Summary:

This activity will walk students through stacking and unstacking of the Adafruit Ultimate GPS Logger Shield (henceforth referred to as the Shield) and interfacing it with the Arduino Mega. By the end of this activity, students will know how to properly stack/unstack the Shield, have an understanding of the difference between direct connect and soft-serial connect, and will be able to parse raw NMEA sentences.

Materials:

Each student should have the following materials, equipment and supplies:

- Computer with Arduino IDE installed
- USB-AB programming cable
- Arduino Mega microcontroller
- Assembled Adafruit Ultimate GPS Logger Shield
    - Includes: GPS Logger Shield, coin cell battery, and 2 female-male jumpers
- Lock Ring Spreader Pliers

Procedure:

Activity A: Stacking/Unstacking a Shield

When a shield is attached to an Arduino, it is called stacking. Stacking and unstacking is not complicated, but it can result in bent pins if done improperly. Always be aware of the Shield's pins and do not force movement.

1. Get the Shield and the Mega. Make sure the Mega is disconnected from power.

   Steps 2 – 3 explain how to stack the Shield. Always be aware of the header pins and if they are at risk of bending.

2. Align the Shield with the Mega, as shown in Figure 1. Do not push down on anything yet. Just ensure the header pins align with the proper Mega pins.

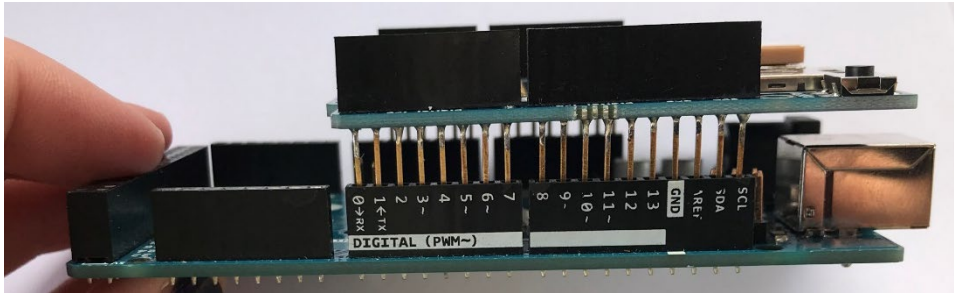

Figure 1: Shown is how to align the Shield with the Mega. The microSD slot is on the side closest to you.

3. Gently (do not shove it) push the Shield onto the Mega by applying even pressure. It is okay to rock the Shield slightly, but do not bend the header pins. Continue until the Shield is fully seated on the Mega; it should look like Figure 2.

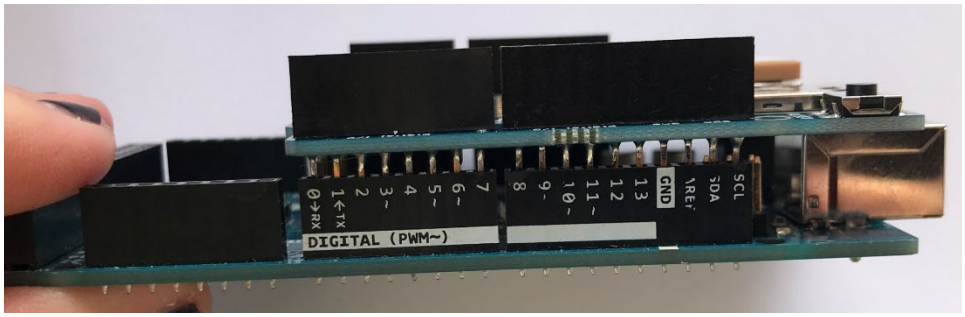

Figure 2: Shown is the Shield fully attached to the Mega. Part of the header pins are still visible. The Shield has a little bit of space between itself and the programming jack.

Steps 4 – 7 explain how to unstack the Shield and the Mega. Always be aware of the header pins and their risk of being bent.

4. The tool shown in Figure 3 is a lock ring spreader pliers. It can be used to unstack the Shield and the Mega while reducing the chances of pins bending. Open and close it a couple of times to get a feel for the instrument

Figure 3: This is the tool you will use to destack the Shield and the Mega. Using this lock ring spreader pliers will help prevent pins from being bent.

5. Stick the pliers in between the Shield and the Mega as shown in Figure 4. Gently push the Shield up just a little. Do not push it so far off that the pins on the other side of the Shield bend.

