CPE 324-01: Advanced Logic Design Laboratory

Lab05

Single Button Texter

Submitted by: Esther Shore

Date of Experiment(s): February 20, 2024 and February 22, 2024

Report Deadline: February 27, 2024

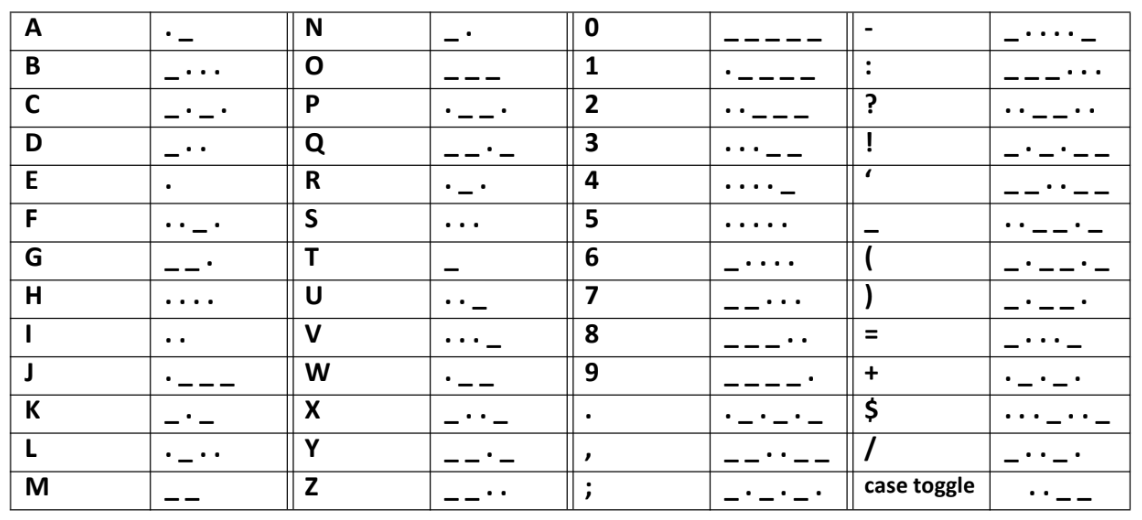
1. Introduction

In this lab we are to develop a single button texter module that can act as a human machine interface using Verilog HDL and finite state machine techniques. The goal is to create a hardware representation capable of translating on/off keying into plain text for simple text message communication. This will be accomplished by creating a module that will decode input patterns from a modified version of international Morse code, allowing users to send messages through a series of button presses. Through integration into a physical test bench environment, we will conduct real-time manual and automated tests to validate the functionality of our design. By varying message speeds and manually sending text, we will demonstrate the effectiveness of our single button texter module.

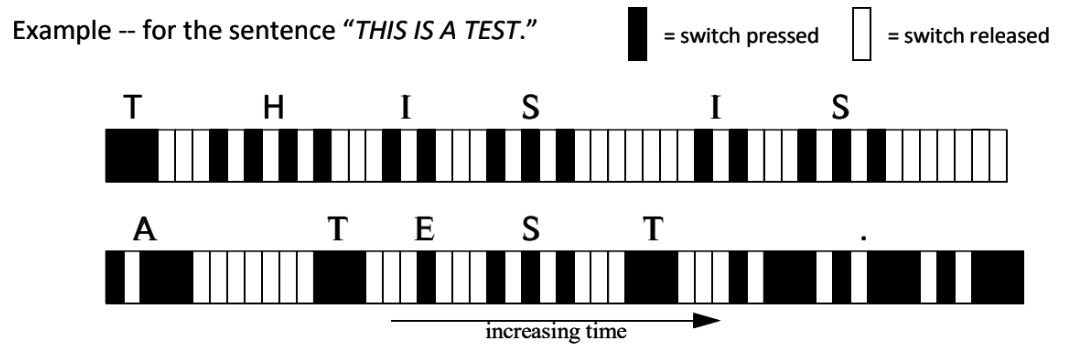
1. Experiment Description

In order to create our single button texter module, which will be able to interpret sequences of presses and releases of a momentarily closed button, we will use the provided project template and design our texter\_control module. The messages will be encoded in a modified version of Morse code as shown in Figure 1.

