ELEX 3330 : Programmable Logic Devices
2016 Winter Term

State Machine Design with VHDL

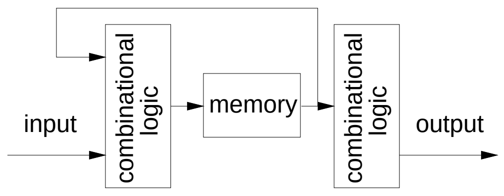
This lecture reviews the design of sequential logic and introduces the use of VHDL to design sequential logic. After this lecture you should be able to:

- *design a state machine from an informal description of its operation, and*
- *write a VHDL description of a state machine.*

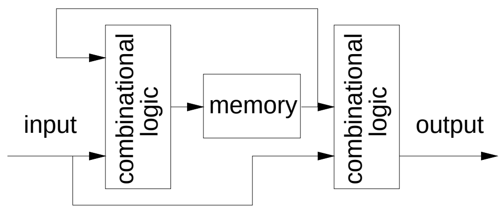
Sequential Logic and State Machines

Sequential logic circuits are circuits whose outputs are a function of their *state* as well as their current inputs. The state of a sequential circuit is the contents of the memory devices in the circuit. All sequential logic circuits have memory.

There are two basic types of state machines. In the *Moore* state machine the output is a function only of the current state:



whereas in the *Mealy* state machine the output is a function of the current state and the current inputs:



Moore state machines are simpler and are often preferred because we can use registered outputs to avoid glitches. However, since their outputs only change on clock edges they cannot respond as quickly to changes in the input.

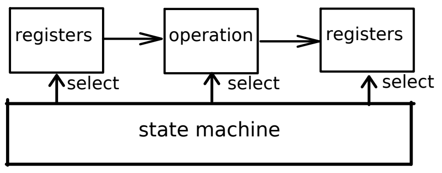
Exercise 1: Which signal in the above diagrams indicates the current state?

Any sequential logic circuit, even the most complex CPU, can be described as a state machine (also called a "finite" state machine or FSM).

However, large sequential circuits such as microprocessors are too complex to be described as a state machine.

Exercise 2: How may possible states are there for a CPU containing 10,000 flip-flops?

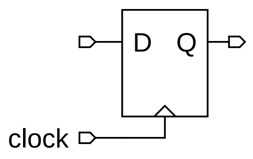
Instead, the design is partitioned into data registers and relatively simple state machines. These state machines then select the arithmetic and logic operations performed on the data data as it is transferred between the registers:



This type of design is called *Register Transfer Level* (RTL[1]) design. In this chapter we will study the design of simple FSMs. Later, we will combine these simple state machines with registers and arithmetic operations to build complex devices.

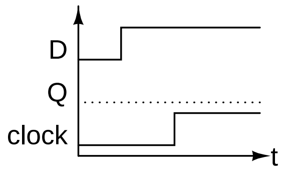
Common Sequential Logic Circuits

The *flip-flop* is the basic building block for designing sequential logic circuits. It's purpose is to store one bit of state. There are many types of flip-flops but the only one we will use is the D (delay) flip-flop.



[1]RTL can also mean Register Transfer Language and Register Transfer Logic

The rising edge of a clock input causes the flip-flop to store the value of the input (typically called *D*) and make it available on the output (typically *Q*). Thus the D flip-flop has a next-state input (D), a state output (Q) and a clock input. The D flip-flop state changes only on the rising clock edge.

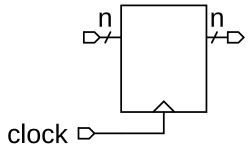


Exercise 3: Fill in the waveform for the Q signal in the diagram above.

Usually all of the flip-flops in a circuit will have the same signal applied to their clock inputs. This *synchronous* operation guarantees that all flip-flops will change their states at the same time and makes it easier to estimate the minimum clock period (or maximum frequency) for the circuit to operate properly.

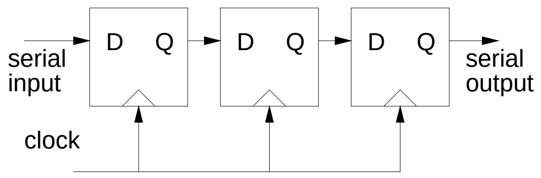
Synchronous design is nearly universal. You should avoid putting logic in clock paths and never do it in this course. Reliable interconnection of circuits that use different clocks requires special techniques that we will study later.

