Finite State Machine w/ Data Path

Last updated 9/24/20
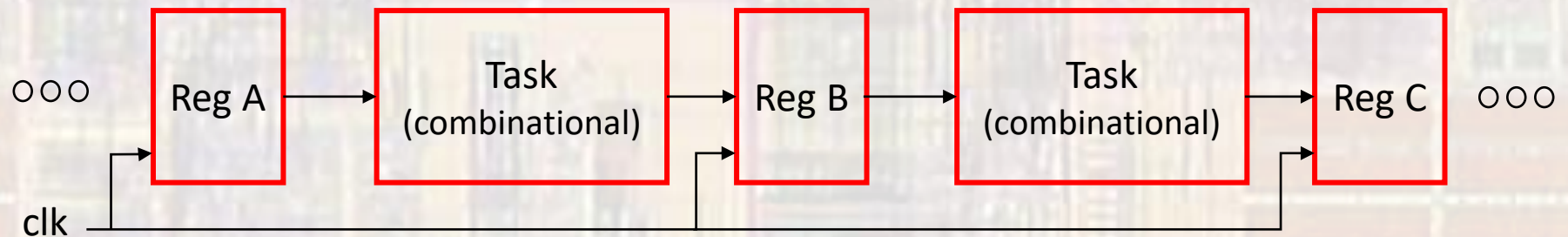
FSMD

These slides review FSM's with Datapath

Upon completion: You should be able to design and simulate FSM's with Datapath
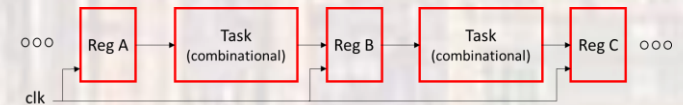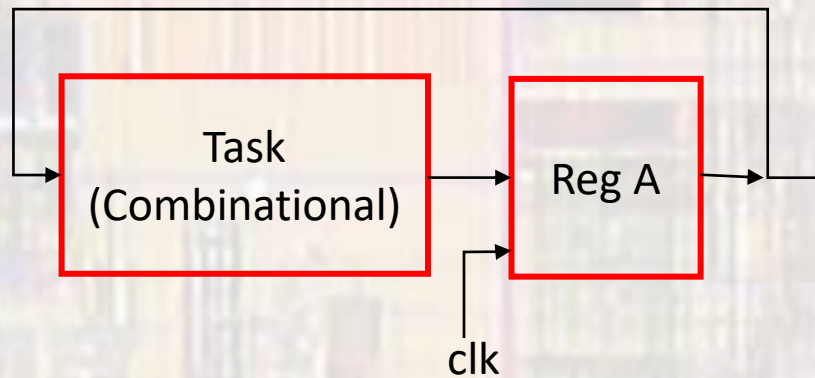
FSMD

- FSMD

  - FSM with Data Path

  - FSM
    - Control the Data Path

  - Data Path
    - Sequential circuitry to accomplish a task
    - Typically register transfer operations

FSMD

- FSMD

- FSM with Data Path

- Data Path
  - Sequential circuitry to accomplish a task
  - Typically register transfer operations

FSMD

- FSMD

Data Path

inputs → Task (Combinational) → Reg A → outputs

clk

Next State Logic

inputs

Register (D-FFs)

Output Logic → outputs

clk

Control Path

FSMD

- Fibonacci – FSMD

$$fib(i) = \begin{cases} 0 & i = 0 \\ 1 & i = 1 \\ fib(i-1) + fib(i-2) & i > 1 \end{cases}$$
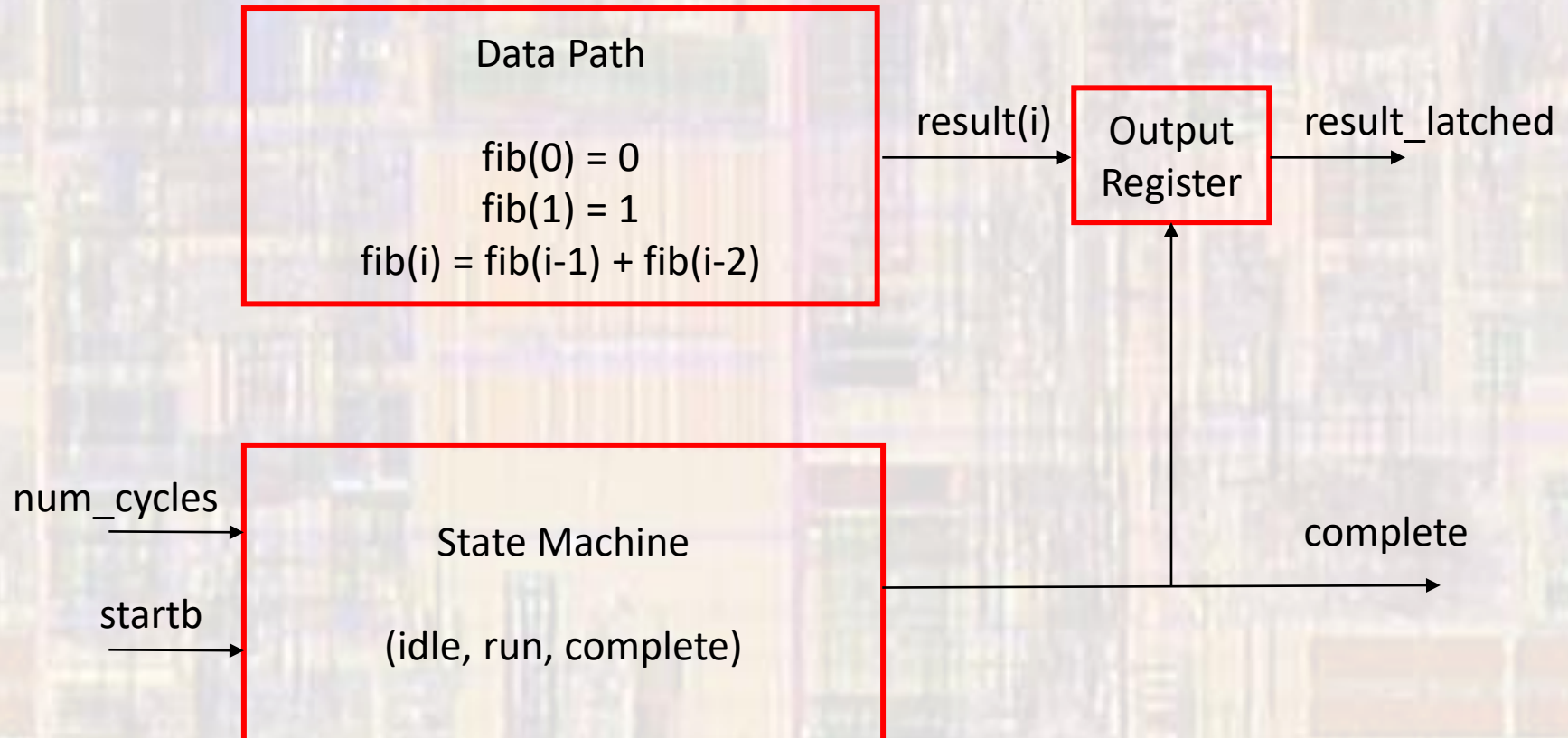
for i= 6
0-1-1-2-3-5-8