First, we had to understand how Morse code works. Each character is represented as a unique sequence of one or more dots and dashes. The dot is equal to one unit of time, the dash is equal to three units of time, and the time between successive dot/dashes in a character coding sequence is one unit of time. There are three units of time between each character within a word and seven units of time between words. The base unit of time depends on the speed that the text is sent and is given by the equation T\_unit = 1200/S, where S is the speed in words per minute, and T\_unit is the unit time in milliseconds. An example of this format is shown in Figure 2 below.

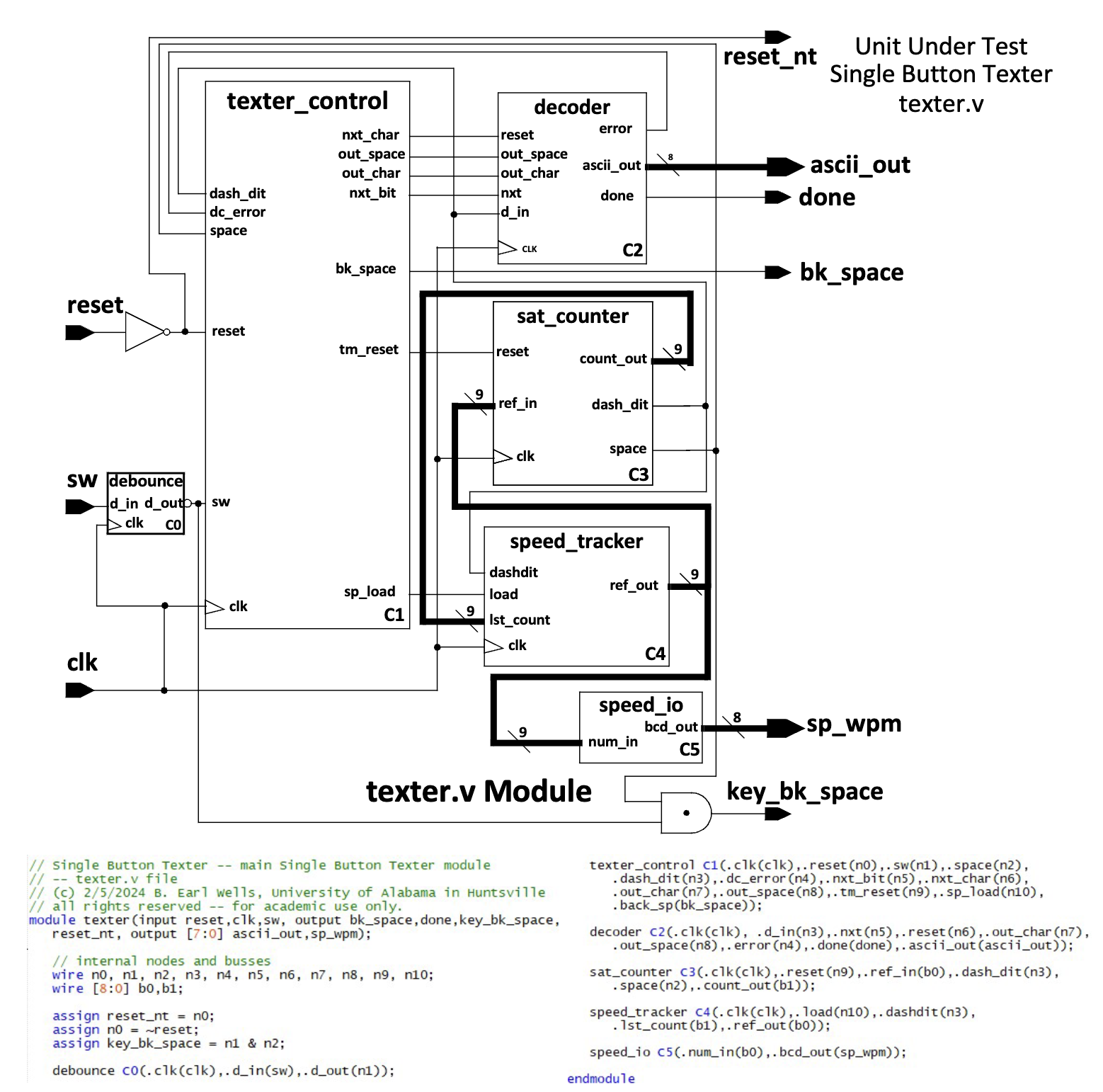


**Figure 1.** Initial Texting Code Format – Modified Morse Code



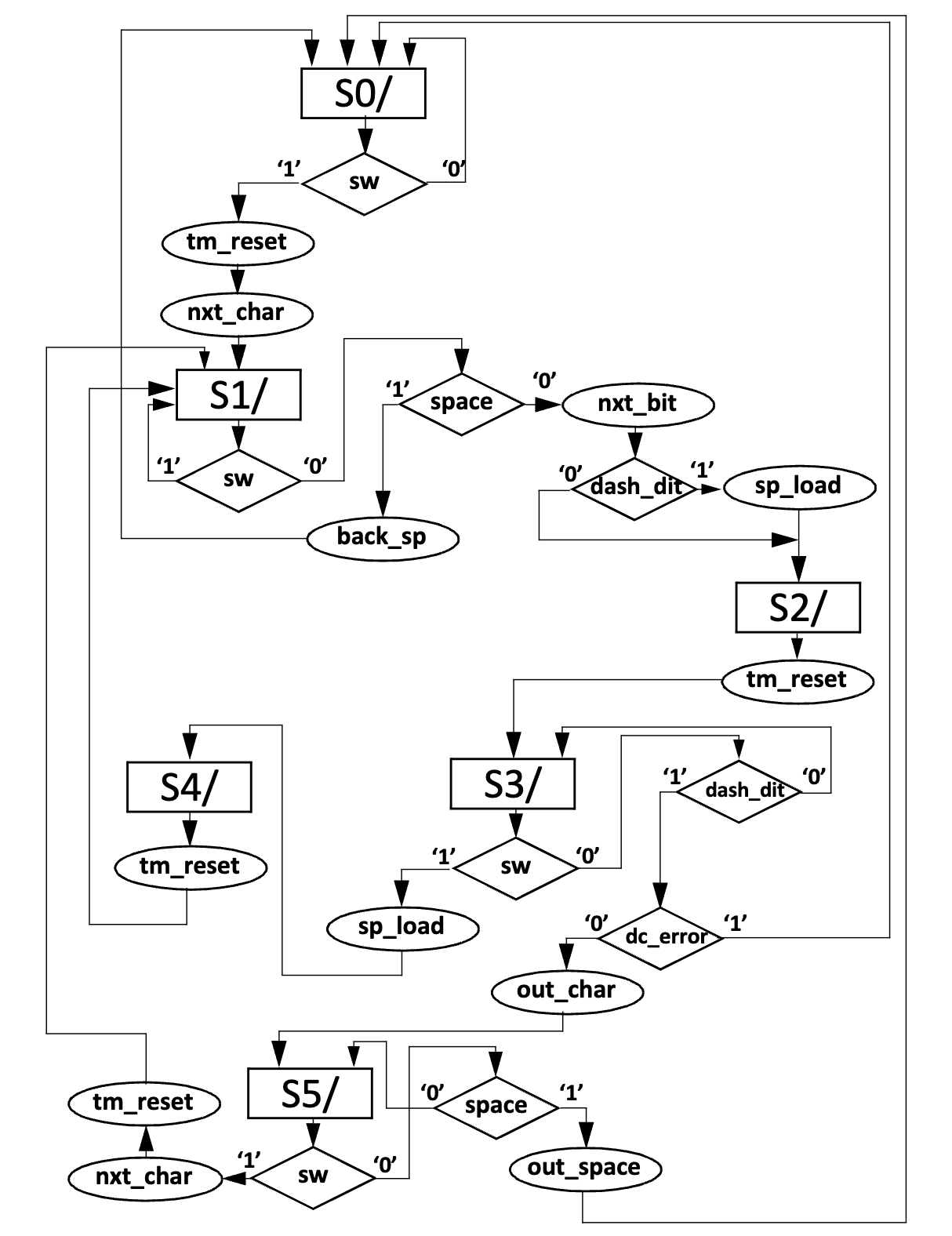
**Figure 2.** Texting Code Format Example

Next, we had to conceptualize the high level design for the texting module as specified in Figure 3. The only module that we are required to develop is the texter\_control module, but it is beneficial for us to also discover how it is integrated into our system. The decoder module is used to shift in the dot/dash bits and to match the bit pattern with a valid Morse code character, outputting the ASCII code representation of the character. The sat\_counter is a saturation counter that determines whether the key press was a dot or dash by checking how long the button was pressed for, or if a space should be generated after the button not being pressed for some time. The speed\_tracker produces the reference number of clock cycles that is used by the sat\_counter. The speed\_io module monitors the reference count produced to estimate the speed in words per minute that the message is being sent.