6. Move the pliers to the other position shown in Figure 4. Again, gently push the Shield up just a little.

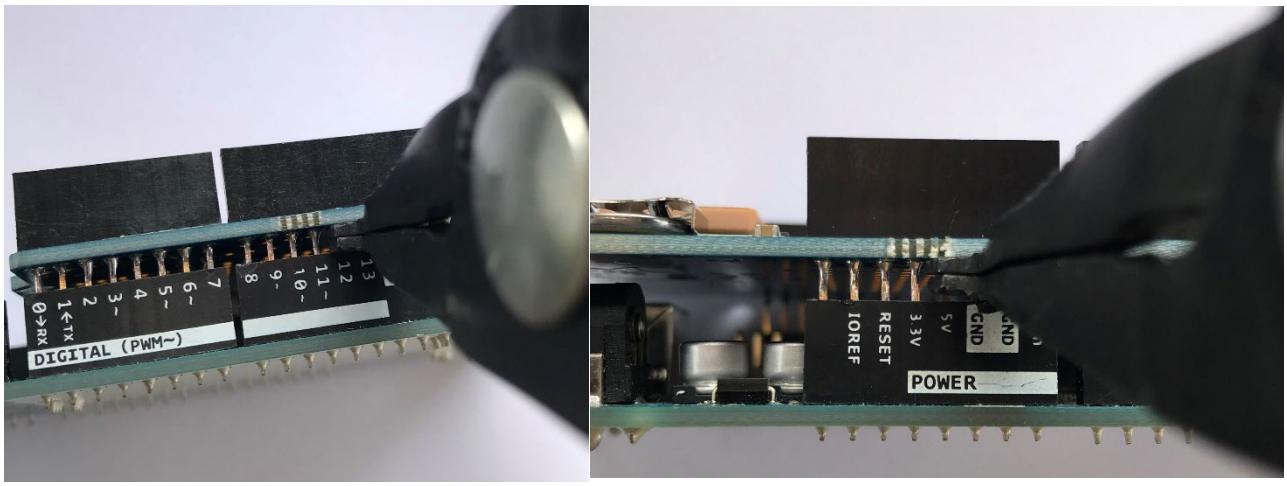

Figure 4: Shown are two of the places the pliers can be placed to separate the Mega from the Shield. Do no fully separate the pieces in one attempt. Go back and forth between these two positions at least three times to ensure that pins do not get bent while separating.

7. Repeat steps 5 and 6 until the Shield can just be lifted off.

Activity B: Direct Connect

1. If not already on the Mega, put the Shield on the Mega. Make sure the switch on the Shield is flipped to Soft Serial. This will allow a sketch to be uploaded on the Mega.

2. Create a new, blank sketch in the Arduino IDE. Upload this sketch to the Mega and open the Serial Monitor.
    a. Make sure the baud rate in the Serial Monitor is set to 9600 baud.

3. After uploading a blank sketch, we can flip the switch from "Soft Serial" to "Direct." This bypasses the Arduino chip so the Shield can communicate directly with the computer.

4. Raw GPS data should start appearing in the Serial Monitor. These are the NMEA sentences talked about in the GPS Lecture. Figure 5 shows an example of what your Serial Monitor might look like.

