



Lab 5 Preparation

Lab 5 Components

- **Part I:** Create a counter
 - Use the synchronized counter circuit described in lectures.
- **Part II:** Slow down the clock
 - Use the counter and the on-board clock to create a slower clock.
- **Part III:** Morse code decoder
 - Decode incoming Morse Code signals!
 - Uses code from Part II 😊

Design Guidelines

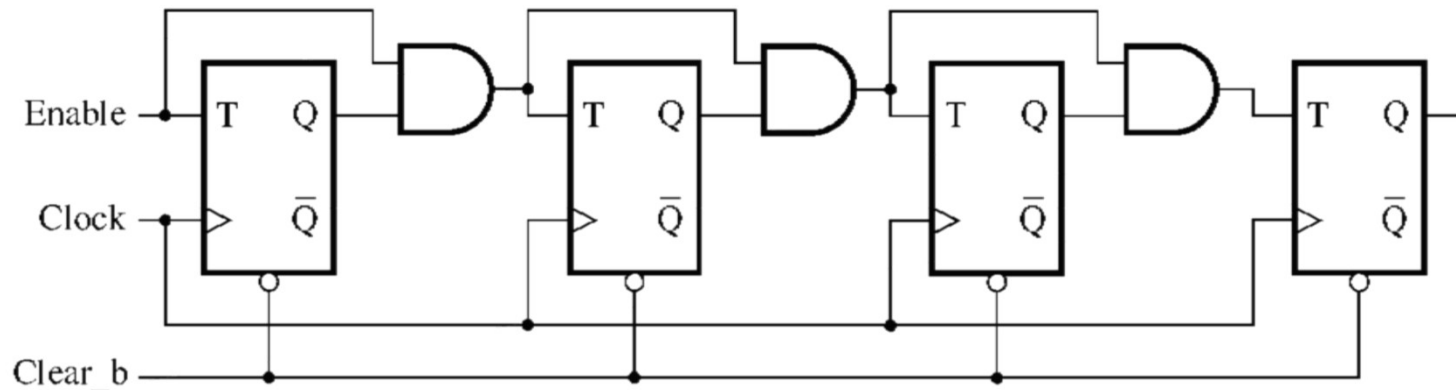
- A note on `always` blocks:
 - **Combinational Circuits** use blocking assignment statements (the `=` operator)
 - **Sequential Circuits** use non-blocking assignment statements (the `<=` operator)

```
always @(*) begin
    q = d;
end
```

```
always @(posedge clock) begin
    q <= d;
end
```

- Do NOT mix assignment types in the same `always` block!

Part I – 4-bit Counter



- Diagram shows a 4-bit synchronous counter, made with T-flip-flops
 - The T flip-flops here have an **active-low asynchronous reset** (Clear_b).
- Need to use hierarchical design to make an 8-bit counter.

Part I (continued)

- Prelab parts:
 - Draw and label 8-bit counter schematic.
 - Write Verilog code for flip-flop and counter
 - Don't use "tff" as a module name.
 - Simulate your counter to confirm correctness.
 - Augment Verilog code with input/output layer.
 - Analyze your design!
 - Logic Utilization (in ALMs)
 - Maximum Frequency (F_{\max})
 - Netlist Viewer

} Tools in
Quartus

Part II: More Counters

- **Main goal:**
 - Display incrementing digits on a hex display at different speeds (e.g., once per second, twice per second, etc)
- **How do we do this?**
 1. Need access to a clock signal.
 2. Need to adjust this clock signal to the right speed.
 3. Need to increment a counter at this adjusted clock speed.

Part II (continued)

- **Step #1:** Finding a clock signal
 - DE1-SoC has a **50MHz clock**
 - Pin `CLOCK_50`
 - Hertz (Hz) => number of cycles per second
 - 50MHz => How many clock cycles per second?
 - Would you be able to see that?

Part II (continued)

- **Step #2:** Slowing down the clock.
 - Load a counter with a countdown value (through a parallel load) and produce an enable signal when the countdown reaches zero.