**Figure 3.** Single Button Texter Logic (Top-Level)

We were provided an algorithmic state machine for the texter\_control design displayed in Figure 4, which we will convert into behavioral Verilog code, tracking the inputs and outputs to and from each state.



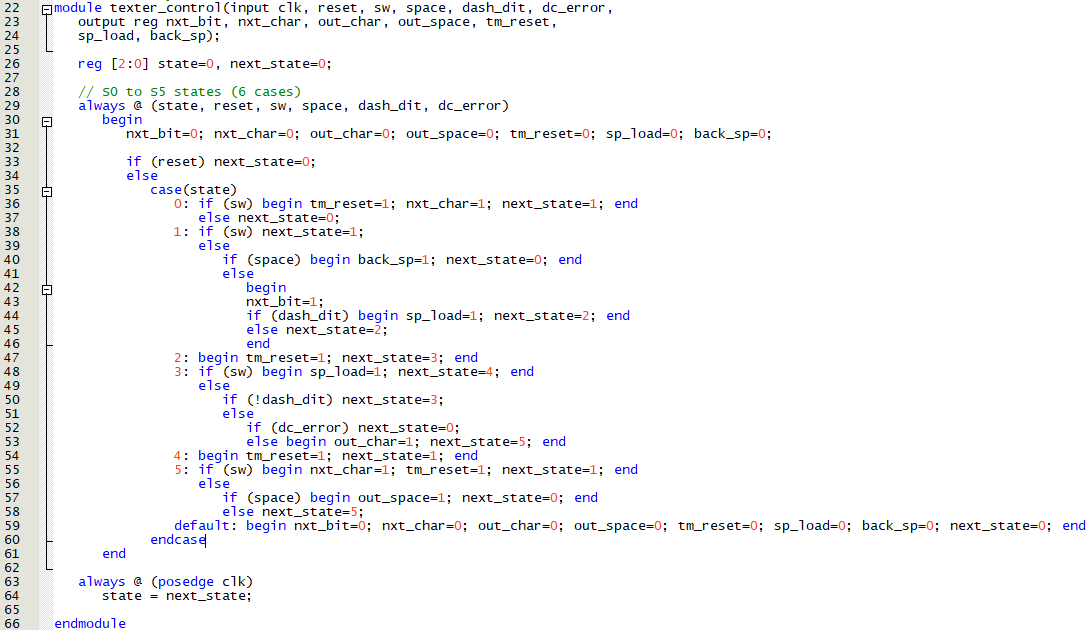
**Figure 4.** texter\_control Algorithmic State Machine Representation

In the physical design setup, the external switch connects to breakout board Pin D10 in addition to VCC and ground. After completing our software implementation of texter\_control, we will then assign the appropriate DE2-115 pins and load our project onto the board.

1. Demonstration

In order to demonstrate the functionality of our single button texter design and texter\_control module, we utilized the provided lab5\_ptb.v testbench to automatically generate all of the outputs as well as manually enter characters via Morse code and view the results on the LCD display. We demonstrated that the design works correctly by varying the speed of the automated message as well as sending my full name manually using the external switch button. The case could also be toggled to upper or lowercase letters via a specific sequence shown in Figure 1.

My texter\_control Verilog design is exhibited in Figure 5, translating the given algorithmic state machine into a case statement with a series of conditional statements.



