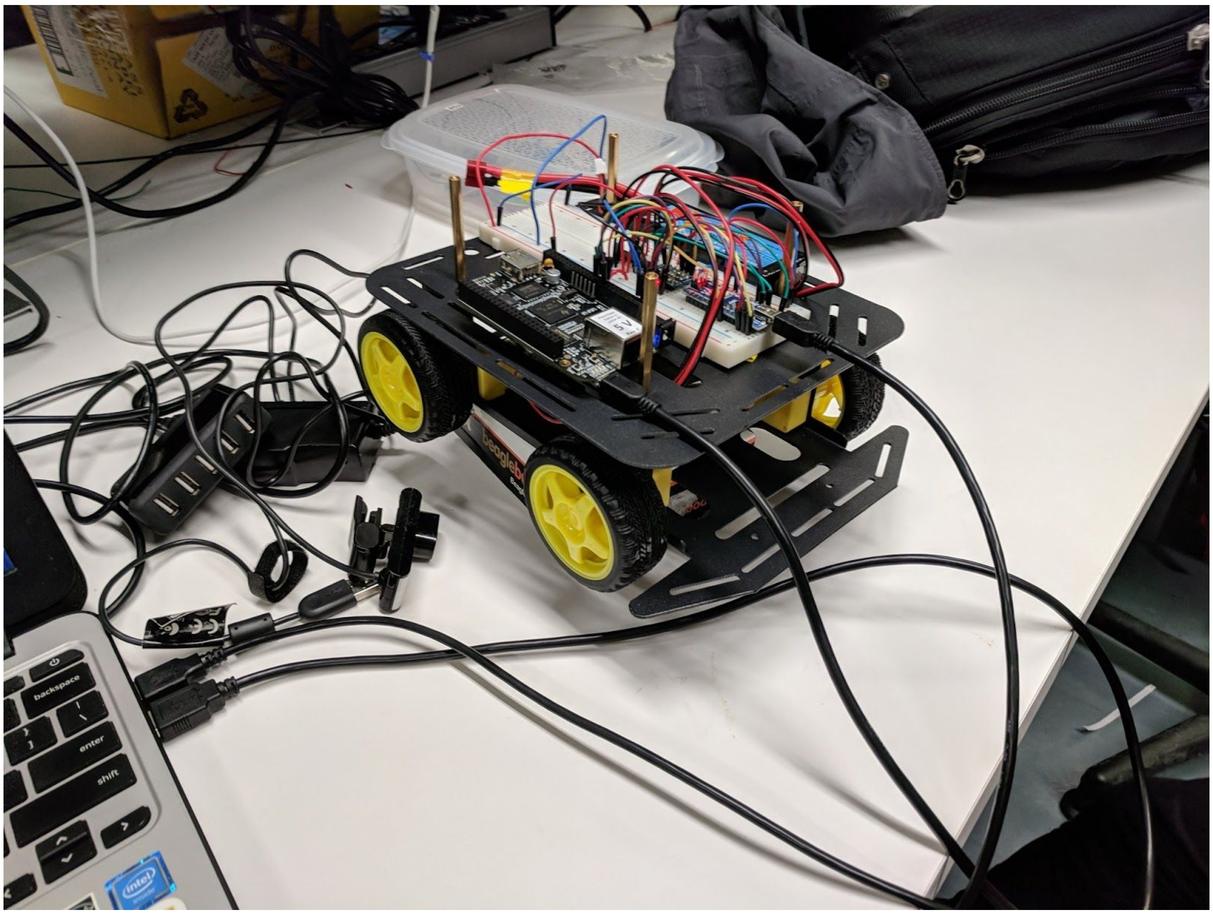
Concordia University
Gina Cody School of Engineering and Computer Science

