ECE 385

Spring 2023
Final Project

GPS Location Logger with Data Plotting

Michael Gamota

Introduction

Our proposed project consists of interfacing an “off the shelf” NEO M6 GPS module which sends latitude, longitude, and time data over a serial Universal Asynchronous Receiver Transmitter (UART) port with a DE10-Lite FPGA board. We will record the latitude, longitude, and time data, storing it in the M9K blocks of on-chip memory. The user will then be able to display the recorded data visually on a VGA monitor. When recording data, the tenths and hundredths place of minutes longitude and latitude are displayed, giving the user live position resolution of about 60 feet latitude and 48 feet of longitude. This live position as well as a message counter is displayed on the hex displays when in “record” mode (SW[0] on) and the timestamp of each location along with the bottom 4 bits of memory location of the data when in “display” mode (SW[1] on).

We planned on demonstrating this by taking the module around campus and then displaying a history of the locations with their respective timestamps on the monitor. However, the GPS quality was poor and the locations transmitted were both inaccurate and unstable (not precise) so we were not able to demonstrate it this way. Instead, we demonstrated the recording of data and plotting of data separately. The recording of data was demonstrated by staying in a stationary location, recording positions for several minutes, and then going through the list of timestamps of those recordings. To demonstrate the plotting, we hardcoded real locations and timestamps in the memory instantiation file (MIF) and showed the plotting of those locations on a real map of the Beckman Quad at the University of Illinois at Urbana-Champaign. We have video documentation as well as pictures of this behavior which will be shown in the report.

Written Description of Final Project Features

UART Byte Buffer:

The UART driver was developed by us from scratch using a state machine. After researching UART protocol using online resources, we figured out how to implement a UART buffer in System Verilog. We referenced the datasheet the NEO-6M GPS module and found that the default settings for the NEO-6M are a 9600 baud rate, 8 data bits, no parity bit, and 1 stop bit. This is all relevant to the implementation of our UART buffer as not all UART messages are structured the same way.

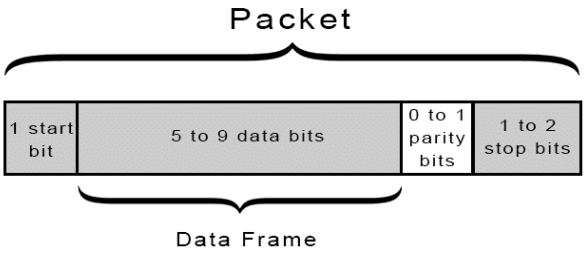

Figure 1: UART Protocol Packet Structure

The UART byte buffer reads in one packet (8 bits) at a time and then sets a flag called “done”. Each byte is an ASCII encoded character which is part of a message type called National Marine Electronics Association (NMEA). The byte buffer is the first layer of the communication stack that interfaces the NEO-6M GPS module with the FPGA. The state machine is shown below and details the 5 different states used. The IDLE state waits for the start bit (0), once the data line thinks there is a start bit, it checks that the value is still 0 halfway through the expected pulse. If so, we move onto the DATA_BITS which samples 8 bits of data and then moves to the STOP_BIT state before returning to the IDLE state.

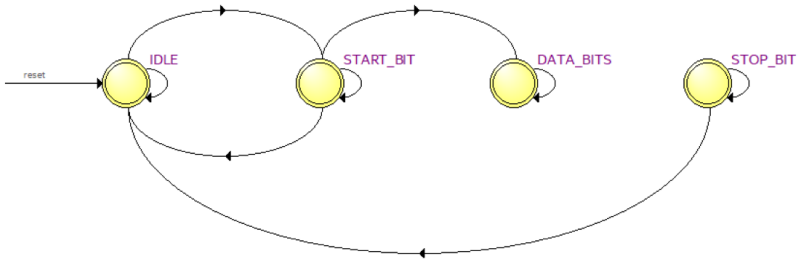

Figure 2: UART receive buffer state machine

NMEA Message Parser:

The next step of extracting the data from the NEO-6M GPS module is assembling the bytes from the UART byte buffer into a “sentence” of ASCII characters. In this case, that sentence is an NMEA message. In my parser, I read every message, but filter out any that do not start with “$GPGLL”. This is a code which indicates the following message will include latitude, longitude, and time data.

$GPGLL, 3723.2475,N,12158.3416,W,161229.487,A,A*41

Table 1-5 GLL Data Format

Name	Example	Unit	Description
Message ID	$GPGLL		GLL protocol header
Latitude	3723.2475		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12158.3416		dddmm.mmmm
E/W Indicator	W		E=east or W=west
UTC Time	161229.487		hhmmss.sss
Status	A		A=data valid or V=data not valid
Mode	*A*		*A=Autonomous, D=DGPS, E=DR (Only present in NMEA v3.00)*
Checksum	*41		
<CR> <LF>			End of message termination

Figure 3: NMEA $GPGLL message (Latitude, longitude, time)

The message parser starts the message(READ_SENTENCE) when it sees $ and once it reaches the "*" before the checksum(CHECKSUM) it appends 2 more bytes to the sentence before moving back into the WAIT_START state.

