Lab 7 – Counters/Dividers – Driving an IR Emitter Using *Parameterized Module* Design Techniques

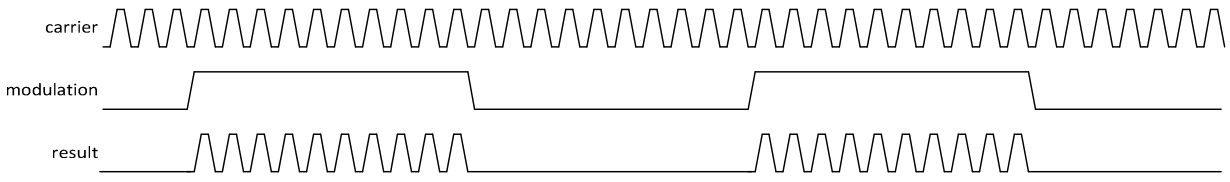
Introduction

The Infrared (IR) light spectrum is beyond the visible spectrum, with electromagnetic (EM) wavelengths in the 700 nm to 1 mm range. For communications it has been applied to short distance, relatively low bandwidth data transfer operations between electronic devices, such as laptops, cell phones, etc. One of its most common applications is transmitting information using remote control devices, for example: TV's, audio equipment and set top boxes.

A basic setup for IR communication is an *IR Emitter* for transmission, which looks and operates like an LED (except the light is invisible), and an *IR Receiver Module*. The IR receiver module is more than just a phototransistor. It has an amplifier, band pass filter and demodulator. This allows the receiver to only capture and amplify signals with a particular carrier frequency.

In this lab you will be designing a circuit that will modulate a signal to drive an IR emitter with characteristics that match a particular IR receiver module. This method is similar to those employed by typical commercial remote control devices. You will be using a counter/divider module that was introduced in the lecture videos.

You will be creating a design to create signals with two frequencies: a carrier frequency and a modulation frequency. The carrier frequency is the fundamental frequency, or the frequency of the carrier wave. The modulation frequency defines how this carrier wave is turned on and off (modulated). In our simple design, the signals with these two frequencies are logically **AND'ed** (multiplied) and the resultant output appears to be *bursts* of carrier frequency signals, with the *burst frequency* being the modulation frequency. The figure below illustrates this concept.



In actual IR devices there are different coding schemes used to transfer data, such that the information (zeros and ones) is translated into the width of the modulated signals, and transmitted serially. We will explore this more in later labs (or you can google *data formats for ir remote control* for more information). A common carrier frequency used for IR devices is 38 KHz.

Your design will be connected to an *IR emitter*, thus generating a 38 KHz modulated, invisible EM wave. It will consist of ten 38 KHz cycles on, ten 38 KHz cycles off, repeating this cycle as long as enabled with *switch 0*. You will first be monitoring this signal with an oscilloscope to see that it meets these specifications. Optionally, you can place your emitter close to a receiver and observe the demodulated signal.

To create these signals you will use a parameterized divider module. You will learn how to implement/instantiate different sized dividers using the same parameterized divider module.

Detailed Specification

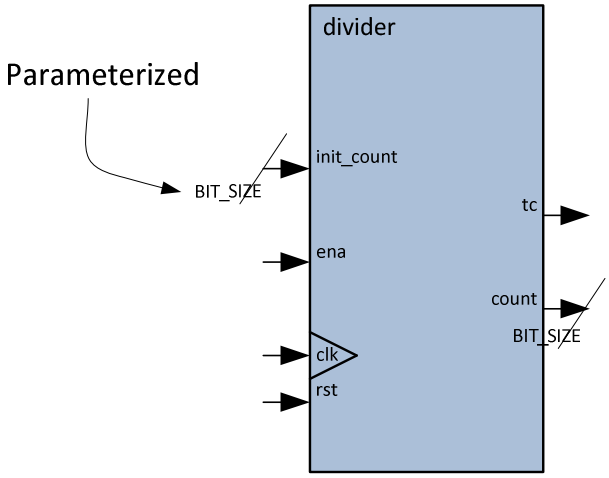
Parameterized module: *divider*

This design is very similar to that described in the videos, except that a ***count*** signal is added to the module output list.