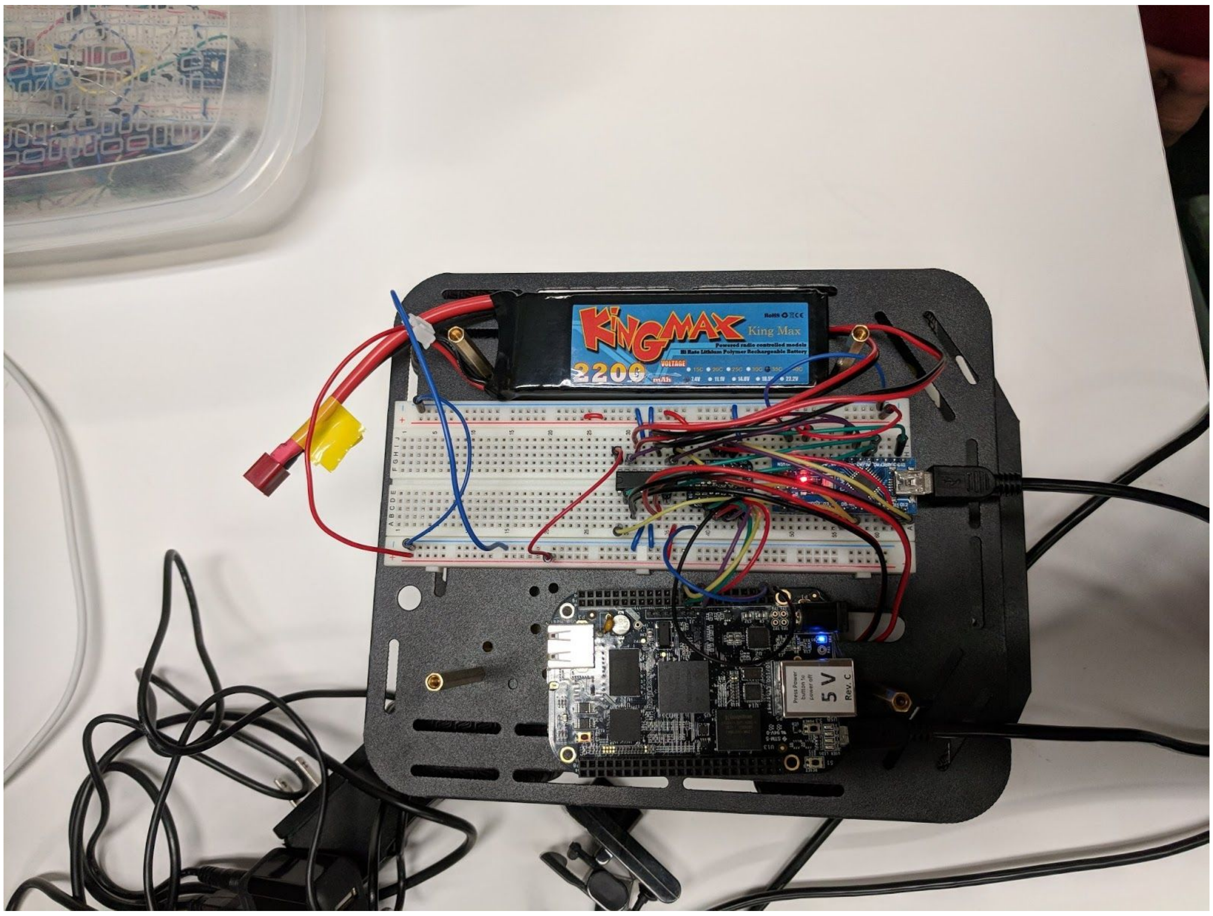
Embedded Systems
SOEN 422/2 --December 2018-- Section MM

