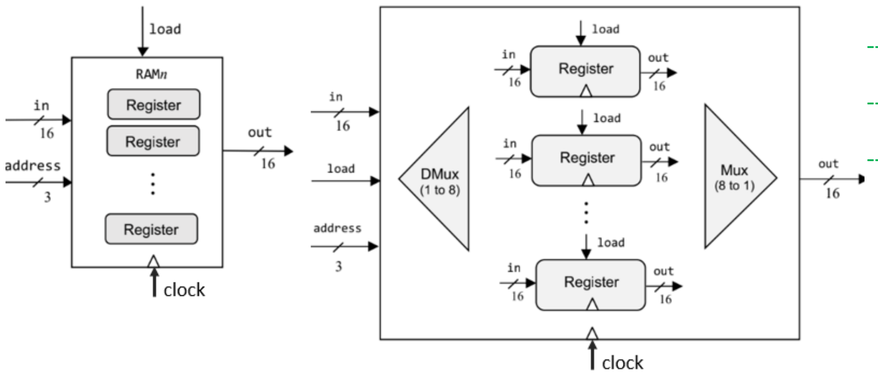


Waveform1.vwf*

Master Time Bar: 100.0 ns Pointer: 68.0 ns Interval: -32.0 ns Start: End:

	Name
0	clock
1	d
18	inc
19	load
20	reset
21	q

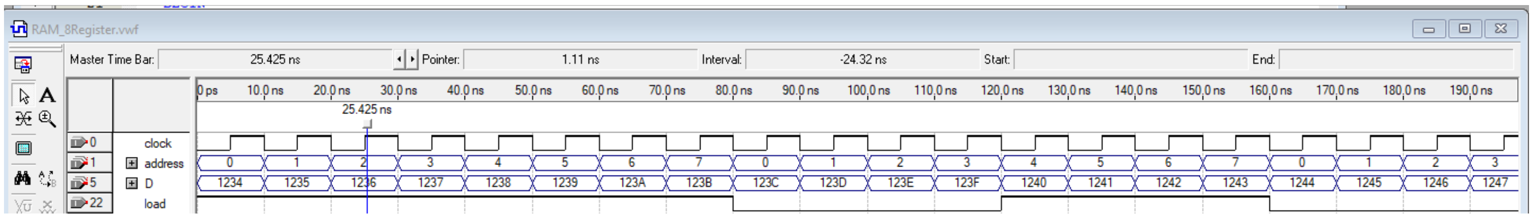
2.5 8 Register RAM



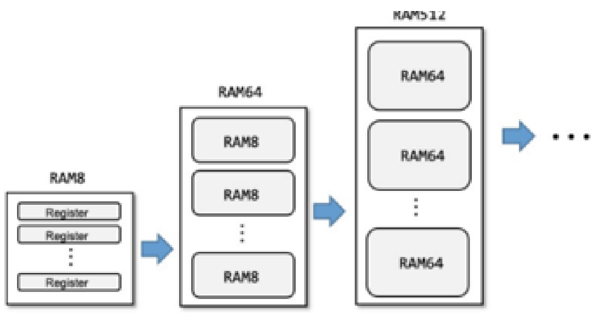
-- Let **M** stand for the state of the register selected by **address**.

-- if load(t – 1) then {**M** = in(t), out(t) = **M**}

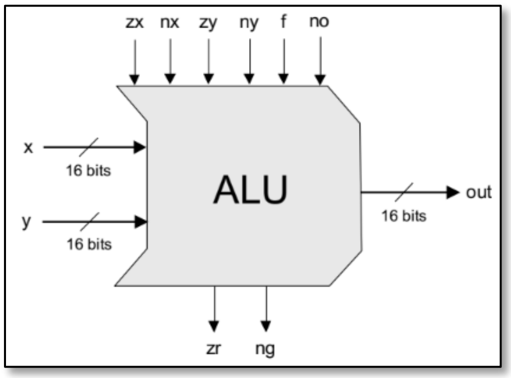
-- else out(t) = **M**


RAM_8Register.vwf
Master Time Bar: 25.425 ns
Pointer: 1.11 ns
Interval: -24.32 ns
Start:
End:
0 ps 10.0 ns 20.0 ns 30.0 ns 40.0 ns 50.0 ns 60.0 ns 70.0 ns 80.0 ns 90.0 ns 100.0 ns 110.0 ns 120.0 ns 130.0 ns 140.0 ns 150.0 ns 160.0 ns 170.0 ns 180.0 ns 190.0 ns
25.425 ns
clock
address 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3
D 1234 1235 1236 1237 1238 1239 123A 123B 123C 123D 123E 123F 1240 1241 1242 1243 1244 1245 1246 1247
load

2.6 Extra Credit – n-Register RAM – RAM64 – made up of 8 RAM8s



2.7 Extra Credit – Arithmetic Logic Unit – ALU – See video Nov. 29 for NAND2Tetris HDL Code explained



```
CHIP ALU {
    IN
        x[16], y[16],// 16-bit inputs
        zx,          // zero the x input?
        nx,          // negate x input?
        zy,          // zero y input?
        ny,          // negate y input?
        f,           // compute out=x+y (if 1)
                     // or out = x & y (if 0)
        no;          // negate out output?
    OUT
        out[16],       // 16-bit output
        zr,            // 1 if (out==0), 0 otherwise
        ng;            // 1 if (out<0),  0 otherwise
    PARTS:
Mux16(a=x,b[0..15]=false,sel=zx,out=zdx); //Zero the x
Not16(in=zdx,out=notx);                   //Not the x
Mux16(a=zdx,b=notx,sel=nx,out=ndx);    //chose x or notx
Mux16(a=y,b[0..15]=false,sel=zy,out=zdy); //Zero the y
Not16(in=zdy,out=noty);                   //Not the y
Mux16(a=zdy,b=noty,sel=ny,out=ndy );  //chose y or noty
Add16(a=ndx,b=ndy,out=xplusy);            //x+y
And16(a=ndx,b=ndy,out=xandy);             //x&y
Mux16(a=xandy,b=xplusy,sel=f,out=fxy);//chose function
Not16(in=fxy,out=nfxy);           //not the output or not
Mux16(a=fxy,b=nfxy,sel=no,out=oo);     //chose if not f
Or16Way(in=oo,out=o);  //for zr (if any 1's in oo, o=1
Not(in=o,out=zr);      //NOT o to detect no 1's (all 0's)
And16(a[0..15]=true,b=oo,out[15]=ng,out[0..14]=drop);//ng
// AND oo with all 1's, ng=out[15]=msb, if 1= neg, 0=pos
Or16(a=oo,b[0..15]=false,out=out); //oo=output
}
```