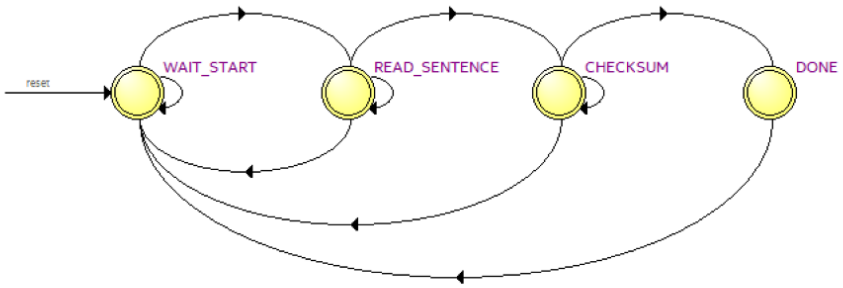

Figure 4: NMEA message constructor and parser state machine

Every time the NMEA parser receives one of these messages, it increments a message counter, and once every 16 messages, the tenths and hundredths place of latitude and longitude as well as the current time (UTC) is stored in on-chip memory. This is implemented by using the message counter as part of combinational logic for the write enable of the memory blocks. When running, we get about 1 message every second, so recording once every 16 messages means we get about 3 locations per minute. These locations using the tenths and hundredths place requires the user to know where they are within 2 miles, but our implementation gives a location within 60 feet North/South and 48 feet E/W of true location which is perfect for tracking locations around a smaller location like a school campus or a smaller section of a school campus.

On-Chip Memory:

Our system uses two different on chip memory blocks: one for location data and one for time data. However, the memories use the same write enable signals and

address variables so the location data and time data are in sync. The location data is stored in a concatenated 16-bit format where the upper 8 bits are longitude data and the lower 8 bits are latitude data. This makes it easy to process the data when we send it to the hex displays and plot the location on the VGA monitor. Additionally, the timestamp is stored in a 16-bit format where the bits [15:12] are the tens place of the hours, [11:8] are the ones place of the hours, [7:4] are the tens place of the minutes, and [3:0] are the ones place of the minutes. We made this design choice to simplify the displaying of the data on the hex displays. This makes it so the hex displays show the timestamps as if it were on a digital clock which makes it very intuitive for the user.

Map and Location Displaying:

The image ROM was created using “Ian’s Helper Tools for converting images to RAM” from the course website. We chose this section of campus because it is relevant to us as ECE students as well as it is a perfect size for the resolution of our GPS module. We wanted to show that we can be precise within 60 feet North/South and 48 feet E/W of true location which is enough to distinguish between buildings and even closer spaced locations. The VGA controller that we used is from Lab 7 and it was fairly easy to implement. The helper tool documentation explains how to connect it to the VGA controller from Lab 7. To display the current location, we use similar logic to that of Lab 6.2 which involves using an always_comb block which uses the circle equation to designate a range of pixels as “user_on” which is used to select between drawing red and drawing the VGA red, green, and blue signals from the map plotter.