FSMD

- Fibonacci – FSMD

  - Input Signals
    - clk, rstb
    - num_cycles – how many numbers to generate - vector
    - startb – start generating (bar)

  - Output Signals
    - complete – generation complete
    - result_latched – Final Fibonacci value

  - States
    - Idle
    - Run
    - Complete

FSMD

- Fibonacci – FSMD

Data Path

fib(0) = 0
fib(1) = 1
fib(i) = fib(i-1) + fib(i-2)

result(i) →

Output
Register

→ result_latched

num_cycles →

State Machine

(idle, run, complete)

startb →

complete →

FSMD

- Fibonacci – FSMD

Data Path

$fib(0) = 0$
$fib(1) = 1$
$fib(i) = fib(i-1) + fib(i-2)$

result(i) → **Output Register** → result_latched

calc

num_cycles →

State Machine

(idle, run, complete)

startb →

complete

FSMD

- Fibonacci – FSMD

  - Control Path

idle

calc ← 0
complete ← 1

startb = 0   F

T

cnt ←
num_cycles

run

calc ← 1
complete ← 0

cnt = 1   F

T

cnt ← cnt - 1

complete

calc ← 0
complete ← 1

FSMD

- Fibonacci – FSMD

  - Data Path

Diagram: A decision block `calc = 0` with branches T (true) and F (false). On the T branch: `fib(i-2) ← 0`, `fib(i-1) ← 1`. On the F branch: `fib(i-1) ← fib(i-1) + fib(i-2)`, `fib(i-2) ← fib(i-1)`.

FSMD

- Fibonacci – FSMD

idle

```
calc ← 0
complete ← 1
```

startb = 0   F

T

cnt ←
num_cycles

run

```
calc ← 1
complete ← 0
```

Cnt = 1   F

T

cnt ← cnt - 1

complete

```
calc ← 0
complete ← 1
```

calc

T   calc = 0   F

```
fib(i-2) ← 0
fib(i-1) ← 1
```

```
fib(i-1) ← fib(i-1)
      + fib(i-2)
fib(i-2) ← fib(i-1)
```

FSMD

- Fibonacci – FSMD - FSM

```
--
-- fibonacci_fsm.vhdl
--
-- created 4/16/17
-- tj
--
-- rev 0
-----------------------------------------
--
-- FSM for fibonacci
-----------------------------------------
--
-- Inputs: rstb, clk, startb, num_cycles
-- Outputs: complete, calc
--
-- NOTE: N is the width of the num_cycles bus
-- 2**N specifies the maximum # of iterations
--
-----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fibonacci_fsm is
    generic(
        N: natural := 5
    );
    port (
        i_clk:          in std_logic;
        i_rstb:         in std_logic;
        i_startb:       in std_logic;
        i_num_cycles:   in std_logic_vector((N - 1) downto 0);

        o_calc:         out std_logic;
        o_complete :    out std_logic
    );
end entity;
```

```
architecture behavioral of fibonacci_fsm is
    --
    -- State Types
    --
    type STATE_TYPE is (Idle, Run, Finish);
    signal state:           STATE_TYPE;
    signal state_next:      STATE_TYPE; --

    --
    -- FSM signals
    --
    signal cnt:             unsigned((N-1) downto 0);

begin
    --
    -- next state logic
    --
    process(all)
    begin
        case state is
            when Idle=>
                if(i_startb = '0') then
                    state_next <= Run;
                else
                    state_next <= Idle;
                end if;
            when Run =>
                if(cnt = 1) then
                    state_next <= Finish;
                else
                    state_next <= Run;
                end if;
            when others =>
                state_next <= Idle;
        end case;
    end process;

    --
    -- State Register logic
    --
    process(i_clk, i_rstb)
    begin
        -- reset
        if (i_rstb = '0') then
            state <= Idle;
        -- rising clk edge
        elsif (rising_edge(i_clk)) then
            state <= state_next;
        end if;
    end process;
```

FSMD

- Fibonacci – FSMD - FSM

```vhdl
--
-- counter logic
--
process(i_clk, i_rstb)
begin
    -- reset
    if (i_rstb = '0') then
        cnt <= (others => '0');
    elsif (i_startb = '0') then
        cnt <= unsigned(i_num_cycles);
    -- rising clk edge
    elsif (rising_edge(i_clk)) then
        cnt <= cnt - 1;
    end if;
end process;
```

```vhdl
--
-- Output logic
--
process(all)
begin
    case state is
        when Idle=>
            o_complete <= '1';
            o_calc <= '0';
        when Run =>
            o_complete <= '0';
            o_calc <= '1';
        when Finish =>
            o_complete <= '1';
            o_calc <= '0';
        when others =>
            o_complete <= '0';
            o_calc <= '0';
    end case;
end process;

end behavioral;
```

FSMD

- Fibonacci – FSMD - data path

```vhdl
---------------------------------------
--
-- fibonacci_datapath.vhdl
--
-- created 4/16/17
-- tj
--
-- rev 0
---------------------------------------
--
-- fibonacci datapath
--
-- Limited to N values
---------------------------------------
--
-- Inputs:  clk, start, count
-- Outputs: fib
--
-- NOTE: N is the width of the num_cycles bus
-- 2**N specifies the maximum # of iterations
--
-- NOTE: The width of the result is calculated from
-- using an equation for the 2**Nth value
--
---------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity fibonacci_datapath is
    generic(
            N: natural := 5
        );
    port (
        i_clk:   in std_logic;
        i_calc:  in std_logic;

        o_fib :  out std_logic_vector((((integer(ceil(log2(((((1.0 + sqrt(5.0))/2.0)**(2**(N-1)) - ((1.0 - sqrt(5.0))/2.0)**(2**(N-1)))/sqrt(5.0)))))) - 1) downto 0)
    );
end entity;
```