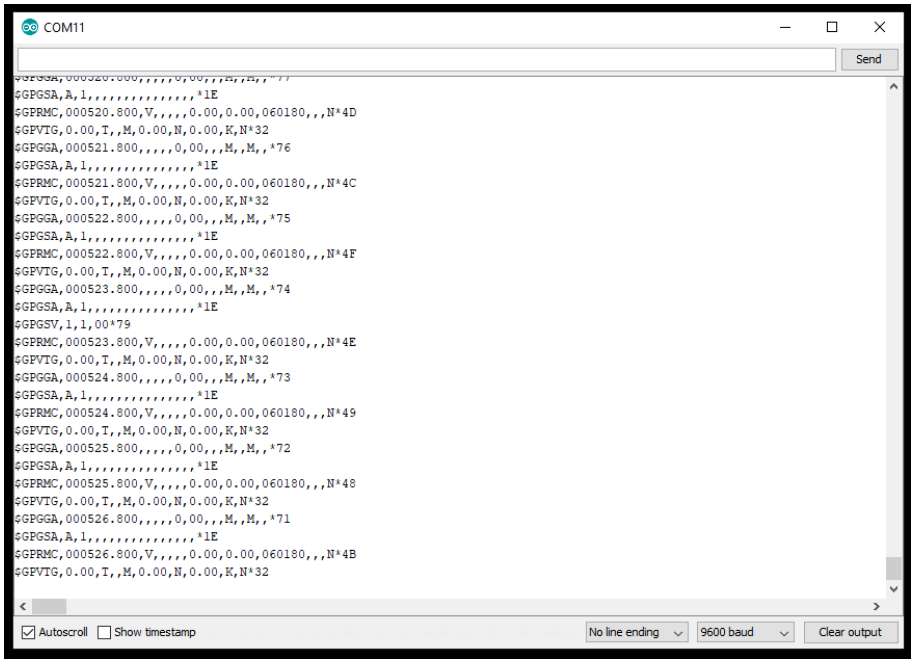

Figure 5: Shown is an example of what raw data from the Shield looks like using Direct Connect. This screenshot was taken when the Shield did not have a fix. If your Shield has a fix, there will be more information instead of blank data fields (the consecutive commas are blank data fields).

5. The rest of the activity works best if you are able to get a fix. If you don't have a fix try moving the Mega and Shield closer to a window. Depending on the location where you are working a fix may not be achievable. The red LED on the Shield will blink once every second if it does not have a fix. It will blink once every 15 seconds if it does have a fix, and the raw GPS data will look more like Figure 6.

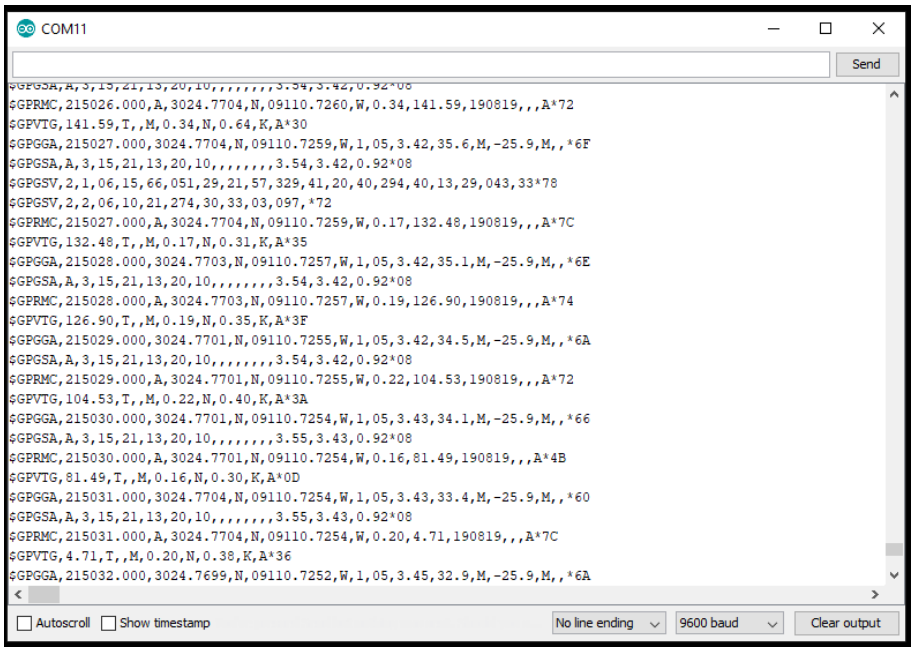

Figure 6: Shown is an example of what raw data from the Shield looks like using Direct Connect. This screenshot was taken when the Shield had a fix. When compared with Figure 5, more data fields are filled.

6. In addition to receiving the raw output of the GPS we can also send to the GPS using the text box at the top of the serial monitor.

7. Before sending commands to the Shield using the Serial Monitor, make sure that the "Both NL & CR" is selected next to the baud rate. This is because all the of the commands end with a <NL><CR>, this option will automatically add them to the string.

