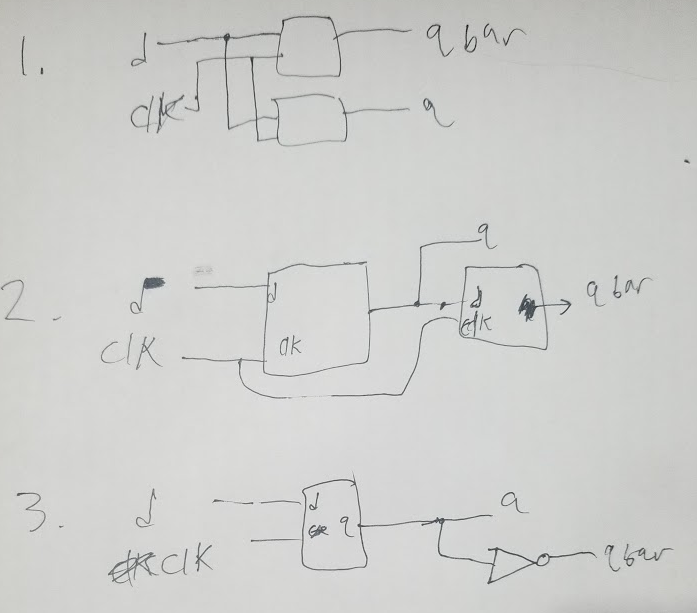
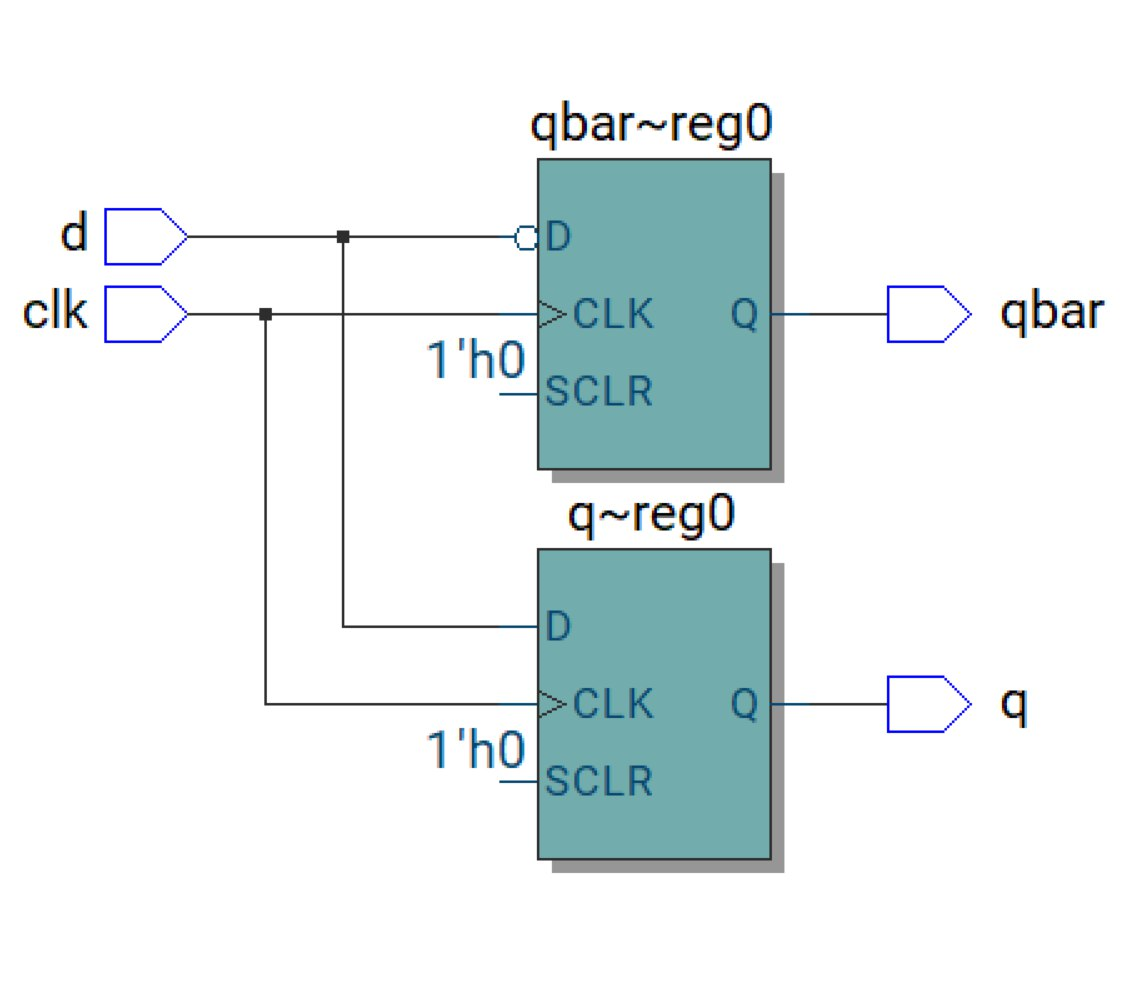
