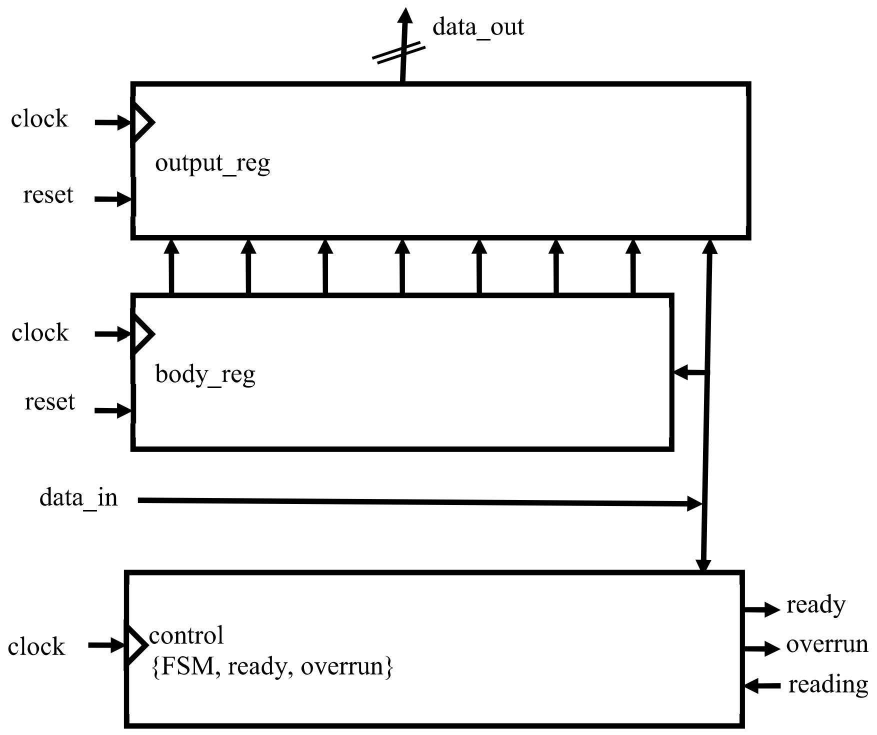
Lab 14-1 Coding State Machines in Multiple Styles

Objective: To code state machines in different styles for synthesis.

In this lab, you code the serial-to-parallel interface receiver as an FSM in different styles. Instead of shifting the matching character, the FSM steps to the next state upon receiving a correct match bit and returns to a previous state while receiving an incorrect match bit.

Read the specification first and then follow the instructions in this lab. Please make sure to use the same variable name as the ones shown in block diagrams.



Specifications

- `data_in`, `reading`, `clk`, and `reset` are single-bit input signals.
- `data_out` is an 8-bit output signal, whereas `ready` and `overrun` are 1-bit output.
- The design is clocked on the rising edge of `clk`.
- Every serial bit transmission has a preamble that needs to be sent before sending the character. For this lab, the preamble value is fixed to 8’hA5.
- After the preamble, the `ready` signal will be high.

Designing a Finite State Machine

1. Change to the *lab15-fsm* directory and examine the files.

2. Write down the following three FSM styles.

   a. A single sequential block (as the *rcvr.v* file partially codes).

   b. Move the "next state" encoding to a separate combinational block.

   c. Also, move all non-FSM registers to a separate sequential block.

3. In your favorite editor, modify the *rcvr.v* file to comply with the coding style.

Verifying the Finite State Machine Design

1. Verify the RTL model using the following commands with Xcelium™.

```
xrun rcvr.v rcvr_test.v (Batch Mode)
```

2. Verify that you see the following message.

```
Message sending: I Love Verilog
At time 35: Put character "I"
At time 39: Got character "I"
At time 77: Put character " "
At time 95: Got character " "
At time 117: Put character "L"
At time 127: Got character "L"
At time 153: Put character "o"
At time 175: Got character "o"
At time 193: Put character "v"
At time 207: Got character "v"
At time 231: Put character "e"
At time 257: Got character "e"
At time 275: Put character " "
At time 297: Got character " "
At time 309: Put character "V"
At time 313: Got character "V"
At time 349: Put character "e"
At time 365: Got character "e"
At time 391: Put character "r"
At time 423: Got character "r"
At time 425: Put character "i"
At time 433: Got character "i"
At time 459: Put character "l"
At time 487: Got character "l"
At time 495: Put character "o"
At time 507: Got character "o"
At time 537: Put character "g"
At time 563: Got character "g"
Message received: I Love Verilog
TEST DONE
```

3. Correct your model until it passes the test.

4. When all models pass their test, synthesize the RTL model using the Genus™ Synthesis Solution.

```
genus -f genus_shell.tcl
```

5. Check to see that the *synthesis succeeded*. Correct your model until it is synthesized.

6. Verify the gate-level model using the following commands with Xcelium.

```
xrun rcvr.vg rcvr_test.v -v ../tutorial.v -vlogext vg
```

7. Check to see the messages I Love Verilog and TEST PASSED as above. If the model fails its test, then you can ask the instructor for help or try again after you study the lecture module *Avoiding Simulation Mismatches*.

8. Correct your design as needed.

End of Lab