- Output ***tc*** – terminal count; this signal gets driven high when the counter is enabled (***ena*** = 1) **and** the *counter registers are zero*.
- Output ***count*** – this is the bussed output signal from the D flip flops. This signal is ***BIT_SIZE*** wide.
- Input *parameter* ***BIT_SIZE*** – bit width of the counter. This *parameterizes* the divider module so that it can be easily reused for other size dividers.
- Input ***clk*** – main system clock (on the Basys3 board is **100 MHz**).
- Input ***rst*** – reset signal; an active high *priority* signal that at the next rising edge of ***clk***, loads the counter registers with the value on the data bus: ***init_count***.
- Input ***ena*** – enable signal; when this signal is high, the counter counts down by one at each rising edge of ***clk***; when the signal is low, the counter *holds* its current count.
- Input ***init_count*** – initial count; this is a bussed input signal whose value gets loaded into the counter registers when a) the ***rst*** signal is active, or b) the ***ena*** signal is active **and** the *counter registers are all zero*. It is ***BIT_SIZE*** wide.

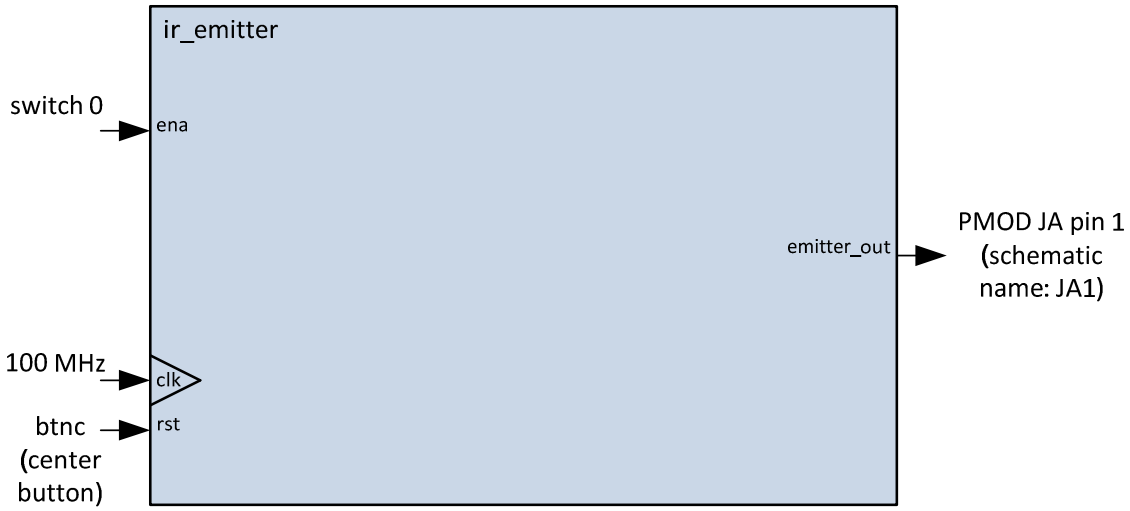
As an example to divide by 10, ***init_count*** should be set to 9, and thus will generate ***tc*** active high for one clock cycle every 10 clock cycles. (What should be the smallest ***BIT_SIZE*** defined for this example?)

A block diagram of this ***divider*** module is shown below.



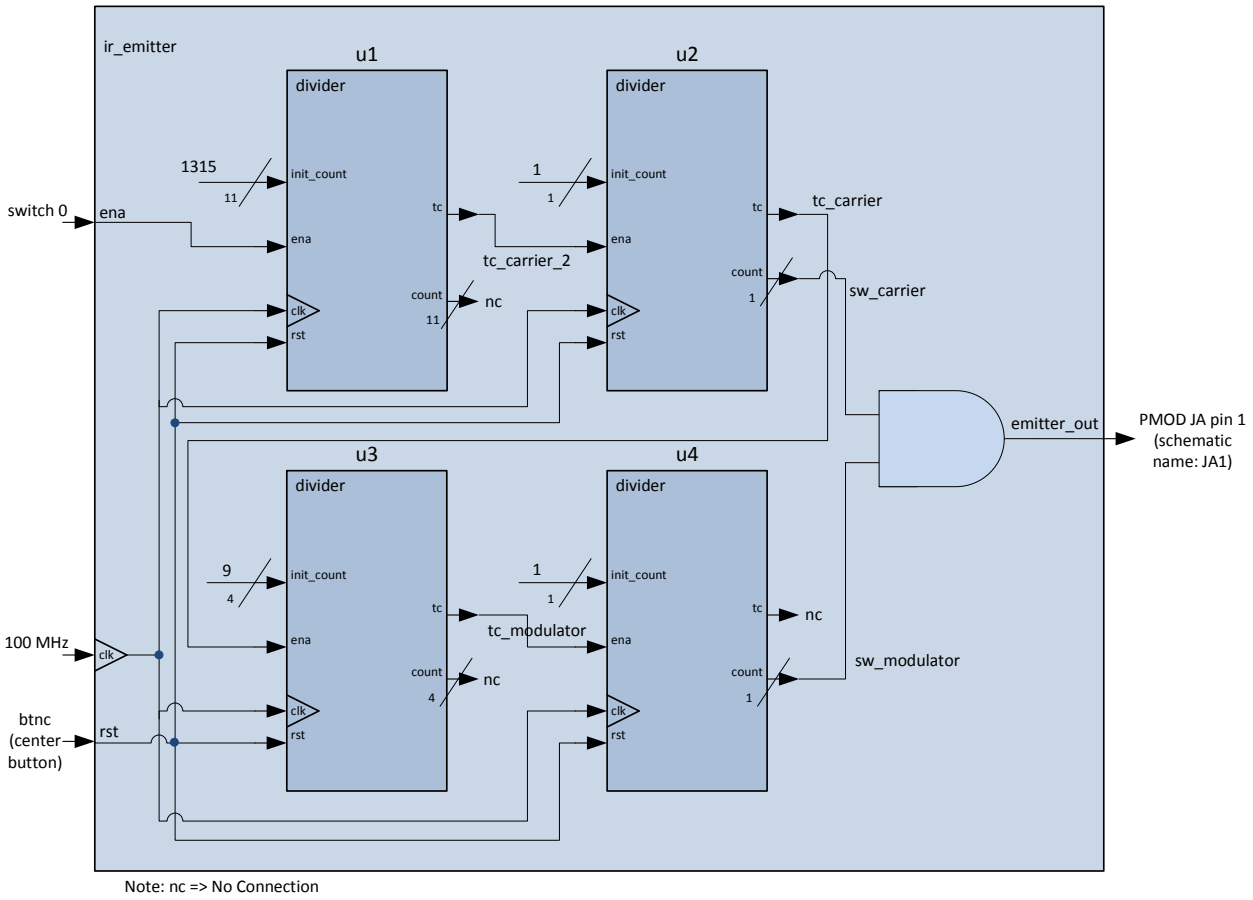
Module *ir_emitter*

The high level specification for this design is to create a modulated 38 KHz carrier wave, with *10 cycles on* and *10 cycles off*, repeating whenever *switch 0* is on. In addition to *switch 0*, other inputs to the design are a *100 MHz clock* and *reset* (*button center: BTNC*) signals. A high level block diagram is shown below. Remember, all of the *D flip flops* in this design are clocked with the same 100 MHz clock – so that we are working with a single clock domain (as discussed in the lectures and videos).



To create module ***ir_emitter***, one parameterized *divider* module needs to be designed. There will be four different implementations of this module using the same divider design, but with different settings of the ***BIT_SIZE*** parameter. The four different implementations effectively divide the *100 MHz* signal into the two signals described in the introduction. This module instantiates the ***divider*** four time to create the required 38 KHz modulated signal. The block diagram for the module is illustrated below.

As shown in the introduction, both the carrier and modulation signals need to be square waves (duty cycles equal to 50%). However, the ***tc*** output signal of the ***divider*** module does not have a 50% duty cycle (except for the unique case of dividing 100 MHz by 2). Thus, one way to create a 50% duty cycle is to create a divide by 2 divider module (which has only one D flip flop) and use the ***count*** output signal from this flip flop. So to generate a square wave, a pair of divider modules is required: the first module divides the clock by *n / 2*, creating a ***tc*** output signal that is twice the desired frequency. This signal is connected to the enable input of a second module that divides by 2 – creating the result of dividing by *n*. A block diagram implementing these ideas in module ***ir_emitter*** is shown below. (Note ***nc*** is not a signal name, but an abbreviation for ***No Connection***.)