8. Below is a table of GPS commands and what they do. To send a command, type or copy it into the Serial Monitor, be careful and make sure you have the entire command and no spaces at the beginning and end. First send the command to turn off all the NMEA sentences

Table 1: GPS Commands

Command	Command Result
$PMTK314,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGLL sentence
$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPRMC sentence
$PMTK314,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPVTG sentence
$PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGGA sentence
$PMTK314,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGSA sentence
$PMTK314,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGSV sentence
$PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn on GPRMC and GPGGA sentences
$PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn on all NMEA sentence output
$PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn off all NMEA sentence output

NOTE: After correctly sending a command, a response will be sent from the Shield "$PMTK001,314,3*36." The $PMTK001 indicates this is an acknowledge packet. The 314 is the command was received, notice how this matches the **314** that we sent. The 3 is a flag that tells the command was valid and was executed successfully. A 0 would indicate an Invalid Command, 1 means a valid command unsupported by that hardware version, and 2 would mean a valid command that vailed to execute.

9. Experiment with all of these commands to achieve different outputs from the Shield. Look at R14.01 NMEA Strings to get more information regarding specific NMEA strings.

   A full list of all possible command strings can be found in the PMTK reference.

Activity C: Connecting to Alternate Serial Connection

1. Let's change to communicating with the GPS Logger Shield using Serial1 connection. First, disconnect the Arduino Mega from the computer. Flip the switch on the Shield so that it is selecting Soft Serial.

2. Connect your jumpers from the from the shield to the TX1 and RX1 of Arduino. Remember we want the TX of the Shield connected to the RX of the Arduino and the RX of the Shield to the TX of the Arduino. If you are having issues connecting to the GPS shield doublecheck your TX and RX connection. The Shield and Arduino Mega should look similar to Figure 7.

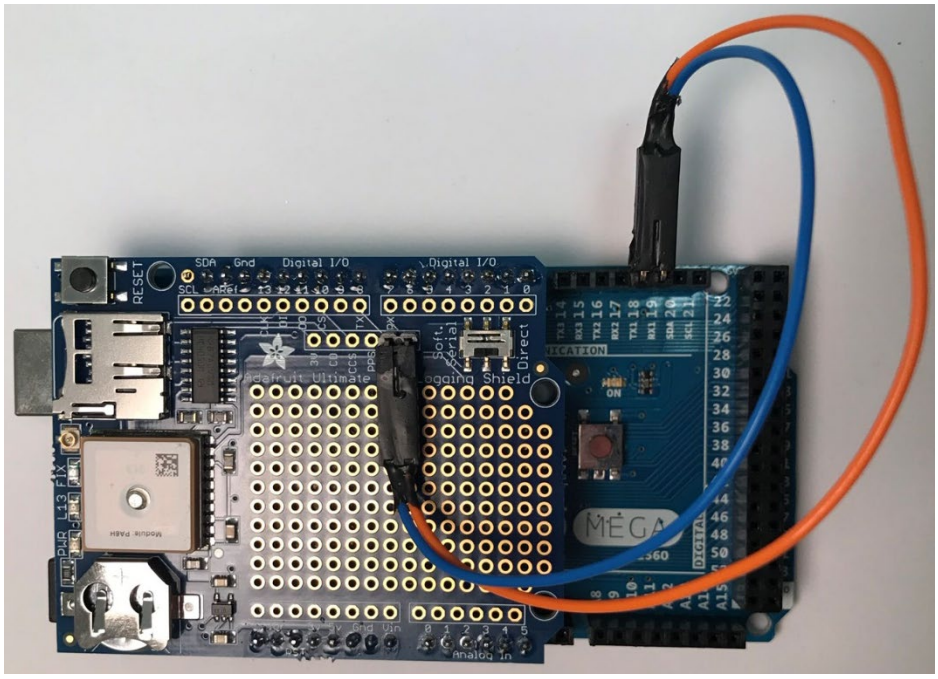

Figure 7: Shown is the Software Serial connection for the Shield and the Mega. The breakout pins RX and TX should be connected to the Mega pins TX1 and RX1, respectively. If these connections are switched, no data will be received from the GPS unit.

3. Next, open up a new sketch in the Arduino IDE and save it as *GPSSerialIntroduction.io*.

4. First, we need to include the library that is needed to communicate with the GPS. This is the Adafruit_GPS library. Figure 8 shows how to include it.

```
24 /* Include statements for required libraries ********************************************/
25 #include <Adafruit_GPS.h>     // Library for using the Adafruit Ultimate GPS Logger
```

Figure 8: This is how a library, specifically the GPS library, is included. This allows the user to access the library's functions

5. Figure 9 defines the pins that will be used for the GPS communication and connecting the hardware port to the GPS. We jumpered the breakout TX and RX to the Mega's Serial 1 (TX1 and RX1), so that is the port that we define as GPSSerial.

```
27 /* Adafruit Ultimate GPS Logger Shield ***********************************************/
28 #define GPSSerial Serial1     // Communicate with GPS using Serial1 (RX and TX 1)
29 Adafruit_GPS GPS(&GPSSerial); // Connect to the GPS on the hardware port
```

Figure 9: This code defines the GPS Serial port.