**Figure 5.** texter\_control.v module

DE-2 User Inputs:

KEY0: decreases the sending speed of the text in automated text mode

KEY1: increases sending speed

KEY2: mode change button

KEY3: display and testbench reset button

DE-2 User Outputs:

LEDG0: on when SW is pressed

LEDG0-LEDG7: lights up when SW held for long time indicating backspace

LEDR0-LEDR7: lights indicates relative speed of text in automated mode

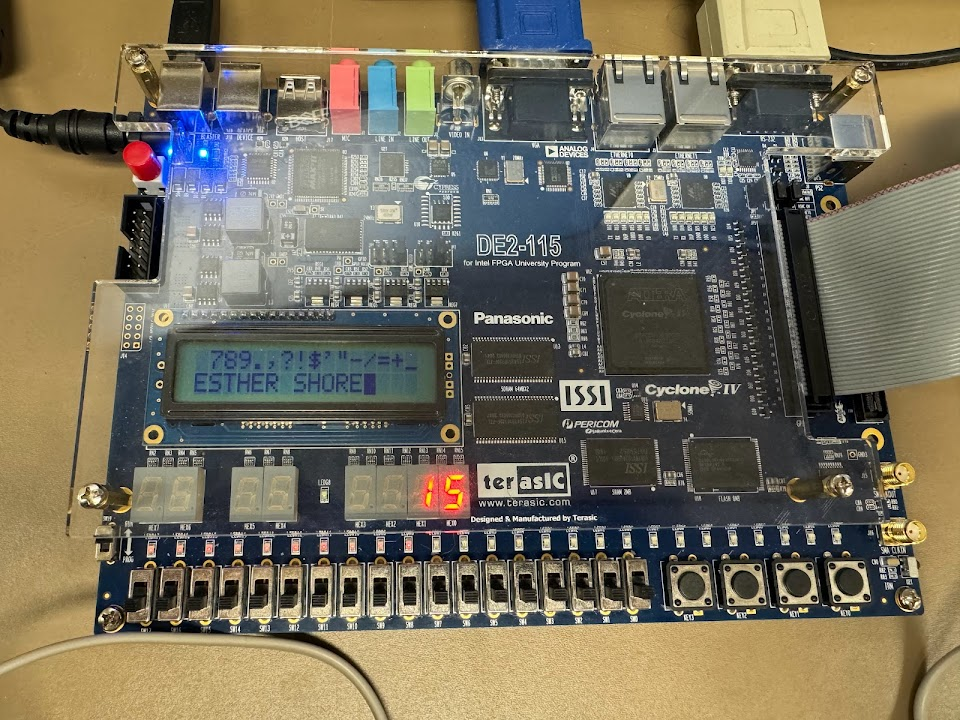
HEX1, HEX0: indicates approximate texting speed in WPM

LCD: text output

For my manual test case, I chose to generate my full name in all caps following the automatic test bench verification, shown in Figure 6.

Morse code for my full name ESTHER SHORE:

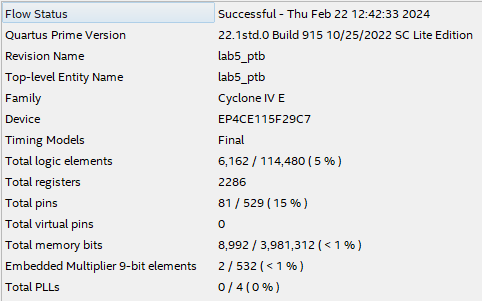
**. . . . \_ . . . . . . \_ . . . . . . . . \_ \_ \_ . \_ . .**

****

**Figure 6.** Demonstration of Functionality

1. Experimental Results

By using the provided testbench, we were able to send a prerecorded morse code texting pattern at various speeds in addition to using an external switch to manually send messages. Successful compilation results are shown in Figure 7. We were able to verify and demonstrate our design successfully using the prior mentioned testbench. In automated text mode, we were able to increase/decrease the WPM and toggle the mode using the keys on the DE2-115 board.



**Figure 7.** Compilation Results

Post-Lab Questions

1. What are the major characteristics of synchronous digital design? What are the differences between the control path and data path? What are the clocking requirements?

The major characteristics of synchronous digital design include a common clock signal to synchronize the components of the system, ensuring reliable operation of actions. Sequential logic, timing analysis, modularity, and finite state machines are also often characteristic of synchronous digital design.

The control path directs the flow of data through the system (logic circuits and finite state machines) while the data path performs arithmetic and logic operations on data (arithmetic logic units and combinatorial logic elements).

The clocking requirements, specifically clock frequency, specifies the maximum frequency at which the design can operate reliably. These specify the characteristics and constraints of the clock signal. To ensure proper timing, minimize clock skew or the variation in arrival times of the clock signal to different components in the system.

