1. **Write an assertion check to make sure that a signal is high for a minimum of 2 cycles and a maximum of 6 cycles.**

```
property ppt;
  @(posedge clk) disable_iff(!reset) $rose(a) |-> ##2 (a[*2:6]);
endproperty
```

The property ppt is defined to check the assertion.

The assertion is triggered on the positive edge of the clk signal and is disabled when the reset signal is active (low).

The triggering event for the assertion is $rose(a), which checks for the rising edge of signal a.

The response part of the assertion is ##2 (a[*2:6]), which means that after the rising edge of signal a, a should remain high for at least 2 cycles (##2) and a maximum of 6 cycles ([*2:6]).

2. **Write an assertion for glitch detection.**

```
property glitch_detect;
  @(posedge clk) disable_iff (!reset)
  ($isunknown(a) && !$isunknown($past(a))) |=> $error("Glitch
detected!");
endproperty
```

property glitch_detect;: This line starts the definition of the property named glitch_detect.

@(posedge clk) disable_iff (!reset): This line specifies that the property is triggered on the positive edge of the clk signal, and it's disabled when the reset signal is active (low).

($isunknown(a) && !$isunknown($past(a))): This is the main part of the property.

$isunknown(a): The $isunknown() function is a built-in SystemVerilog function that returns true if the signal a is in an unknown state (X). This part checks if the signal a is unknown at the current sampling point (positive edge of clk).

$past(a): The $past() function is a built-in SystemVerilog function that returns the value of the signal a at the previous clock edge. This part checks if the signal a was known (not unknown) at the previous clock edge.

&&: This is the logical AND operator, which combines the conditions.

|=>: This is the "assertion implication operator," which means "if the condition on the left-hand side is true, then the condition on the right-hand side should be true as well."

$error("Glitch detected!"): If the condition specified on the left-hand side of the implication ($isunknown(a) && !$isunknown($past(a))) is true (i.e., a glitch is detected), then the $error statement is triggered. The $error statement terminates the simulation immediately and prints the specified error message, which is "Glitch detected!" in this case.

3. **If there's an uncorrectable err during an ADD request, err_cnt should be incremented in the same cycle and an interrupt should be flagged in the next cycle.**

```
property error_detect;
  @(posedge clk) disable_iff(!reset)
  (add_request && uncorrectable_err) |->
  (err_cnt <= err_cnt + 1;) ##1 interrupt;
endproperty
```

The error_detect property checks for two conditions:

When there is an add_request and an uncorrectable_err at the same time on the positive edge of the clk (and reset is not active), it triggers the assertion.

In response to the triggering event, it increments err_cnt in the same cycle and flags the interrupt signal high in the next cycle (one clock cycle after the triggering event).

4. **Are following assertions equivalent:**

@(posedge clk) req |=> ##2 $rose(ack);
@(posedge clk) req |-> ##3 $rose(ack);

Both assertions essentially state the same condition:

"When there is a rising edge of req, the ack signal should rise ($rose(ack)) within 2 clock cycles after the rising edge of req."

The difference in the two assertions is the timing offset specified after the implication (|=> and |->), but in this case, they are equivalent because the $rose(ack) is specified to happen within the same time window (2 clock cycles) in both assertions.

So, in conclusion, the two assertions are indeed equivalent and describe the same behavior.

5. Write an assertion: everytime when the valid signal goes high, the count is incremented.

```
property increment_on_valid;
  @(posedge clk)
  disable_iff(!reset)
  valid && !$past(valid) |=> count == count + 1;
endproperty
```

The increment_on_valid property checks if the valid signal rises (goes from low to high), and when it happens, it increments the value of the count variable by 1. The assertion will check this behavior during simulation, and if the property fails, it indicates that the count was not incremented correctly when the valid signal went high.

6. If the state machine reaches STATE=active1, it will eventually reach STATE=active2.

```
property ppt;
  @(posedge clk)
  disable_iff(!reset)
  (STATE == active1) |=> ##1 (STATE == active2);
endproperty
```

The property ppt captures the condition where the state machine transitions from STATE=active1 to STATE=active2 within one clock cycle after the positive edge of the clk signal, provided that the reset signal is not active. The assertion will check this behavior during simulation, and if the property fails, it indicates that the state machine did not reach STATE=active2 in the expected timing from STATE=active1

7. When there's a no_space_err, the no_space_ctr_incr signal is flagged for exactly once clock.

```
property flag;
  @(posedge clk)
  disable_iff(!reset)
  no_space_err && !$past(no_space_err) |=> ##1
no_space_ctr_incr;
endproperty
```

When the positive edge of the clk signal occurs, if the reset signal is not active, and the no_space_err signal goes from low to high (rising edge), then in the very next clock cycle, the no_space_ctr_incr signal should be flagged (set to high) for exactly one clock cycle.

8. If signal_a is active, then signal_b was active 3 cycles ago.

```
property ppt;
  @(posedge clk)
  disable_iff(!reset)
  signal_a && $past(signal_b, 3);
endproperty
```

When the positive edge of the clk signal occurs, if the reset signal is not active, and signal_a is active (true) at the current clock cycle, then three clock cycles ago, signal_b was also active (true).

9. For a synchronous FIFO of depth = 16, write an assertion for the following scenarios. Assume a clock signal(clk), write and read enable signals, full flag and a word counter signal.

a) If the word count is >15, FIFO full flag set.

b) If the word count is 15 and a new write operation happens without a simultaneous read, then the FIFO full flag is set.

```
// Assertion: FIFO full flag is set when word count is > 15
assert property fifo_full_when_word_count_gt_15;
  @(posedge clk) disable_iff (!rst_n)
  (wordcnt > 15) |=> fifo_full;
endproperty
```

```
// Assertion: FIFO full flag is set when word count is 15 and a new
write happens without a simultaneous read
assert property fifo_full_when_word_count_eq_15_and_new_write;
  @(posedge clk) disable_iff (!rst_n)
  (wordcnt == 15 && write_en && !read_en) |=> fifo_full;
endproperty
```

10. Write an assertion checker to make sure that an output signal never goes X?

```
// Property: Output signal should never be X
property no_x;
  @(posedge clk)
  disable_iff(!reset)
  !$isunknown(output_signal);
endproperty
```

When the positive edge of the clk signal occurs, if the reset signal is not active, then the output_signal should never be in an unknown state (X).