7-Segment Decoders and State Machines

Version 2: Added LED pin numbers, removed conditional assignment option, added pin assignments and .qsf file

Introduction

In this lab you will modify the kitchen timer developed in the previous lab by: (i) generating a 100 Hz clock to control the timer, (ii) displaying the time remaining on a 7-segment LED display and (iii) changing how the pushbutton switches control the timer.

Requirements

Your design should generate a 100 Hz clock signal from the 50 MHz clock and use it as the clock for all other logic. Quartus should recognize the divided clock signal as a clock.

The time remaining should be displayed as a hexadecimal digit on a 7-segment LED display instead of a 4-bit binary number. Your design should use an array as a lookup table instead of a selected or conditional assignment.

The operation of the timer should be modified so that the pushbutton switches control the timer as follows:

- pressing and releasing only the left button will cause the time to increase by 1
- pressing and releasing only the right button will cause the time to decrease by 1
- pressing both buttons and releasing them will toggle the run/stop state

In no case should the time remaining be decremented when at zero or incremented when at 15.

Your design should comply with the course VHDL coding guidelines including that it be synchronous – all registers (except for one clock divider) should use the same clock and not use asynchronous sets or clears.

Pre-Lab

Read the material below and write a VHDL description of the timer that meets the requirements above. The following sections provide more details on each requirement.

It is strongly recommended that you check your VHDL description for syntax errors before the lab.

If you've installed Quartus on your PC you can also follow the procedure below, complete the lab at home and bring your circuit to the lab to have it checked.

Debouncing

Instantiate one **`clk_debounce`** entity for each pushbutton. This is a synchronizer/debouncer designed to operate with a 100 Hz clock. The VHDL file is available on the course web site and should be added to your project.

Clock Divider

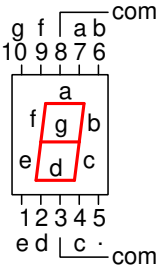
The generated clock should come from the output of a flip-flop to ensure it has no glitches. Glitches are short pulses resulting from differences in propagation delays through combinational logic. The generated clock does not need to have a 50% duty cycle since only the rising edges will be used.

7-segment Decoder

Modify your design to display the time remaining in hexadecimal on a single-digit 7-segment LED display.

A 7-segment display decoder converts a 4-bit binary value to signals that turn the individual segments of a 7-segment display on or off as appropriate.

The conventional labels for the segments and the pinouts for the LED in your parts kit is:



I connected the LED segments to the CPLD using the bottom row of pins on connector P2:

wire color	CPLD pin	segment	LED pin
black (0)	30	`e`	1
brown (1)	34	`d`	2
red (2)	36	`c`	4
orange (3)	38	`dp`	5
yellow (4)	40	`-`	
green (5)	42	`b`	6
blue (6)	44	`a`	7
violet (7)	48	`f`	9
gray (8)	50	`g`	10
white (9)	52	`com`	-

The display in your parts kit[1] has a `com`mon anode. This pin should be connected to an output (pin 52 above) through a 1 kΩ resistor. This pin should be forced high (e.g. `com <= '1';`). Typically a current-limiting resistor is used on each segment to obtain the same brightness regardless of the number of segments that are lit. But for our purposes one resistor will do.

A lookup table using an array of 8-bit `std_logic_vector` values is a simple way to implement a 7-segment decoder. The values of the entries in the array are readily available[2].

You can use the VHDL aggregate syntax to assign a vector value to a number of discrete signals. For example if `decoder` is the 7-segment decoder array (for example declared as `std_logic_vector (7 downto 0)`) and `num` is an `unsigned` value you could use:

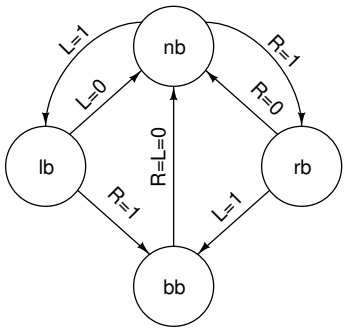
```
(dp,a,b,c,d,e,f,g) <= decoder(to_integer(num)) ;
```

Pushbutton Control

State Machine Description

We can describe the pushbutton controller as a state machine. The inputs to the state machine are the two synchronized pushbuttons and the outputs are the current state and the next state.