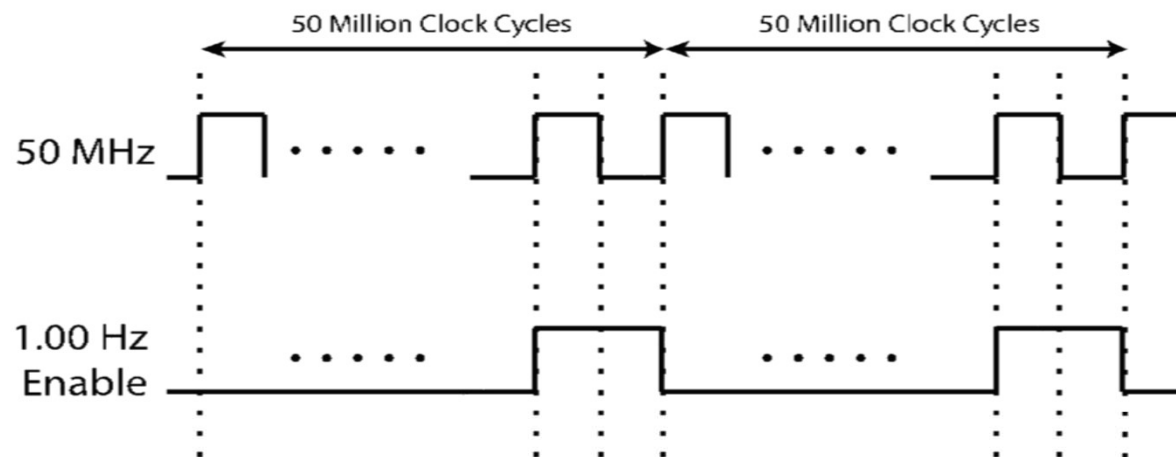


Figure 3: Timing diagram for a 1 Hz enable signal

- Another module will use this enable signal to determine whether it will change on the next clock pulse or not.

Part II: An Aside

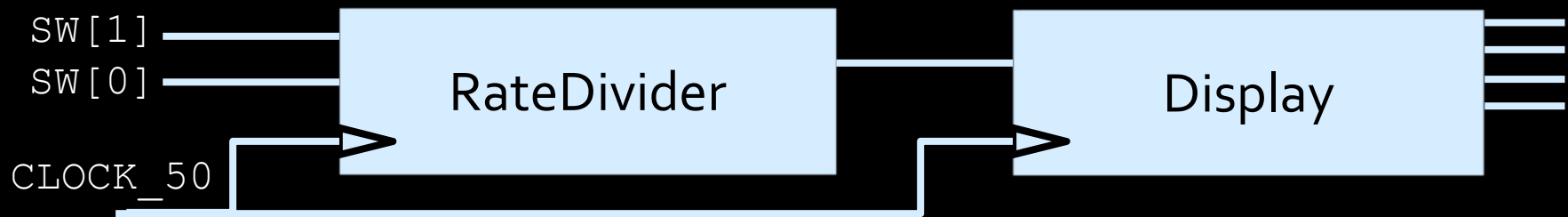
- Here is an alternate implementation of counters:

```
reg [3:0] q;  
  
always @(posedge clock)  
begin  
    if (Clear_b == 1'b0)  
        q <= 0;  
    else  
        q <= q + 1;  
end
```

- Things to note:
 - **Non-blocking** assignment (sequential circuit)
 - Means that assignments are implemented in parallel, not series.
 - Synchronous active-low `Clear` signal

Part II (cont'd)

