# Digital System Design

by
Dr. Lesley Shannon
Email: lshannon@ensc.sfu.ca
Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc350

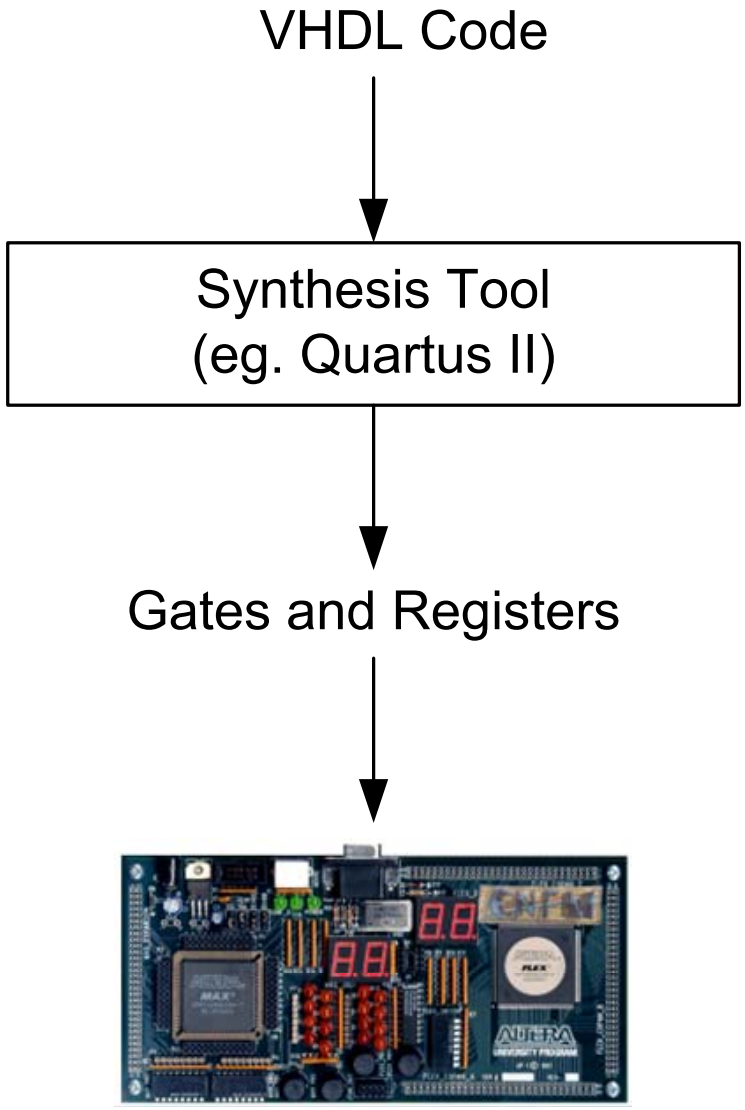
***Simon Fraser University***

Slide Set: 3
Date: January 19, 2009

# Slide Set Overview

- Making your VHDL synthesizable
  - If you learn VHDL from a book, write your code using the constructs they describe, and try to compile it to an hardware, it probably won't work!
    - Why?
    - I'll tell you in this slide set, and also talk about how to make sure it does work.
  - This is probably the most important slide set of all!
    - It will save you tons of debugging time in the lab if you understand it