FSMD

- Fibonacci – FSMD - data path

```vhdl
architecture behavioral of fibonacci_datapath is
    constant result_w: integer:= integer(ceil(log2(((((1.0 + sqrt(5.0))/2.0)**(2**(N-1)) - ((1.0 - sqrt(5.0))/2.0)**(2**(N-1)))/sqrt(5.0)))));
    -- data path signals
    signal f_iminus1:        unsigned((result_w - 1) downto 0);
    signal f_iminus1_next:   unsigned((result_w - 1) downto 0);
    signal f_iminus2:        unsigned((result_w - 1) downto 0);
    signal f_iminus2_next:   unsigned((result_w - 1) downto 0);

begin
    --
    -- data path - D next state logic
    --
    process(all)
    begin
        if (i_calc = '0') then
            f_iminus2_next <= (others => '0');
            f_iminus1_next <= (to_unsigned(1,result_w));
        else
            f_iminus2_next <= f_iminus1;
            f_iminus1_next <= f_iminus1 + f_iminus2;
        end if;
    end process;

    --
    -- Datapath register logic
    --
    process(i_clk)
    begin
        -- rising clk edge
        if (rising_edge(i_clk)) then
            f_iminus2 <= f_iminus2_next;
            f_iminus1 <= f_iminus1_next;
        end if;
    end process;

    --
    -- Output logic
    --
    o_fib <= std_logic_vector(f_iminus1);

end behavioral;
```

FSMD

- Fibonacci – FSMD

```vhdl
----------------------------------------
--
-- fibonacci_fsmd.vhdl
--
-- created 9/7/18
-- tj
--
-- rev 0
----------------------------------------
--
-- FSMD for fibonacci
--
----------------------------------------
--
-- Inputs: rstb, clk, startb, num_cycles
-- Outputs: result_latched, complete
--
-- NOTE: N is the width of the num_cycles bus
-- 2**N specifies the maximum # of iterations
--
-- NOTE: The width of the result is calculated from
-- using an equation for the 2**Nth value
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity fibonacci_fsmd is
    generic(
        N: natural := 6
    );
    port (
        i_clk             : in std_logic;
        i_rstb            : in std_logic;
        i_startb          : in std_logic;
        i_num_cycles      : in std_logic_vector((N - 1) downto 0);

        o_result_latched  : out std_logic_vector(((integer(ceil(log2((((1.0 + sqrt(5.0))/2.0)**(2**(N-1)) - ((1.0 - sqrt(5.0))/2.0)**(2**(N-1)))/sqrt(5.0))))) - 1) downto 0);
        o_complete        : out std_logic
    );
end entity;
```

FSMD

- Fibonacci – FSMD

```vhdl
architecture behavioral of fibonacci_fsmd is
    --
    -- structural signals
    --
    signal calc:        std_logic;
    signal result:      std_logic_vector((((integer(ceil(log2((((1.0 + sqrt(5.0))/2.0)**(2**(N-1)) - ((1.0 - sqrt(5.0))/2.0)**(2**(N-1)))/sqrt(5.0))))) - 1) downto 0);

    ---------------------------------
    -- Component prototype
    ---------------------------------
    COMPONENT fibonacci_fsm
        generic(
        N: natural := 5
        );
        port  (
            i_clk:          in std_logic;
            i_rstb:         in std_logic;
            i_startb:       in std_logic;
            i_num_cycles:   in std_logic_vector((N - 1) downto 0);

            o_calc:         out std_logic;
            o_complete :    out std_logic
        );
    END COMPONENT;

    COMPONENT fibonacci_datapath
    generic(
            N: natural := 5
        );
    port  (
        i_clk:   in std_logic;
        i_calc:  in std_logic;

        o_fib :  out std_logic_vector((((integer(ceil(log2((((1.0 + sqrt(5.0))/2.0)**(2**(N-1)) - ((1.0 - sqrt(5.0))/2.0)**(2**(N-1)))/sqrt(5.0))))) - 1) downto 0)
    );
    END COMPONENT;
```
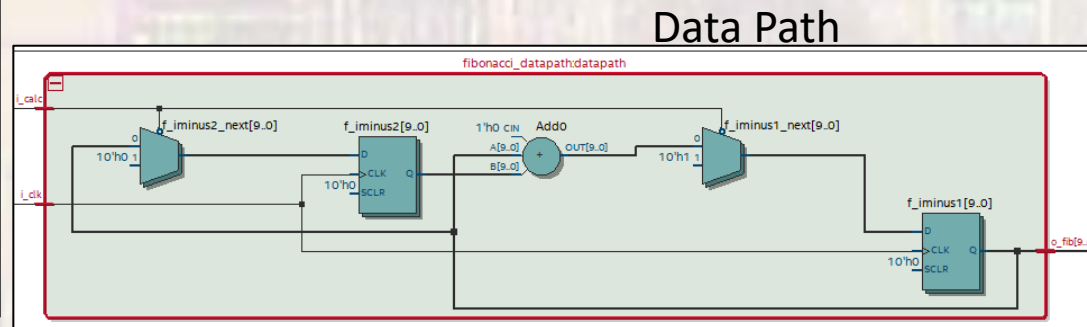
FSMD