The functionality of the four dividers is:

1. One divider (instance ***u1***) will be designed to generate a 76 KHz (38 KHz multiplied by 2) pulse train (***tc*** signal) as was illustrated in the lectures and videos. The internal ***tc*** signal is ***tc_carrier_2***. Note the ***BIT_SIZE*** for this module is 11, and the ***init_count*** is 1315. There are no connections to the output signal: ***count***. The enable (***ena***) signal in the

input list of the ***ir_emitter*** is connected to the enable signal of this instance. At the top level it is connected to ***switch 0***.

2. This signal from ***u1***, ***tc_carrier_2***, will then connect to the enable input of another divider (instance ***u2***) to create a square wave by dividing by 2. ***BIT_SIZE*** is 1 and ***init_count*** is 1 for this instance. By connecting ***tc_carrier_2*** to the enable signal, this divider will only count down for the one clock period that ***tc_carrier_2*** is active – or every 1/78 KHz. The ***tc*** output of this module (a frequency of 38 KHz), is connected to internal signal: ***tc_carrier***. The square wave signal, ***sw_carrier***, is the output ***count*** signal of this instance.
3. The third divider (instance ***u3***) will create the *10 cycles on* and *10 cycles off* modulation. ***BIT_SIZE*** is 4 and ***init_count*** is 9 for this instance. Its enable (***ena***) is controlled by the terminal count from the 38 KHz square wave generator, or ***tc_carrier***. Its terminal count (***tc_modulator***) signal needs to drive the enable of a final divider instance. The ***count*** output signal from this instance is not connected to anything.
4. The final divider (instance ***u4***) generates the modulation square wave signal (***sw_modulator***) by toggling from high to low and back to high for groups of 10 cycles of the carrier wave. ***BIT_SIZE*** is 1 and ***init_count*** is 1 for this instance, just as for ***u2***. The terminal count output (***tc***) is not connected to anything.

The square wave output signals (***sw_carrier***, ***sw_modulator***) are **AND'ed** together to produce the output signal: ***emitter_out***. A global reset signal is also connected to *button 0* (***btnC***). This resets all of the D flip flops in your design.

For completeness, a description of the top level signals for module ***ir_emitter***:

- Output ***emitter_out*** – modulated signal to drive an IR emitter. The top level i/o wrapper connects this to ***PMOD JA pin 1***.
- Input ***clk*** – main system clock (on the Basys3 board is **100 MHz**).
- Input ***rst*** – reset signal; connected to *button 0* and wired to all of the reset signals of the internal modules.
- Input ***ena*** – enable signal; connected to *switch 0* and wired to the enable signal of the carrier frequency counter.

Design Implementation

There are two design steps to complete this lab.

- Design the counter/divider module, ***divider.sv***, based on the above specification and the example work done in the lectures.

- Design the top level module, ***ir_emitter***, as per the block diagram shown in the specification section. It is basically four instantiations of the divider and one **AND** gate. I recommend a *continuous assign* statement for the **AND** gate.

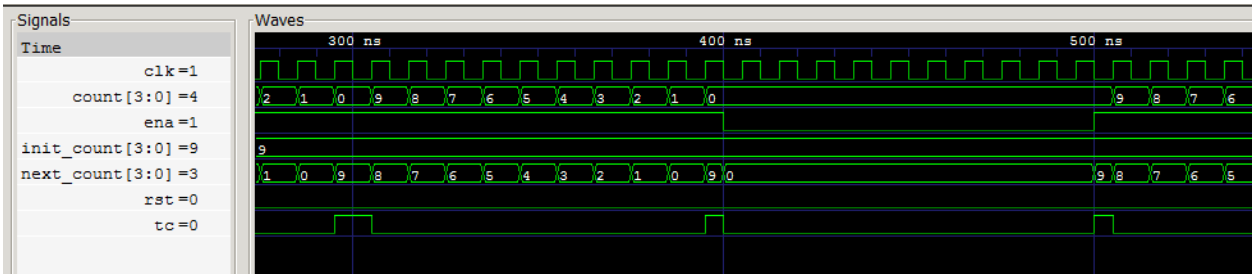
Simulation/Verification

There are two simulation command files (right click each of the *.sim files and Run) associated with the testbenches that match the two design steps above:

