

EXPERIMENT

5

CONTROL UNIT DESIGN

VERSION S07

In this experiment, you will design a finite state machine controller, and integrate it into a design using the logic functions you have created previously. After performing this experiment, you should be able to:

- 1) Design a one flip-flop per state control unit to implement a given algorithmic state machine (ASM) diagram,
- 2) Use simulation to test the operation of your controller independent of other circuit modules, and
- 3) Integrate your controller into a fully functional design, performing testing and debugging as needed to ensure proper operation.

5-1 PRELAB

In this prelab, you will be designing a state machine to act as a control unit for a frequency counter. Perhaps the most obvious way to implement a frequency counter is to simply count signal changes ($C = \text{number of } 0 \rightarrow 1 \text{ edges}$) for a fixed period of time (T), then the signal frequency can be calculated as $F_{\text{SIGNAL}} = C / T$. We have already created all of the major building blocks (i.e. BCD counters and decoders, divide-by-N counter) that we will need for this in previous labs, now we need to design and implement a state machine to control the operation of those blocks. In particular, we will be implementing the controller as a state machine using the one FF per state method (also referred to as a one-hot controller); as preparation for this prelab and lab you should first review Mano and Kime [1] pages 406-410.

Fewer step-by-step instructions are provided for tasks you have completed in earlier labs, as you should be becoming fairly accomplished in using the Altera Quartus II toolset. The issues of tool complexity and the learning curves associated with the tools are something that you will deal with routinely as a design engineer. Since we will often be designing with the most recent technology, the tool sets are often immature and less than robust. So, we will need to be able to fight our way through the various errors and omissions (and our own self-inflicted troubles) that we will discover in our hardware, software and documentation in order to bring our design to fruition. If you need to, review the previous labs as a reminder of how to accomplish tasks that you encounter in this lab.

OVERVIEW

This lab will build on the functional blocks that you created in experiments 3 and 4, and we will add a finite state machine (FSM) to create a useful machine. We will be developing a frequency counter based on the block diagram shown below in Figure 1. The blocks that are highlighted in thicker blue lines are logic that you have already designed or that will be provided for you. All of the items in the thicker blue blocks are contained in a schematic sheet `FREQCNT.BDF` that will be provided for you to use. The contents of `FREQCNT.BDF` are shown below in Figure 2. Although the logic is largely complete, you will still need to add synchronizing flip-flops to the external inputs, add circuitry to generate a pulse on input transitions, and make the various connections to your controller. You will design your controller on a separate sheet (`CONTROL.BDF`), then create a symbol for it, add that symbol `FREQCNT.BDF`, and make the necessary connections.