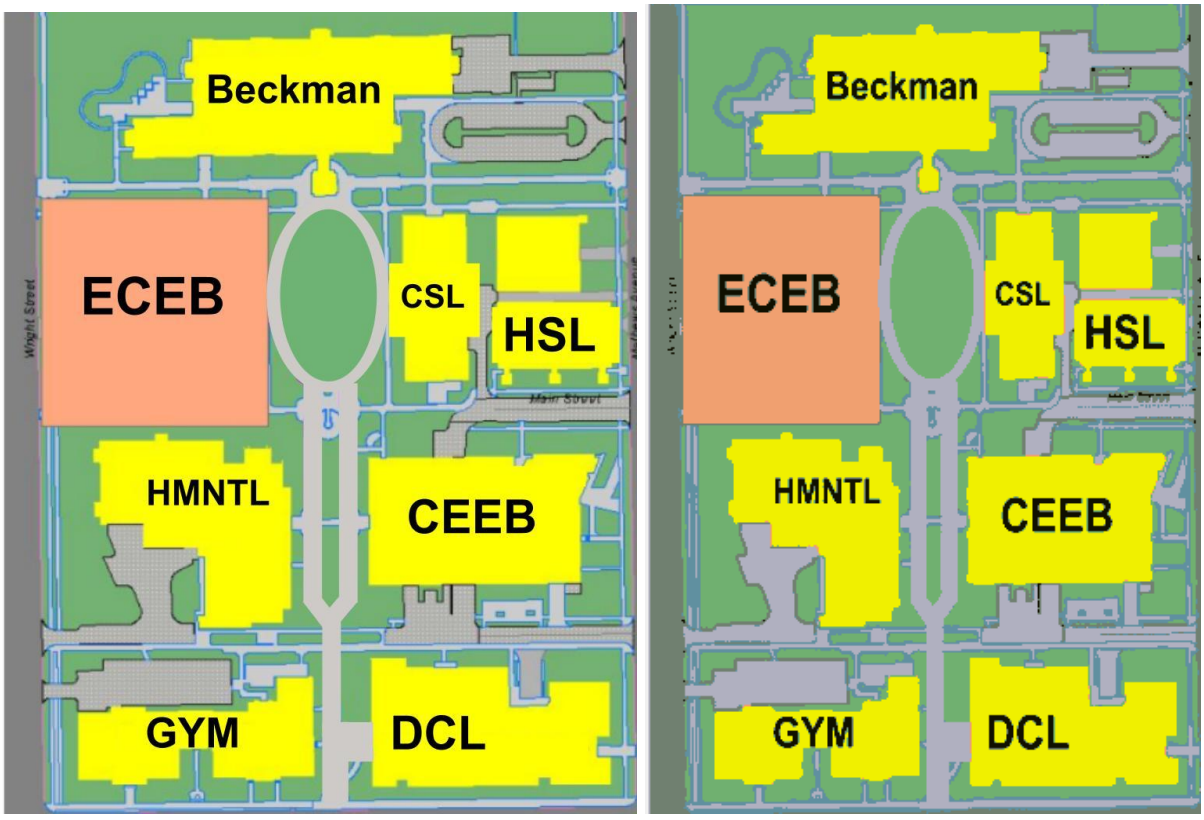

Figure 5: Input image (left) and output palletized image (right)

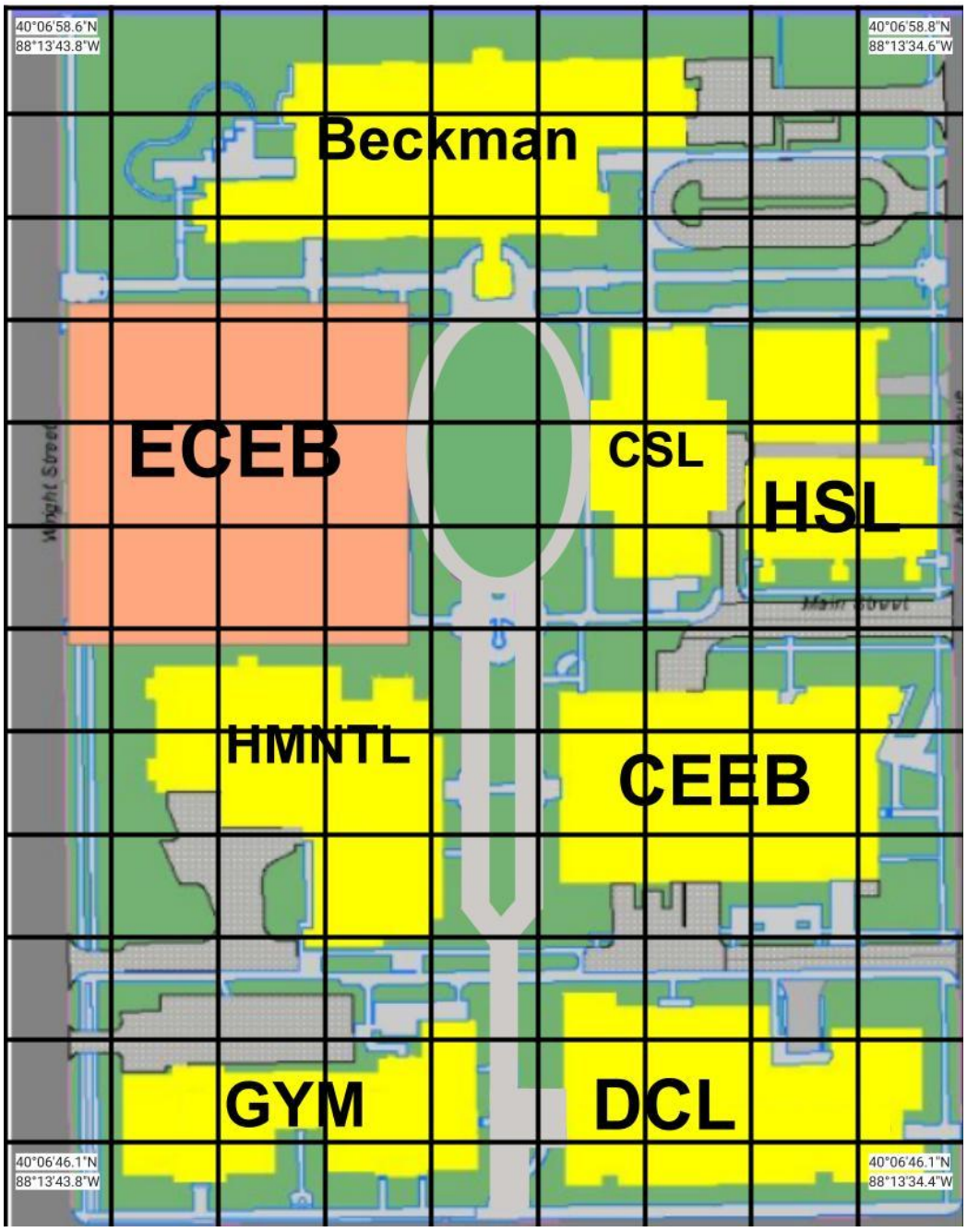

Figure 6: GPS accuracy allows position resolution as shown above

Hex Display Multiplexer:

We also implemented a hex display multiplexer which is used to create a more complete user experience. When in “record” mode, the hex display shows the user their current position and has a “message counter”. When in “display” mode, the hex display

shows the corresponding time for the plotted location and the lowest 4 bits of the memory address. Additionally, the left-most hex display block displays a 1 if the user has accidentally flipped both the “display” and “record” switches or if neither is flipped.