- ***tb_divider.sim*** is used to simulate the *divider* module. The testbench is set up for a divide by 10 operation. This is not a self checking testbench, but prints the transition times of the terminal count (***tc***) signal. A successful design will have the following output – note the change in the pattern between 395 and 505 nanoseconds:

```
Testing divide by 10 – view and analyze timing diagram.
Time for tc going  low: 0.000000 nsec
Time for tc going high: 95.000000 nsec
Time for tc going  low: 105.000000 nsec
Time for tc going high: 195.000000 nsec
Time for tc going  low: 205.000000 nsec
Time for tc going high: 295.000000 nsec
Time for tc going  low: 305.000000 nsec
Time for tc going high: 395.000000 nsec
Time for tc going  low: 400.000000 nsec
Time for tc going high: 500.000000 nsec
Time for tc going  low: 505.000000 nsec
Time for tc going high: 595.000000 nsec
Time for tc going  low: 605.000000 nsec
Time for tc going high: 695.000000 nsec
Time for tc going  low: 705.000000 nsec
Time for tc going high: 795.000000 nsec
Time for tc going  low: 805.000000 nsec
Time for tc going high: 895.000000 nsec
Simulation complete!!!
```

You will also have to look at the timing diagram to check your design. Download the *tb_divider.vcd* file and view the signals in the divider module with the GTKWave viewer as shown below. Note the differences in the *count* and *next_count* signals for *ena* equal 1 and *ena* equal 0. If it doesn't match the behavior below, your design is not correct. Save and insert a similar timing screenshot for your divider design in your lab report.



If you like you can change the divide by 10 function by trying other divisors by editing the testbench. Don't forget to change ***BIT_SIZE*** as needed.

- ***tb_ir_emitter.sim*** is used to simulate the *ir_emitter* module. This is also not a self checking test bench. A successful design will have the output:

```
Index            0: Frequency 37.993921 kHz
Index            1: Frequency 37.993921 kHz
Index            2: Frequency 37.993921 kHz
Index            3: Frequency 37.993921 kHz
Index            4: Frequency 37.993921 kHz
Index            5: Frequency 37.993921 kHz
Index            6: Frequency 37.993921 kHz
Index            7: Frequency 37.993921 kHz
Index            8: Frequency 37.993921 kHz
Index            9: Frequency 3.453993 kHz
Index           10: Frequency 37.993921 kHz
Simulation complete!!!
```

This testbench is measuring the time between positive edges of the *emitter_out* output signal. It then inverts the measurements to display the equivalent frequency for each positive edge-to-positive edge cycle. The first 9 lines display the differences of the first burst of 10 cycles. The 10th line (index 9) is the difference between the 10th and 11th cycles, which is the gap between the bursts of 10 cycles. This time should be 11 times longer, or have a frequency 11 times smaller than the carrier frequency (since the time between positive edges for the off time is 10 off cycles plus one).

If your simulation does not ***exactly*** match this display, debug and modify your design until it does.

Synthesis

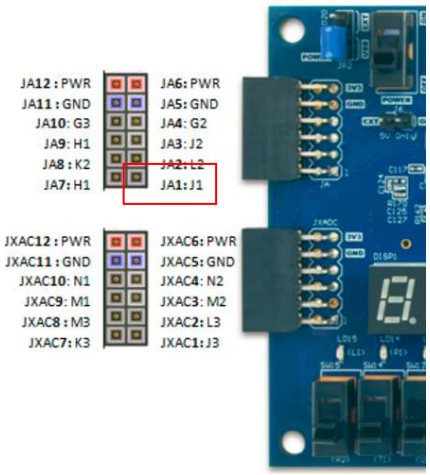
Generate a bit file as you did in past labs (right click and Run on: *lab7_top_io_wrapper.tcl*). For this lab it will be called: ***lab7_top_io_wrapper.bit***. Copy it to your memory stick to load and run on your Basys3 board.

The ***rst*** signal is connected to the center pushbutton. The ***ena*** signal is connected to ***switch 0***.

In your lab writeup justify why the ***init_count*** for the carrier signal divider is equal to 1315.