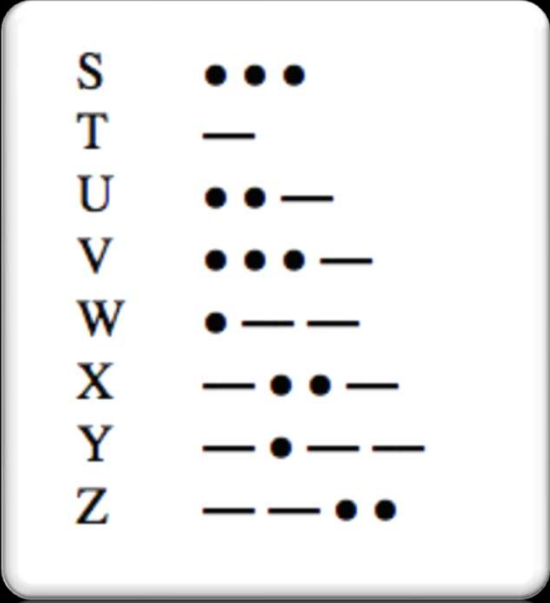
- **Step #3:** Updating the display counter.
 - You will need 2 counters for this:
 - A `RateDivider` counter (from previous step)
 - A `Display` counter (that feeds to the 7-segment decoder)
 - Both will be synchronized to the same 50MHz clock.



- Recall the purpose of an `Enable` signal in a counter.
 - How often do you want the `Display` counter to increment?
 - `SW[1]` and `SW[0]` will control that rate.

Part III – Morse Code

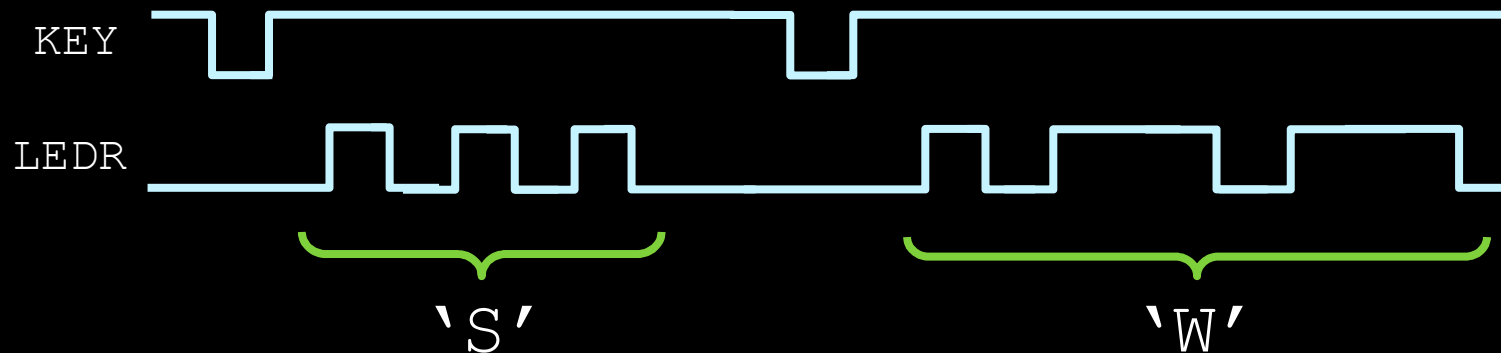
- **Morse Code:** “A method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment.”



S	• • •
T	—
U	• • —
V	• • • —
W	• — —
X	— • • —
Y	— • — —
Z	— — • •

Part III (continued)

- You will be transmitting individual letters using a single red LED
 - **Dot** => 0.5 seconds LED on
 - **Dash** => 3 * 0.5 seconds LED on
 - **Pause** (between symbols or in the end of transmission)
=> 0.5 seconds LED off



Part III (continued)

- How do you do this?
 - **Step #1:** Create a Lookup Table (LUT)
 - Switch values as input, binary representation of the corresponding Morse code letter to transmit as output.
 - **Step #2:** Create a shift register
 - When the `KEY` input sends a signal, load a shift register with the current value from the lookup table.
 - Shift out a bit on each clock cycle and send it to the LED.

Part III (continued)

- How to decide on the binary representation in the LUT
 - Each bit corresponds to 0.5 seconds of light (1) or no light (0).
 - Example: • – (“dot dash”)
 - Multiple ways this could be represented:
 - 101110
 - 10001110
 - 10111000
 - 10011100000000
 - All of these look like “dot dash” in the end, so it’s up to you (or up to the longest letter you encode)
- How do you make the shift register move bits out every half a second?
 - Rate divider from Part II 😊

Part III (continued)

- How it all comes together:

