MIE438 - Microprocessors and Embedded Microcontrollers

Automatic Bicycle Transmission System

Final Project Report - Team PID

Parthkumar Parmar -
Isaac Luong -
Danyal Rehman

Table of Contents

List of Tables

List of Figures

1 Summary of Project Objectives

The primary goal of the original project was to design an Automatic Bike Transmission System based on applied torque, with the intention of enabling the rider to fully focus on steering the bike as opposed to shifting gears at regular intervals. Additionally, it allows the team to incorporate multiple aspects of embedded systems programming within one project; the system requires reading sensor data, wireless transmission, data processing, information display, and actuation. Lastly, the project allows the team to couple electronics with mechanical design giving thorough exposure to practical design constraints and hardware.

1.1 Proposal Revisions

After receiving feedback on the original project proposal, the team decided to slightly change the scope of the project. The new scope of the project was the development of a load-controlled wireless drivetrain. The new system was tasked with being able to read torque from a load cell using an Arduino, wirelessly transmit the data to a PIC microcontroller, and in turn send a custom PWM signal from the PIC to a DC motor to control its speed. In addition, commands could be entered into a computer connected to the Arduino which would transmit user commands to the PIC in order to control the direction of the motor rotation.

1.2 Microcontroller Selection

The project originally consisted of two microcontrollers – an Arduino and PIC. The Arduino was selected since the team members are familiar with its operation and capabilities. The PIC was selected so that the team could obtain experience in using a microcontroller other than an Arduino. Additional emphasis was given to accessing, understanding and using microcontroller features that were demonstrated in lecture and labs that Arduinos generally conceal and take care of, for the user. After changing the project scope based on the interim report feedback, the team decided to keep both microcontrollers for the new project scope. However, rather than attempting to add the mechanical complexity of a gear transmission system to the final design, the team focused on exploring the PIC microcontroller?s capabilities and its direct applications to this project.

1.3 Microcontroller Evaluation

Having completed the project, the team concludes that both the Arduino and PIC were able to fulfill their objectives satisfactorily. Arduinos could have been used for both microcontrollers but that would have lowered the difficulty of the project significantly due to its ease of use and extensive libraries. Conversely, the team feels confident that PICs could now be used for both microcontroller components of the project. However, one significant downside to this approach is established during the debugging state. Since the Arduino has a built in serial monitor in the boards and IDE, it can be easily setup to display messages and data. The PIC does not have a serial monitor and the team had to rely on using a blinking LED to assist with system debugging before the LCD screen was setup. In acknowledgement of this, it is not practical to hookup an LCD to the PIC as a "serial monitor" for every project.

2 Final Design

2.1 Embedded Nature

The reason that our design requires the use of an embedded system is its portable nature. It can be noted that if the goal was to design a commercial product out of this prototype, there will be no change in the device's overall functionality over its life. Additionally, it can be noted that the system is to be performing its task highly predictably. These are both typical traits of an embedded system. All of the aforementioned points show different aspects of an embedded system, for which, microcontrollers are ideal to use.

2.2 Design

From a high-level perspective, the design can be compartmentalized into the following stages:

1. There is an input into the system by the user that determines which direction the motor is going to spin.
2. The system uses a load sensor which translates the pressure applied into a measurable electrical signal using strain gauges.
3. The analog signal exiting the load cell is initially amplified, and then recorded by the Arduino in a digital form.
4. The Arduino uses serial communication to write the data to a Bluetooth master module.
5. The module transmits the received data to a slave module which is coupled with a PIC18 microcontroller for wireless communication.
6. The PIC18 then sends a PWM signal that is proportional to the load applied, to actuate the attached motor[1].
7. Additionally, the LCD display is also updated to illustrate the motor's rotation direction as well as the load values read.

Highlighted below in Figure 1 illustrating the final design and setup.