1. During compilation, there were many warning messages. List at least five different types of messages that you observed. For each of these messages, specify what you perceive to be their meaning. Also specify why these messages could be ignored and the design still functioned correctly.

Warning (18236): Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment NUM\_PARALLEL\_PROCESSORS in your QSF to an appropriate value for best performance.

This warning states that the number of parallel processors is not defined, which can lead to resource management inefficiencies. It can be ignored since we are not so concerned with performance and the functionality was not affected.

Warning (10230): Verilog HDL assignment warning at sw\_driver.v(146): truncated value with size 32 to match size of target (23)

The warning indicates that a value is being trimmed down from 32 to 23 bits. It is not intentional and could lead to loss of precision but does not affect the functionality since we are mostly dealing with data that is less than the target size.

Warning (10030): Net "m\_clk\_range.data\_a" at sw\_driver.v(13) has no driver or initial value, using a default initial value '0'

It is warned that no initial value was defined for a specific net. It may be driven by another source or the default value is sufficient, so the functionality is maintained and the warning can be ignored.

Warning (127007): Memory Initialization File or Hexadecimal (Intel-Format) File "C:/Users/eas0035/Documents/lab5/db/lab5\_ptb.ram0\_usend\_78d4ead.hdl.mif" contains "don't care" values -- overwriting them with 0s

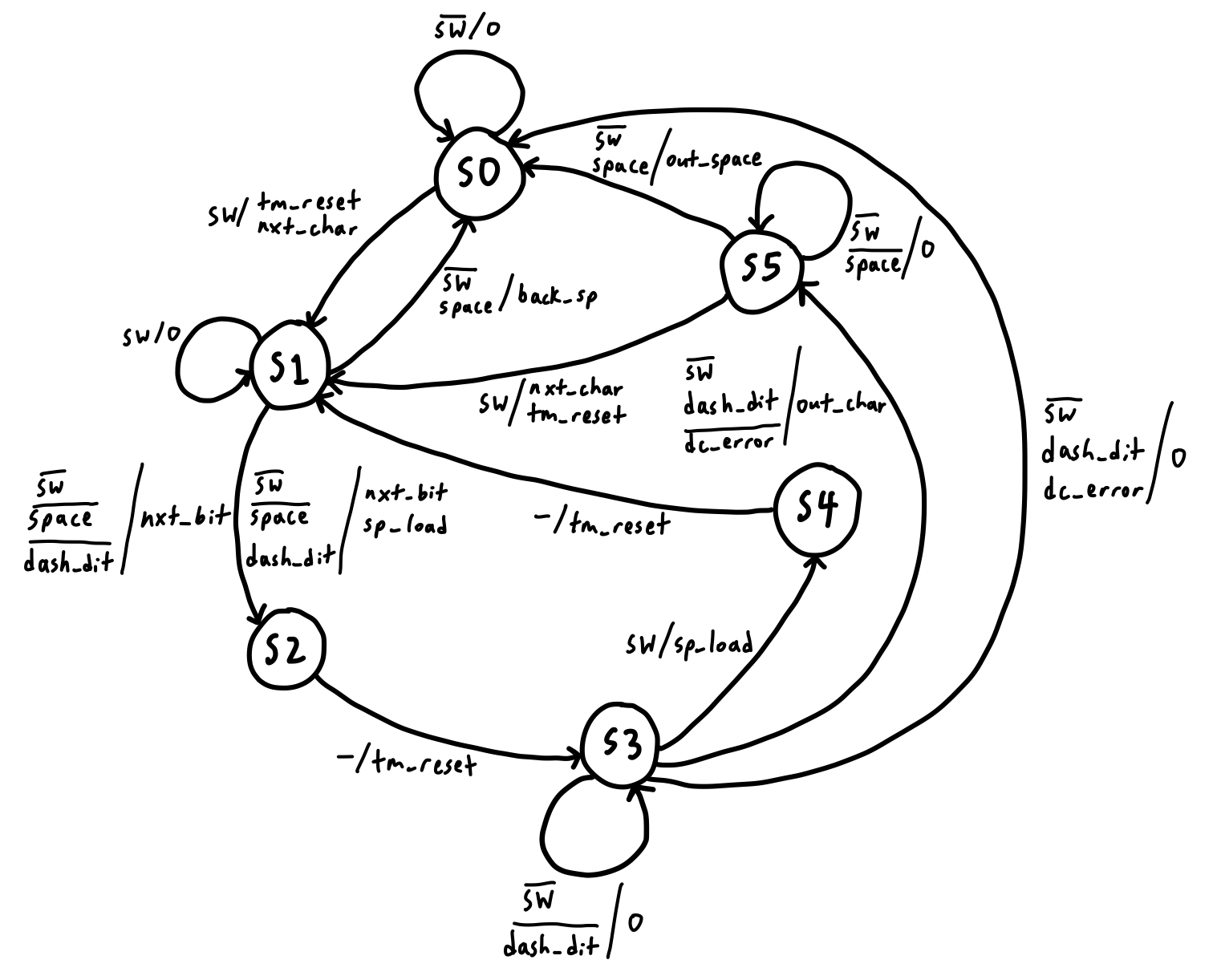
This warning says that a memory initialization file has “don’t care” values that are being overwritten with 0s. This doesn’t affect the functionality of the design and is not critical for proper operation, so 0s are acceptable placeholders.

