TIMER PERIPHERAL

This lab has you create a timer peripheral such as that typically found in a microcontroller.

BACKGROUND

Review the notes on creating bus-based peripherals and parameterized Verilog.

TASK DESCRIPTION

You are to implement a timer peripheral with the following features:

- 32-bit timer, 32-bit period register
- Status, Control register

The module interface is given below (note the parameter definitions and default values):

```
module timer32(clk, reset, din, dout, wren, rden,addr)
    );
input clk, reset, wren, rden;
input [31:0] din;
output [31:0] dout;
input addr[2:0];

parameter PERIOD = 32'h0000000F;  //must have this initial value
parameter ENBIT = 1'b0;
```

The input/output definition is:

- *clk* – clock input
- *reset* – high true reset
- *rden* – read enable for dout bus
- *din* – data input bus
- *dout* – data output bus, reflects the contents addressed by the *addr* input when *rden* is high, else *dout* is 0.
- *addr* – address bus for internal registers
- *wren* – write line for a register. The register selected by *addr* is written on the rising clock edge when *wren* is high.

Register addressing and reset behavior:

Register	Address	Value at reset
Timer	0b00	0
Period	0b01	Set to parameter `PERIOD`
Status/Control	0b10	All bits 0, except for enable bit, which is set to

		parameter `ENBIT`
(dout = 0 for this)	0b11	

Status/Control register definition:

- Bit 0: Enable bit, timer counts up when '1', frozen when '0'.
- Bit 1: TMR Flag – this is set to a '1' when the period register matches the timer contents. This bit is cleared when a read is done of this register – it can also be cleared by a write to the register
- Bit 2: Toggle bit – this bit is complemented each time the period register matches.

For the control register, a write operation takes precedence over any other action.
For the TMR flag, if a read operation (clears the TMR flag) and a period match (sets the TMR flag) occur in the same clock cycle, then the period match takes precedence.

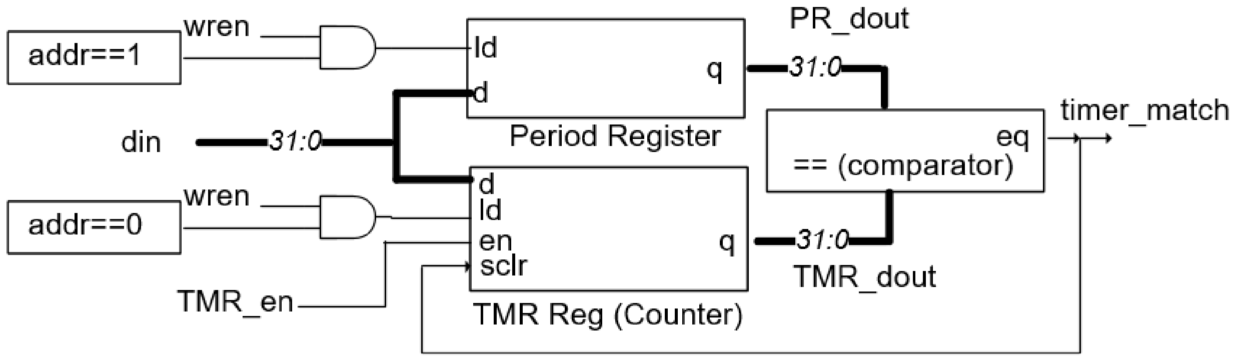
PART 1 DELIVERABLE: TIMER MODULE

Create a folder and a project named *lab7_p1* and finish implementing the *timer32.v* module that is found in the *lab7_p1_files.zip archive*. Verify both behavioral simulation and Post-route simulation using the *tb_timer32.v* testbench.

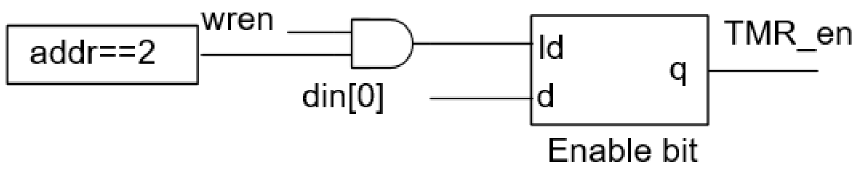
Fill out the requested information in the report.doc file.

Below are the various pieces that you have to implement in your timer module. This is the part 1 deliverable. In these pseudo schematics, the clock input is not shown to the registers.