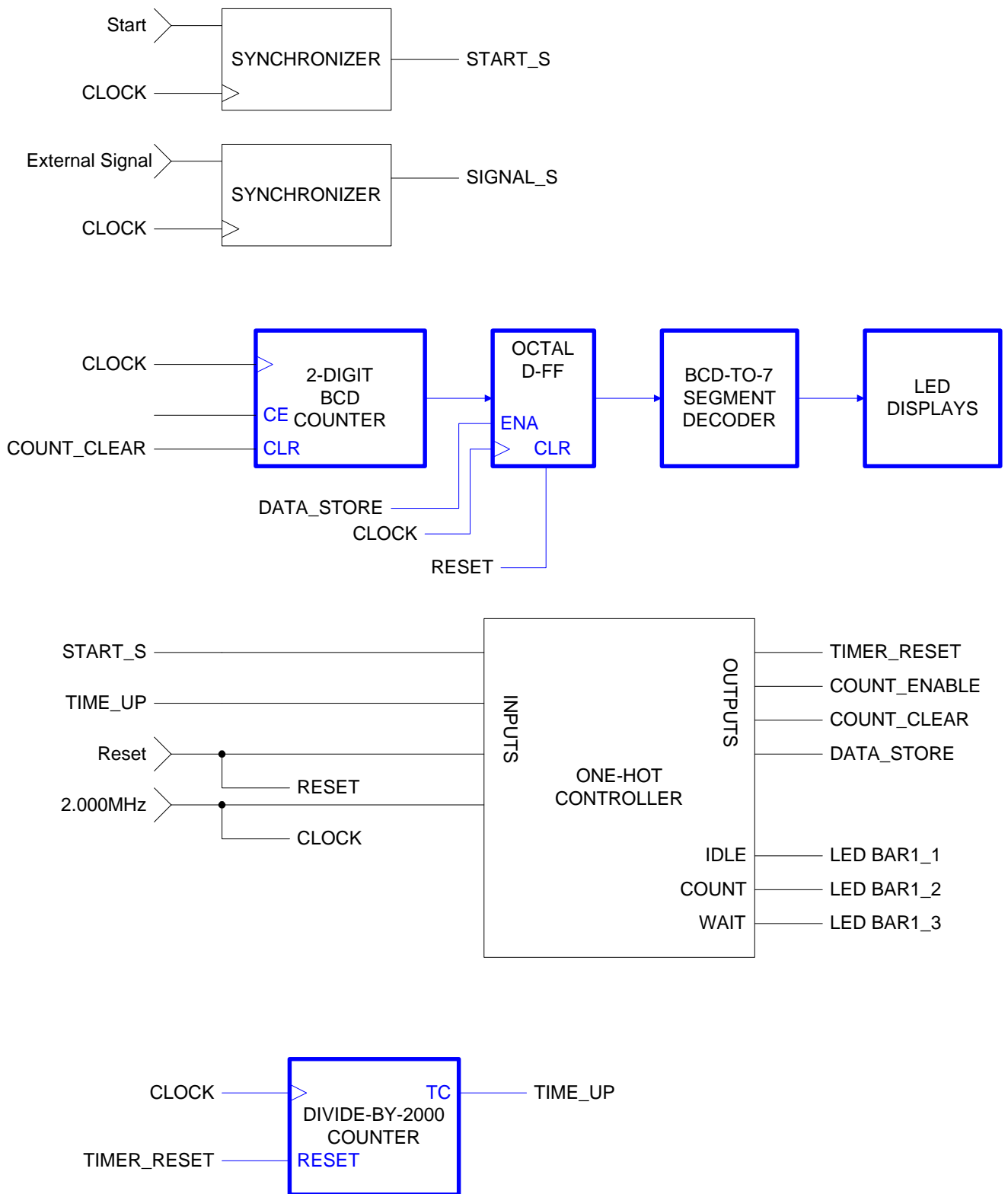


Figure 1 Frequency Counter Block Diagram

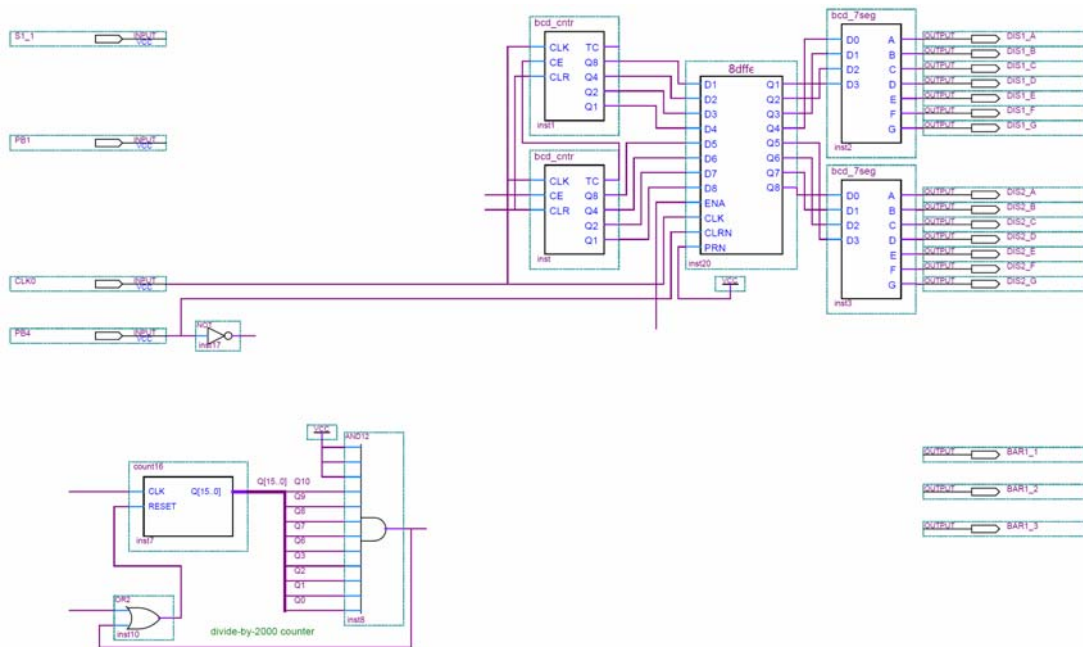


Figure 2 FREQCNT.BDF

One new issue that we will have to deal with now is the fact that the signal that we will be trying to count will not be synchronized to our system (FPGA) clock signal. So, it is possible that it could change at any time, with potentially unpredictable results. Consider the input *Start* in Figure 1. It will be used as a signal in the next-state logic for two state flip-flops. If it were to change at the wrong time, it is very possible that it could be read as a 1 at one flip-flop and a 0 at the other. (This would be due to the fact that the delay in the two signal paths is not likely to be identical.) This would cause our state machine to enter an invalid state. So, before using *Start*, we synchronize it (creating *START_S*) so that the signal to the next-state logic can only change at a known time (right after the clock edge).

In order to count the signal transitions on the frequency counter input, you will need to generate a pulse every time the signal transitions from a 0 to a 1. This pulse will then be used in conjunction with the controller's COUNT_ENABLE signal to enable the BCD counter, causing the count to increment by one on each 0→1 transition of the input signal. This 0→1 transition detection can be done with a very simple state machine (just two states), and one input and one output. You are not required to place the circuitry for this state machine until the in-lab portion of the experiment, but you are to prepare and turn in a state table, input equation, and output equation for it with your prelab report. (Of course, it always wise actually implement your circuit and run a quick simulation to test it before handing anything in.)