- Fibonacci – FSMD

```vhdl
begin
    -------------------------------------
    -- Device instantiations
    -------------------------------------
    fsm: fibonacci_fsm
        generic map(
            N => N
            )
        port map(
            i_clk              => i_clk,
            i_rstb             => i_rstb,
            i_startb           => i_startb,
            i_num_cycles       => i_num_cycles,
            o_calc             => calc,
            o_complete         => o_complete
    );

    datapath: fibonacci_datapath
        generic map(
            N => N
            )
        port map(
            i_clk        => i_clk,
            i_calc       => calc,
            o_fib        => result
    );

    --
    -- latch the result so it doesnt reset
    --
    process(i_clk, i_rstb)
    begin
        if(i_rstb = '0') then
            o_result_latched <= (others => '0');
        elsif(rising_edge(i_clk)) then
            if (o_complete = '0') then
                o_result_latched <= result;
            end if;
        end if;
    end process;

end behavioral;
```
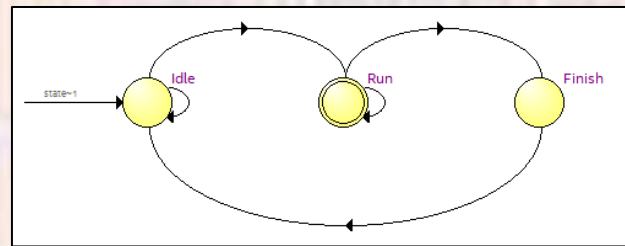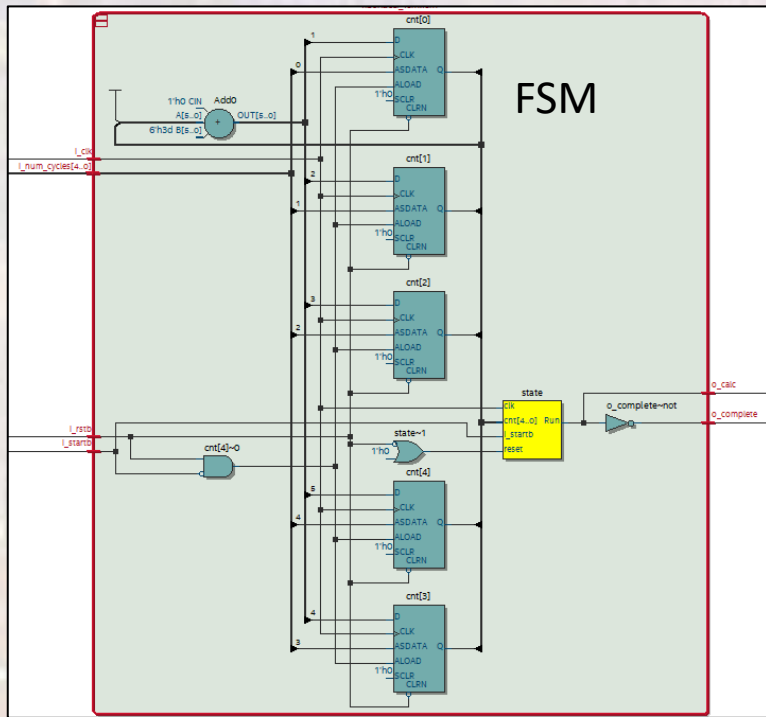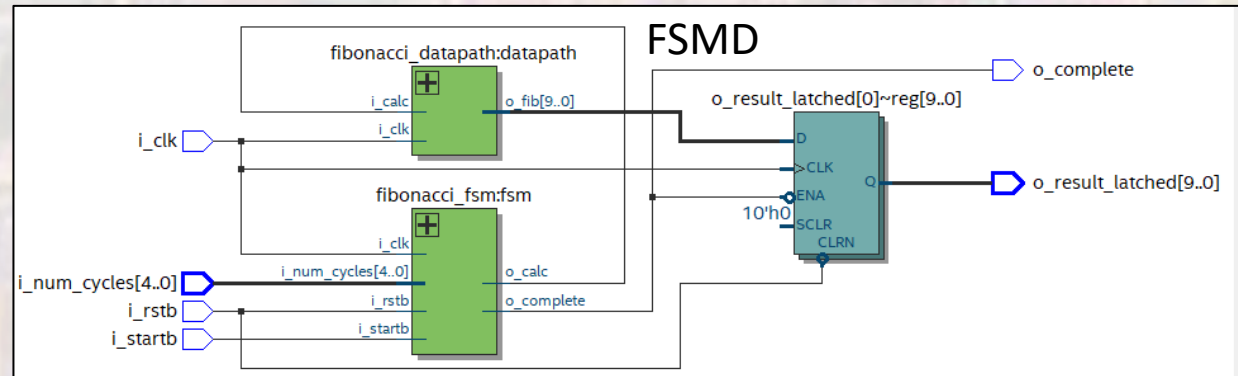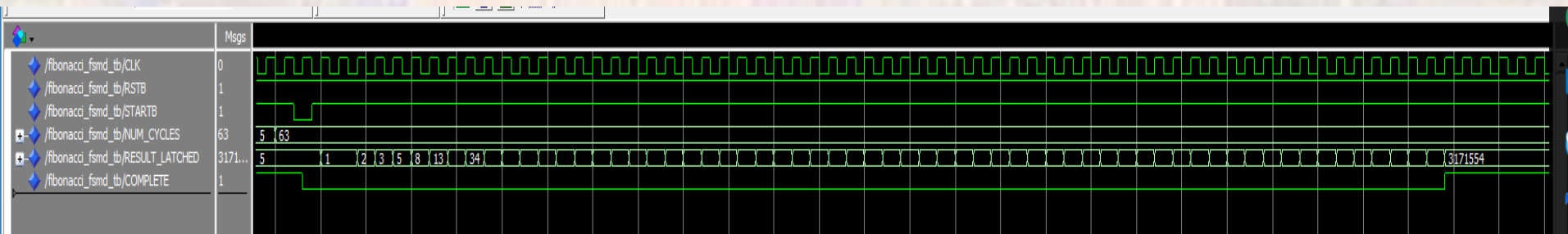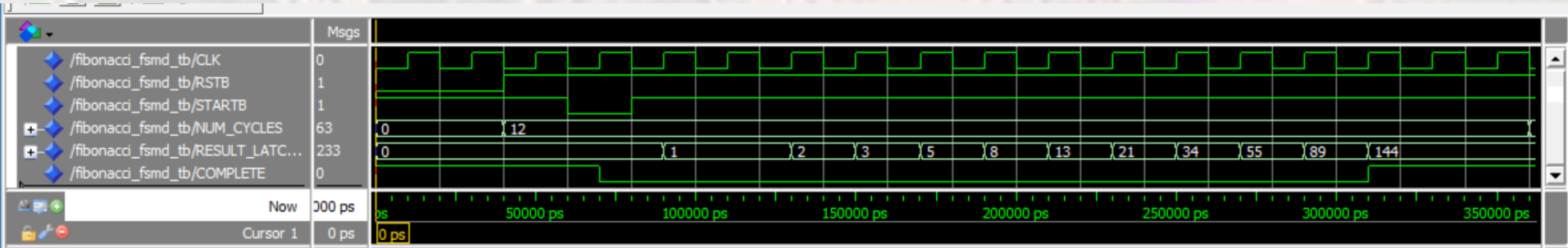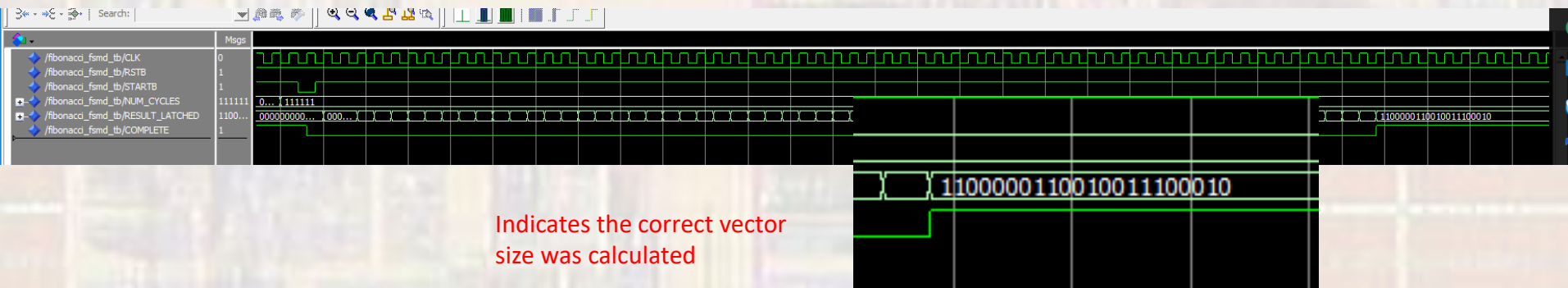
FSMD