A *register* is several D flip-flops with their clocks tied together so that all the flip-flops are loaded simultaneously. A *latch* is a register that whose output follows the input (is *transparent*) when the clock is low.



Exercise 4: What would be another name for a 1-bit register?

A *shift register* is a circuit of several flip-flops where the output of each flip-flop is connected to the input of the adjacent flip-flop:



On each clock pulse the state of each flip-flop is transferred to the next flip-flop. This allows the data shifted in at one "end" of the register to appear at the other end after a delay equal to the number of stages in the shift register. The flip-flops of a shift registers can often be accessed directly and this type of shift register can be used for converting between serial and parallel bit streams.

Exercise 5: Add the parallel outputs to the shift register diagram.

A *counter* is a circuit with an *N*-bit output whose value increases by 1 with each clock. A *synchronous counter* is a conventional state machine and uses a combinational circuit (an adder) to select the next count based on the current count value. A *ripple counter* is a simpler circuit in which the the Q output of one flip-flop drives the clock input of the next counter stage and the flip-flop input is its inverted output.

Exercise 6: Is a ripple counter a synchronous logic circuit?

Design of State Machines

This section describes how to design a Moore state machine, the only kind you will need to design in this course.

Definition

Step 1 - Inputs and Outputs

The first step is to accurately identify the inputs and outputs. This is important because the rest of the design effort will be wasted if there are missing inputs or outputs.

Step 2 - States

List all possible combinations of output values. For a Moore state machine each of these will correspond to a state.

Step 3 - State Transitions and Additional States

Use the definition of the machine's behaviour to list the input condition(s) required for each possible state transition.

Split up states whose next-state transition is not uniquely determined from the current state and the input. These additional states will be represented by flip-flops which are not connected to outputs.

Documentation

The FSM can be described as a state transition table that lists the next state for each possible combination of state and input.

A FSM can also be described as a state transition diagram using circles for states. Arrows representing transitions connect the states. States are labelled with the corresponding output values and transitions with the corresponding input conditions.

Implementation

A Moore state machine uses one flip-flop for each output and additional flip-flops to represent the added states.

Up to 2^n states can be represented with *n* flip-flops (e.g. 3 flip-flops can encode up to 8 states). Another common alternative is to use one flip-flop per state.

Exercise 7: If we used 8-bits of state information, how many states could be represented? What if we used 8 bits of state but added the condition that exactly one bit had to be set at any given time (a so-called "one-hot encoding")?

We then design the combinational logic that determines the next state based on the current state and the input. The design of this combinational circuit proceeds as described earlier.

We also need to apply a clock signal to the clock inputs of each flip-flop. The FSM will change state on every rising edge of this clock.

Practical circuits also require some means to initialize (reset) the state. A synchronous reset can be included in the FSM design as another input. Asynchronous reset signals set/reset all the flip-flops independently of the clock. They can simplify the design and are required if we can't be certain that the system will always be in a valid state (e.g. on power-up).

Example: Resetable 2-bit Counter

A resetable 2-bit counter has one input (reset) and two one-bit outputs. There are four valid combinations of the outputs (00, 01, 10 and 11) and thus (at least) four states. The transition conditions are to go from one count to the next higher count if the reset input is not active, otherwise go to the zero state.

If we use the variables R as the reset, Q0 and Q1 to represent the outputs, and Q0' and Q1' as the next output, the state transition table would be as follows:

Q1	Q0	R	Q1'	Q0'
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	0	0	0
X	X	1	0	0

where 'X' is used to represent all possible values (often called a "don't care").

In this case no additional state variables are required since each state transition is uniquely defined by the current state and the input.

We can obtain the following sum-of-products expressions for these equations:

$$\begin{aligned} Q1' &= \overline{Q1}\,Q0\,\overline{R} + Q1\,\overline{Q0}\,\overline{R} \\ Q0' &= \overline{Q1}\,\overline{Q0}\,\overline{R} + Q1\,\overline{Q0}\,\overline{R} \end{aligned}$$

Exercise 8: Write the tabular description and draw the schematic of a resetable 2-bit counter with demultiplexed outputs (only one of the four outputs is true at any time). You can assume the counter will always be reset before being used. How does this counter compare to the previous one in terms of number of flip-flops and the complexity of the combinational logic?

Exercise 9: Draw the state transition diagram.