# From HDL to Gates



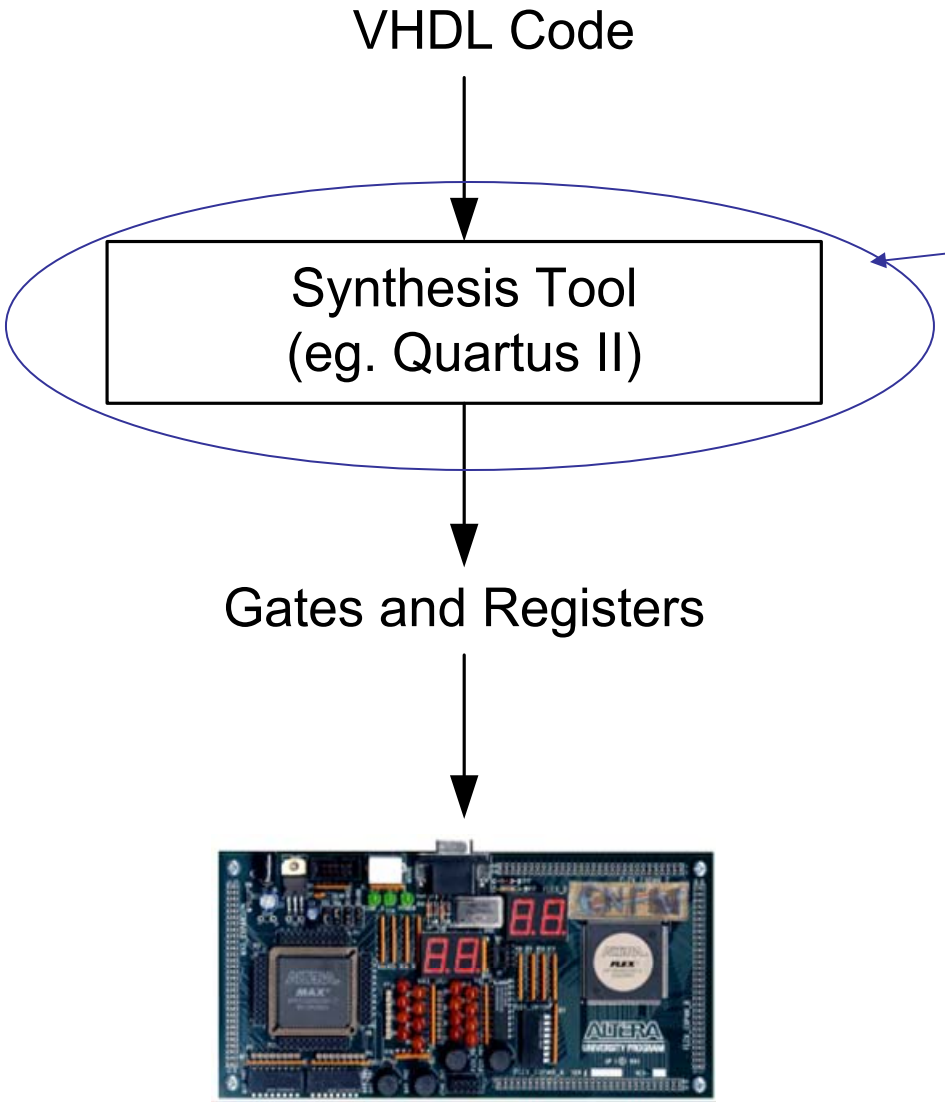

The synthesis tool "compiles" the VHDL into gates. Makes hardware out of VHDL code

The gates are implemented on a chip (in our case, on an FPGA)

# If you only learn one thing during this course

When you are describing hardware in VHDL, you are only describing the behaviour. The actual circuit will be synthesized (by the tools, e.g. Quartus II) to gates. The FPGA then implements the gates. The FPGA does ***NOT*** execute the VHDL code directly!!!!

# Unsynthesizable HDL




What do you think happens if the synthesis tool can not make hardware for your VHDL?

**The gates will not implement the behaviour you specified…**

**Your circuit won't implement the behaviour you specified**

# “Synthesizable” VHDL

- Not all VHDL Code can be synthesized by current tools
  - This isn’t limited to Altera’s tools
- Synthesizable VHDL is a subset of VHDL that can be synthesized by current tools
- If you write VHDL that is not synthesizable:
  - Tools will not be able to create hardware
  - Sometimes it will try, but end up with something that is “not quite right”
    - Sometimes it gives an error message, sometimes not

MORAL: If you are going to synthesize, always write Synthesizable VHDL!

# But what sort of VHDL is Synthesizable?

- In general, it depends a bit on the tools
  - RTL-level Synthesis Tools: Fairly small subset it can handle
  - Behavioural Synthesis Tools: larger subset

- In the next few slides, I am going to show you the minimum set that is synthesizable by all tools. If you restrict your VHDL to this small set, your code will be synthesizable by all tools

# Synthesizable processes

- To make sure your VHDL is synthesizable, every process must be one of ***THREE*** types:
  - Type 1: Purely Combinational – All outputs are a function only of the current inputs (not on the previous inputs)

```
process(SEL, A, B)
begin
    if (SEL = '0') then
            Y <= A;
    else
            Y <= B;
    end if;
end process;
```