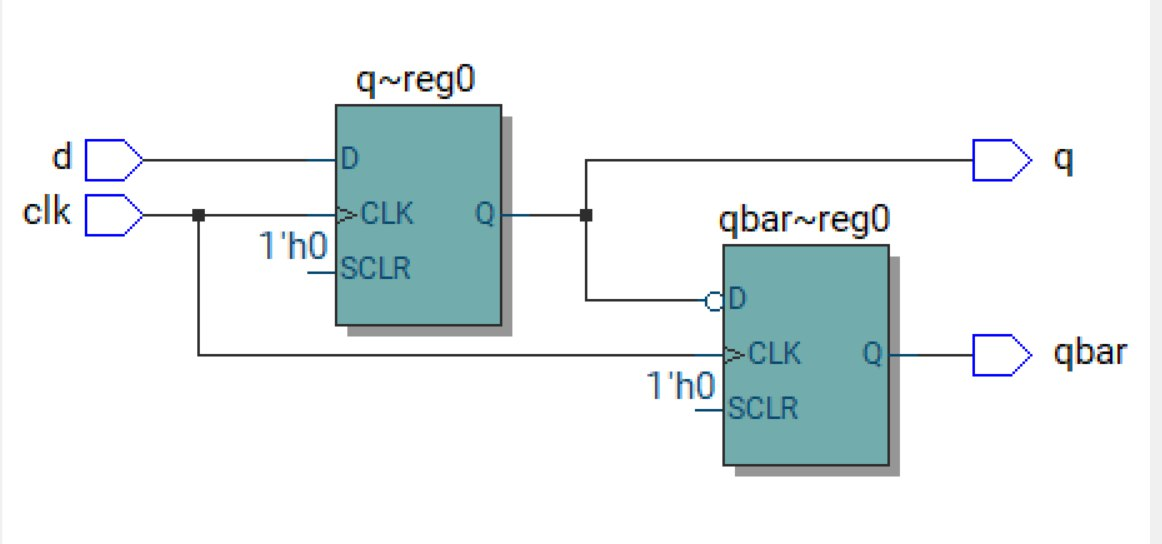
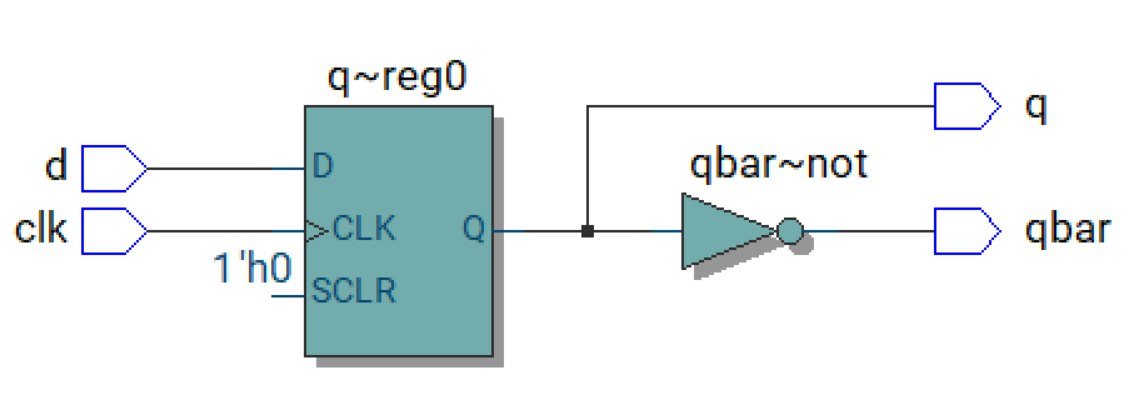