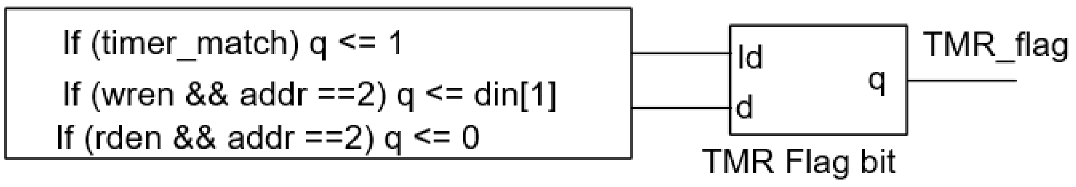
Timer and Period registers (the period register power reset value must be the value of the PERIOD parameter).



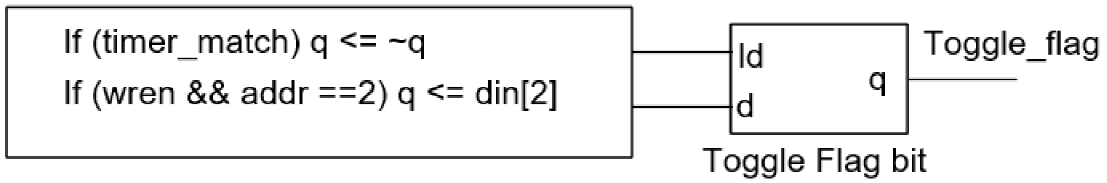
Status/Control register, bit 0 (Enable bit). 1-bit register that is loaded on write to address 2 from bit 0 of the databus. This is just a one-bit register.



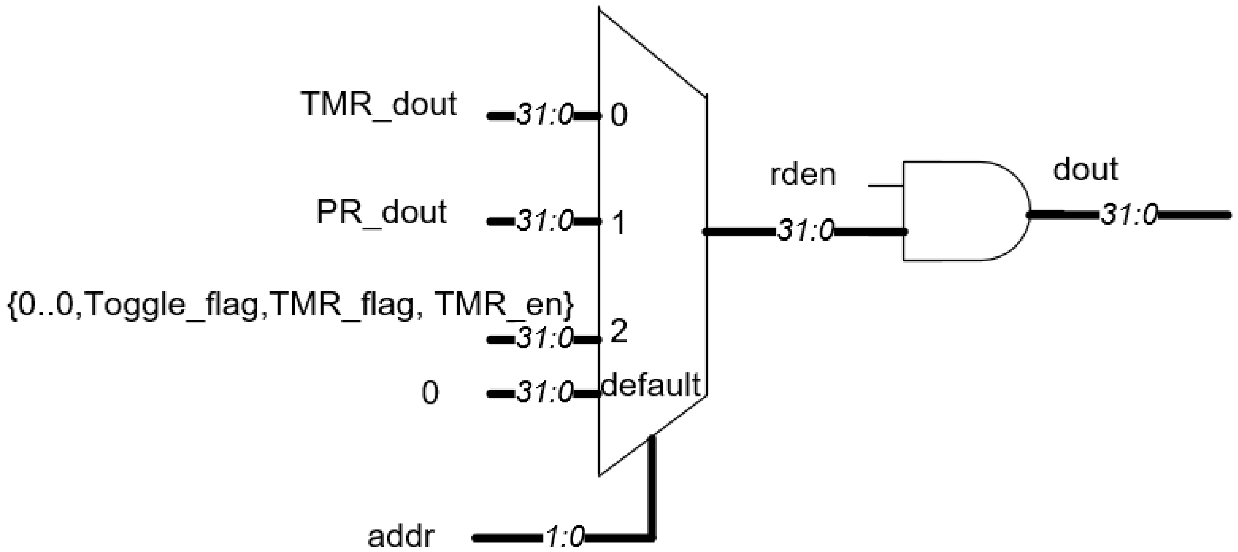
Status/Control register, bit 1 (TMR flag). 1-bit register that is set on timer match, written from the databus, and auto-cleared on read. This is just a one-bit register.


If (timer_match) q <= 1
If (wren && addr ==2) q <= din[1]
If (rden && addr ==2) q <= 0
ld
d
q
TMR_flag
TMR Flag bit

Status/Control register, bit 2 (Toggle bit). 1-bit register that is toggled timer match, written from the databus. This is just a one-bit register.