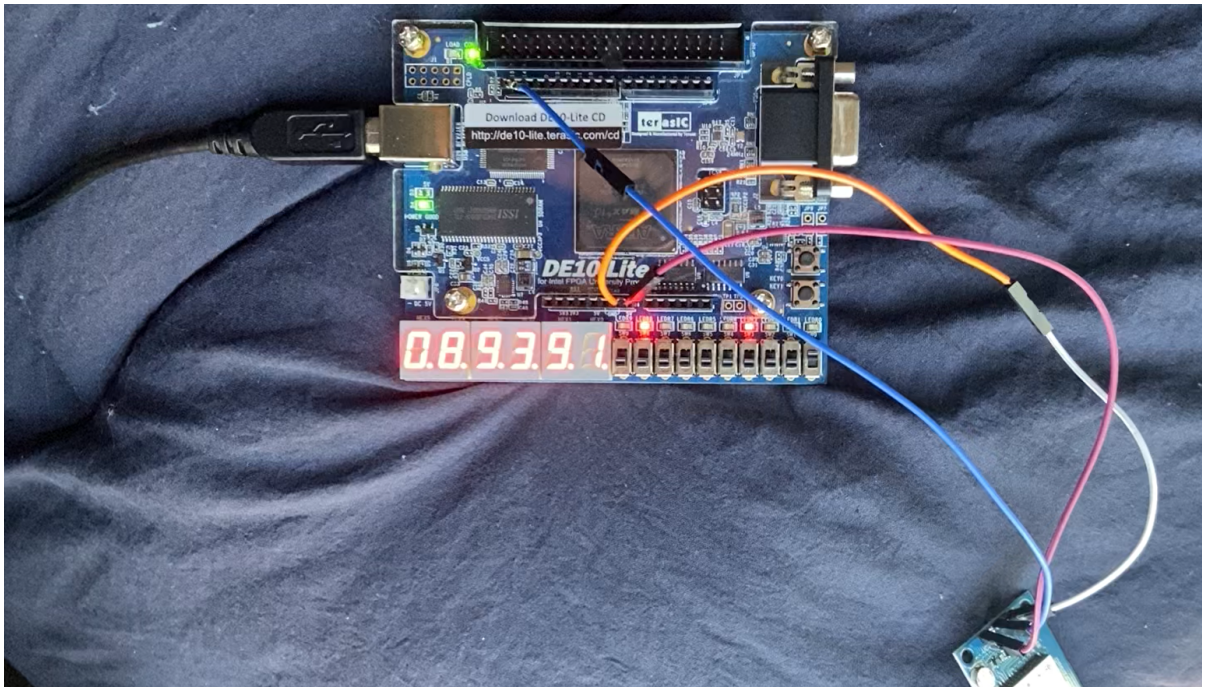

Figure 7: “Record” mode, right 4 hex displays show location, 2nd left shows message counter

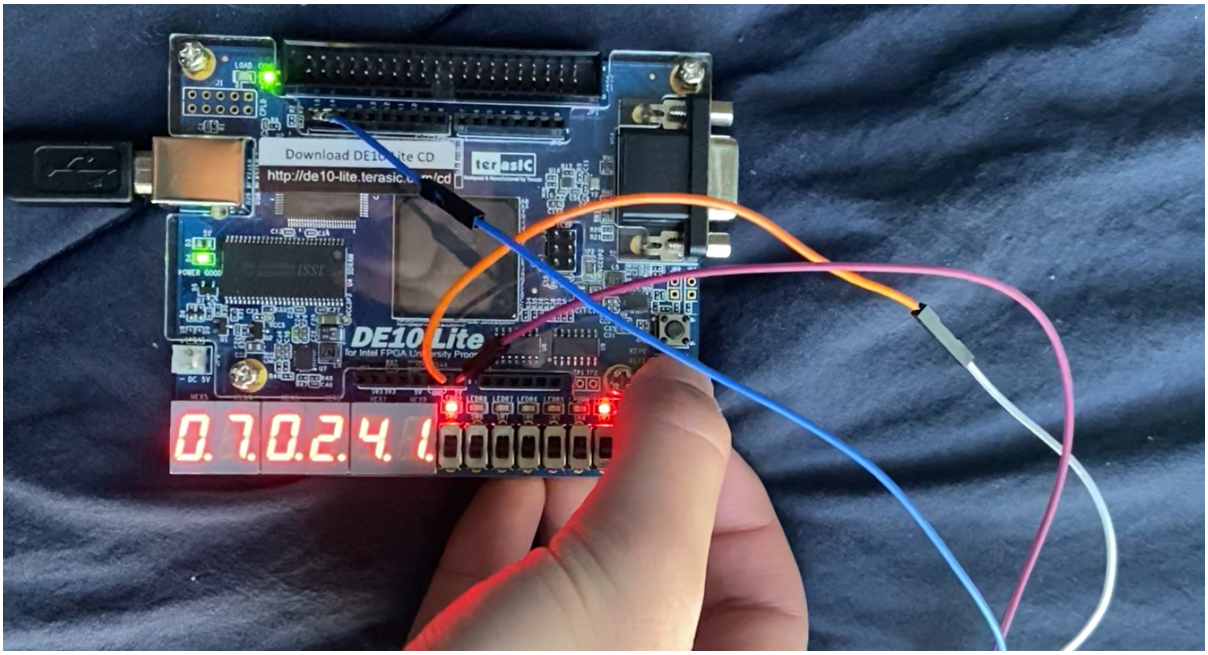

Figure 8: “Display” mode, right 4 hex displays show timestamp, 2nd left shows address [3:0]

