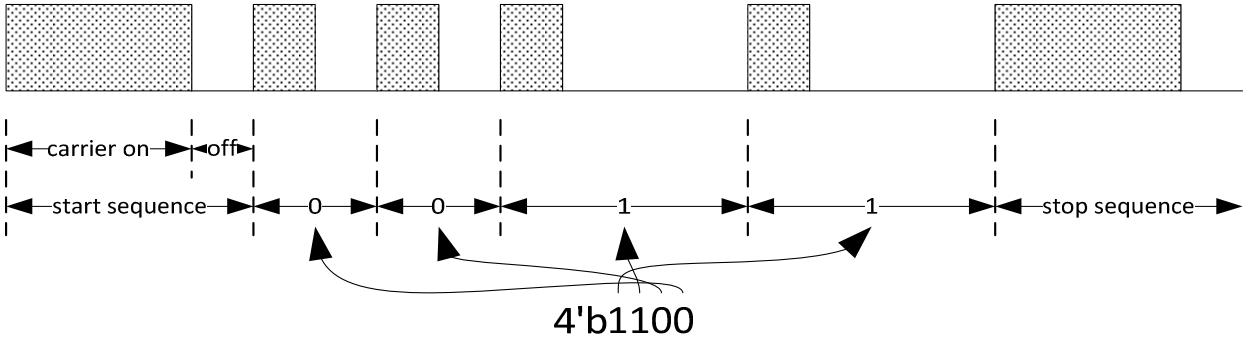
Lab 10 – Pulse Distance Coding Part 2
Designing a Controller

Introduction

In Lab 8 you designed a *datapath* for a Pulse Distance Coding transmitter. In this lab you will be designing a Finite State Machine (*FSM*) which is the main component of a *controller* for this application. This controller will wait for an external push button, and then send the control signals that will serially transmit 4-bits of data to an IR emitter. A unique start and stop sequence will be added to the NEC R8C/27 protocol. The figure below illustrates an example pattern for a 4'b1100 data transmission with a unique start and stop sequence defined for this lab. Refer to Lab 8 for the specifics on the timing for the zero and one pulse distance codes.



The 4-bit data pattern will be defined by four switches on the Basys 3 board. You will test your design by implementing it in hardware, and transmitting an IR signal to a receiver design that the TA will have.

You will be reusing several modules from previous labs.

Detailed Specification:

A block diagram for the top module, ***ir_emitter_top***, is illustrated below. Note there are 4 inputs:

a) the clock,
b) the reset signal (center button),
c) the start signal (left button) and
d) four switches to provide 4-bits of data.

There is one output, the emitter. This will be connected to an IR emitter on the Basys 3 hardware. The signal ***(nc)*** means there is ***no connection*** to the output signal of the instance.

The main input to the FSM, ***ir_emitter_fsm***, is a start signal (pulse), ***strt***, which controls the start of the FSM and thus, the serial data transmission. This signal is generated by pressing the left button (***btnL***), which first is first synchronized using two *D flip-flops*. The synchronized signal then drives the ***btn_in*** signal on ***u_bdb***, the *pulse button debounce* module we developed in the lectures/videos. When receiving the ***btn_in*** signal, this module outputs a ***zero_to_one*** signal (one clock wide) that will start the FSM. (The ***one_to_zero*** output is not connected (***nc***).)

The debounce interval time, ***dbt***, is generated from a *divide by 1,000,000* divider module from a previous lab. This will generate a stream of pulses separated by 0.01 second – effectively debouncing the button press.