Exercise 1: flip flops inference

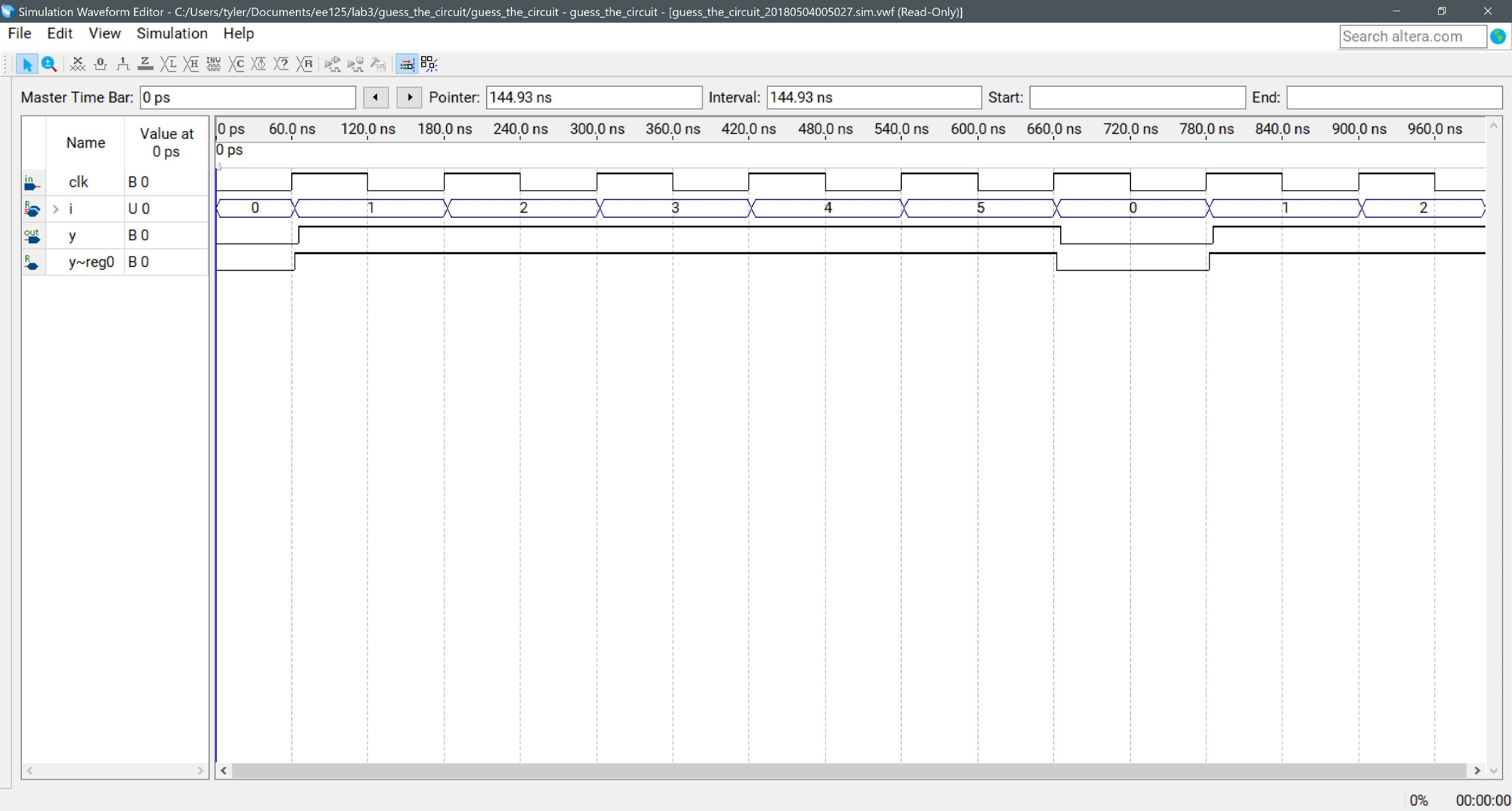
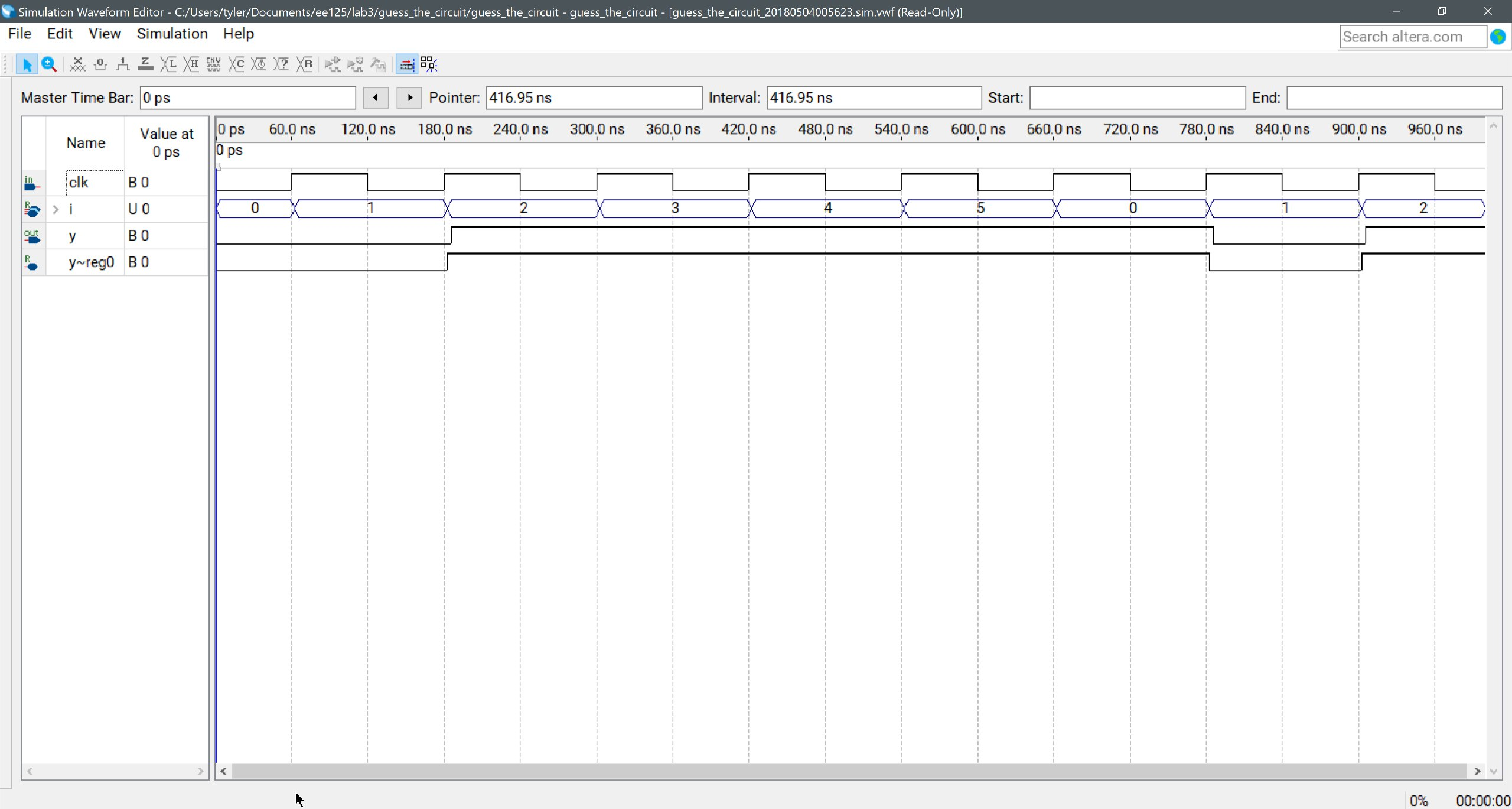
1. Drawings