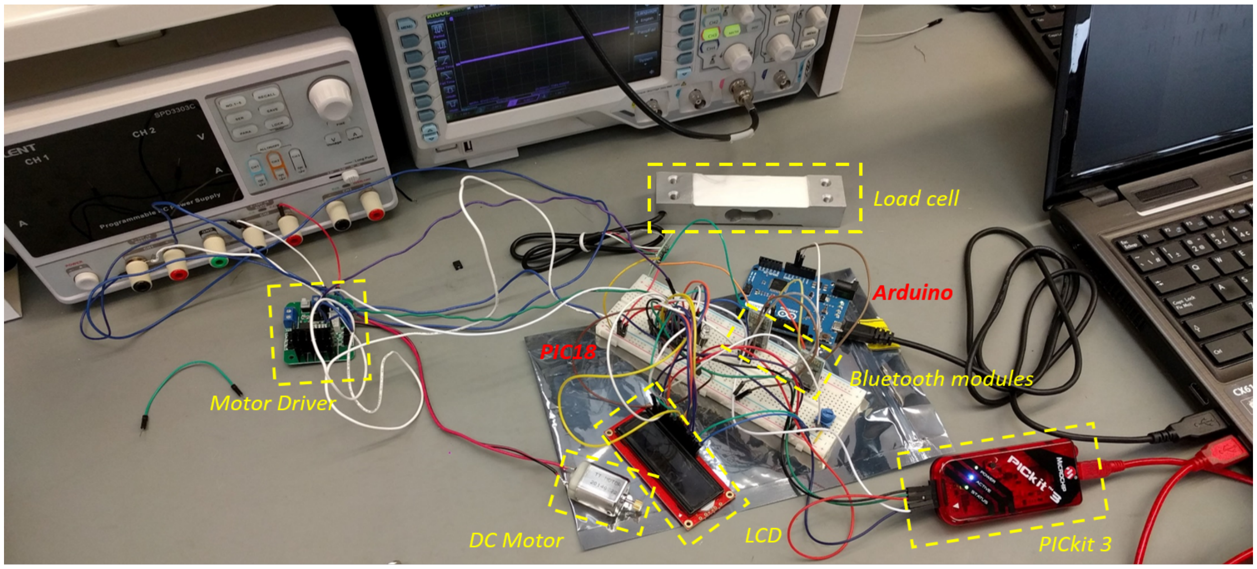

Figure 1: Final Design Setup

Lastly, the reason the team identified the PIC18 over an Arduino was simply for the learning experience. Due to the well documented nature of the Arduino, the team were of the stance that it can be learned at any point. However, a PIC requires significantly more effort given the team's lack of experience. Additionally, the team were interested in the application of concepts learned within lectures and labs, thus, revalidating the requirement for a PIC microcontroller.

Additionally, all the I/O devices that were used within the system are highlighted in Table 1.

[1]The PIC18 exploits interrupts to receive the incoming data, and also differentiates user input from load values (*done using the first character that is received, $ or **).

Type	Device	Connection to	Reasoning
Input	Load Cell	Arduino	To measure the applied load
Input	Potentiometer	LCD Screen	To adjust the contrast of the LCD screen
Input	Keyboard	Arduino (*via hardware serial port*)	To allow user input for commanding the movement of the motor
Input/Output	Bluetooth	Arduino and PIC	To transfer the data from Arduino to PIC
Output	LCD Screen	PIC	To display the load sensor value and user inputs/current motor mode
Output	Motor	PIC	To vary in output speed based on PWM signals from the PIC
Output	LEDs	PIC	To indicate the reception of the data and to act as a visual event flag

Table 1: Arduino Connection Description

3 Hardware

The hardware sections are subdivided into 2 sections; Arduino and the PIC18's circuits respectively.

3.1 Subcomponent 1 - Arduino and Circuit

The Arduino selected was the Arduino Leonardo. Since the task of the Arduino was to read data from a sensor and the computer and to transmit data over Bluetooth to the PIC, the team ensured that the Arduino had enough digital pins to provide these functions. Although two serial communication ports were needed (one for the computer and one for the Bluetooth), the team was aware that a software serial port could be created using any pair of digital pins on the Arduino so the single hardware serial port provided on the Arduino Uno was not an issue. In addition two digital pins were necessary for communication with the load cell amplifier – a data line and a clock line. The Arduino was also used to provide power to some of the electronics in the circuit. The Arduino was not the main focus of the projects so many of its features were left unused such as its analog to digital converters, PWM features, and the various communication protocols other than serial. This includes the analog pins, the communication pins, and the PWM pins. Please see Appendix 8.2 for the the electrical schematic of the Arduino subsystem.

Please see Table 2 for additional details regarding pin setup and functionality.