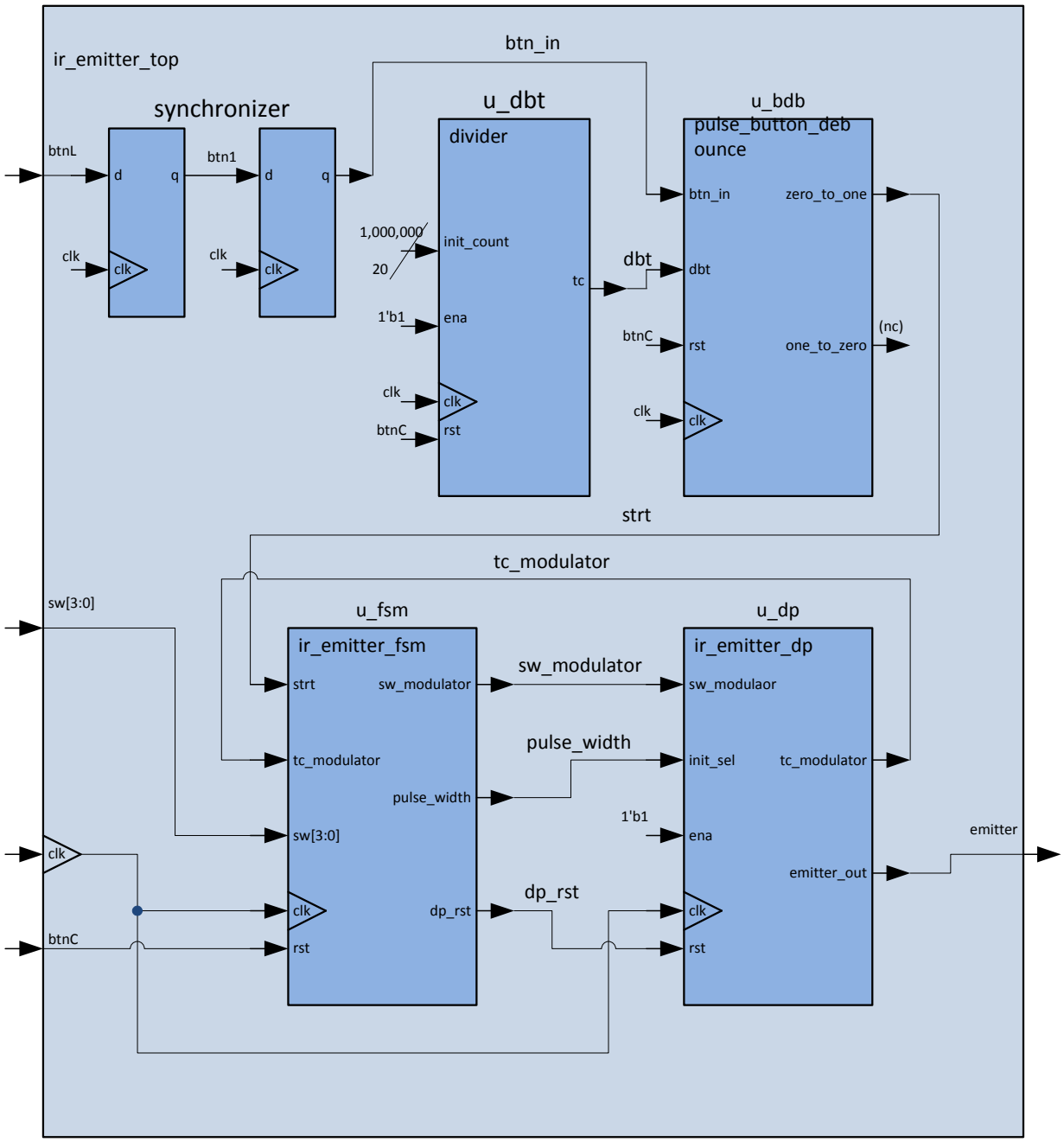
The center button (***btnC***) supplies the reset signal to all modules. It is not debounced for this lab.

The ***ir_emitter_fsm*** module provides three output control signals (***sw_modulator***, ***pulse_width*** and ***dp_rst***) that connect to the *datapath* module, ***ir_emitter_dp***. The main inputs to this module are:

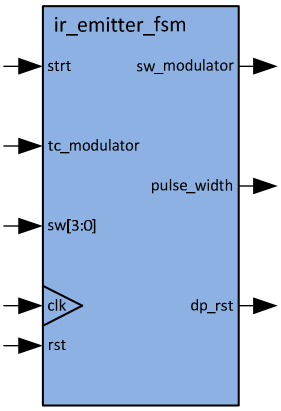
a) the start button pulse,
b) the 4-bit data value and
c) the ***tc_modulator*** signal from the *datapath* module (from Lab 8).