The following state transition diagram shows the states (e.g. `lb` for "left button was pushed") and the conditions that cause state transitions (e.g. `R=0` means the right pushbutton is not pushed).



The Quartus documentation recommends using enumerated types for states. This allows it to optimize the representation of the states. In particular, CPLDs and FPGAs often use "one-hot" encodings. An example of defining states using an enumerated type would be:

```
type bstate_t is ( nb, lb, rb, bb ) ;
signal state, state_next : bstate_t ;
```

The state transitions can be defined using one conditional assignment per state to compute the state transitions out of each state and a selected assignment to choose one of these depending on the current state. This is shown by the following code fragment:

```
signal nb_next, ..., bb_next : bstate_t ;
...
bb_next <= nb when r = '0' and l = '0' else bb ;
nb_next <= lb when l = '1' else rb when r = '1' else nb ;
...
with state select state_next <=
  nb_next when nb,
  ...
  bb_next when bb ;

state <= state_next when rising_edge(clk);
```

Actions

For the pushbutton switches to operate as described above we can only take action after both buttons have been released because we can't determine the order in which the buttons will be pressed when the user presses both.

Changes in other parts of your circuit (e.g. time remaining counter or the run/stop state) can be made conditional on the current state, the next state or both.

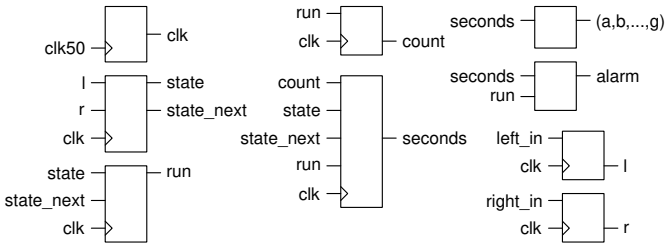
For example, if the state is `lb` and the next state is `nb` then the left button is being released and we should increment the time remaining. This should

[1] Datasheet on course web site.

[2] Note that these are active-high while your outputs should be active-low; a `not` operator will invert the bits.

only happen on the state transition so it has to be conditional on the values of both the current state and the next state. For example:

```
seconds_next <=
  ...
  seconds + 1 when state = lb and state_next = nb and
     seconds /= 15 else
  ...
```



Pre-Lab Documentation

You should prepare the following and be able to show it to the lab instructor at the start of the lab:

1. Block diagrams of the CPLD functional blocks (see below). Follow the conventions used in the lectures and show registers, multiplexers, logic functions, inputs and outputs. This will help you understand how your design works and to write the VHDL.

2. A schematic, including pin numbers, showing the CPLD connections to the pushbutton switches and to the 7-segment LED display. An accurate schematic will help you avoid wiring errors.

3. The VHDL code for your design, commented as described in the previous lab. You will need to have this before your lab since you will not have time to both write and troubleshoot your design in the time allotted to the lab.

Procedure

Synthesize your VHDL description, program the CPLD, interface it to the peripherals described above and demonstrate your working design to the lab instructor.