- Fibonacci

FSMD

- Fibonacci

Results and intermediate values

N = 6

Indicates the correct vector
size was calculated

FSMD

- Multiplication
  - Shift and add

```
    1110              11111110                 11111110
  x 0011            x 00000011               x 00000011
                                              11111110
                                              11111110
                                              00000000
                                              00000000
                                              00000000
                                              00000000
                                              00000000
                                              00000000
                                              11111010
```
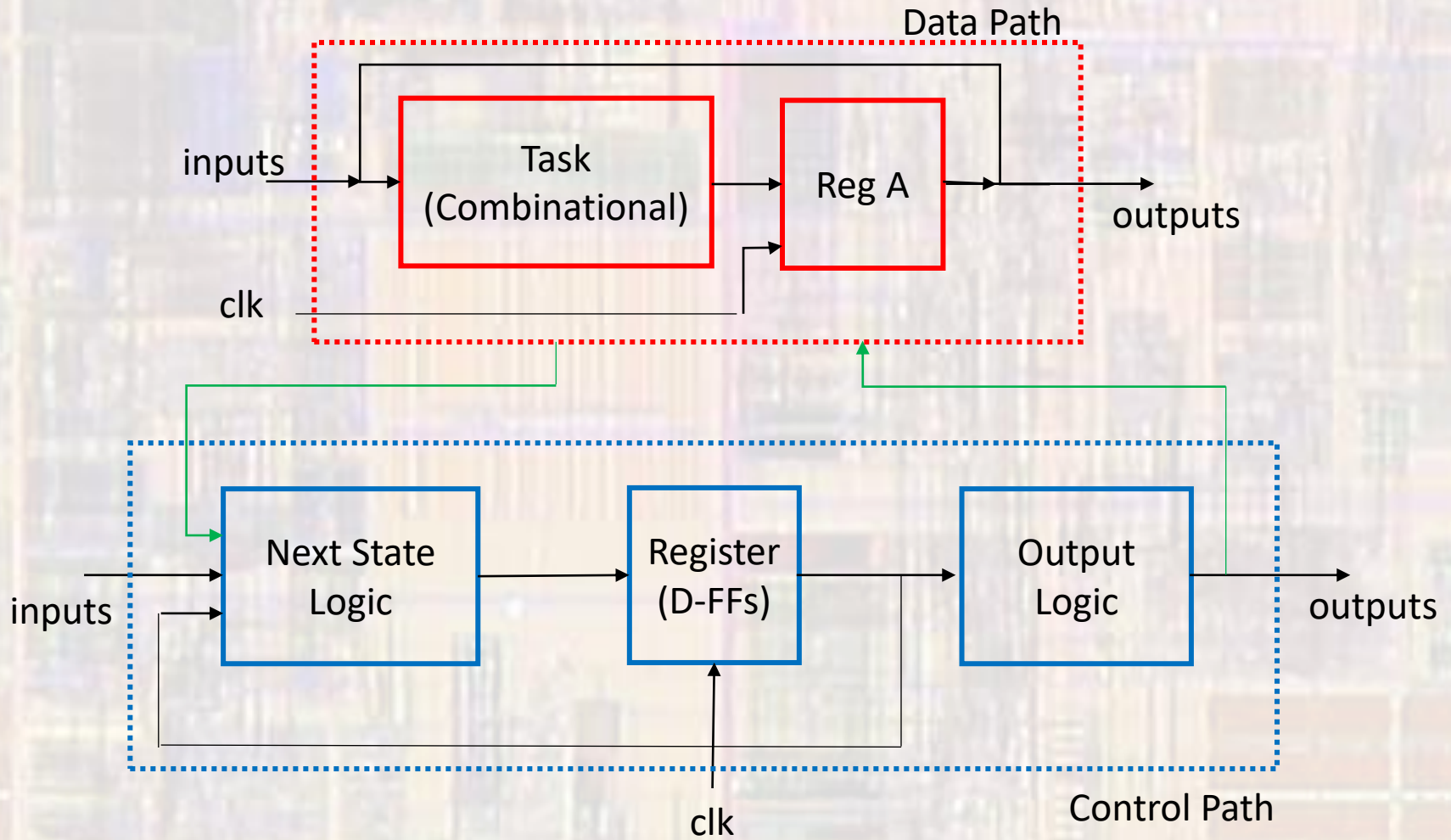
FSMD

- Multiplication
  - Want a process that is repetitive
    - Repetitive addition multiplication

      5 x 4 = 5 + 5 + 5 + 5   i.e. the multiplicand added the multiplier times

    - Requires integers – or non-fractional binary numbers for the multiplier
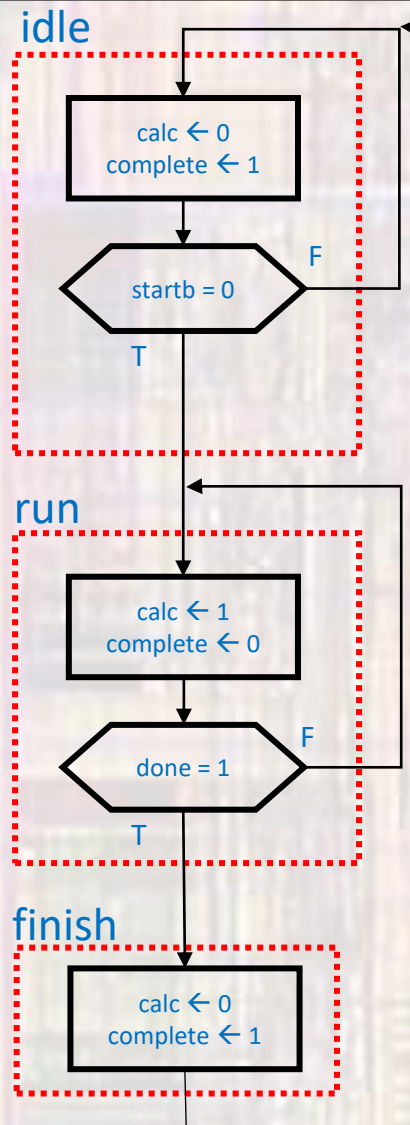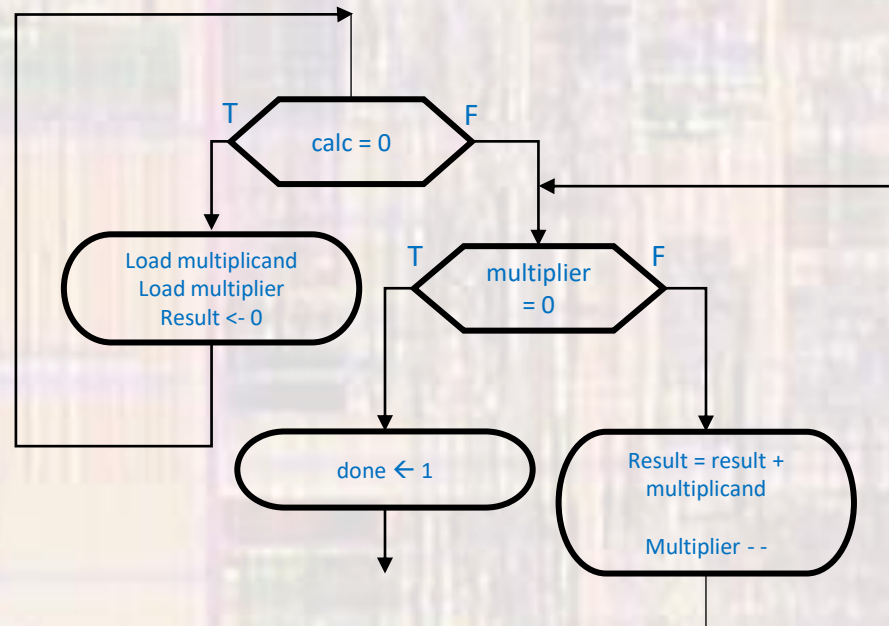
FSMD