Block Diagram

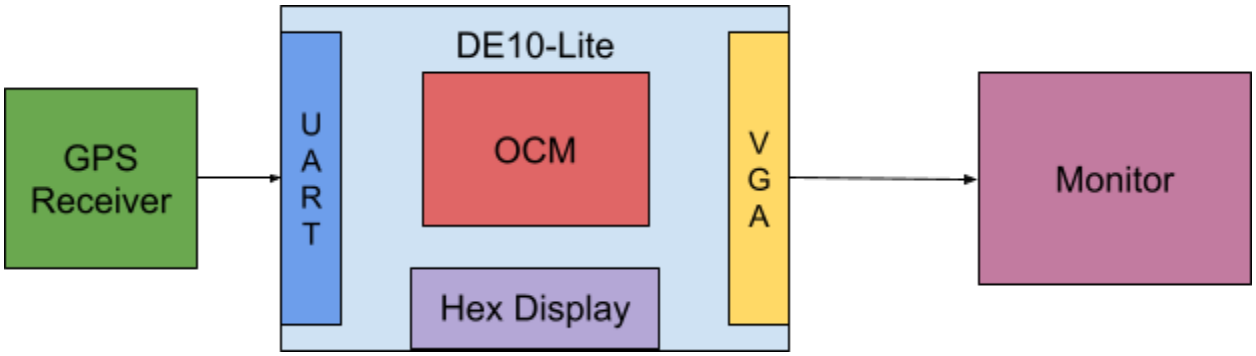

Figure 9: System Level Block Diagram

RTL Viewer

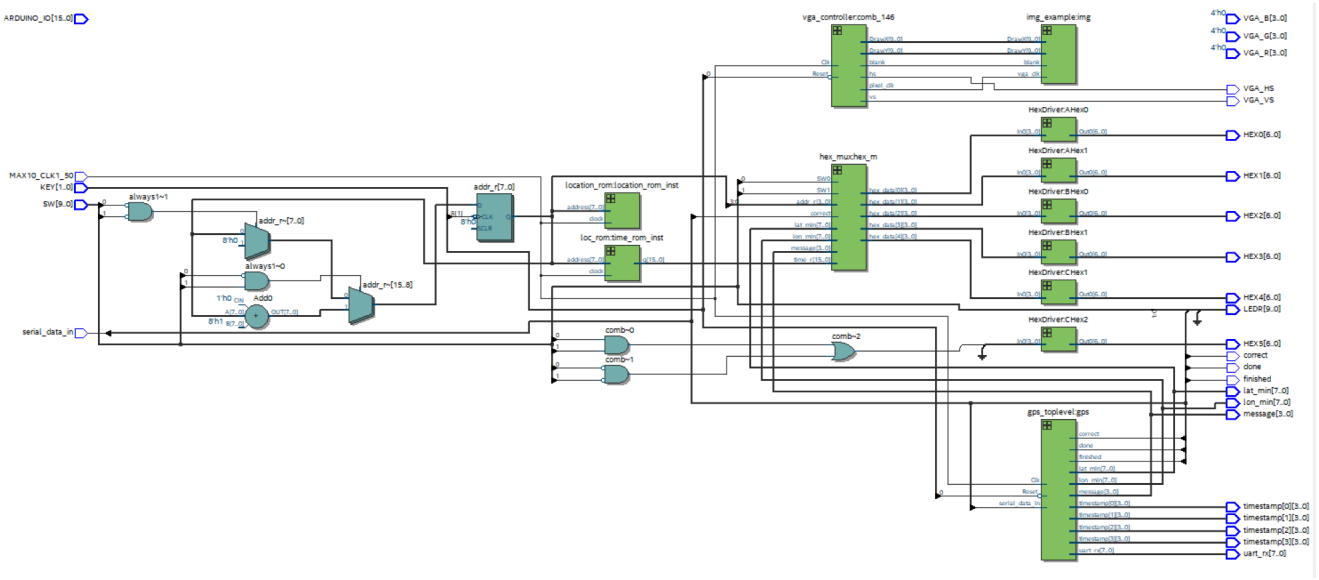

Figure 10: RTL Block Diagram

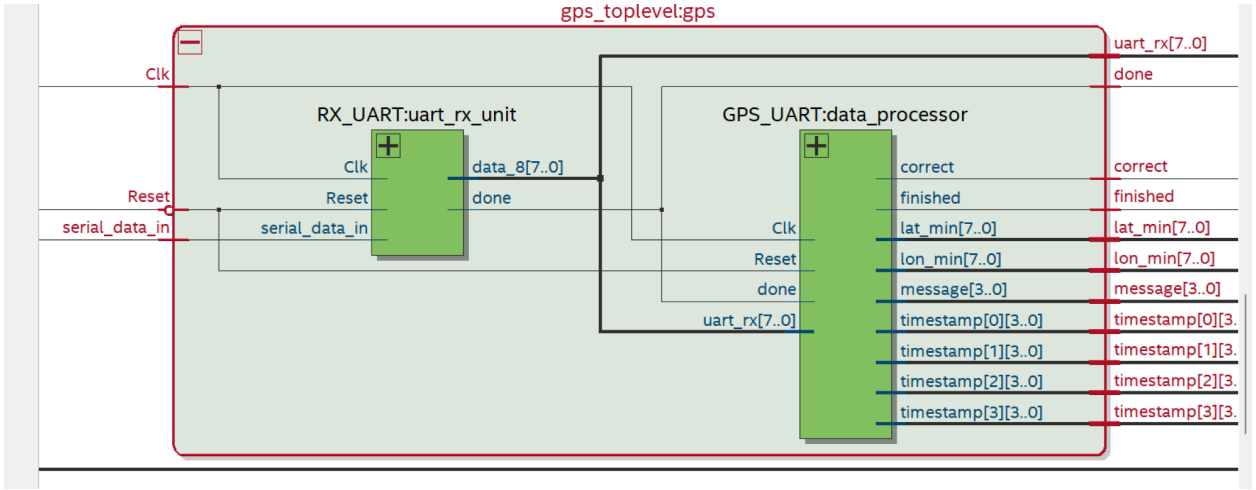

Figure 11 : GPS Module UART Processor Stack

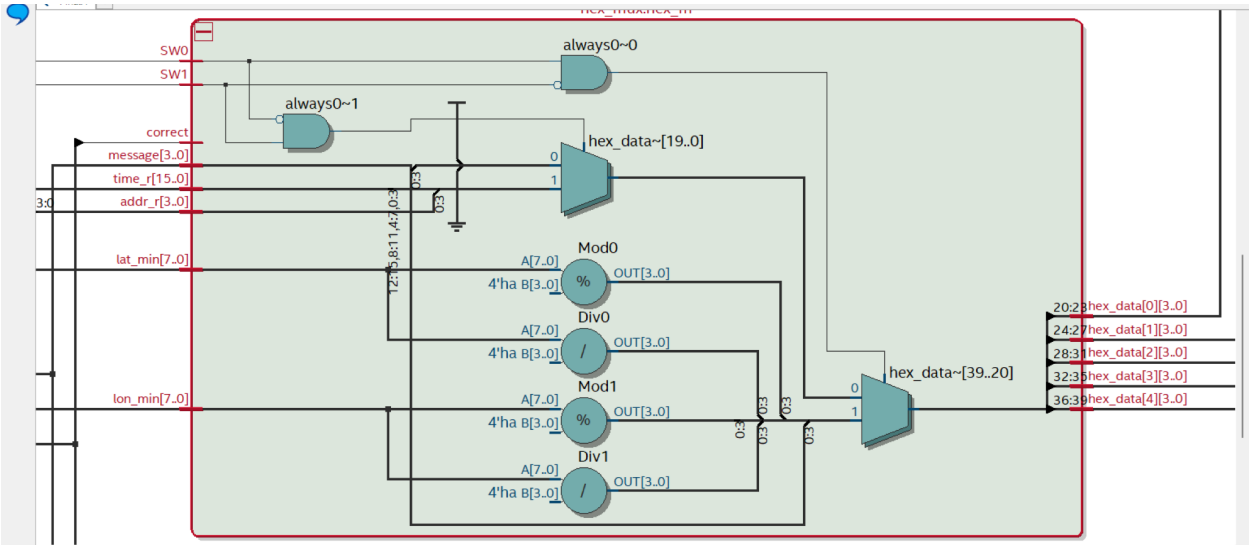

Figure 12: Hex Display Multiplexer

Simulation

The below simulation traces show the UART buffer and NMEA parser in action. The testbench feeds in serial data at the 9600 baud rate used by the real NEO-6M module. The highlighted channel is the “sentence” which is shown near the end of transmission. The testbench/done channel pulses every time a byte has been received. The testbench/finished channel pulses when a full sentence has been received, indicated by the end of the checksum.

