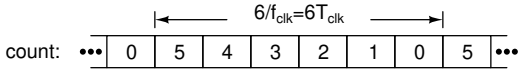
Common Circuits

This lecture covers a few additional commonly-used circuits.
After this lecture you should be able to: write Verilog and draw a block diagram for a timer, clock divider, edge detector and synchronizer.
Updated Feb 2.

Timer and Clock Divider

If we set a register to $N-1$ and decrement it on every clock cycle it will reach zero[1] after N clock cycles. This happens after a delay of N/f_{clk} seconds where f_{clk} is the clock frequency.

If the counter is re-initialized to $N-1$ when it reaches zero, the counter values will repeat periodically with frequency of f_{clk}/N. For example, if $N=6$:



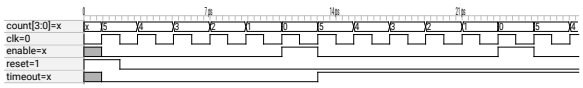
By taking other actions (e.g. decrementing another counter or inverting an output) when the counter reaches specific values (e.g. zero) we can carry out these actions after a certain delay or at a certain rate.

The following example shows how a counter can be used to implement a resetable timer and to generate a periodic enable signal:

```
module ex38 ( input logic clk, reset,
              output logic timeout, enable ) ;

   // timer duration or divided clock period
   localparam period = 6 ;

   logic timeout_next, enable_next ;
   logic [3:0] count, count_next ;

   // resetable clock divider
   assign count_next
     = reset ? period-1 :
       count ? count - 1'b1 : period-1 ;
   always_ff @(posedge clk) count = count_next ;

   // latch a '1 first time count reaches zero
   assign timeout_next
     = reset ? '0 :
     count ? timeout : '1 ;
   always_ff @(posedge clk) timeout = timeout_next ;

   // periodic signal
   assign enable_next
     = count_next ? '0 : '1 ;
   always_ff @(posedge clk) enable = enable_next ;

endmodule
```

[1]Timers traditionally count down to zero because no additional hardware is required to determine the final value – the subtracter's borrow bit indicates when the count has reached zero.

and the signals:



Exercise 1: Draw the block diagram for the Verilog above.

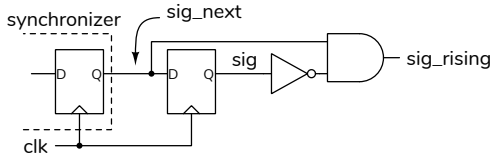
Exercise 2: Is the enable output a square wave? How could you create a square wave? What would be the period?

Exercise 3: How would the output differ if `enable_next` was based on `count` rather than `count_next`?

As described below, it is usually a bad idea to use a periodic signal generated this way as a clock (in the `@(posedge ...)` part of a conditional assignment).

Edge Detector

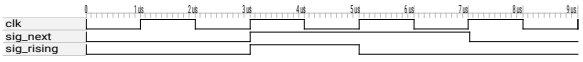
To detect rising edges on a signal that is not a clock, such as a press of a pushbutton, we can store the previous value of the signal in a flip-flop and compare it to the current value. If the previous value was low and the current value is high then there must have been a change in level from low to high (i.e. a rising edge):



The Verilog for this circuit would be:

```
module ex39
  ( input logic clk, sig_next,
    output logic sig_rising )  ;

   logic sig ;

   always @(posedge clk) sig = sig_next ;

   assign sig_rising = ~sig & sig_next ;

endmodule
```

and an example of its behaviour is:



Exercise 4: What is the duration of the `sig_rising` output?

Exercise 5: How would you detect a falling edge?

Synchronizer

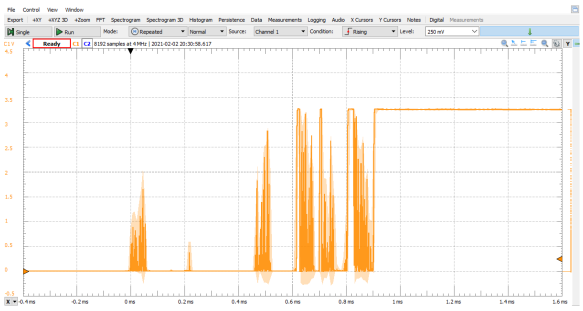
Since inputs such as those from pushbuttons are asynchronous to the clocks used in your circuit, there is no way to ensure that the setup time requirements of the various flip-flops in your design will be met.

We can minimize the likelihood of metastable events[2] by passing asynchronous inputs through a flip-flop called a synchronizer[3]. The output of this flip-flop will be synchronous with the clock which allows us to verify that the setup requirements of the other flip-flops in our design will be met.

Exercise 6: Draw the schematic of a synchronizer.

Debouncer

Mechanical switches briefly interrupt the connection when they switch. This "switch bounce" produces multiple signal edges. Here's an example:

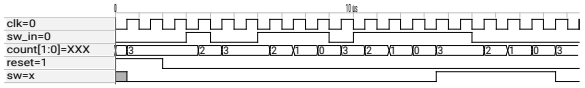


If a switch is being used in a circuit that is edge-sensitive, such as the clock for a register, the signal from the switch must be "debounced" so that there is only one transition for each switch operation. This can be done by delaying recognition of changes in the switch input until the signal has been stable for a time longer than the maximum duration of the "bounces". The required duration depends on the switch, but typical maximum bounce durations would be a few milliseconds.

The Verilog for a debouncer circuit might be:

```
module ex40
  #(parameter N=16'hffff)
   ( input logic reset, clk, sw_in,
     output logic sw ) ;

   logic [$clog2(N)-1:0] count, count_next ;
   logic sw_next ;

   // reset count while input matches output
   assign count_next
     = reset ? N-1 :
       sw_in ==? sw ? N-1 : count - 1'b1 ;
   always_ff @(posedge clk) count = count_next ;

   // change output if not reset in N clock cycles
   assign sw_next
     = reset ? 0 :
       !count ? sw_in : sw ;
   always_ff @(posedge clk) sw = sw_next ;

endmodule
```

and an example of its behaviour when the required "stable" duration is 4 clock periods is:



Exercise 7: What is the bounce duration in the waveform above? What value of **N** would achieve a delay of ten times this with a 50 MHz clock?

Multiple Clocks

It is usually not a good idea to use signals generated by logic, for example the different bits of a counter, as clocks. Among other issues, clock signals generated by logic circuits will have more uncertainty in their timing which in turn will reduce the speed at which your design can operate.

However, power consumption is sometimes more important than speed and reducing the clock rate reduces power consumption because power consumption is linearly related to the clock rate.

Battery powered designs might divide a clock to a much lower frequency and use that as the sole clock in the design[4].

The use of multiple clocks that are not derived from the same clock (i.e. generated by different oscillators) poses different problems and special techniques are needed to cross "clock domains."

[2] Meaning that flip-flop outputs do not settle by their specified t_{CO}.

[3] Synchronizers typically use two flip-flops in series to make them more robust.

[4] Watches and battery-powered timers typically use an oscillator operating at $2^{15} = 32768$ Hz.