- Multiplier

FSMD

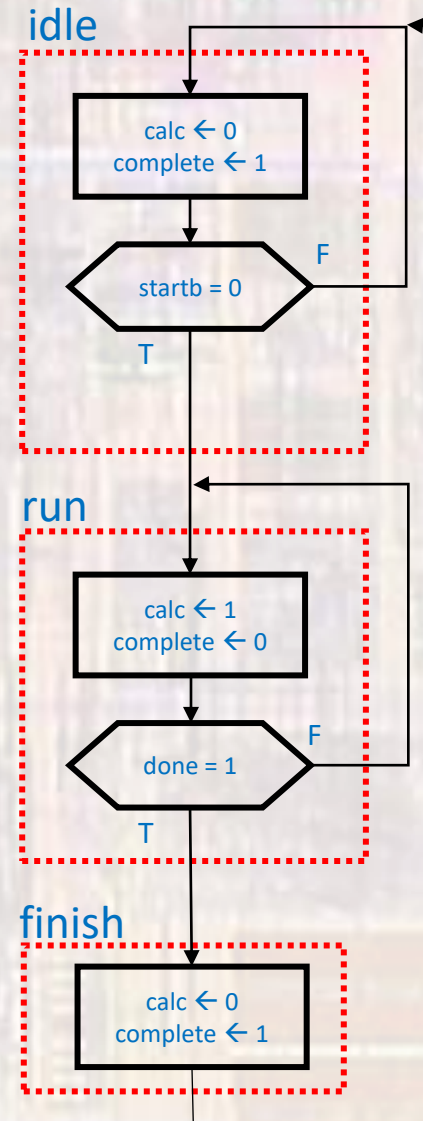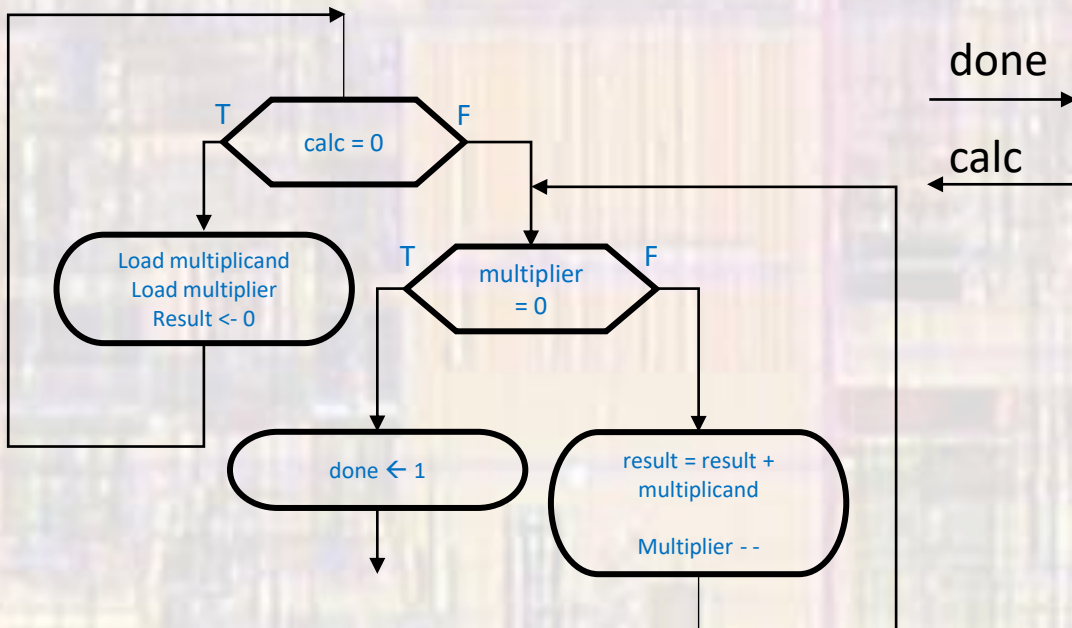- Multiplier – FSMD

  - Control Path

idle

calc ← 0
complete ← 1

startb = 0   F

T

run

calc ← 1
complete ← 0

done = 1   F

T

finish

calc ← 0
complete ← 1

FSMD

- Multiplier – FSMD

  - Data Path

FSMD

- Multiplier – FSMD

idle

calc ← 0
complete ← 1

startb = 0   F

T

run

calc ← 1
complete ← 0

done = 1   F

T

finish

calc ← 0
complete ← 1

done

calc

T   calc = 0   F

Load multiplicand
Load multiplier
Result <- 0

T   multiplier
= 0   F

done ← 1

result = result +
multiplicand

Multiplier - -

FSMD

- Multiplier – FSMD - FSM

```
------------------------------------------
--
-- mult_rep_add_4bit_fsm.vhdl
--
-- created 9/12/18
-- tj
--
-- rev 0
------------------------------------------
--
-- FSM for repetitive addition multiplier
-- tells the datapath when to calculate (calc) and uses
-- done from the datapath to know its complete
------------------------------------------
--
-- Inputs: rstb, clk, startb, done
-- Outputs: calc, complete
--
------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mult_rep_add_4bit_fsm is
    port (
            i_clk           :  in std_logic;
            i_rstb          :  in std_logic;
            i_startb        :  in std_logic;
            i_done          :  in std_logic;

            o_calc          :  out std_logic;
            o_complete      :  out std_logic
    );
end entity;
```

```
architecture behavioral of mult_rep_add_4bit_fsm is
    --
    -- State Types
    --
    type STATE_TYPE is (Idle, Run, Finish);
    signal state:         STATE_TYPE;
    signal state_next:    STATE_TYPE;

begin
    --
    -- next state logic
    --
    process(all)
    begin
        case state is
            when Idle=>
                if(i_startb = '0') then
                    state_next <= Run;
                else
                    state_next <= Idle;
                end if;
            when Run =>
                if(i_done = '1') then
                    state_next <= Finish;
                else
                    state_next <= Run;
                end if;
            when Finish =>
                state_next <= Idle;
        end case;
    end process;
```