Feature	Functionality
EUSART/Hardware Serial (USB port)	Serial communication with computer
GPIO (*2 pins*)	Software Serial - Communication with Bluetooth (*master*)
GPIO (*2 pins*)	Communication with HX711 amplifier (*data and clock*)

Table 2: Arduino Connection Descriptions

3.2 Subcomponent 2 - PIC18 and Circuit

The selection process for the PIC, however, was more detailed since it needed to be connected to more devices and perform more functions. The hardware connected to this microcontroller consisted of a LCD screen, a bluetooth module, a motor driver, and the PICkit 3. Since many peripherals on the PIC can share pins, it was important to ensure that the selected microcontroller did not have pin overlap for the necessary peripherals. The LCD screen required 11 pin connections with the PIC – 8 pins for writing data and 3 pins for selecting register select, read/write, and enable lines. The PIC18 had three memory-mapped ports available and it was important to the team that one port could be used entirely for the 8 data lines. This ensured that writing data to the LCD was as simple as writing a binary or hex number to one of the ports. In this project, PORTA was used for writing data to the LCD as pins on PORTB and PORTC were mapped to other peripherals necessary to the project. PORTC contained the TX1/RX1 pins needed for the use of the EUSART1 peripheral to communicate with the Bluetooth module. PORTB contained enhanced PWM peripheral used to command the motor driver. The R/S, R/W, and EN pins for the LCD as well as the INA and INB pins for controlling the motor mode did not need to be assigned to any specific pins were allocated according to wiring convenience. One of the microcontrollers was used for setting up the PWM module; however, no external connections were needed for this peripheral. Features that were not used include its ADCs, comparators, communication peripherals other than EUSART, and external interrupt pins. Refer to Appendix 8 for the electrical schematic of the PIC subsystem.

Please see Table 3 for additional details regarding pin setup and functionality.

Feature	**Functionality**
EUSART (TX, RX)	Communication with Bluetooth (*slave*)
GPIO (*11 pins*)	Interface with LCD (*PORT B + RS/RW/EN pins*)
GPIO (*2 pins*) - PWM	Interface with DC motor (*INA/INB + EN*)

Table 3: PIC18F23K22 Connection Descriptions

4 Software

Refer to Appendix 8.3 and 8.4 for the Arduino and PIC18F23K22 code respectively.

4.1 Program Description

4.1.1 Reading and Transmitting Data

Upon program startup, both microcontroller systems await data input. The PIC waits for data transmission from the Arduino over Bluetooth. The Arduino waits for data from the load cell amplifier, which is connected to its digital pins, or a user command entered into the computer which it receives from its hardware serial port.

The team used an Arduino library developed for the HX711 load cell amplifier to communicate with the sensor [1]. Data is read from the load cell amplifier using a clock line and a data line. The clock line is an output from the Arduino to the amplifier while the data line is an output from the amplifier to the Arduino. In operation, the data line is held high and the clock line is held low until data is ready to be sent from the amplifier. When this condition is met, the data line is set low and the Arduino sends out 25 clock pulses to the amplifier. As the amplifier receives each clock pulse, a bit of data is pulsed back to the Arduino across the data line. After receiving 24 bits of data, the Arduino will send the 25th clock pulse which will inform the amplifier to bring the data line back to a high state until data is ready again [2]. The 24 bits of data sent from the amplifier form a binary number in 2's complement. The corresponding load is calculated and cast into an 8-bit integer.

Once data is received from either source, the Arduino sends two bytes of data to the PIC through the Bluetooth module. The first byte is an identifier character (either * or $) which tells the PIC whether the following byte is data from the load cell or a user command. Since serial data is sent only when the load cell value changes or a user enters a command – the data can be sent irregularly and infrequently – the PIC captures the data from the wireless transmission through an interrupt service routine (ISR) using its EUSART peripheral. In this ISR, which is entered whenever data is available from the Bluetooth, the PIC will record the byte in a 64 element buffer and reset the data received interrupt flag. In the main code for the PIC, the program continuously polls to see whether there is any data in the serial buffer. If there is data available, the PIC will read two bytes from the before. As mentioned before, the first byte will tell the PIC whether the following byte belongs to the load cell or is a user command. After reading the second byte, the PIC will execute instructions based on the data it received.

4.1.2 Displaying and Driving Motor

The PIC is connected to a LCD screen and a motor driver. The team developed a library for the LCD screen in the PIC environment with functions such as screen initialization, writing commands to the screen (e.g. clear screen, move to first line, move to second line), and writing data to the LCD (*e.g. writing one byte or writing a string of data*). The motor can be driven in three different modes which are commanded with user input: stop, forward, and reverse. Based on the data it received from the Arduino, the PIC will update the LCD with a new load value or new user command.