Warning (292013): Feature LogicLock is only available with a valid subscription license. You can purchase a software subscription to gain full access to this feature.

The warning states that a certain feature cannot be accessed without a proper subscription license. The LogicLock feature is not essential to this design, so the warning is ignored without impacting the function.

1. Create an equivalent Extended State Diagram that has the same functionality as the Algorithmic State Machine representation of the texter\_control module that was presented in Figure 3 of this lab.

This involves nodes that represent the states and arcs that represent transition between states, with labeling of the arcs with inputs/outputs. Only inputs that impact a transition from one state to another are present and only outputs that are true for a given transition are listed.



**Figure 8.** texter\_control Extended State Diagram Representation

1. Explain the function of each of the six states in the system, do you think it would be possible to reduce the number of states and still preserve the functionality?
2. S0: initial state for switch being pressed
3. S1: determine dot/dash or space
4. S2: reset saturation counter
5. S3: process Morse code input and outputs corresponding ASCII character, detects dc error
6. S4: loads saturation counter
7. S5: handles release of the switch button

It might be possible to reduce the number of states and still preserve the functionality. S2 and S4 combined could potentially handle the saturation counter, while S1 and S5 effectively handle the switch press. Attempting to consolidate any of these states would require careful consideration to ensure no errors or inefficiencies are generated.

1. In what ways does this physical testbench differ from traditional test benches that are used for simulations? What are the advantages and disadvantages to utilizing the physical testbench approach over that of a simulation only approach that utilizes a traditional testbench? What would be the challenges associated with creating this testbench?

This physical testbench differs from traditional test benches that are used for simulations in that it interfaces directly with external physical components rather than simulating them. By doing so, it also allowed for real-time interaction with the hardware, letting the user enter key inputs manually and viewing the output on the hardware displays. This is one advantage of the physical testbench approach over that of a simulation only approach that utilizes a traditional testbench. Another advantage is that a physical testbench considers environmental factors such as hardware constraints that limit signal integrity and timing. The challenge with creating this type of testbench is that it often requires more resources such as the hardware itself and can be more challenging to thoroughly test and debug a wide array of possible scenarios.

1. What are the general design trade-offs one must consider when choosing when components should be implemented in hardware or software? Consider such items as performance requirements, the ease of implementation, the ability to expand and alter the system as new requirements emerge, the ability to execute in a real-time/deterministic manner, the energy utilization, etc.

While this design could have been implemented in software using commercially available embedded instruction set process environments (micro-controllers) such as Arduino or Raspberry Pi, it is important to consider the trade-offs between hardware and software implementations. Hardware usually offers higher performance and deterministic execution but is generally more difficult to modify and consumes more energy. On the other hand, software is easier to develop and modify but may not meet strict timing requirements and could be less energy-efficient. Hardware designs also often have higher upfront costs and longer development times.

1. Conclusion

This lab successfully demonstrated the development of a single button texter module using Verilog HDL and finite state machine techniques. Through integration into a physical test bench environment, the functionality of the design was thoroughly validated, showcasing its ability to translate Morse code inputs into plain text messages for communication. The experiment helped us learn about the characteristics of synchronous digital design, clocking requirements, and the distinction between control and data paths. We also examined the design trade-offs between hardware and software implementations.