Since a complete discussion of the issues surrounding asynchronous signals is beyond the scope of this lab, you will simply implement a commonly used synchronization scheme. Anytime you are designing synchronous logic that has asynchronous signals as inputs, you need to ensure that they are synchronized in some way so that they cannot change during the set-up and hold time requirements of the synchronous system's flip-flops. Synchronizers are constructed by using flip-flop chains clocked by the synchronous clock, as shown below in Figure 3 for a two flip-flop synchronizer – *in this lab you will use only a single flip-flop*. For the frequency counter, we will need to synchronize both the input signal (that we are measuring the frequency of) and the input from PB1 (*Start*) that we are using as a control input to our state machine. Note that although the flip-flops are drawn in Figure 3 without set/clear inputs, you will want to use set/clear inputs as appropriate to ensure the initial condition of the flip-flops is as specified in the project requirements.

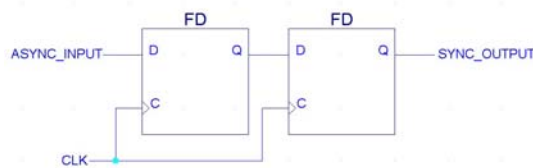


Figure 3 Synchronizer

One more thing we will need to do is to add a synchronous reset capability to the divide-by-2000 counter that you designed in Experiment #4. This is a very simple change, we simply OR the external reset input with the reset logic, as shown below. Note that the logic in FREQCNT.BDF already incorporates this change.

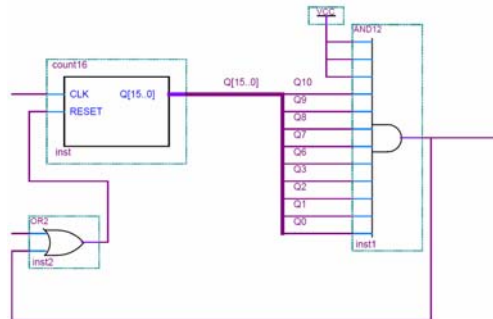


Figure 4 Divide-by-2000 Counter with Reset Input

PRELAB DESIGN AND SIMULATION

To begin your controller design;