Hardware testing steps:

a) Using a breadboard connected a *PMOD interface* or other connections as per the TA's instructions, probe ***PMOD JA pin 1*** with an oscilloscope to see that you have a modulated signal similar to your waveform simulation. Measure the carrier frequency and the number of cycles in the *on burst*.



Optional Evaluation with a Receiver (as per TA's instructions)

b) Connect an *IR Emitter* to this pin (long lead is the anode (+ side)). Connect the cathode to ground.

c) Build the IR receiver on the breadboard using the power connections from the Basys3 board. You will need the *IR receiver module*, a resistor and a capacitor (see *Application Circuit* below). Do not connect its output to anything; just use a second channel on the oscilloscope to monitor the demodulated output signal. Measure the modulation periods for the on time and off time. Also measure the time shift from the start of transmission of the carrier burst on the *IR emitter* to when the *IR receiver module* turns on/off its output. Because of the characteristics of the IR emitter signal, it will be more stable to trigger on the IR receiver module's output to make these measurements.

Transmitter/Receiver Hardware

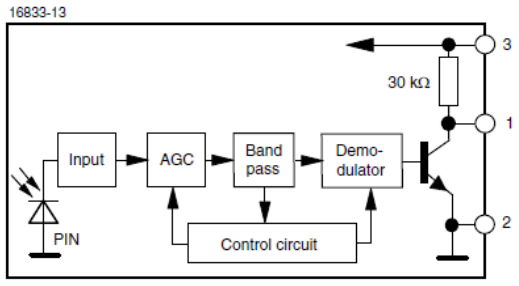
As an emitter you will be using a Vishay *High Power Infrared (940 nm) Emitting Diode*, (*TSAL4400*). It will be driven directly from the ***PMOD JA pin 1*** connection.

On the receiving side, you will use a Vishay *IR receiver module* (*TSOP 38238*). It has a peak EM sensitivity at 940 nm. Its internal circuitry is designed to match to a carrier frequency of 38 KHz. The data sheets for these devices are available online. A pertinent section of the data sheet for the receiver is shown below. Note in the *Block Diagram* the input signal from the photodiode first goes through an automatic gain control (AGC) amplifier to normalize the input signal to an appropriate gain. Then follows a bandpass filter which is designed for a 38 KHz center frequency. The demodulator block removes the 38 KHz carrier frequency and preserves the modulation frequency. The modulated signal finally drives an output transistor driver. Note the internal pull up resistor. With no carrier signal, the output is high. When a carrier signal is detected, the output is low (the transistor is switched on) – thus an active low signal.

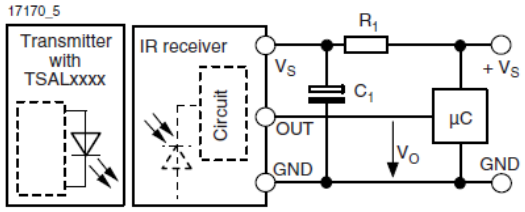
The *Application Circuit* illustrates a typical interface to a microcomputer – or an equivalent digital system (i.e., FPGA). A resistor and capacitor should be added to the interface as shown. You will not be connecting this to the FPGA for this lab, only monitoring it with an oscilloscope.

CARRIER FREQUENCY	(AGC2)		(AGC4)	
	PINNING			
	1 = OUT, 2 = GND, 3 = V_S	1 = OUT, 2 = V_S, 3 = GND	1 = OUT, 2 = GND, 3 = V_S	1 = OUT, 2 = V_S, 3 = G
30 kHz	TSOP38230	TSOP39230	TSOP38430	TSOP39430
33 kHz	TSOP38233	TSOP39233	TSOP38433	TSOP39433
36 kHz	TSOP38236	TSOP39236	TSOP38436	TSOP39436
38 kHz	TSOP38238	TSOP39238	TSOP38438	TSOP39438
40 kHz	TSOP38240	TSOP39240	TSOP38440	TSOP39440
56 kHz	TSOP38256	TSOP39256	TSOP38456	TSOP39456

BLOCK DIAGRAM



APPLICATION CIRCUIT



R_1 and C_1 are recommended for protection against EOS. Components should be in the range of 33 Ω < R_1 < 1 kΩ, C_1 > 0.1 µF.