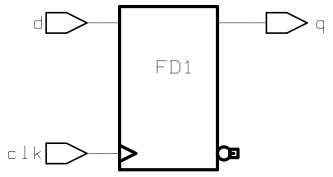
Sequential Circuits in VHDL

The design of sequential circuits in VHDL requires the use of the `process` statement. Process statements can include *sequential* statements that execute one after another as in conventional programming languages. However, for the logic synthesizer to be able to convert a process to a logic circuit, the process must have a very specific structure.

In this course you may only use the simple three-line process shown below. Using processes solely for registers reinforces that you are designing hardware. Also, the rules for synthesizing sequential statements within procedures are too complex to cover in this course. **In this course you may only use processes to generate registers and may only use the single-if structure shown below.**

As an example, the following VHDL description of a D flip-flop synthesizes to the following circuit:

```
entity d_ff is port (
   clk, d : in bit ;
   q : out bit ) ;
end d_ff ;

architecture rtl of d_ff is
begin
   process(clk)
   begin
      if clk'event and clk = '1' then
         q <= d ;
      end if ;
   end process ;
end rtl ;
```

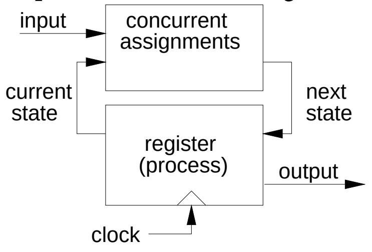


The expression `clk'event` (pronounced "clock tick event") is true when the value of `clk` has changed since the last time the process was executed. This is how we model memory in VHDL.

Within the process the output q is only assigned a value if `clk` changes and the new value is 1. Between clock edges the output retains its previous value. It's necessary to check for `clk`=1 to distinguish between rising and falling edges of the clock.

FSMs in VHDL are implemented using concurrent assignment statements (e.g. selected assignments) to generate the next state as a function of the current state and input values. The `process` above generates the flip-flops that define the current state.

This corresponds to the following block diagram:



A VHDL description for a 2-bit counter could be written as follows:

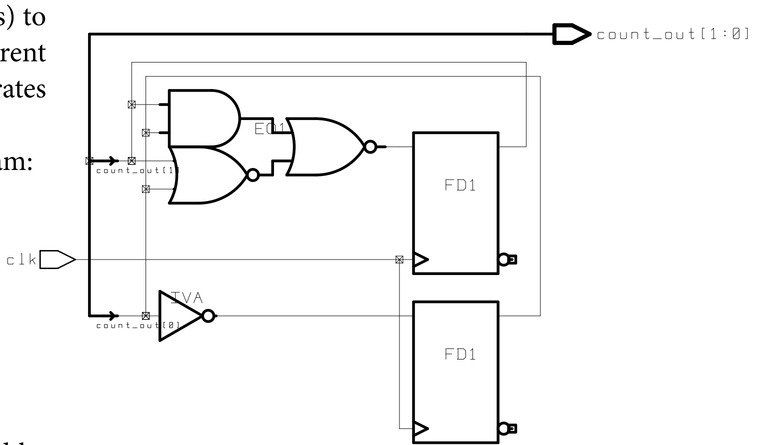
```
entity count2 is port (
   clk : in bit ;
   count_out : out bit_vector (1 downto 0) ) ;
end count2 ;

architecture rtl of count2 is
  signal count, next_count : bit_vector (1 downto 0) ;
begin
   -- combinational logic for next state
   with count select next_count <=
      "01" when "00",
      "10" when "01",
      "11" when "10",
      "00" when others ;

   -- combinational logic for output
   count_out <= count ;

   -- sequential logic
   process(clk)
   begin
      if clk'event and clk = '1' then
         count <= next_count ;
      end if ;
   end process ;
end rtl ;
```

Exercise 10: Draw the block diagram corresponding to this VHDL description. Use a multiplexer for the selected assignment and and a register for the state variables. Label the connections, inputs and outputs with the corresponding VHDL signal names.

Note that two signals need to be defined for each flip-flop or register in a design: one signal for the input (`next_count` in this case) and one signal for the output (`count`).

Also note that we must define a new signal (`count`) for an output if its value is to be used within the architecture. This is simply a quirk of VHDL.

The synthesized circuit is:



Exercise 11: Identify the components in the schematic that were created ("*instantiated*") the different parts of the VHDL code.

Exercise 12: Modify the diagram of the 2-bit counter by adding a (synchronous) reset input and a 2-input, 2-bit multiplexer. Draw a block diagram showing the modified circuit. Label the connections and i/o with appropriate signal names. Write the corresponding VHDL description.