# For purely combinational processes

**Rule 1:** Every input (that can affect the output(s)) must be in the sensitivity list.

**Rule 2:** Every output must be assigned a value for every possible combination of the inputs

# Examples of not synthesizable code

This would *not* be synthesizable (violates Rule 1):

```
process (A, B)
  begin
    if (SEL = ‘0’) then
      Y <= A;
    else
      Y <= B;
    end if;
  end process;
```

This would *not* be synthesizable (violates Rule 2):

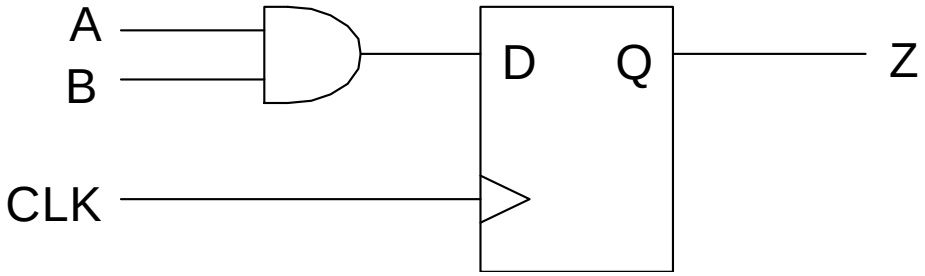
```
process (SEL, A, B)
begin
    if (SEL = ‘0’) then
        Y <= A;
    end if;
end process;
```

# Synthesizable Processes

**Type 2:** Purely Sequential: Each output changes *only* on the rising *or* falling edge of a single clock

```
process (CLK)
 begin
  if (CLK'event and CLK='1') then
      Z <= A and B;
  end if;
end process;
```

optional



# For purely sequential processes

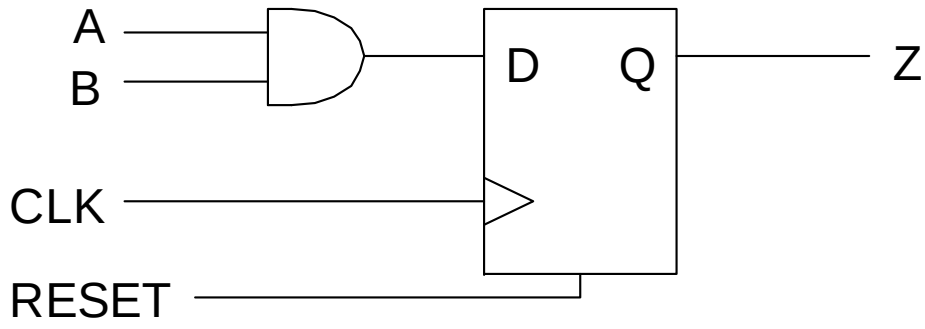
**Rule 1:** Only the clock should be in the sensitivity list

**Rule 2:** Only signals that change on the same edge of the same clock should be part of the same process

# Synthesizable Processes

**Type 3:** Purely Synchronous with asynchronous set or reset

```
process (CLK, RESET)
  begin
   if (RESET='1') then
       Z <= '0';
   elsif (CLK'event and CLK='1') then
       Z <= A and B;
   end if;
 end process;
```



# For purely sequential processes with Asynch Rst/St

**Rule 1:** Sensitivity list includes clock and set/reset signal

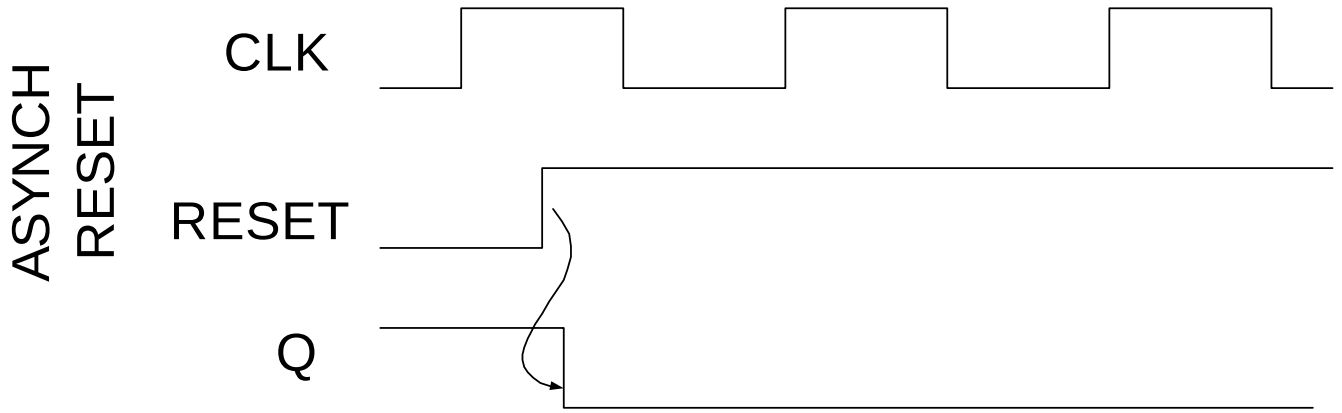
**Rule 2:** Need the clk'event clause (Recall why?)