- 1) Create a new Quartus project named **FREQCNT**. Copy the files `bcd_7seg.bdf`, `bcd_cntr.bdf`, and `count16.v` from your earlier lab work into the project directory. Update the pin assignments for the project as done for previous labs.
- 2) Download the files `FREQCNT.BDF` and `CONTROL.BDF` (you will need to unzip these files from `lab5.zip`) from the course web page and place them into your project directory. `CONTROL.BDF` contains all of the required inputs and outputs for your controller.
- 3) The schematic sheet `CONTROL.BDF` is where you will place all of your logic to implement the state machine design, and then you will create a symbol from this file and place the symbol into `FREQCNT.BDF`.
- 4) The required external signals are summarized below. Also, note that PB1 and PB4 are active-low, so you will need to invert these signals.

Type	Signal name	Net name	FPG A Pin	Purpose
INPUT	2.000MHz	CLK0	10	Master clock for FPGA
INPUT	Start	PB1	28	Signal to initiate a frequency capture
INPUT	Reset	PB4	35	Master reset signal, when asserted it should place controller in state IDLE and reset all synchronizer flip-flops to 0.
INPUT	External signal	S1_1	36	External signal to be measured – you must place S1-1 in the OFF position.
OUTPUT	IDLE	BAR1_1	48	Indicator for state IDLE, connects to Q_{IDLE}
OUTPUT	COUNT	BAR1_2	49	Indicator for state COUNT, connects to Q_{COUNT}
OUTPUT	WAIT	BAR1_3	50	Indicator for state WAIT, connects to Q_{WAIT}

CONTROL UNIT LOGIC DESIGN

The control unit you design will operate as specified in the algorithmic state machine (ASM) diagram shown below in Figure 5. There are three states;

IDLE – both counters are held in reset, and the controller will remain in this state until PB1 is pressed.

COUNT – the BCD counter is enabled, and the controller remains in this state until the divide-by-2000 counter reaches its terminal count

WAIT – the controller remains in this state until PB1 is released. In WAIT, the DATA_STORE output is one to clock the BCD counter data into the register between the BCD counters and the decoders.

Since there are three states, you will need to use three flip-flops. Although the one flip-flop per state design methodology may not use the minimal number of flip-flops, it does greatly simplify the design required for next state and output logic. It often will result in the fastest implementation of a state machine, but that is not an issue here. The basic premise of the one flip-flop per state design methodology is that only a single state flip-flop will contain a logic 1 at any time, and that the flip-flop with the 1 in it is by definition the current state. So, the input logic for each state flip-flop simply reflects that there are only two times when we want to place a 1 on the D input of the flip-flop;

1. The flip-flop currently has a 1 (it is the current state), and the conditions for moving to the next state have not been met, or
2. The previous state flip-flop has a 1, and the conditions for moving to the next state have been met.

