Simple state machines in VHDL
or
The answer to Question 4.

Nigel Gunton 10/02

School of Electrical & Computer Engineering

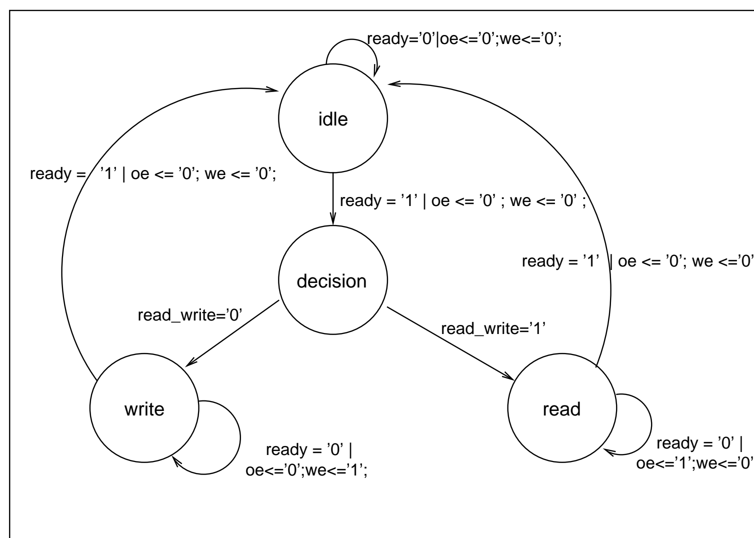
Grouping	individual
Prerequisites	Knowledge of *nix and emacs
Courses	CRTS, CSE, any?
Requirements	GNU/Linux or Unix system with Alliance tools, Stamina
Summary	Provides an introduction to more of the language and the tools used for UQC149S2
Duration	2 hrs

Background

VHDL descriptions of state machines are based on a ’case’ statement. This worksheet looks at the implementation of an interface between a CPU and memory as a state machine. The specification is provided as follows

4) "A controller is used to enable and disable the write enable (`we`) and output enable (`oe`) signals of a memory buffer during read and write transactions. The signals `ready` and `read_write` are outputs of a microprocessor and inputs to the controller. `we` and `oe` are outputs of the controller. A new transaction begins with the assertion of `ready` following a completed transaction (or upon power-up, for the initial transaction). One clock cycle after the commencement of a transaction, the value of `read_write` determines whether it is a read or write transaction. If `read_write` is asserted, then it is a read cycle; otherwise, it is a write cycle. A cycle is completed by the assertion of `ready`, after which a new transaction can begin. Write enable is asserted during a write cycle and output enable is asserted during a read cycle" (From ’VHDL for Programmable Logic’, K. Skahill)

From this can be derived the following state diagram. The events and actions have been described using VHDL fragments.



Study the statement and the diagram and make sure that you understand the following.

? Why are there 4 states?

? What is the role of the ready signal.

? Write the entity for the state machine.

Implementation

Once you have written an entity for the design; copy the two files found in `/usr/local/alliance/examples/rw_buf/` into a new subdirectory in your home directory. You should have `rw_buf.fsm` and `rw_buf.pat`. Open up the `.fsm` file in emacs

? How does your entity compare with this one. If there are any differences then make sure that you understand why and whether the differences are important.

Now find the line that says

```
if (clk = '0' and not clk'stable) then
```

and edit it to read

```
if (clk = '1' and not clk'stable) then
```

This is because the library that we will be using requires clock triggers to be on the rising edge and not the falling edge.

- Add `vdd` and `vss` inputs to the entity and save the file.
- Convert the `.fsm` to a data-flow description, `.vbe` by using the tool `syf`[1]. For now just use the line

```
syf -a -VEI rw_buf rw_buf
```

You should now have a file `rw_buf.vbe`. Test this by using the simulator, `asimut`, and view the resulting pattern file. Make sure that the results conform to the specification of the buffer given at the beginning of this document. How do you know that the results conform? Discuss this with your lab tutor.

Now convert the `rw_buf.vbe` into a structural model (`.vst`) by using `boog` and then simulate the new structural model using the same input pattern file but using a new name for the output pattern file. View the results. Are they correct? Ensure that you understand the sequence of events displayed in the output waveforms.

Common Errors

Open up the `rw_buf.fsm` file in emacs and study the case statement. Each of the sections such as

```
when wrte => oe <= '0';
             we <= '1';
             if ready = '1'  then
                next_state <= idle;
             else
                next_state <= wrte;
             end if;
```

changes state based on a particular input (`ready` in this example). Note that there is a a state assignment for each possible value that `ready` can hold.