FSMD

- Multiplier – FSMD - FSM

```
--
-- State Register logic
--
process(i_clk, i_rstb)
begin
    -- reset
    if (i_rstb = '0') then
        state <= Idle;
    -- rising clk edge
    elsif (rising_edge(i_clk)) then
        state <= state_next;
    end if;
end process;
```

```
--
-- Output logic
--
process(all)
begin
    case state is
        when Idle=>
            o_complete <= '1';
            o_calc <= '0';
        when Run =>
            o_complete <= '0';
            o_calc <= '1';
        when Finish =>
            o_complete <= '1';
            o_calc <= '0';
    end case;
end process;
end behavioral;
```

FSMD

- Multiplier – FSMD – Data path

```vhdl
-------------------------------------------
--
-- mult_rep_add_4bit_datapath.vhdl
--
-- created 9/12/18
-- tj
--
-- rev 0
-------------------------------------------
--
-- Data Path for repetitive addition multiplier
-- adds the multiplcand multiplier number of times
-------------------------------------------
--
-- Inputs: clk, start, multiplicand, multiplier
-- Outputs: result, complete
--
-------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mult_rep_add_4bit_datapath is
    port (
        i_clk             : in std_logic;
        i_multiplicand    : in std_logic_vector(3 downto 0);
        i_multiplier      : in std_logic_vector(3 downto 0);
        i_calc            : in std_logic;

        o_result          : out std_logic_vector(7 downto 0);
        o_done            : out std_logic
    );
end entity;
```

```vhdl
architecture behavioral of mult_rep_add_4bit_datapath is
    --
    -- structural signals
    --
    signal multiplicand_sig:     unsigned(3 downto 0);
    signal Multiplier_sig:       unsigned(3 downto 0);
    signal Multiplier_sig_next:  unsigned(3 downto 0);
    signal result_sig:           unsigned(7 downto 0);
    signal result_sig_next:      unsigned(7 downto 0);
    signal zero_fill:            unsigned(7 downto 0);

begin
    --
    -- datapath registers
    --
    process(i_clk)
    begin
        if (rising_edge(i_clk)) then
            if(i_calc = '0') then
                multiplicand_sig <= unsigned(i_multiplicand);
                multiplier_sig <= unsigned(i_multiplier);
                result_sig <= (others => '0'); --result_sig;
            else
                multiplicand_sig <= unsigned(i_multiplicand);
                multiplier_sig <= multiplier_sig_next;
                result_sig <= result_sig_next;
            end if;
        end if;
    end process;
```

FSMD

- Multiplier – FSMD – Data path

```vhdl
--
-- next state logic
--
process(all)
begin
    if(multiplier_sig = 0) then
        multiplier_sig_next <= multiplier_sig;
        result_sig_next <= result_sig;
        o_done <= '1';
    else
        multiplier_sig_next <= multiplier_sig - 1;
        result_sig_next <= ("0000" & multiplicand_sig) + result_sig;
        o_done <= '0';
    end if;
end process;

--
-- output logic
--
o_result <= std_logic_vector(result_sig);

end behavioral;
```

FSMD

- Multiplier – FSMD

```
---------------------------------
--
-- mult_rep_add_4bit_fsmd.vhdl
--
-- created 9/7/18
-- tj
--
-- rev 0
---------------------------------
--
-- FSMD for repetitive addition multiplier
-- adds the multiplcand multiplier number of times
---------------------------------
--
-- Inputs: rstb, clk, start, multiplicand, multiplier
-- Outputs: result, complete
--
---------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mult_rep_add_4bit_fsmd is
   port (
        i_clk           : in std_logic;
        i_rstb          : in std_logic;
        i_startb        : in std_logic;
        i_multiplicand  : in std_logic_vector(3 downto 0);
        i_multiplier    : in std_logic_vector(3 downto 0);

        o_result_latched : out std_logic_vector(7 downto 0);
        o_complete       : out std_logic
   );
end entity;
```

```
architecture behavioral of mult_rep_add_4bit_fsmd is
   --
   -- structural signals
   --
   signal calc:          std_logic;
   signal done:          std_logic;
   signal complete_sig:  std_logic;
   signal result:        std_logic_vector(7 downto 0);

   ---------------------------------
   -- Component prototype
   ---------------------------------
   COMPONENT mult_rep_add_4bit_fsm
      PORT(
           i_clk      :   IN STD_LOGIC;
           i_rstb     :   IN STD_LOGIC;
           i_startb   :   IN STD_LOGIC;
           i_done     :   IN STD_LOGIC;
           o_calc     :   OUT STD_LOGIC;
           o_complete :   OUT STD_LOGIC
      );
   END COMPONENT;

   COMPONENT mult_rep_add_4bit_datapath
      port (
           i_clk           :  in std_logic;
           i_calc          :  in std_logic;
           i_multiplicand  :  in std_logic_vector(3 downto 0);
           i_multiplier    :  in std_logic_vector(3 downto 0);

           o_done          :  out std_logic;
           o_result        :  out std_logic_vector(7 downto 0)
      );
   END COMPONENT;
```
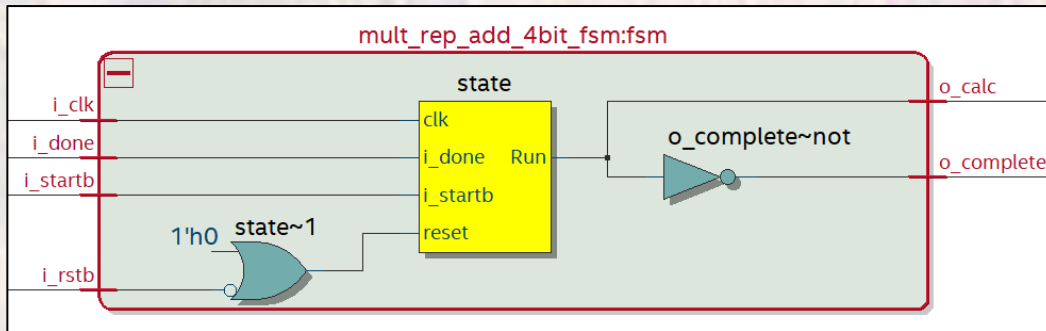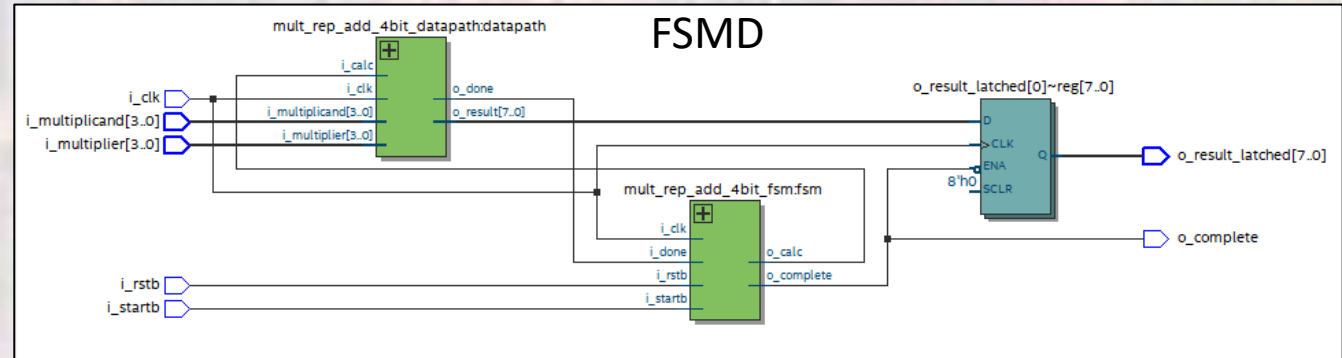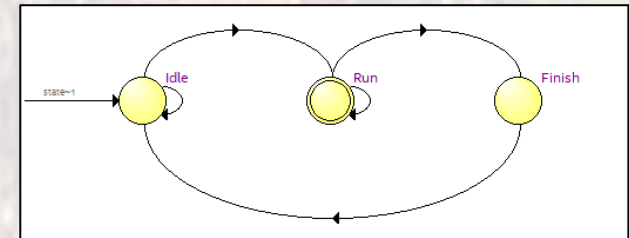
FSMD