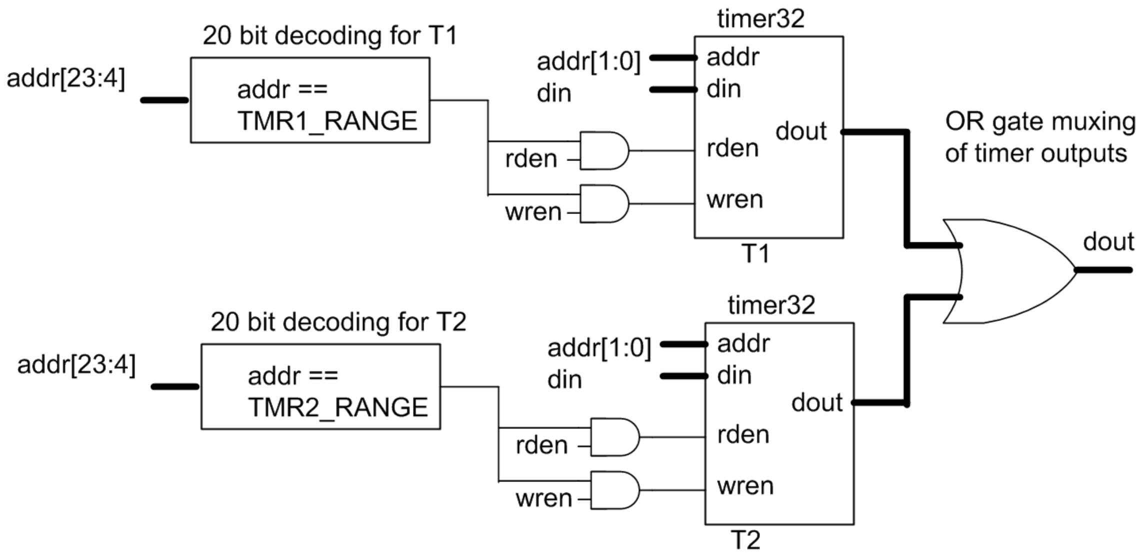
DOUT bus logic (this is best implemented with an **always *** combinational block, using a case statement for the mux, which is inside of an IF statement that tests *rden*. The default value for *dout* should be 0 at the beginning of the always block). Note that for addr == 2 (Control/Status register), the lower 3 bits are a concatenation of the Toggle/TMR/TMR_EN one-bit registers and the upper 28 bits are zero).



PART2 DELIVERABLE: INSTANTIATE TWO TIMER MODULES, SEPERATELY ADDRESSED

Create a folder and a project named *lab7_p2* and finish implementing the *timer32bus.v* module that requires instantiating two of the timer32 modules you completed in Part1 (see schematic below). The *timer32bus.v* Verilog file is found in the *lab7_p2_files.zip* archive. You will need to add your *timer32.v* module as a design source to the project.

After you have the timer module completed and tested, you are to implement two of these modules with address bus decoding as shown below.



The module interface for this logic in is shown below, the parameters `TMR1_RANGE` and `TMR2_RANGE` define the address ranges for Timer1, Timer2 respectively.

```
module timer32bus(clk, reset, din, dout, wren,rden, addr);
input clk, reset, wren,rden;
input [31:0] din;
output [31:0] dout;
input [23:0] addr;    //24 bit address

//20-bit decode, compare against addr[23:4]
parameter TMR1_RANGE = 20'h9250A;   //20 bit decode
parameter TMR2_RANGE = 20'h3C74D;   //20 bit decode
```

Verify both behavioral simulation and Post-route simulation of your *timer32bus* module using the *tb_timer32bus.v* testbench.

Fill out the requested information in the report.doc file.