6. We want to define a constant for to enable the balloon mode, this particular command is not defined in the library so we need to create our own constant.

```
#define BALLOONMODE "$PMTK886,3*2B" //This defines the command string for ensure the GPS is in balloon mode.
                                    //This command is not defined in library so we need define it here and send the command
                                    //in the setup()
```

Figure 10: The line defines the string to set the GPS in to "Balloon Mode" which has an altitude limit of 80km in order to prevent issues with possible altitude cutoffs

7. We want to read the GPS using an interrupt. This means we do not actively query the GPS serial communication to read that data. Instead, a new character is automatically read every millisecond. In figure 11 we will define a function that we will use to turn the interrupt on and off as well as a global variable that lets us check status of the interrupt. We will write the useInterrupt() function and the actual interrupt function after setup.

```
31 /* Other global variables ***********************************************************/
32 boolean usingInterrupt = false;    // Flag for if GPS interrupt should be enabled
33 void useInterrupt(boolean);        // Define function to enable/disable interrupt
```

Figure 11: This code allows the use of an interrupt (the interrupt function will be defined later) to read the GPS. setup() should finish before the interrupt is enabled, so usingInterrupt is originally defined as being false

8. In setup(), we initialize the two serial objects for the two Serial ports we are going to use: the Serial Monitor communication(Serial) and the GPS communication(Serial1).

a. The Serial Monitor baud rate should be set to 115200. This will allow the Mega to print to the Serial Monitor without missing GPS data.
b. The GPS baud rate should be set to 9600 (a standard for GPS communication). Start this communication by with GPS.begin(9600);.

```
/* Begin serial communications ***************************************/
Serial.begin(115200);    // Begin communication with the computer
GPS.begin(9600);         // Begin communcation with the Adafruit GPS
```

Figure 12: This code sets up the 2 Serial ports we will be using at different Baud Rates, the GPS at 9600 and the USB serial with 115200.

9. Inside the setup we want configure the GPS to output the Figure 13 shows code that tells the GPS what types of information should be received and the rate at which the information should be sent.
    a. We use the predefined string constants from the library with the sendCommand() function to send the command. It is useful to have a block of code with all the NMEA sentences we might want so we can just comment and uncomment the appropriate line to get the output we want.
    b. The Library defines a number of command strings. The library documentation is available at https://github.com/adafruit/Adafruit_GPS . The commands are defined in the Adafruit_PMTK.h file.
    c. One thing to note is the NMEA output commands overwrite each other so if I call RMCONLY followed by GGAONLY you will get out GGA data only not RMC

```
/* Define data being requested from the GPS and at what rate **********/
GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);      // Set GPS update rate
GPS.sendCommand(BALLOONMODE);

// These define what NMEA sentence(s) will be received. Uncomment one
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GLLONLY);
GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_VTGONLY);
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GGAONLY);
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GSAONLY);
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GSVONLY);
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_ALLDATA);
//GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_OFF);
```

Figure 13: This code defines what information will be sent by the GPS and the rate at which the information will be sent. This will have the GPS send the module version, no antenna status, and only the RMC NMEA string. It will send updated information (i.e. a RMC sentence) every 1 second.

10. For now, just select RMC only and the 1 Hz rate and Balloon mode commands.

11. Finally after setting up the GPS we want enable the interrupt Below the commands from Figure 11, enable the interrupt that will read the GPS serial buffer. Do this by calling the useInterrupt() function and inputting true.

```
  /* Enable the interrupt that reads the GPS ****************************/
  useInterrupt(true);
}
```

Figure 14: As the final part of setup we want to call the useInterrupt with true to turn on the interrupt and begin reading the GPS