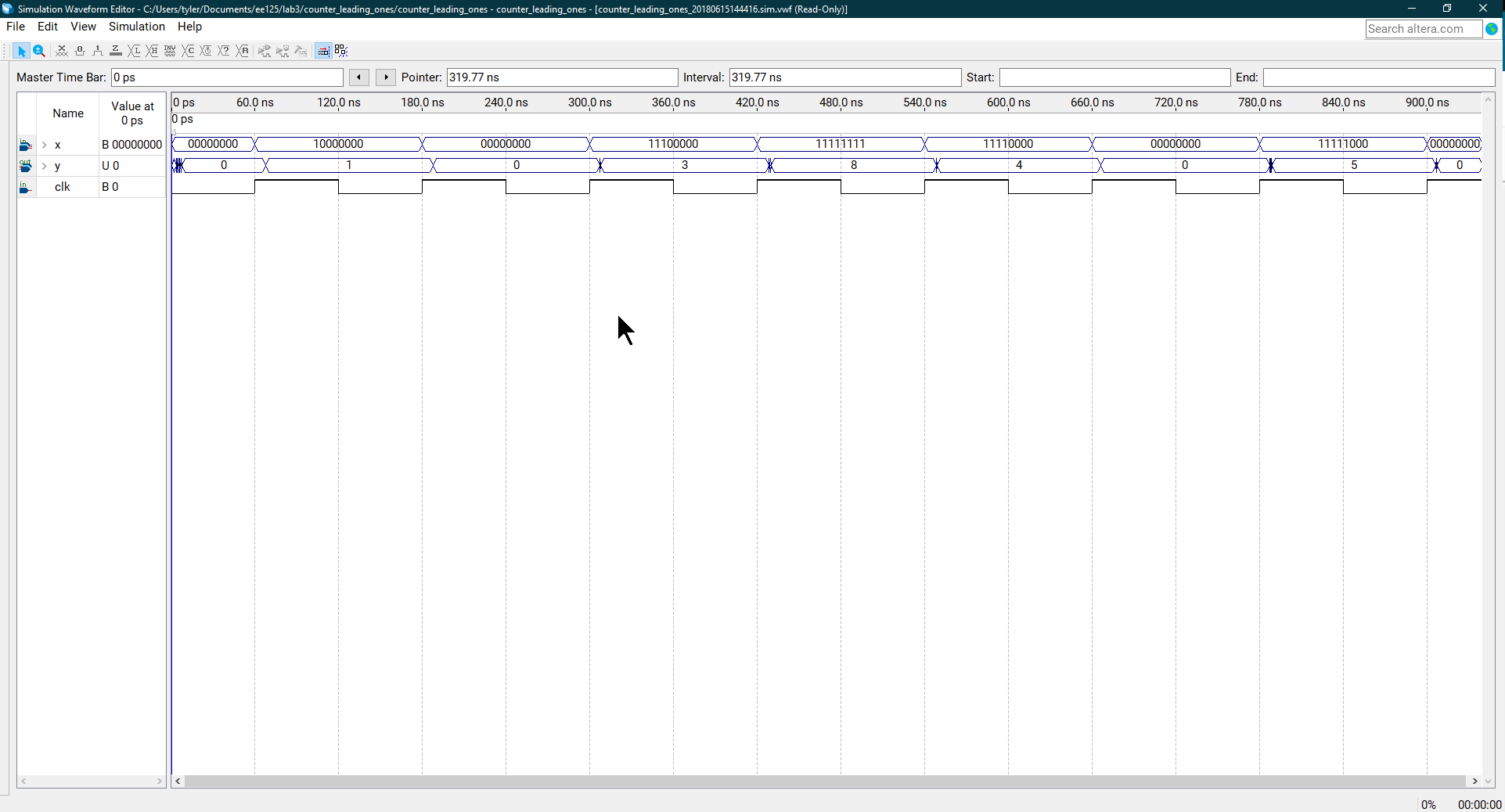
1. RTL views
2. 
3. 
4. 