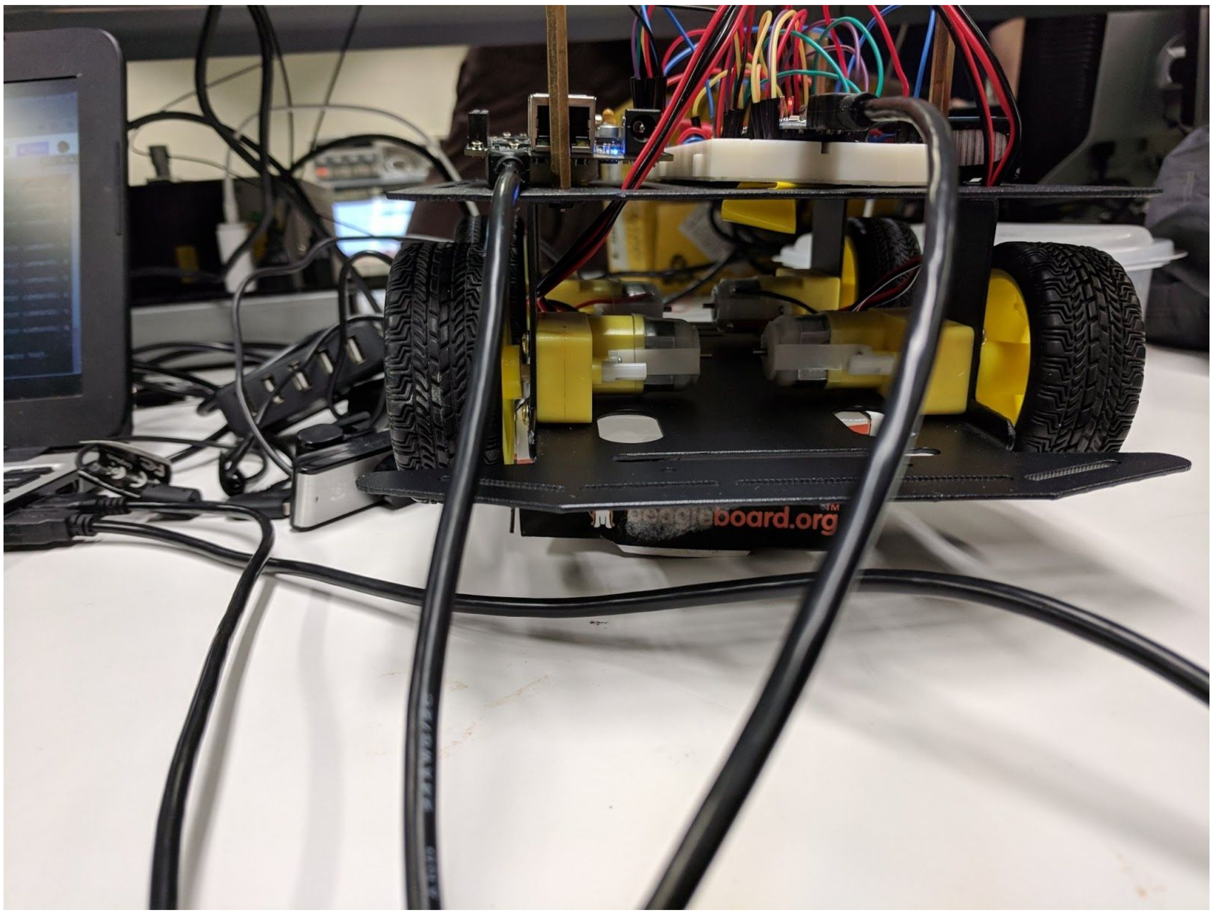
Project Final Report

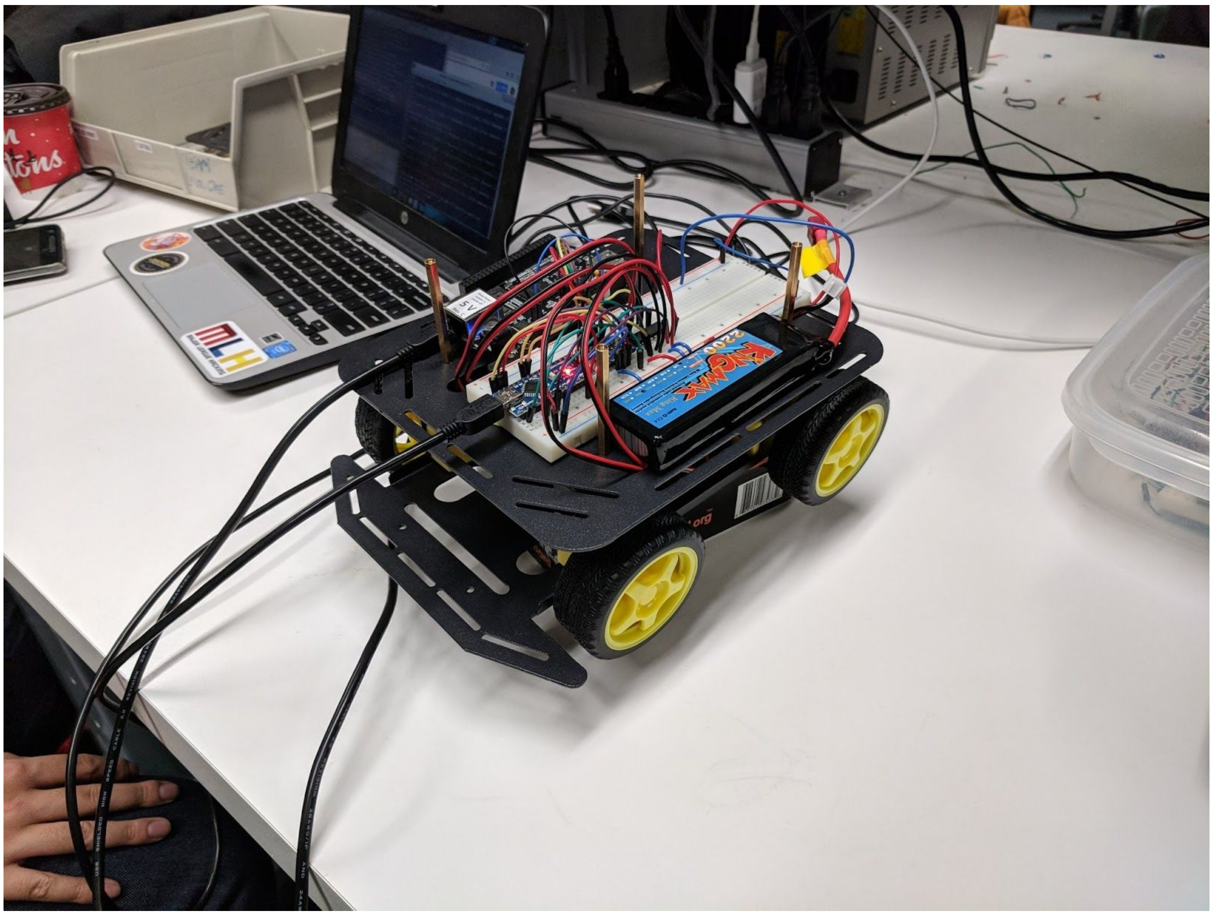
Our report for the final project, “Rally Cam”, which uses an Arduino Atmega328p and a Beaglebone black.

Team Level One	
Member Name	**Member IDs**
JIMMY LE	26546986
JOHN HUA	27056958
MAXIM NGUYEN	27564171
ZIAD YARBOUH	27762585






beagleboard.org



0.Table of Contents

1. Introduction

The complexity of this project is meant to demonstrate the culmination of knowledge gathered throughout this course. It covers a wide range of topics such as bare-metal functions, SPI communication, motor control, and Beaglebone scripts, as well as new things, such as using level shifters, webcams, and connecting all the moving parts. After everything is done, we hope to add yet another experience to the ones we’ve already gained.

2. Project Description

The goal of this project was to make a remote controlled car with a mounted camera. The user would be able to control the acceleration and direction of the vehicle, while displaying a live camera feed.

2.1 The name and purpose

Our project is named the “Rally Cam” and it was built to provide live footage in motion.

2.2 Unit function and performance

The car is supposed to move and manoeuvre with the skill of the driver. Through the use of synchronized motors, it should accelerate and change directions according to the user’s input. The camera is expected to show the vehicle’s point of view during usage.

2.3 Project Hardware

The vehicle has two mounted microcontrollers and four wheels powered by four DC motors. There are two platforms on the vehicle, the lower one houses the motors, battery pack, and the arduino. The top platform holds the beaglebone black, the usb camera, and the usb hub. There are usb wires that run from the car to the laptop that controls it. On the laptop, there will be a camera feed that displays everything from the point of view of the car.

2.3.1 The arduino Nano

The arduino is the SPI slave that receives commands from the beaglebone black SPI master. We designed the code so that depending on what signal it received from the master (the beaglebone), it would match this input to the appropriate command to enable the motors to rotate in the specified direction.

2.3.2 The Beaglebone Black

The Beaglebone Black is the microcontroller that does the most work between the two boards. It sends commands to the Nano, receives user input and displays a camera feed. It communicates to the arduino through serial peripheral interface (SPI) communication. After the user specifies the direction command, it sends an integer command as a hexadecimal to the nano. The nano reads the integer and engages the motors appropriately.

2.3.3 The USB camera

The camera came with existing device drivers so after it was plugged into the beaglebone, it was ready to display a camera feed.