12. Below setup(), define the interrupt service routine, as in Figure 12. When enabled, it reads one character from the GPS serial port. This is the actual routine that reads the GPS and triggers every 1 ms.
    a. Interrupt routines work differently than regular functions. We use with the SIGNAL() or ISR() to tell the compiler this is a Interrupt Service Routine. Inside the parenthesis we put the signal that we want to trigger the routine.
    b. The specifics of these signals require reading into the actual Microcontroller Datasheet (AtMega2560 chip not the Arduino board). This particular signal occurs when the TIMER0 timer matches a specific value value.
    c. TIMER0 is the a counter that is used to generate the millis() clock ticks so it resets every 1 ms. This means that it will match a specific counter value 1 time every 1 ms, which means this ISR will get called every 1 ms.
    d. When the signal occurs the program will execute the code inside the function, in this case it just calls the GPS.read(); function. Note because the GPS library builds the NMEA sentence as we call read() we do no need to worry about the variable c.

```
66 SIGNAL(TIMER0_COMPA_vect) {
67 /******* Interrupt Service Routine ************************************************
68  *  This is the interrupt service routine. It reads a character from the GPS.
69  ********************************************************************************/
70   char c = GPS.read();      // Read a character from the GPS
71 }
```

Figure 15: This is the function that reads a character from the GPS serial buffer. It operates using Timer0, an internal timer on the Mega. Using Timer0 means that this function will run every millisecond when the interrupt is enabled.

13. Next, define the function that enables/disables the interrupt. Figure 16 shows this code. This code does 3 things, it picks the value that TIMER0 has to match, turns the interrupt on or off, and sets the usingInterrupt flag. If the interrupt is to be enabled, this flag is true. If the interrupt is to be disabled, this flag is false. This variable lets us check elsewhere in our code if the interrupt is turned on or off.

```
73 void useInterrupt(boolean v) {
74 /******* useInterrupt ************************************************************************
75  *  This function is enables or disables the SIGNAL(TIMER0_COMPA_vect) interrupt.
76  ********************************************************************************************/
77   // millis() uses Timer0, so we'll interrupt in the middle and call the function above
78   if (v) {
79     OCR0A = 0xAF;
80     TIMSK0 |= _BV(OCIE0A);
81     usingInterrupt = true;
82   }
83
84   // Do not call the interrupt function COMPA anymore
85   else {
86     TIMSK0 &= ~_BV(OCIE0A);
87     usingInterrupt = false;
88   }
89 }
```

Figure 16: This function enables or disables the interrupt. If it is passed true, it enables the interrupt. If it is passed false, it disables it.

14. We recommend using this particular set of interrupt functions everywhere you want to use the GPS to ensure you do not miss an output.
15. The main loop should just be an if statement. If a new NMEA sentence is received, print the sentence to the Serial Monitor. Figure 17 shows how loop() should look.

```
91 void loop() {
92 /******* loop ************************************************************************
93  *  This function runs repeatedly. It prints new NMEA strings to the Serial Monitor
94  ********************************************************************************************/
95   // If a new NMEA sentence is received, print it on the Serial Monitor
96   if (GPS.newNMEAreceived()) Serial.print(GPS.lastNMEA());
97 }
```

Figure 17: This is main loop. Every time it runs, it checks if a new NMEA sentence has been fully received. If it has, then the sentence is printed to the Serial Monitor.

16. Plug in the Arduino Mega and upload the sketch. Open the Serial Monitor and set the baud rate to 115200. When running this should just copy the output of the GPS to the Serial Monitor.

17. Try using other GPS commands to turn on different NMEA outputs. Make sure to try every command from Figure 13 at least once and understand the information that is being printed out on the Serial Monitor.

18. Using the GPS library documentation, try commands to change the update rate. Does the new output in the Serial Monitor make sense?

Activity E: Parsing Data Introduction

1. While recording the raw GPS data can be useful we often want to record specific pieces of the data with our other data. It is often beneficial to parse NMEA sentences to extract specific information. This activity will show how to use the GPS library to parse information automatically.
2. The setup used before will work fine and does not require alteration except for we want to turn all of the outputs.