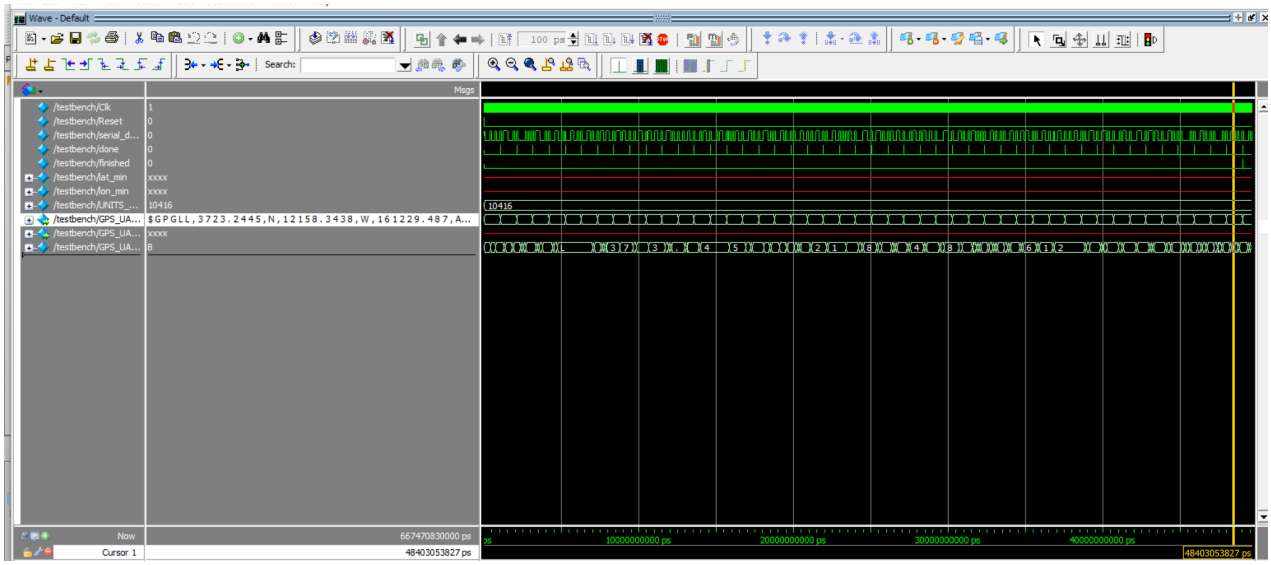

Figure 13: UART/GPS stack simulation trace

The below simulation shows how a single NMEA message is received and the lon_min and lat_min values are updated. This testbench file being executed sends a generic NMEA message with random coordinates at the proper baud rate (9600) which simulates the function of the GPS module. The bottom channel /testbench/uart_rx receives the message one byte at a time. Once the message is received, the /testbench/finished and /testbench/correct signals both go high and /testbench/lat_min and /testbench/lon_min hold the values from the message.