The camera used is a Logitech (V-UAR38) QuickCam. The specs are as follows:

- Connectivity: USB 2.0
- 1.3 megapixel sensor
- Length of Wire: 80cm

2.3.4 The four DC motors

The motors used are DCRobot’s 6V 180 RPM DC motors. The specs are as follows:

- Input voltage range: 3-7.5V DC
- Stall current: 2.8A
- Performance characteristics (6V, no load): 180 RPM, producing 11.11 oz-in of torque

2.4 Project Software

2.4.1 Arduino C code

The Nano’s microcontroller has an SPI slave code written in C flashed onto it. This code contains data for slave and USART initialization. It listens for new commands by polling for information.

2.4.2 Beaglebone black python code

The Beaglebone has a python script that constantly listens for user input and sends the appropriate command through SPI.

3. Hardware Design

Our “Rally Cam” uses an Arduino Atmega328p and a Beaglebone Black as microcontrollers. We have four DC motors and a USB camera mounted on the rig.

3.1 System Design

The project has two microcontrollers that communicate through SPI lines. The Arduino is connected to an H bridge that controls two sets of two motors. Each motor is running in tandem. In other words, the same PWM signal affects two motors at a time. The achieved effect is that two wheels on the same side will spin in sync.

In addition, there is a level shifter that steps down the voltage between the Arduino Nano and the Beaglebone Black. To power the camera, there is a USB hub that provides power to both, camera and keyboard.

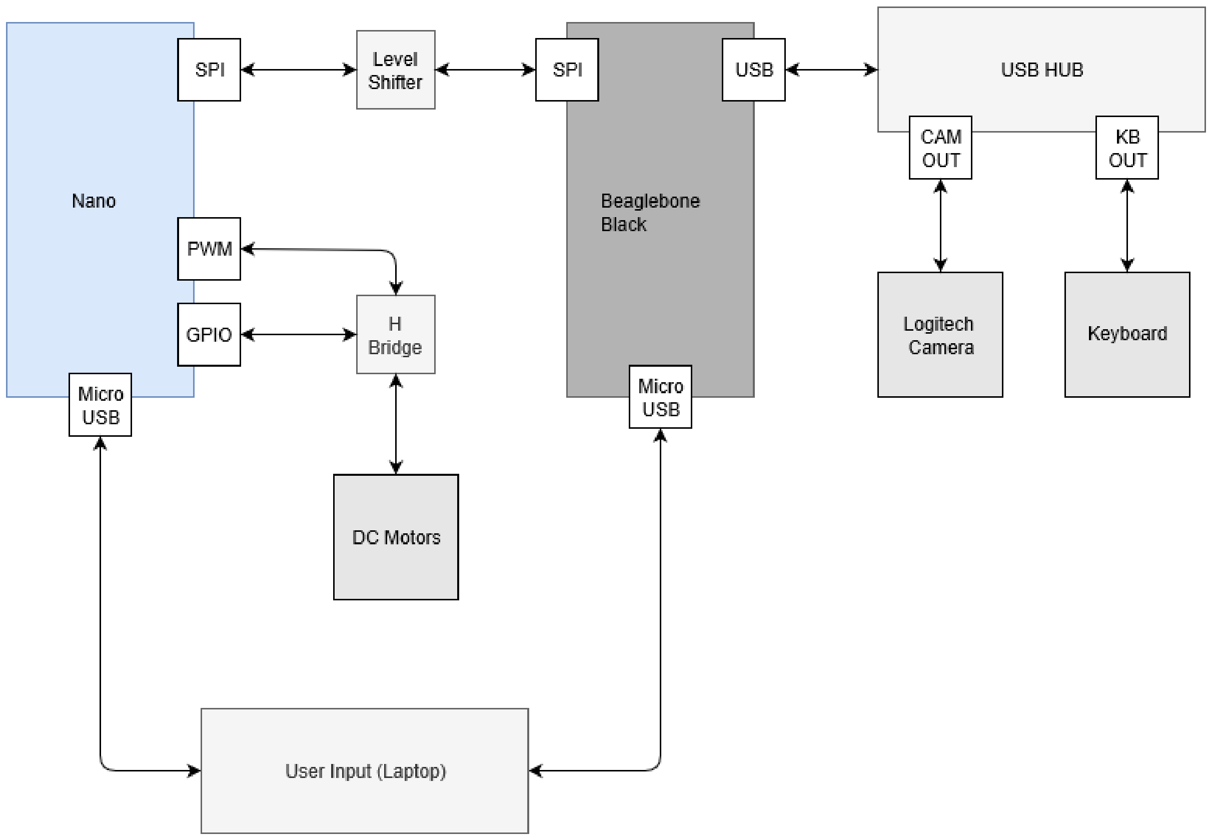


Figure 1 Overall Design

3.2 Subsystem Design

3.2.1 H bridge

The Arduino Nano is the microcontroller that interacts with the H bridge. On the H bridge, it has two power sources: Vcc1 and Vcc2. Vcc1 is powered through the Nano with 5V and Vcc2 is powered by an external battery pack of at least 7V.