Based on the user commanded motor mode, the PIC will drive the motor with a specific speed and direction using one of its PWM peripherals and two digital pins which control the input pins (*INA or INB*) of the attached motor driver. In stop mode, the PIC will not output any PWM signal and the input pins of the motor driver will both be set to zero. In forward mode, INA will be set high and INB will be set low while the PIC outputs a PWM signal to the motor drivers enable pin that is proportional to the force being applied to the load cell. The function of the code in reverse mode is identical to that of forward mode except the logic levels for INA and INB are reversed.

The entire program flow for the two microcontrollers with respect to the project is shown in Figure 2 below.

`design_process_flowchart.png`

Figure 2: Design Process for Revised Project Scope

4.2 Programming Language Selection

The project was programmed in a high-level language (*i.e. C*) for both the Arduino and the PIC microcontroller. This language was selected for the Arduino since the team is most familiar with its usage and the programming language is used by the Arduino IDE. The PIC microcontroller was also programmed using C. Although the IDE for programming the PIC – MPLAB X – does provide the ability to program the microcontroller using assembly, similar reasons for using C on the Arduino stand for the PIC. In addition, the assembly language instruction set and format used in the PIC environment is different from what was studied in class and used on the Dragon12-Plus2 Evaluation Boards during the labs. The team decided that it would be better to spend time and effort in learning how to program a PIC microcontroller and use its peripherals in C, which the team is already familiar and comfortable with, rather than attempting to learn about a new microcontroller and learning how to program in an unfamiliar assembly language at the same time. It should be noted that MPLAB X has a plugin that provides a graphical user interface for programmers to configure the system and peripherals and generates source and header files for the system based on the user selections (*see Appendix 8.4 for more detail*). The team was able to successfully use the PIC microcontroller and its peripherals and would likely use C again rather than assembly. Assembly would likely only be used if the program is fairly simple or needs to be heavily optimized in a way that the compiler is unable to replicate (*e.g. code needs to be executed extremely quickly*).

5 Bill of Materials (BOM)

The table given below summarizes the major hardware/sensors used in this project.

6 Final Result and Potential Enhancements

The team was unable to develop the mechanical system envisioned in the original project. However, the project was successful in giving group members the experience they wanted in using a PIC microcontroller. Overall, the system was successful in transmitting and decoding multiple types of data from different sources over a single channel of communication. The team was able to incorporate an ISR to help with the reading and storing of serial data. The system was also able to translate a change in force applied to the load cell into a proportional change in duty cycle for the PWM signal. Despite this, the team was unable to calibrate the applied load into an accurate measurement – the system only showed whether there was an increase and decrease in load and not what the exact value of the load was. As such, the full range of the load cell sensor could not be fully used and mapped to the duty cycle of the PWM signal.

In terms of potential improvements to the project, a mechanical drive train system could be built for better evaluation of the system. This would enable the team to use encoders for sensor feedback of the motor. In this way, the team would be able add a control system to the project using the PIC microcontroller. The force applied to the load cell could be mapped to a specific motor speed and the encoders would be used to send feedback to the PIC which would adjust the PWM signal accordingly using PID.

7 Conclusion

The primary goal of this project was to design and build an embedded system which can imitate a wireless drive train system. In order to achieve this objective, the team made use of the content learned during the course lectures as well as practical sessions. The team?s prior familiarity with Arduinos and circuit building fundamentals from other related courses helped in making this project successful. One of the goals of this project was to explore and gain experience in using a microcontroller other than an Arduino. This is because the Arduino environment is designed for hobbyists/beginners and ease of use and achieves this goal through its extensive libraries and the wrapping of many microcontroller features into "pre-packaged" functions. The team selected a PIC18 microcontroller to be used as part of the project and now feels confident comfortable enough that it would be possible to replace the Arduino in the project with another PIC. The team thoroughly enjoyed learning various aspects of embedded system as well as capabilities of microcontrollers and its use in the real world applications.