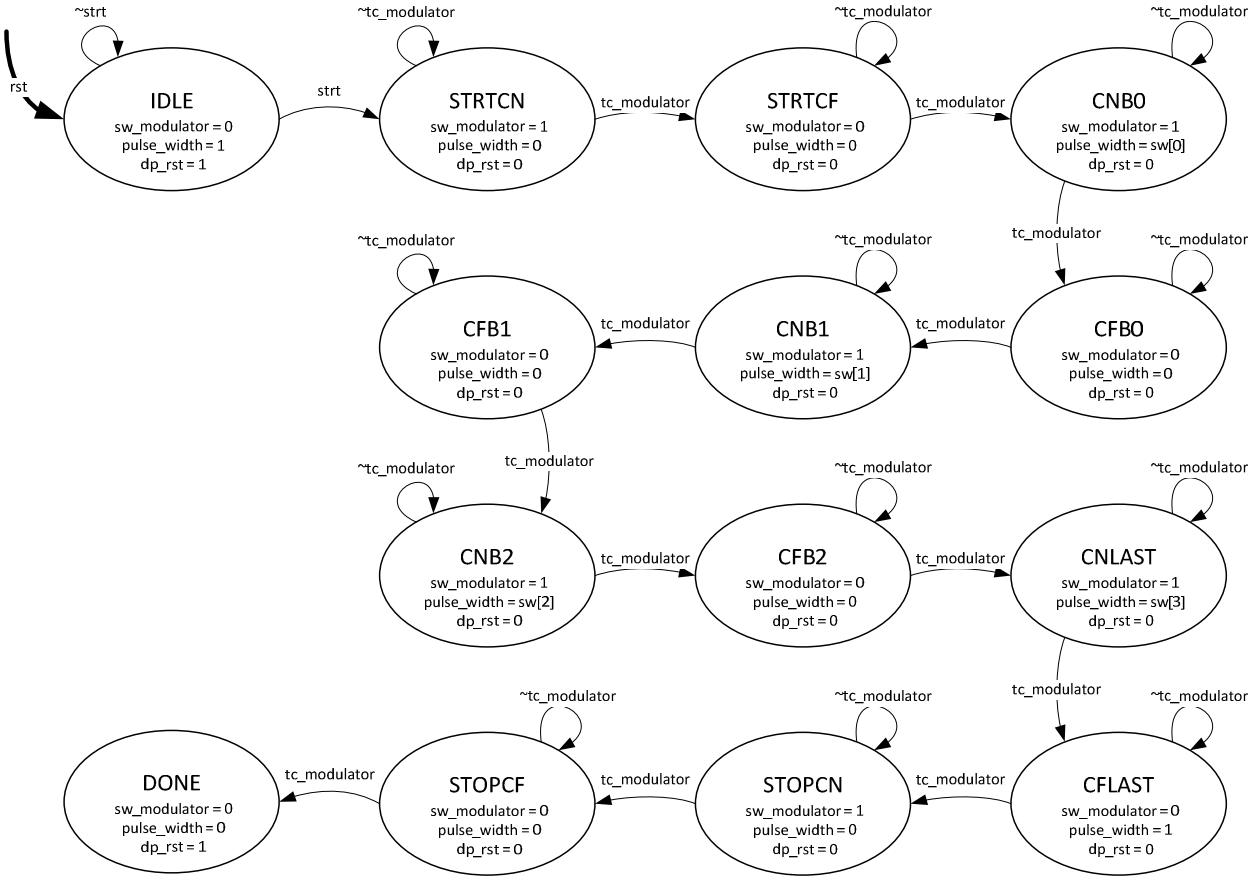
More details of the inner FSM design are provided below.

Finally, the datapath module, ***ir_emitter_dp***, supplies the ***emitter*** signal based on the control signals supplied by the FSM.



The details of the FSM are supplied in the block diagram, state machine diagram and table below. Except for IDLE and DONE, note how the states are in defined pairs for turning the carrier on (**CN**) and turning the carrier off (**CF**). The pulse width, ***pulse_width***, is defined one state ahead of when it is needed, as it is loading an ***init_value*** from the multiplexer (see previous lab) when the ***tc_modulator*** signal is pulsed.





State	Description	Details
IDLE	Idle	Arrives here after a reset. Activates the reset signal on the datapath module (***db_rst***). Sets the pulse width for the next state to be *long*.
STRTCN	Start, carrier on	Turns the carrier on for the *long* period.
STRTCF	Start, carrier off	Turns the carrier off for the *short* period.
CNB0	Carrier on, bit 0	Turns the carrier on for the *short* period.
CFB0	Carrier off, bit 0	Turns the carrier off for a period defined by switch 0 (0 => short; 1 => long).
CNB1	Carrier on, bit 1	Turns the carrier on for the *short* period.

State	Description	Details
CFB1	Carrier off, bit 1	Turns the carrier off for a period defined by switch 1.
CNB2	Carrier on, bit 2	Turns the carrier on for the *short* period.
CFB2	Carrier off, bit 2	Turns the carrier off for a period defined by switch 2.
CNLAST	Carrier on, last bit	Turns the carrier on for the *short* period.
CFLAST	Carrier off, last bit	Turns the carrier off for a period defined by last switch (switch 3). Sets the pulse width for the next state to be *long*.
STOPCN	Stop, carrier on	Turns the carrier on for the *long* period.
STOPCF	Stop, carrier off	Turns the carrier off for the *short* period.
DONE	Done	Stays in this state forever – until a reset is activated. Activates the reset signal on the datapath module.