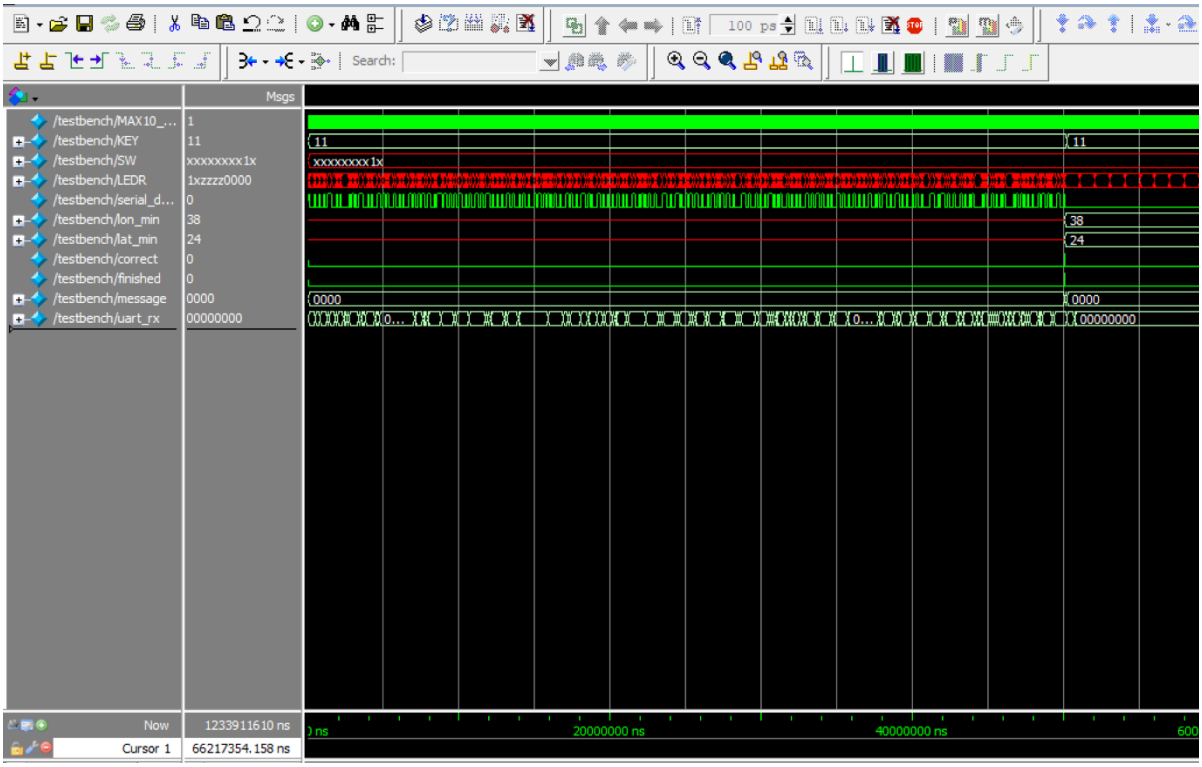

Figure 14: System level simulation trace

Module Descriptions

1) Module: final_project.sv
    a) Inputs: input MAX10_CLK1_50, input [2:0] KEY, input [9:0] SW, input serial_data_in, input ARDUINO_IO
    b) Outputs: output [7:0] HEX0 (there are 6 of these: HEX0, HEX1, etc), output VGA_HS, output VGA_VS, output [3:0] VGA_R, output [3:0] VGA_G, output [3:0] VGA_B, output [7:0] lon_min, output [7:0] lat_min, output correct, output finished, output [3:0] message, output [7:0] uart_rx, output done, output [3:0] timestamp[4]
    c) Description: This is our top level module which links the gps toplevel, the switches, buttons, hex displays, instantiates memory, and holds the logic for the VGA drawing process as well as the mode states.
    d) Purpose: The purpose of this module is to create our system by connecting all the appropriate subsystems together.
2) Module: RX_UART.sv
    a) Inputs: input serial_data_in, input Clk, input Reset
    b) Outputs: output done, output [7:0] data_8
    c) Description: This module uses a state machine to decode a 9600 baud serial UART input and outputs a “done” signal at the end of the message and outputs the message in data_8
    d) Purpose: This receive buffer is how the hardware converts the serial UART signal from the GPS module into a byte which can be stored/parsed
3) Module GPS_UART.sv
    a) Inputs: input logic Clk, input logic Reset, input logic done, input logic [7:0] uart_rx
    b) Outputs: output logic finished, output logic [7:0] lat_min, output logic [7:0] lon_min, output logic correct, output logic [3:0] message, output logic [3:0] timestamp [4]
    c) Description: This module uses a state machine to determine the start and end of each message string, which starts with “$” and ends with “*xx” where xx is a checksum value
    d) Purpose: This module is used to construct, filter, and parse NMEA messages which are sent byte by byte into the RX_UART buffer. We extract the lat_min, lon_min, and timestamp values from the NMEA message. The module also has a message counter which increments every time a $GPGLL message is received.
4) Module:gps_toplevel.sv
    a) Inputs: input Clk, input Reset, input serial_data_in,

   b) Outputs: output [7:0] lon_min, output [7:0] lat_min, output correct, output finished, output [3:0] message, output [7:0] uart_rx, output done, output [3:0] timestamp[4]
   c) Description: This module links the UART byte buffer with the NMEA message parser
   d) Purpose: The purpose of this module is to add another layer of abstraction to our design, creating a module which only outputs the necessary data from the message instead of the entire NMEA message
5) Module: HexDriver.sv
   a) Inputs: input [3:0] In0
   b) Outputs: output logic [6:0] Out0
   c) Description: In this module, a four-bit input is transformed into its respective HEX value with the help of a case statement.
   d) Purpose: The purpose of this module is so the output can be displayed on HEX displays on the FPGA board.

6) Module: hex_mux.sv
   a) Inputs:input SW0, input SW1, input [7:0] lat_min, input [7:0] lon_min, input [15:0] time_r, input [3:0] message, input [3:0] addr_r, input correct