8 Appendices

8.1 PIC18 Subassembly

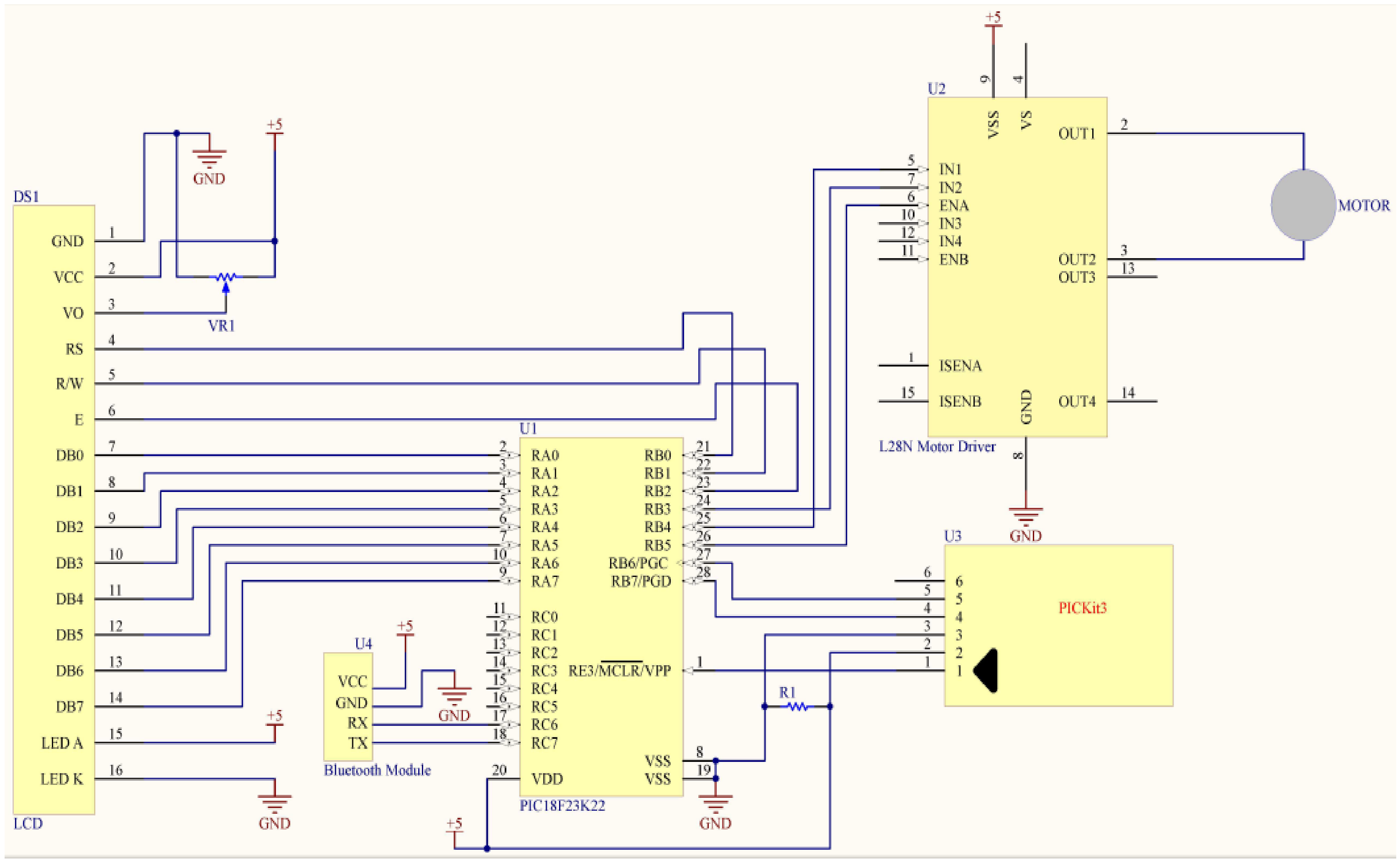

Figure 3: PIC Subassembly

8.2 Arduino Subassembly

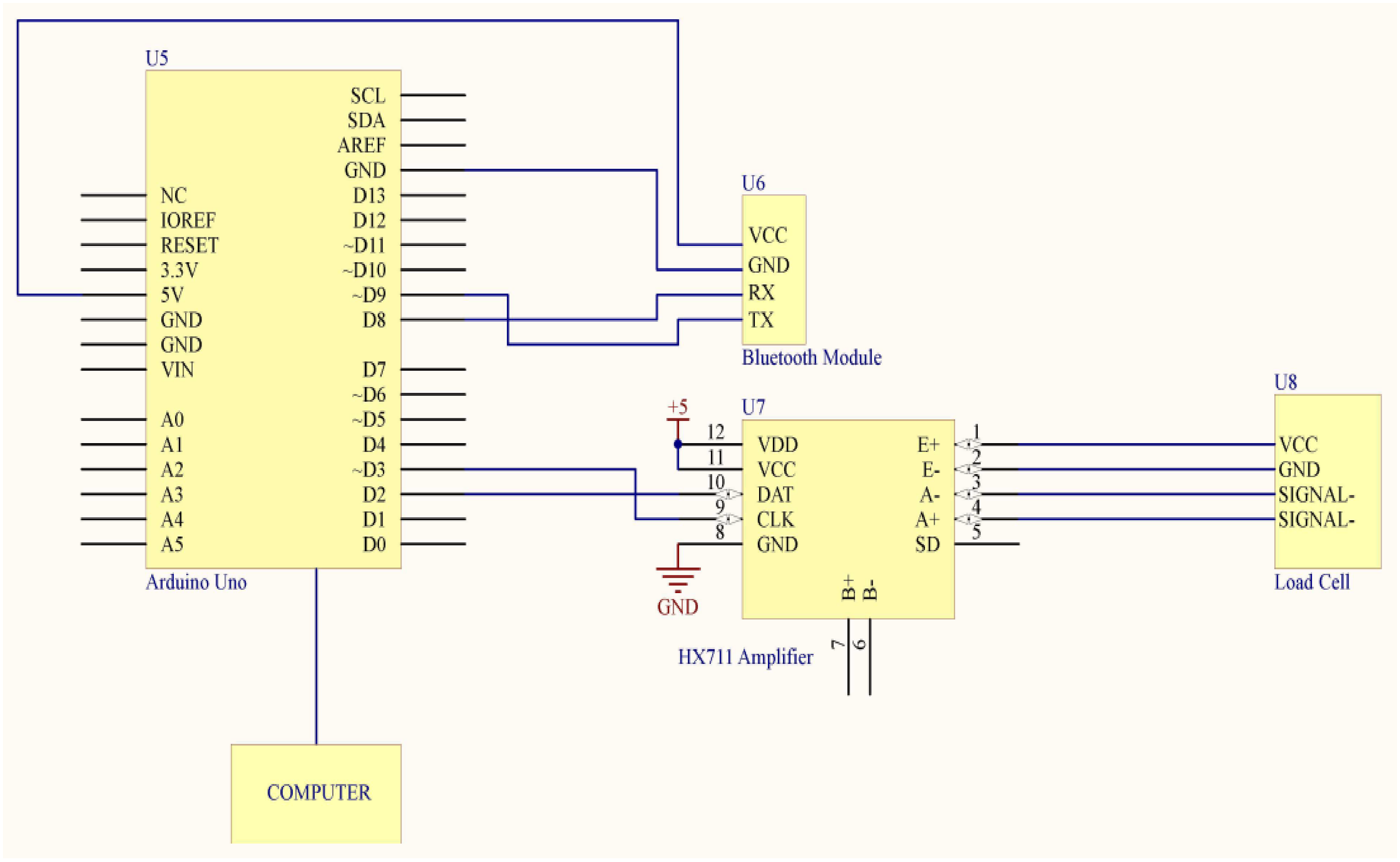

Figure 4: Arduino Subassembly

8.3 Arduino Code

```
#include <SoftwareSerial.h>
#include "HX711.h"
#define rxPin 7
#define txPin 8
#define calibration_factor -7050.0 //This value is obtained using the SparkFun_HX711_Calibration sketch
#define DOUT 2
#define CLK 3

SoftwareSerial picSerial(rxPin, txPin); //Cretes SoftwareSerial port using pins 7 and 8

HX711 scale(DOUT, CLK); //Creates HX711 object using pins 8 and 9

char userIn; //Store serial input from computer
int prevLoad; //Store previous load cell data
int currentLoad; //Store current load cell data

void setup()
{
  //Setup serial ports
  Serial.begin(9600); //Hardware serial
  picSerial.begin(9600); //Software serial

  //Setup HX711 and scale
  scale.set_scale(calibration_factor);
scale.tare(); //Assuming there is no weight on the scale at start up, reset the scale to 0

  Serial.println("SYSTEM␣INITIALIZED");
  Serial.println();
}

void loop()
{
//Obtains data from the load cell and sends it to the PIC if the new value is different form the previous value
  prevLoad = currentLoad;
  currentLoad = scale.get_units();
  if (!(currentLoad == prevLoad))
  {
picSerial.write('*'); //Sends identifier of '*' to the PIC so that it knows that the following data is from the load cell
    picSerial.write(currentLoad);
    Serial.print("Load␣Value:␣");
    Serial.println(currentLoad);
  }

  if (Serial.available())
  {
    userIn = Serial.read();
picSerial.write('$'); //Sends identifier of '$' to the PIC so that it knows that the following data is user input
    picSerial.write(userIn);
    Serial.println();
    Serial.print("Command␣Entered:␣");
    Serial.println(userIn);
    Serial.println();
  }
}
```

8.4 PIC18 Code