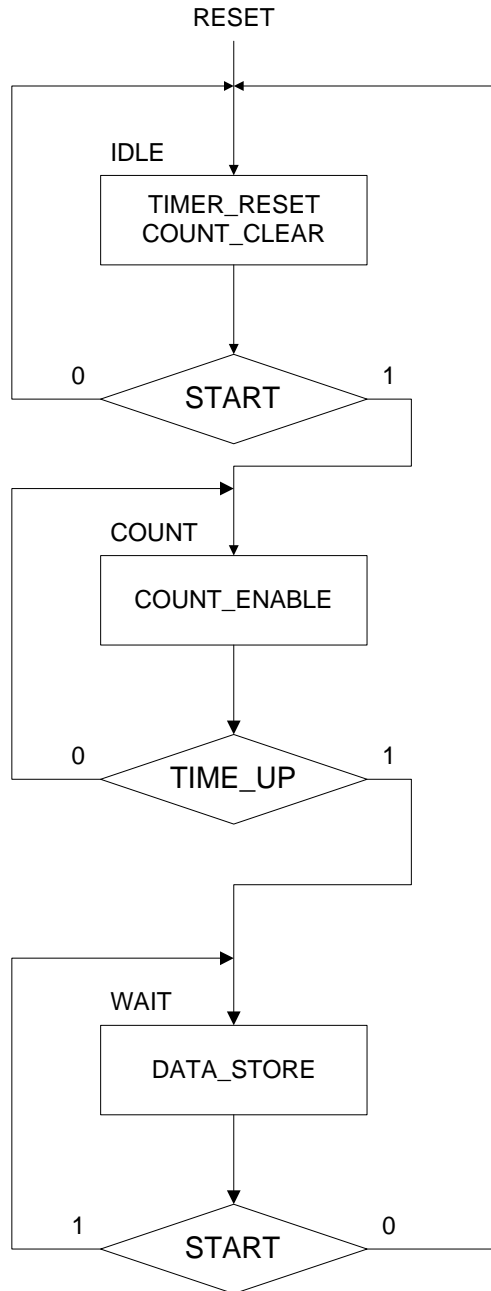


Figure 5 Frequency Counter ASM Diagram

Now, let's get this controller designed;

1. Using the ASM diagram, develop the expressions that correspond to the two conditions specified above, and then write the expression for each D_{STATE} . A table has been provided for you on the last page of this document to fill in and turn in with your prelab report.
2. Using the ASM diagram and the block diagram, develop the expressions for the controller output signals $TIMER_RESET$, $COUNT_CLEAR$, $COUNT_ENABLE$, and $DATA_STORE$. A table has been provided for you on the last page of this document to fill in and turn in with your prelab report.
3. Place the required circuitry in your $CONTROL.BDF$ schematic to implement the required functionality.

CONTROL UNIT SIMULATION

Now, you will simulate just the controller to see if it responds properly to its input signals;

1. Add CONTROL.BDF to your project and make it the top level entity.
2. Open the simulator tool.
3. Add the input signals (START, TIME_UP, RESET, CLOCK) and output signals (TIMER_ RESET, COUNT_ CLEAR, COUNT_ENABLE, DATA_STORE, IDLE, COUNT, WAIT)
4. Create waveforms for the inputs and run a functional simulation to show that your controller behaves as shown in the ASM diagram.
5. Print copies of the simulation trace as needed to show the correct operation of your controller. Annotate the traces to indicate specifically how/when you are demonstrating proper control unit operation.

1. What are the significant differences between using an asynchronous clear versus a synchronous reset on a FF?
2. Why don't we need to synchronize the RESET input?

EXPERIMENT 6 PROJECT PROPOSAL

Review the Experiment 6 project requirements. You will be required to furnish a more formal project proposal with the Experiment 6 prelab report, but if your project idea ends up being unsuitable for one reason or another your team may end up having to start over after doing significant work on it. To minimize that sort of thing, **as a team**, you will submit a single page document containing a narrative and/or diagrammatic description of your proposed project to be accomplished in Lab 6. This will allow your instructor a chance to quickly review it and give you rapid feedback before you invest any significant time in it. Only one submission is required for each team.

PRELAB SUBMISSION

Submit:

1. answers to all questions,
2. state diagram, input equation, and output equation for the $0 \rightarrow 1$ transition detector,
3. a printout of your CONTROL.BDF,
4. a printout of the simulation trace for your control unit,
5. the completed table for control unit state FF input logic,
6. the completed table for control unit output logic, and
7. your team's initial proposal for your Experiment 6 project.

5-2 LAB WORK

WARNING: All lab results and all answers to questions or discussion are to appear in the lab reports of individual students. All tangible lab results are to be identical (unless indicated otherwise). When a printout of results is specified, a copy should be made for each team member. All answers to questions or discussion are to be the work of individual students, not the lab team. Evidence of collaboration on these aspects of a report within or between teams will be noted and is subject to University disciplinary action.

EQUIPMENT NEEDED

In addition to the equipment already on the lab bench, your instructor will have you check out a plastic tray containing:

1. an FPGA prototyping board.
2. a power supply module for the prototyping board,
3. two scope probes,
4. a logic analyzer probe,
5. and, a coaxial cable with BNC and mini-grabber connectors

WARNING – LAB EQUIPMENT HANDLING: Much of the lab equipment is small and delicate. In particular, this applies to the XESS prototyping boards, scopes, scope probes, and logic analyzer probes. So please be careful and handle the equipment with a light and careful touch and do not use the scope probes with the grabbers removed. Perform wiring on the board **ONLY** with the power disconnected.