Exercise 2: guess the circuit

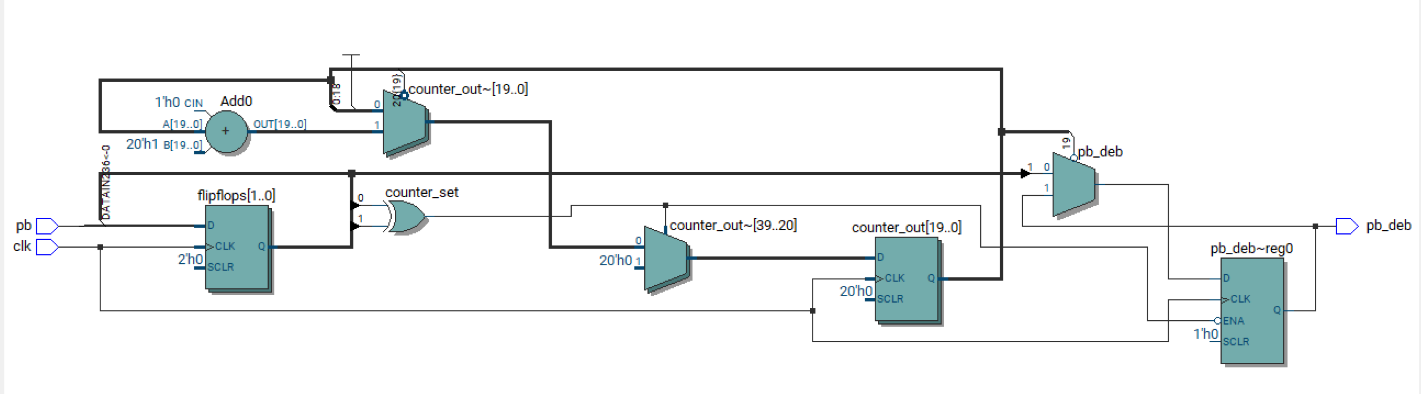
* 1. Lines 8 and 14, as it declares a variable which needs a register/flip flop to store its value. 3 for the integer variable and 1 for the output signal.
  2. Line 12, as the signal will need a register/flip flop to store its value, and line 8 as we need to maintain the value of y somewhere. 3 flip flops to store 2^3=8 numbers on line 12, plus the 1 for y and we have 4 total.

1. The hardware will not change, it will need the same number of registers.
2. In lieu of reproducing the sketch (502 disability for handwriting legibility), I know the y will be offset by one clock in code2 because signal doesn’t update till the next clock cycle unlike the variable y in code1.
3. They both take 4 registers.
4. Simulation comparison
5. 
6. 

Exercise 3: leading ones counter

1. VHDL code
   1. LIBRARY ieee;
   2. USE ieee.std\_logic\_1164.all;
   3. use ieee.numeric\_std.all;
   4. USE ieee.math\_real.all;
   5. *----------------------------*
   6. ENTITY counter\_leading\_ones IS
   7. GENERIC (
   8. *-- number of input bits*
   9. BITS: INTEGER := 8
   10. );
   11. PORT (
   12. *-- input in stl format*
   13. x: IN STD\_LOGIC\_VECTOR(BITS-1 DOWNTO 0);
   14. *--num of bits, take the log base 2, then convert to an int.*
   15. y: OUT STD\_LOGIC\_VECTOR((integer(log2(real(BITS)))) DOWNTO 0)
   16. );
   17. END ENTITY;
   18. ARCHITECTURE sequential OF counter\_leading\_ones IS
   19. BEGIN
   20. counter: PROCESS(x)
   21. variable count: UNSIGNED(y'RANGE);
   22. BEGIN
   23. count := to\_unsigned(0, y'LENGTH);
   24. FOR i IN x'RANGE LOOP
   25. CASE x(i) IS
   26. WHEN '1' => count := count + 1;
   27. WHEN OTHERS => EXIT;
   28. END CASE;
   29. END LOOP;
   30. y <= std\_logic\_vector(count);
   31. END PROCESS counter;
   32. END ARCHITECTURE;
2. Sequential was simpler by far
3. Simulation results 

Exercise 4: Switch debouncer

1. Use a DFF to check that the signal isn’t changing for T\_deb time. 
2. You can precompute the counter size (20 bits) to not need to store T\_deb in a register somewhere. Trade off isn’t that good though and could take >8 more logic elements to save a few registers.
3. 23 registers in my implementation for the dev board, varied between fpga model though.