- Multiplier – FSMD

```
----------------------------------------
-- Device instantiations
----------------------------------------
fsm: mult_rep_add_4bit_fsm
    port map(
        i_clk        => i_clk,
        i_rstb       => i_rstb,
        i_startb     => i_startb,
        i_done       => done,
        o_calc       => calc,
        o_complete   => complete_sig
    );

datapath: mult_rep_add_4bit_datapath
    port map(
        i_clk             => i_clk,
        i_calc            => calc,
        i_multiplicand    => i_multiplicand,
        i_multiplier      => i_multiplier,
        o_done            => done,
        o_result          => result
    );

--
-- latch the result so it doesnt reset
--
process(i_clk)
begin
    if(rising_edge(i_clk)) then
        if (complete_sig = '0') then
            o_result_latched <= result;
        end if;
    end if;
end process;

--
-- output logic
--
o_complete <= complete_sig;

end behavioral;
```
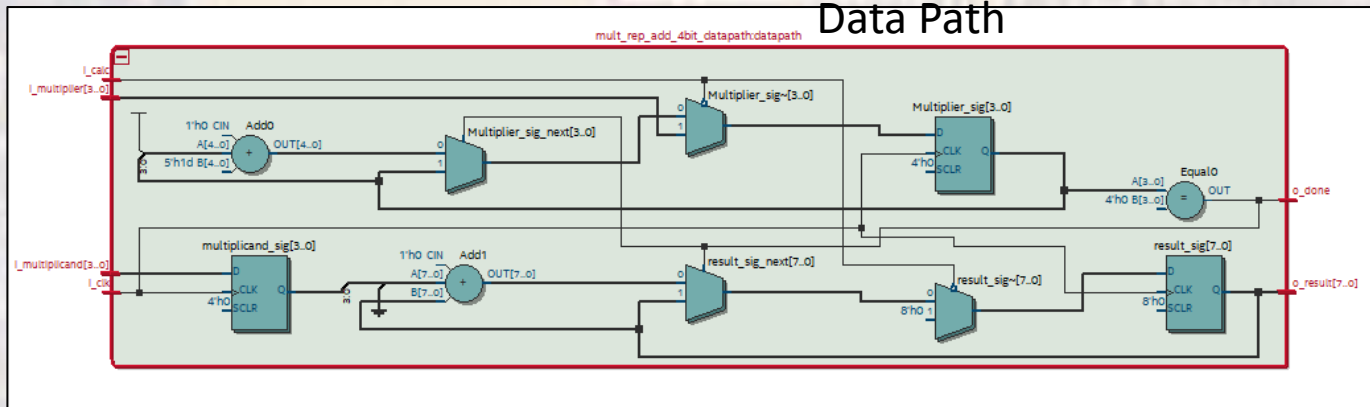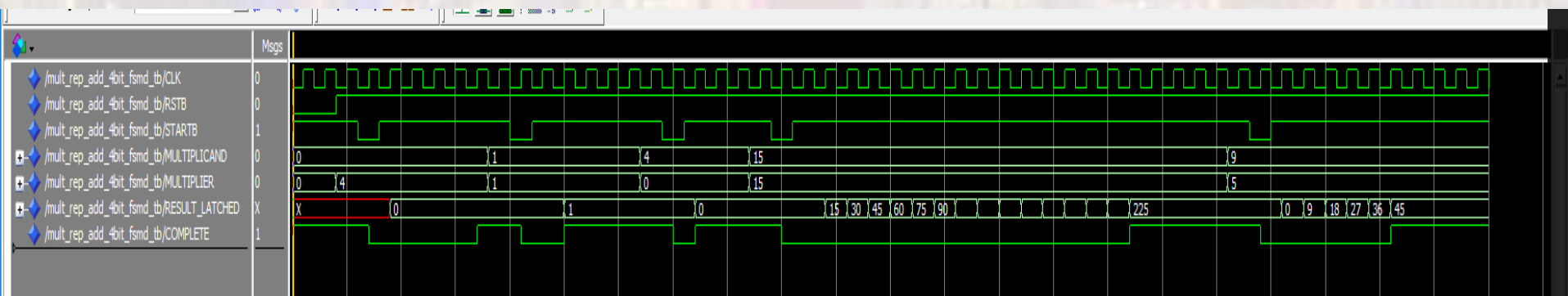
FSMD

- ## Multiplier

FSMD

FSM

Data Path

FSMD

- Multiplier - FSMD

Results and intermediate values

Note: Takes "multiplier" number of clock cycles
Takes a variable number of clock cycles