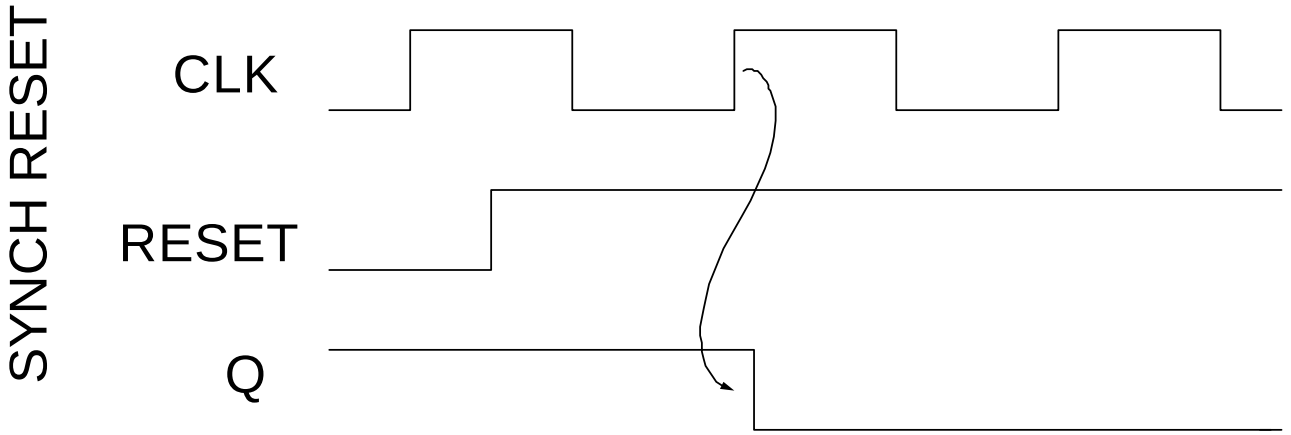
**Rule 3:** Inside the first part of the if statement, must assign either 0 or 1 (not anything else).

# Final Rule (the most important rule of all)

- If you want to synthesize your circuit, every process must fall exactly into one of these categories. Every process. Every single one. No execeptions.
  - If one of your processes doesn't, you need to break it up into blocks, where each block does fit into one of these categories.
- Note: This is a little bit conservative… Some synthesizers will handle a few patterns not described here. But DON'T COUNT ON IT!!

# Consider a flipflop with a synchronous reset



# Consider a flipflop with a synchronous reset

- Can we describe this sort of flipflop using any of the three types of processes described earlier?

  - Recall, the types are:
    - Type 1: Purely Combinational
    - Type 2: Purely Synchronous
    - Type 3: Purely Synchronous with Asynchrounous Set/Reset

# Consider a flipflop with a synchronous reset

```
process (CLK)
begin
   if (CLK'event and CLK='1') then
       if (RESET = '1') then
            Q <= '0';
        else
           Q <= D;
       end if;
 end process;
```

# What about a Moore FSM?

- Remember, a Moore machine has outputs that depend only on the current state.
- Can we define this in one process?