4. VHDL code
   1. LIBRARY ieee;
   2. USE ieee.std\_logic\_1164.ALL;
   3. USE ieee.numeric\_std.ALL;
   4. USE ieee.math\_real.ALL;
   5. ENTITY switch\_debouncer IS
   6. GENERIC (
   7. *--50MHz*
   8. fclk : INTEGER := 50\_000\_000;
   9. *-- counter size (20 bits gives 20.9ms with 50MHz clock)*
   10. *-- 2^20 \* 1/(50MHz) ~= 20ms*
   11. *-- time to debounce in ms*
   12. t\_deb : INTEGER := 20
   13. );
   14. PORT (
   15. clk, pb : IN STD\_LOGIC;
   16. pb\_deb : OUT STD\_LOGIC
   17. );
   18. END ENTITY switch\_debouncer;
   19. ARCHITECTURE arch OF switch\_debouncer IS
   20. *--input flip flops*
   21. SIGNAL flipflops : STD\_LOGIC\_VECTOR(1 DOWNTO 0) := (OTHERS => '0');
   22. *--sync reset to zero*
   23. SIGNAL counter\_set : STD\_LOGIC := '0';
   24. *-- we need to divide by 1000 after multiplying t\_deb by fclk to adjust for t\_deb being in ms*
   25. SIGNAL counter\_out : UNSIGNED(INTEGER(CEIL(LOG2(REAL(t\_deb \* fclk / 1000)))) - 1 DOWNTO 0) := to\_unsigned(0, INTEGER(CEIL(LOG2(REAL(t\_deb \* fclk / 1000)))));
   26. BEGIN
   27. *--determine when to start/reset counter*
   28. counter\_set <= flipflops(0) XOR flipflops(1);
   29. PROCESS (clk)
   30. BEGIN
   31. IF rising\_edge(clk) THEN
   32. flipflops(0) <= pb;
   33. flipflops(1) <= flipflops(0);
   34. *--reset counter because input is changing*
   35. IF (counter\_set = '1') THEN
   36. counter\_out <= to\_unsigned(0, counter\_out'LENGTH);
   37. *--stable input time is not yet met*
   38. ELSIF (counter\_out(counter\_out'LEFT) = '0') THEN
   39. counter\_out <= counter\_out + 1;
   40. *--stable input time is met*
   41. ELSE
   42. pb\_deb <= flipflops(1);
   43. END IF;
   44. END IF;
   45. END PROCESS;
   46. END ARCHITECTURE;
5. Demo was shown to you in class. Below is the top level code for said demo.
   1. LIBRARY ieee;
   2. USE ieee.std\_logic\_1164.ALL;
   3. USE ieee.numeric\_std.ALL;
   4. USE ieee.math\_real.ALL;
   5. USE work.ALL;
   6. ENTITY toplevel IS
   7. GENERIC (
   8. *-- our clock frequency*
   9. fclk : INTEGER := 50\_000\_000;
   10. *-- number of input bits*
   11. BITS : INTEGER := 8;
   12. *-- time to debounce in ms*
   13. t\_deb : INTEGER := 20
   14. );
   15. PORT (
   16. clk, rst, pb : IN STD\_LOGIC;
   17. ssd\_out\_1, ssd\_out\_2: OUT STD\_LOGIC\_VECTOR(6 DOWNTO 0)
   18. );
   19. END ENTITY toplevel;
   20. ARCHITECTURE toplevel\_arch OF toplevel IS
   21. SIGNAL counter\_output\_1, counter\_output\_2 : STD\_LOGIC\_VECTOR((INTEGER(log2(real(BITS)))) DOWNTO 0);
   22. SIGNAL pb\_deb : STD\_LOGIC;
   23. BEGIN
   24. debouncer : ENTITY work.switch\_debouncer
   25. GENERIC MAP(fclk, t\_deb)
   26. PORT MAP(clk, pb, pb\_deb);
   28. counter\_1 : ENTITY work.counter(on\_falling)
   29. GENERIC MAP(BITS)
   30. PORT MAP(clk => pb\_deb, rst => rst, y => counter\_output\_1);
   32. counter\_2 : ENTITY work.counter(on\_falling)
   33. GENERIC MAP(BITS)
   34. PORT MAP(clk => pb, rst => rst, y => counter\_output\_2);
   36. ssd\_1 : ENTITY work.ssd\_interface
   37. GENERIC MAP(BITS)
   38. PORT MAP(counter\_output\_1, ssd\_out\_1);
   40. ssd\_2 : ENTITY work.ssd\_interface
   41. GENERIC MAP(BITS)
   42. PORT MAP(counter\_output\_2, ssd\_out\_2);
   43. END ARCHITECTURE toplevel\_arch;