When we are apparently interested in only one value of the signal as in

```
when idle => oe <= '0';
             we <= '0';
             if (ready = '1')  then
                next_state <= decision;
             else
                next_state <= idle;
             end if;
```

where we remain in the current state if the input does not have the value `'1'` it is tempting to write

```
when idle => oe <= '0';
             we <= '0';
```

[1] rtfm

```
if (ready = '1')  then
   next_state <= decision;
end if;
```

and assume that we will automatically remain in the current state. In software this is an acceptable assumption.

Edit the `rw_buf.fsm` file, removing the else clause as in the fragment above. Convert it using `syf` as before and then simulate it. Make sure that you write the results into a new pattern file and then compare the results with the earlier results.

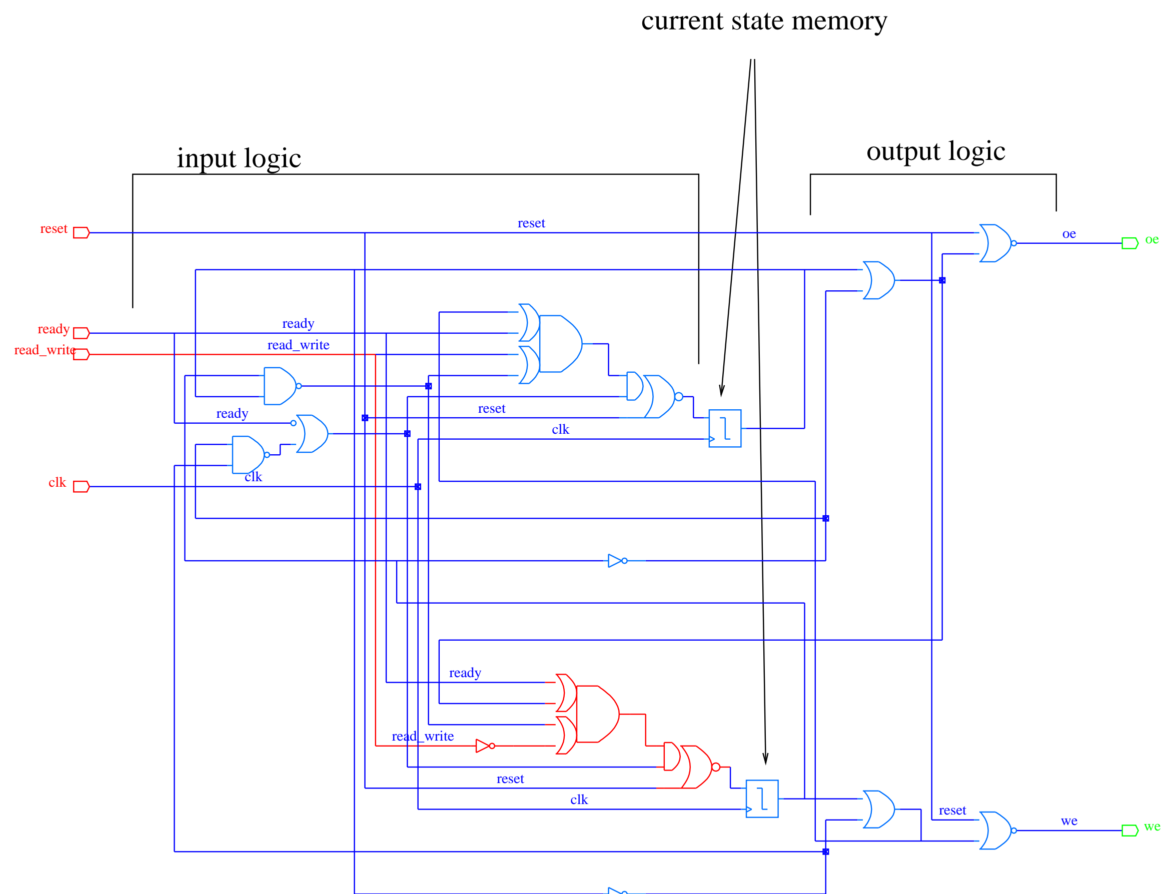
? What has happened.

? Why.

This is one of the commonest errors when people with a software background start developing state-machines in VHDL. The state-machine is in hardware and the current-state value is updated on every clock tick by logic applied to the inputs. If you do not explicitly assign a state for all input conditions then the synthesis tools will make assumptions about what the next-state should be.

Moore or Mealy?

Use the tool `xsch` to view the structural model of the read_write buffer (`.vst`). It will look similar to the example below. It can be seen to fall into three sections that correspond to the sections presented in the lecture. Is it a Moore state-machine or a Mealy state-machine? The red coloured signals indicate the critical path in the design. Use the `View`→ `Layers` menu to switch off some of the labeling in the diagram, e.g. 'io-gate' and 'net'.



rw_buf state machine at the logic gate level

Further testing.

Does the pattern file provided test all possible input conditions? If not then modify the input pattern file to provide for this and also ensure that your pattern file has meaningful labels in it.

Finally, why a state called `wrte` and not `write`?