Design

You will be using modules from your previous labs: ***ir_emitter_dp***, and ***divider***. These will be automatically linked in the simulation and synthesis scripts.

You will be provided the ***pulse_button_debounce*** module and the ***ir_emitter_top*** module with the synchronizer and all modules instantiated.

A skeleton file is provided for the Finite State Machine module, ***ir_emitter_fsm***. This is the only module that you need to design.

Simulation/Verification

In the simulation view, there is one test bench wrapper for the above module: ***tb_ir_emitter_top.sv***. This file should not be edited.

Verification step:

- ***ir_emitter_top.sim*** is used to simulate the *ir_emitter_top* module.

The simulation will take about several seconds to complete, as in the front end it must simulate 0.02 seconds of actual run time to debounce the start button.

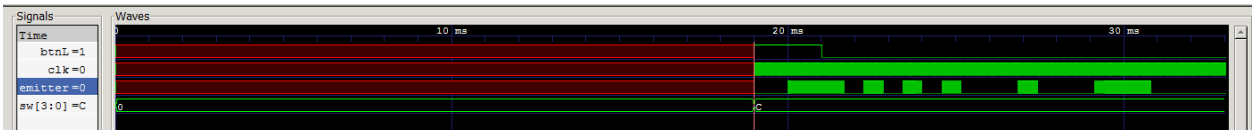
This is *not* a self checking testbench. It measures what I call the *Modulation delta time*. This is the time between 38 kHz bursts – i.e., the off times of the emitter's carrier frequency. The first one is about 0.02 seconds (20000000 ns), as it waits for the button debounce time (x2). The remaining 4 delta times match the pattern for a 4'b1100 switch setting (refer to the first figure in this write up).

Your output listing in your ***tb_ir_emitter_top.log*** file should match the following:

```
## run 35000us
Modulation delta time: 20000000 ns
Modulation delta time: 605360 ns
Modulation delta time: 605360 ns
Modulation delta time: 605360 ns
Modulation delta time: 1710800 ns
Modulation delta time: 1710800 ns
Simulation complete!!!
```

You should also look at your timing diagram from the simulation. (Download the ***tb_ir_emitter_top.lxt2*** file and view it in your local *gtkwave* waveform viewer.) It should appear representative of the figure below. To speed up the simulation and reduce the waveform file size, the signals from time 1,120 ns to 19,001,120 ns (waiting ~20 msec = 2 x debounce times) are not written to it. If you need to see signals in this time region, comment out the ***$dumpoff;*** and ***$dumpon;*** instructions in the testbench file: ***tb_ir_emitter_top.sv***. The missing signal region is shown below in red.

The simulated 4-bit input data pattern is 4'b1100. You should see this pattern serially represented in your timing diagram as show below – with the start and stop sequence.



If it does not match, there are likely problems with your FSM design. You should look at the signals specific to the FSM module (***u_fsm***). Correct your design and simulate until it matches the pattern.

Synthesis

Generate a bit file as you did in past labs (right click and Run on: *lab10_top_io_wrapper.tcl*). For this lab it will be called: ***lab10_top_io_wrapper.bit***. Copy it to your memory stick to load and run on your Basys3 board.

The center button is the reset, and left button is the start (transmit). Connect an IR emitter supplied by the TA to your PMOD interface. Connect the emitter anode to **JA pin 1**; the cathode to ground **JA pin 5 or 11**.

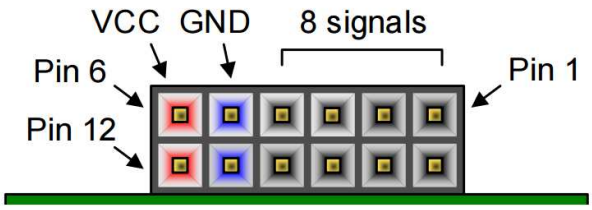


Figure 20. Pmod ports; front view as loaded on PCB.

Your TA will have a receiver module that will test your design. Try it with several different switch combinations. Recall from the state diagram that the FSM is locked in the DONE state once the single transmission is complete. So before trying another code, you have to reset the design.

If you think you are having hardware problems even though the simulation looks good, you can probe **JA pin 1** with an oscilloscope. However, because it is not a continuous signal, you need to set the oscilloscope to run in a single trigger mode.