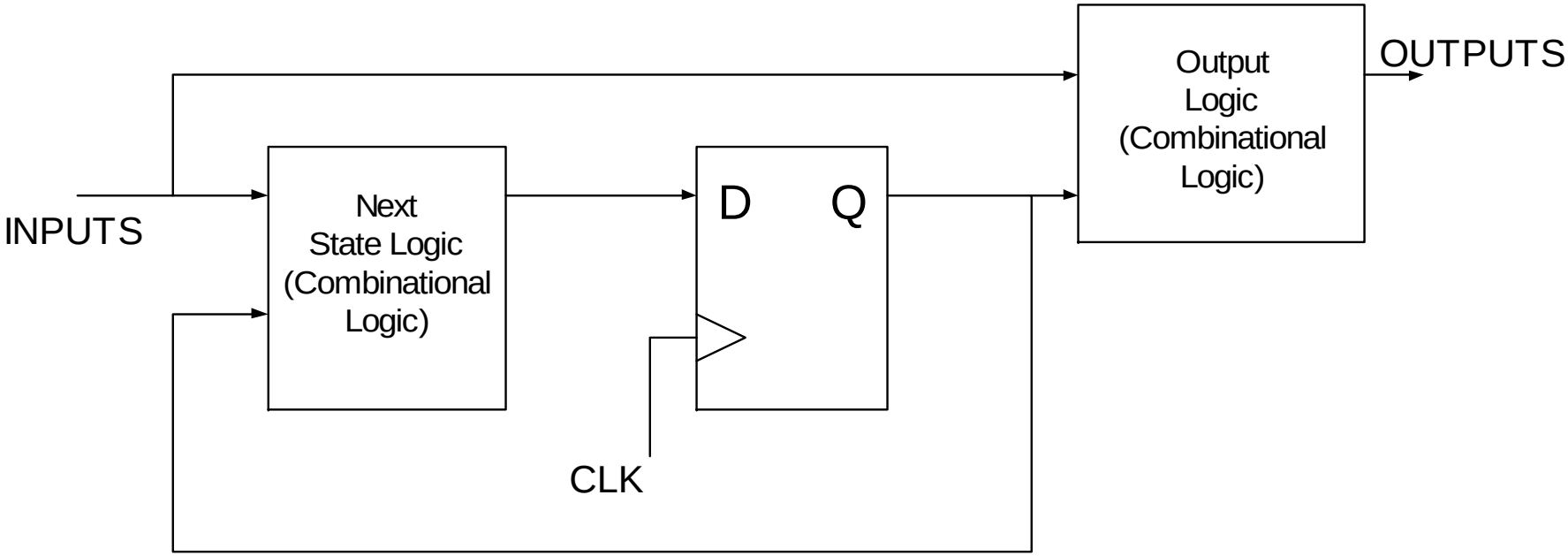
# Moore FSM sample code

```
process(clk)
variable PRESENT_STATE : bit_vector(1 downto 0) := “00”;
begin
      if (clk’event and clk=‘1’) then
        case PRESENT_STATE is
              when “00” => if (in_sig = ‘0’) then
                                PRESENT_STATE := “01”;
                           else
                              PRESENT_STATE := “00”;
                              end if;
              when “01” => PRESENT_STATE := “10”;
              when “10” => PRESENT_STATE := “11”;
              when “11” => PRESENT_STATE := “00”;
         end case;
         Z <= PRESENT_STATE(0) and not PRESENT_STATE(1);
      end if;
   end process;
```