PART 3: INSTANTIATING 7 TIMERS

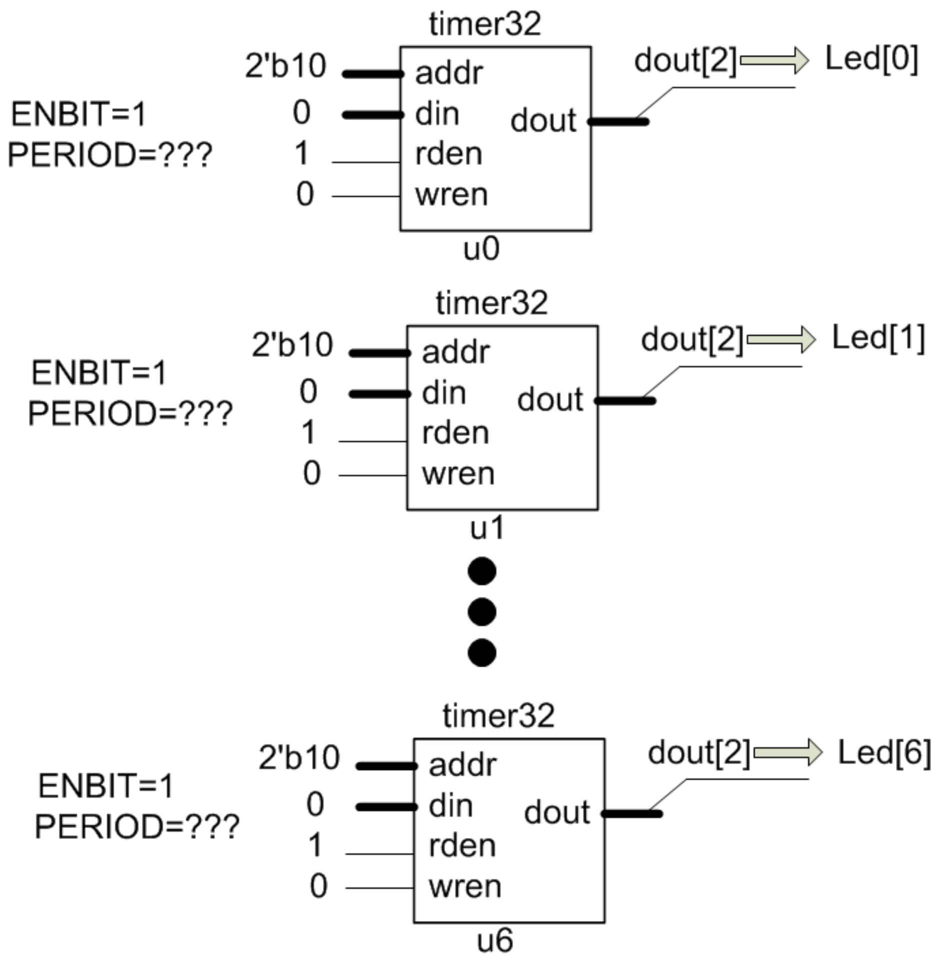
This requires you to create a design to be loaded into the Basys board. Create a project and folder named *timer_p3* . The zip archive has a file named *timer_p3.v* and within this file is the *timertop* module whose interface is shown below. Add this file and also your timer module file to the project as design sources. The zip archive also has a Basys constraints file, you need to add this to the project as a constraints file. There is no testbench to add to this project.

```
module timertop(LED,SW,board_clk);
output [6:0] LED;
input [0:0] SW;
input board_clk;
```

The *timertop.v* Verilog has an internal clock named ***clk*** that is 50 MHz generated from a Mixed Mode Clock Manager (MMCM) block named 'clk_wiz' (see previous lab for how to generate this IP block). The 50 MHz clock should be used as your system clock. This module also has code that generates a reset signal from SW0 on the Basys3 board.

You will need to add more Verilog code to the *timertop* module that instantiates seven copies of the *timer32* module as shown below. The address bus is hardwired to 2b'10 and the *rden* input to '1', thus outputting the status register on the output. Bit 2 of the output (the toggle bit) is tied to an LED. The ENBIT parameter of each timer module is '1', so the timers are running at power up. Set the PERIOD parameter such that the timer periods are 0.125 s, 0.25 s, 0.5 s, 1.0 s, 2.0 s, 4 s, and 8 s. Because the LED is tied to the toggle bit, it will be ON for the timer period and then OFF for the timer period (e.g., ON for 8 seconds, then OFF for 8 seconds). Visually, the LEDs will appear to count in a binary pattern. To check your design, use an online stopwatch to time the LEDs – LED6 should come on at 8 seconds, off at 16 seconds, on at 24 seconds, etc.

The clock input of the each timer is tied to *clk*, and the reset input to the switch 0 input (*reset*). Add the included *Basys3_Master.xdc* file to your project as was done in the first lab exercise in order to map this to the Basys board.



To get credit for this part, you need to demo this to the TA as well as submitting this project.

Associated files

The ZIP archives associated with this lab contains the following files:

- *timer32.v* -- complete this module (lab7_p1_files.zip)
- *tb_timer32.v* – test bench for timer 32 (lab7_p1_files.zip)
- *report.doc* – report file that needs to be filled out (lab7_p1_files.zip)
- *timer32bus.v* – complete this module for part 2 (lab7_p2_files.zip)
- *tb_timer32bus.v* – testbench for part 2 (lab7_p2_files.zip)
- *timer_p3*.v – top-level module for mapping to Basys3 (lab7_p3_files.zip)
- *Basys3_master*.xdc – constraints file for part 2 (lab7_p3_files.zip)

Submission

For submission, create a directory named 'lab7_*netid*', i.e., (lab7_rbr5).

Copy your *lab7_p1, lab7_p2, lab7_p3* project directories to this directory.
Copy your completed *report.doc* to this directory.

Create a ZIP archive of this directory and submit it.