3. First test calling the GPS.Parse() function. Figure 15 shows the command to parse a NMEA sentence. If a parse is successful, the sentence "Parse was successful!" and the sentence will be printed to the Serial Monitor.
    a. The command GPS.parse(GPS.lastNMEA()) returns a Boolean. If the parse was successful, it returns true. If it failed, if for example the string passed has a bad checksum it returns false. Because of this, it is beneficial to use the command as the condition of an if statement. If the parse is successful, then the if statement runs.
    b. Also remember that since we have multiple NMEA sentences being output so we will have multiple successful parses in a single second.

```
91  void loop() {
92  /******* loop ********************************************************************************
93   *  This function runs repeatedly. It prints new NMEA strings to the Serial Monitor.
94   ********************************************************************************************/
95    // Check if a new NEMA sentence has been received
96    if (GPS.newNMEAreceived()) {
97
98      // Try to parse the NMEA sentence.  If it can't be parsed, exit the if statement.
99      if (GPS.parse(GPS.lastNMEA())) {
100       Serial.println("Parse was successful!");
101       Serial.print(GPS.lastNMEA());
102     }
103   }
104 }
```

Figure 18: For the goal of printing "Parse was successful!" and the NMEA sentence, this is an example of how the loop() function could look. This is not the only way to achieve this end result.

4. After we parse the sentences the internal variables of the GPS object get updates. For example we can get the hours, minutes, and seconds from the GPS using GPS.hour, GPS.minute, and GPS.seconds. Notice these are variables and not functions but variables so we do not use parenthesis.

5. Let's have loop() print out the current seconds from the parsed information. Beneath Line where we print the lastNMEA in Figure 15, add the lines *Serial.print("The current seconds are: ");* and *Serial.println(GPS.seconds);*. Upload the sketch. You should see the seconds printed out.

6. Edit the loop() function so that it prints out the following to the Serial Monitor after a successful parse:

   Parse was successful!
   Current time is HH:MM:SS

   where HH:MM:SS is the current UTC time. Show an instructor before proceeding.
   a. UTC stands for Coordinated Universal Time. It will not be the same as your time. This is the same as GMT. In Louisiana depending on daylight saving, UTC will be 5 or 6 hours ahead of you.

7. Now we may not need all that data every single second, in flight we may only need the data to updated at a slower rate. We can set up variables that we use as timers so we output variables every 5, 7 and 13 seconds. Because we want to access these timers in multiple spots we want to make them global variables so we want declare them before the setup

```
/* Timers ***************************************
unsigned long timer5s = millis();
unsigned long timer7s = millis();
unsigned long timer13s = millis();
```

Figure 19: Initialization of the 3 timers we are going to use to output date. Setting them equal to millis() gives them the current millisecond value as the current time. Also, we have picked 3 prime numbers. It is often useful have thing occur on prime numbers to minimize the time they will occur at the same time.

8. First edit the sketch so that every 5 seconds, the current time is printed to the Serial Monitor as well as the GPS fix quality and number of satellites being used for the GPS fix. With the exception of the time, make sure that all the information is labelled.
   a. Hint: It will be beneficial to use variables and if statements.

b. Hint: The function millis().

```
// Do this every 5 seconds
if ((millis()-timer5s) > 5000) {
  // Print current time
  Serial.print(GPS.hour); Serial.print(":");
  Serial.print(GPS.minute); Serial.print(":");
  Serial.println(GPS.seconds);

  // Print fix and satellite information
  Serial.print("Fix: "); Serial.println(GPS.fix);
  Serial.print("# of satellites: "); Serial.println(GPS.satellites);
  Serial.println("");       // For readibility, it's nice to have an extra line

  // Reset timer
  timer5s = millis();
}
```

Figure 20: This should if statement will occur when the current time is 5000 milliseconds greater than the last time the if statement executed. It outputs the desired variables and then sets the 5 second timer to the current time. Note this uses the Arduino timer for the 5 seconds which is not as accurate as the GPS. We could also use the GPS.seconds, but would need to be concerned about affects of failed GPS communication or loss of fix.

9. Now add code so that every 13 seconds, it prints out the speed and current date. Include a label for the speed.

10. Every 7 seconds print out the most recent GLL NMEA string and position information (altitude, longitude, and latitude). For this you will need check if the sentence is a GLL (Remember it will start with $GPGLL) and copy it to another variable.

This activity did not cover all possible internal variables, be sure to look at the GPS library documentation to find the other variables. Also in general having extra sentences on is not a good idea because they can potentially overwrite information before you can read it. If you loop takes starts to take ~1 second/Number of sentences you will not parse the sentences before the LastNMEA gets overwritten. To combat this determine the minimum number of sentences you need and call GPS.parse() frequently.

NOTE: Included in the sample code is sample code showing how to manually parse the GPS strings which is useful for reference but will not be done as a separate activity.