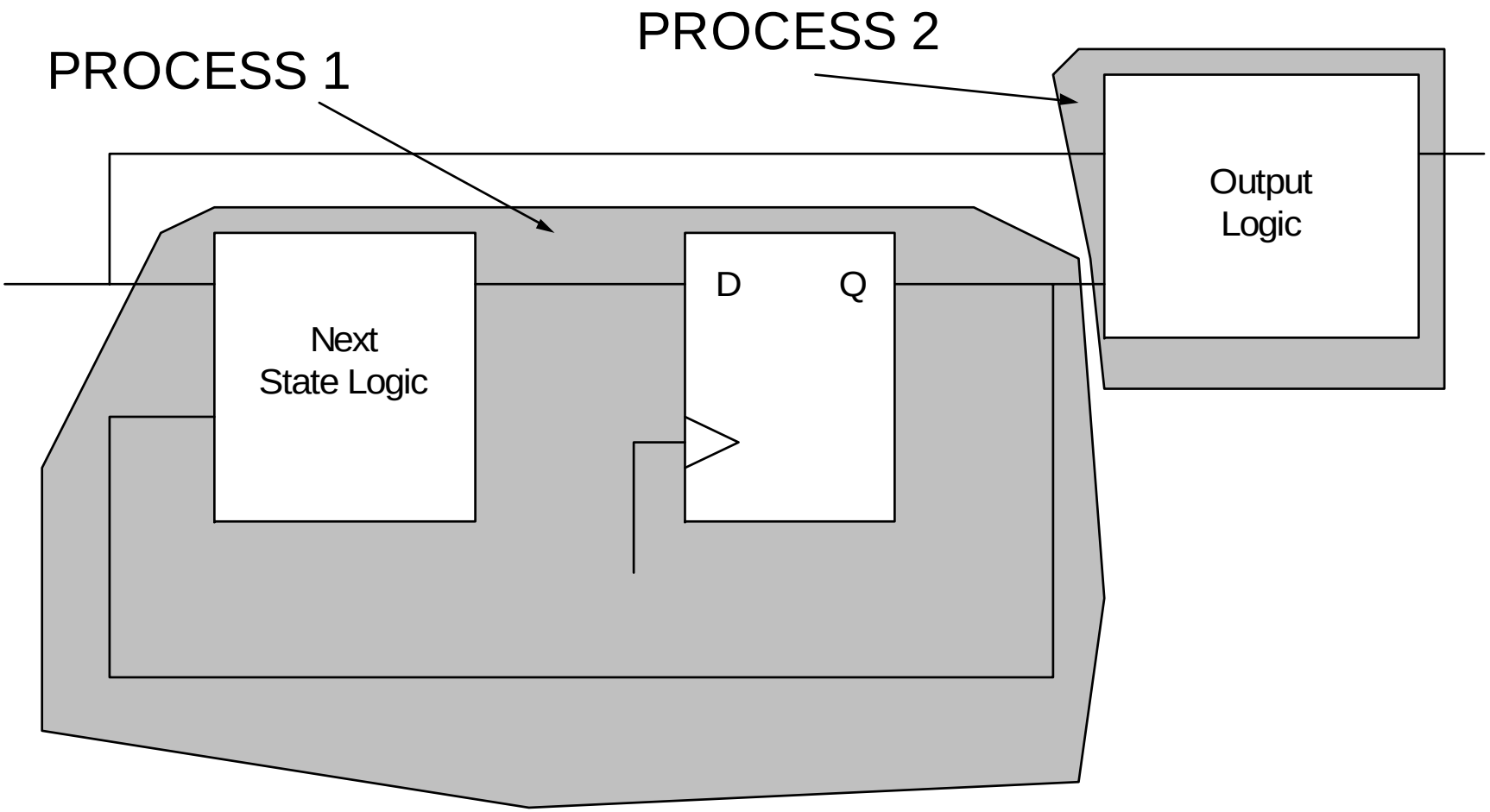
# What about a Mealy FSM?

- Remember, a Mealy machine has outputs that depend on both the current state and current inputs(s).
- Can we define this in one process?

# A Mealy state machine



# A Mealy state machine in 2 processes



# A Mealy State Machine

```
architecture behavioural of mealy is
    signal current_state : bit_vector(1 downto 0) := “00”;
    begin
        process (clk)
        begin
            if (clk’event and clk=‘1’) then
                case current_state is
                    when “00” => if (in_sig=‘0’) then current_state <= “01”;
                                                 else current_state <= “00”;
                            when “01” => ….
                end case;
            end if;
        end process;
        process (current_state, in_sig)
        begin
              out_sig <= some function of current state and in_sig
        end process;
    end behavioural;
```

# For State machines, remember

- Moore Machines
  - Can do in one process (some people use two)
- Mealy Machines
  - Must break into two processes (some people use three)
- Some people separate next state logic from flipflops

# Key points for synthesizable HDL

- Processes must be:
  - Type1: Purely Combinational
  - Type2: Purely Synchronous
  - Type3: Purely Synchronous with Asynchronous set/reset

This will ensure your code is synthesizable

If not, there is a good chance your circuit won't work

Remember when debugging for each process to:

- Identify which of the three types of process it is
- For that type, follow the pattern in this slide set ***exactly***

# Questions

- Does the FPGA execute HDL code?

- What types of processes are guaranteed to be synthesizable?

# Questions

- What are the rules for combinational processes?

- What are the rules for sequential processes?

# Questions

- What are the rules for synchronous processes with asynchronous set/reset signals?

- How many processes do you need minimally to implement a Moore FSM?

# Questions

- How many processes do you need minimally to implement a Mealy FSM?

- Why might someone use more than the minimal number of processes to implement an FSM?