   b) Outputs:, output logic [3:0] hex_data [5]
   c) Description: In this module, we create a MUX to change the output on the hex display
   d) Purpose: The purpose of the module is to control the hex display so that it depends on the mode of the system, either “Record” or “Display”.

7) Module: VGA_controller.sv
   a) Inputs: input Clk, Reset
   b) Outputs: output logic hs, vs, pixel_clk, blank, sync, output [9:0] DrawX, DrawY
   c) Description: In this module, the controller outputs the DrawX and DrawY signals to depict where the pixels must be drawn.
   d) Purpose: The purpose of this module is to control the horizontal and vertical sync to draw the pixels
8) Module: loc_ram.v
   a) Inputs: input [9:0] address_a, input [9:0] address_b, input clock, input [15:0] data_a, input [15:0] data_b, input wren_a, input wren_b
   b) Outputs: output [15:0] q_a, output [15:0] q_b
   c) Description: This module is an on-chip memory M9K block which is configured as 2-port RAM

    d) This module is where location values (lon_min and lat_min) are read from and written to
9) Module: time_ram.v
    a) Inputs: input [9:0] address_a, input [9:0] address_b, input clock, input [15:0] data_a, input [15:0] data_b, input wren_a, input wren_b
    b) Outputs: output [15:0] q_a, output [15:0] q_b
    c) Description: This module is an on-chip memory M9K block which is configured as 2-port RAM
    d) This module is where timestamp values are read from and written to
10) Module loc_rom.v
    a) Inputs: input clock, input [7:0] address
    b) Outputs: output [15:0] q
    c) Description: This module is an on-chip memory M9K block which is configured as 1-port ROM
    d) This module is where the hardcoded timestamps are located for the plotting demonstration
11) Module location_rom.v
    a) Inputs: input clock, input [7:0] address
    b) Outputs: output [15:0] q
    c) Description: This module is an on-chip memory M9K block which is configured as 1-port ROM
    d) This module is where the hardcoded locations are located for the plotting demonstration

Design Resource and Statistics

LUT	1,349
DSP	0
Memory (BRAM)	469,120
Flip-Flop	569
Frequency	135.96 MHz
Static Power	90.06 mW
Dynamic Power	17.21 mW
Total Power	124.49 mW

Conclusion

This project was a great learning experience; it required us to understand a communication protocol (UART), build a driver/buffer to receive data sent in UART format, process that data, store it, read it, and display it visually in an intuitive way. Our project utilized the skills we learned in Lab 6 to do VGA display but also required research into UART and how to design a communication buffer in SystemVerilog. This project can be seen as a prototype for a consumer/industrial product which keeps track of locations of people or even animals. We enjoyed the freedom of the final project and the only thing which could make the instructions clearer would be some type of outline for the report, but we feel like we structured this report similar to and provided as much if not more detail than the past 7 labs reports.