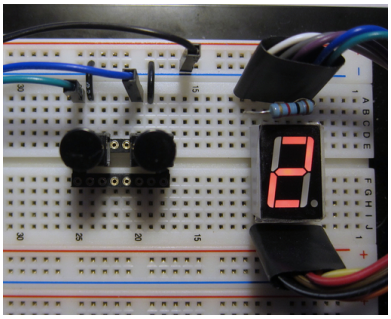
Hints

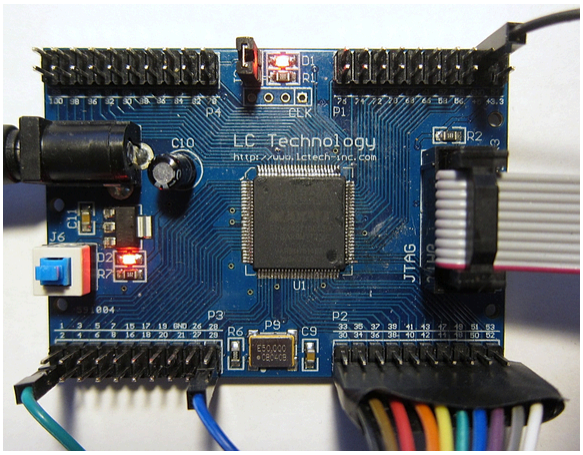
Block Diagrams

Based on the functional description we can determine the inputs and outputs of each functional block. The result might be:

Layout

The photos below show how the board can be connected to the pushbutton switches and the 7-segment LED display. The individual pins are held together with tape as a substitute for a connector. This allows all of the connections to be made at once.





Pin Assignments

These are the pin assignments shown above:

To	Assignment Name	Value	Ena
left_in	Weak Pull-Up Resistor	On	Yes
right_in	Weak Pull-Up Resistor	On	Yes
left_in	Location	PIN_2	Yes
clk50	Location	PIN_12	Yes
right_in	Location	PIN_29	Yes
e	Location	PIN_30	Yes
d	Location	PIN_34	Yes
c	Location	PIN_36	Yes
dp	Location	PIN_38	Yes
b	Location	PIN_42	Yes
a	Location	PIN_44	Yes
f	Location	PIN_48	Yes
g	Location	PIN_50	Yes
com	Location	PIN_52	Yes
alarm	Location	PIN_77	Yes

Quartus Settings Files

You can import the Quartus Settings File (`lab5pins.qsf`) on the course web site if you wish (**Assignments** > **Import Assignments...**). This is a text file with tcl commands to configure pin numbers and pull-ups for the signal names shown above. These match the signal names and pin headers used above.

Since you will have to make these assignments manually on the lab test, it's best not to do this unless you are confident you know how to make pin and pull-up assignments.

Demonstration and Marking

The preparation mark will be based on coming to the lab with the documentation described above.

The lab instructor will determine the demonstration part of the lab mark by checking that your circuit meets the requirements. In particular:

- a fully-synchronous design. The lab instructor will check **Compilation Report** > **Fitter** > **Resource Section** > **Global & Other Fast Signals** to verify that only two signals are present: the 50 MHz input clock (on pin 12) and your 100 Hz generated clock.
- the use of an array as a 7-segment LED decoder
- that your circuit works as described in the Requirements section.

Then the lab instructor will ask you to make a small change to the behaviour of your design to make sure you understand how it works. For example, the instructor could ask you to modify the actions taken by the pushbuttons, to limit the minimum or maximum timer values, the timer rate, etc.

Optional Extensions

If you found this lab too easy, you can try adding[3] the following features (listed in order of increasing complexity).

By adding a timer that is reset on entry into a state we can implement state transitions that depend on time. For example:

- if the left or right button is held down for more than 1 second the time remaining increments/decrements at a rate of approximately 4 seconds per second, and/or
- pressing both buttons for more than one second sets the time remaining to the initial value computed in the previous lab, and/or
- pressing both buttons for more than two seconds sets the initial time remaining to the current time remaining.

The timer can be reset on state transitions (e.g. `btimer_next <= to_unsigned(100,8) when state_next /= state ;`).

Another extension would be to save and restore the initial time remaining to the CPLD's on-board "UFM" flash memory (use the ALTUFM LPM block and a parallel interface).

Decoding CPLD Date Codes

Out of curiosity I wet the package to check the CPLD's date code:

which indicates it was built in TSMC's Fab 8 (Altera fab code 9M) in week 25 of 2017. Altera was purchased by Intel in 2015.

[3] You must leave the required functionality intact.