DESIGN INTEGRATION

1. Open your project, add FREQCNT.BDF to the project, and make it the top level entity.
2. Open CONTROL.BDF and create a symbol from it. Add the symbol to FREQCNT.BDF and complete the logic in FREQCNT.BDF. You will need to add;
 - a. Synchronizers
 - b. Logic so that the BCD counters are only enabled when COUNT_ENABLE is a 1 and the input has transitioned from a 0 to a 1.
 - c. Connections as required.
3. Implement your design.

CHECKPOINT: Show your instructor your schematic and your Pin-Out File verifying the correct pin assignments.

FREQUENCY COUNTER TEST

1. Ensure the Wavetek signal generator is OFF. Connect the coaxial cable with mini-grabbers to **MAIN OUT** on the Wavetek. Do **NOT** connect the Wavetek to the FPGA board yet.
2. Turn on the oscilloscope and connect the channel A1 probe to the mini-grabbers on the cable from the Wavetek (the black mini-grabber is ground).
3. Turn on the signal generator power, and adjust the signal generator to obtain a 0-3.3V square wave at 25KHz. Verify the signal using the oscilloscope.

CHECKPOINT: Have your instructor observe the signal generator waveform **BEFORE** connecting to the FPGA board.

4. Turn off the signal generator.
5. Connect the black mini grabber and the scope probe ground to the GROUND pins on the FPGA board. Connect the red mini-grabber and the scope probe to the S1_1 pin of JP7. Ensure that switch S1_1 is in the open (off) position.
6. Perform the POWER UP sequence.
7. Verify that you still have a 0-3.3V square wave at 25KHz.
8. Download your freqcnt.sof file to the FPGA board.

9. If your circuit is working properly, you should be able to press PB4 (Reset) and the controller should go to the IDLE state (BAR1_1 illuminated) when it is released. Then, each time you press PB1 (Start) the display should update to "25" and the controller should remain in the WAIT state (BAR1_3 illuminated) until PB1 is released. Since the BCD counter is enabled for a 1ms interval, the display will be directly readable in KHz. (Since it is in state COUNT for only 1ms, you will not be able to see BAR1_2 illuminate.)
10. Vary the frequency of the signal generator and verify the operation of your frequency counter. Make any necessary changes to your design, and test its operation again. When you are certain it is working properly, notify your instructor.

CHECKPOINT: Have your instructor observe the operation of your frequency counter.

11. Perform the POWER DOWN sequence.
12. Print out both pages of your schematics for submission with your postlab report.

1. If your frequency counter display indicates "25", does it mean that the input signal must be 25KHz? If not, what are the other possibilities?
2. What is the minimum number of flip-flops that we could have used to design this FSM? What are all of the possible advantages and disadvantages of designing it that way versus using the one FF per state method?

5-3 REPORT

Your individual report on 8.5 by 11 inch pages should follow the report format available on the course web page and consist of:

- 1) A cover with the experiment number, experiment name, your name, name(s) of other team member(s), and your instructor's name, your lab section, the number on the equipment tray used, and name on the PC used.
- 2) Description of Laboratory Exercise and Conclusions as shown in the report format,
- 3) Your individual answers to all of the questions given in the green-bordered boxes (online version has color), and
- 4) Copies of all printouts for the lab properly annotated and in the order produced.
 - a. printouts of FREQCNT.BDF and CONTROL.BDF

REFERENCES

1. MANO, M. M AND C. R. KIME. *Logic and Computer Design Fundamentals*, 1st and 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1997 and 2000.

To be completed and turned in with your prelab report:

Input logic for state flip-flops

State	Expression specifying the conditions required to remain the current state	Expression specifying the conditions required to become the current state	Input logic expressions
IDLE			$D_{IDLE} =$
COUNT			$D_{COUNT} =$
WAIT			$D_{WAIT} =$

Logic for controller outputs

Output signal logic expressions
TIMER_RESET =
COUNT_CLEAR =
COUNT_ENABLE =
DATA_STORE =