3.2.2 PWM

There are two pulse width modulation, PWM, signals that come from pins PD5 and PD6. These are connected to 1,2EN and 3,4EN respectively. The PWM signal correlates to the 8-bit timer0 on the Nano. Since we are using phase correct mode, the timer counts from 0 to 255 and back down to 0. So, the motor will be set when the output compare register, OCR, value is equal to the timer count.

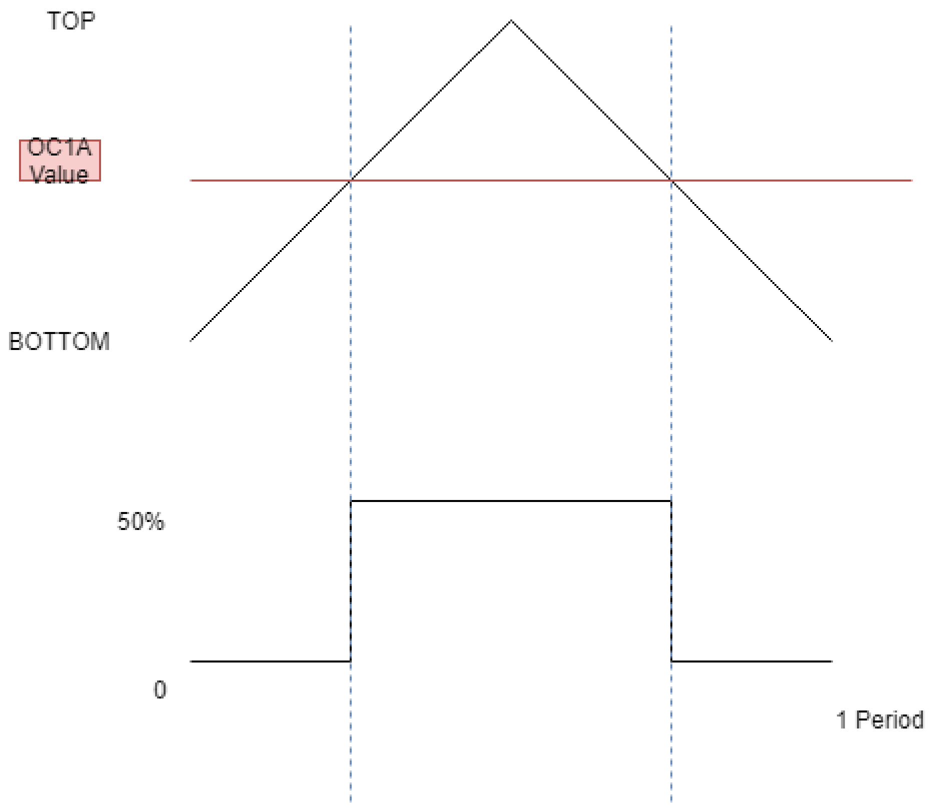


Figure 2 PWM

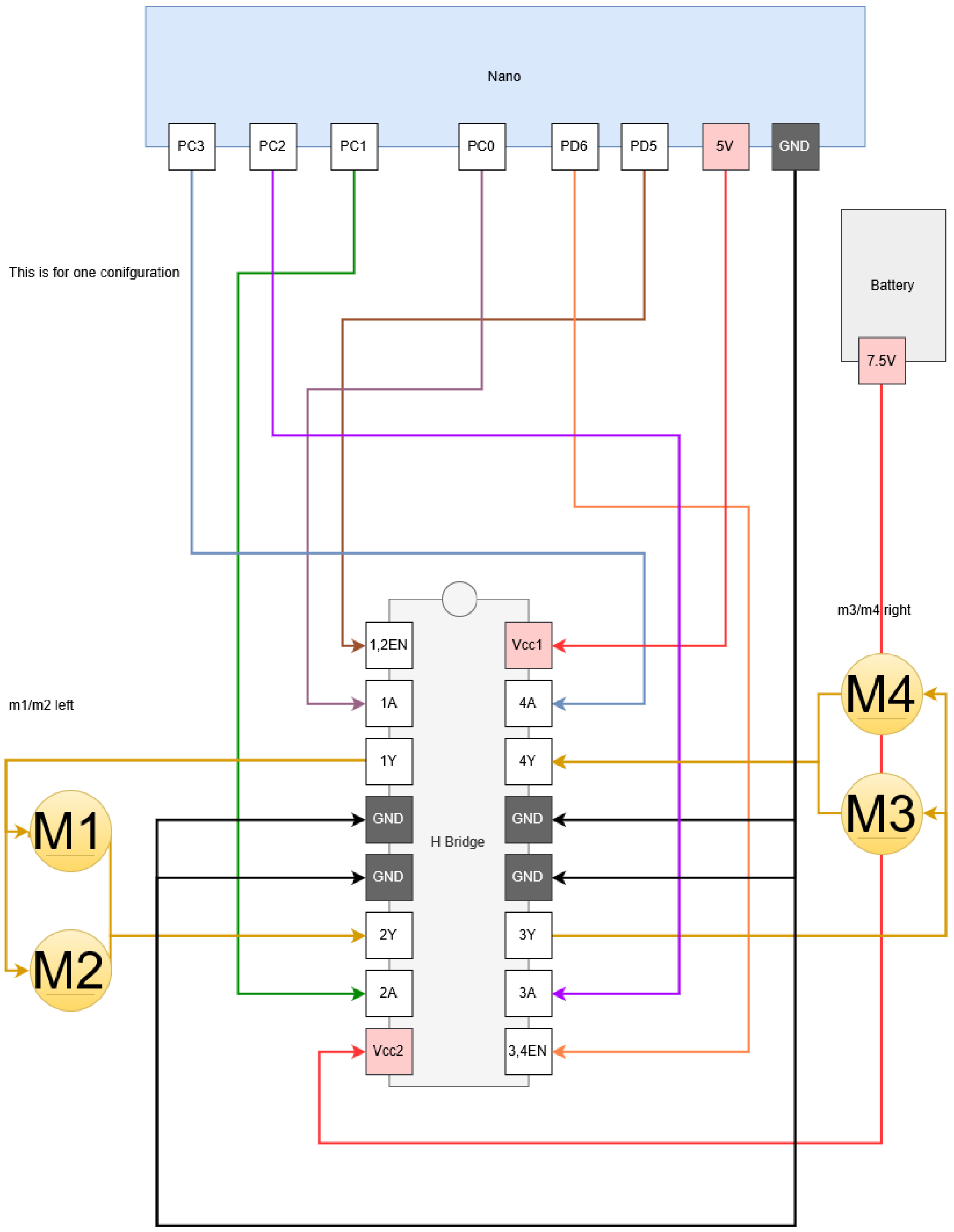


Figure 3 DC Motor Setup

The GPIO pins PC1 through PC4 provide inputs for the H bridge, 1A through 4A. The inputs and the enable are “logically ANDed” and the result will either be high or low. If it is high, the motor runs and vice versa.

Input A	Input EN	Output
H	H	H
L	H	L
Any	L	L

Table 1. H bridge table

3.3 System intercommunication

3.2.3 Level Shifter

The level shifter shifts down the voltage by using resistors and it shifts up the voltage by using the MOSFET transistor.

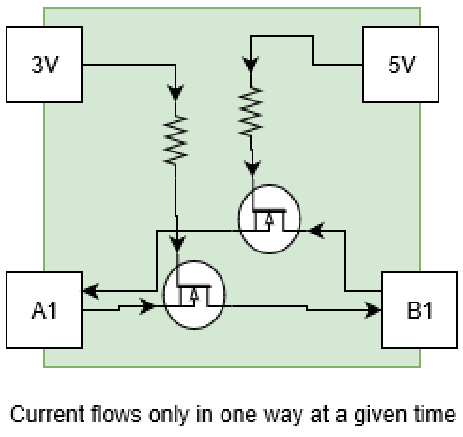


Figure 4 Level shifter concept

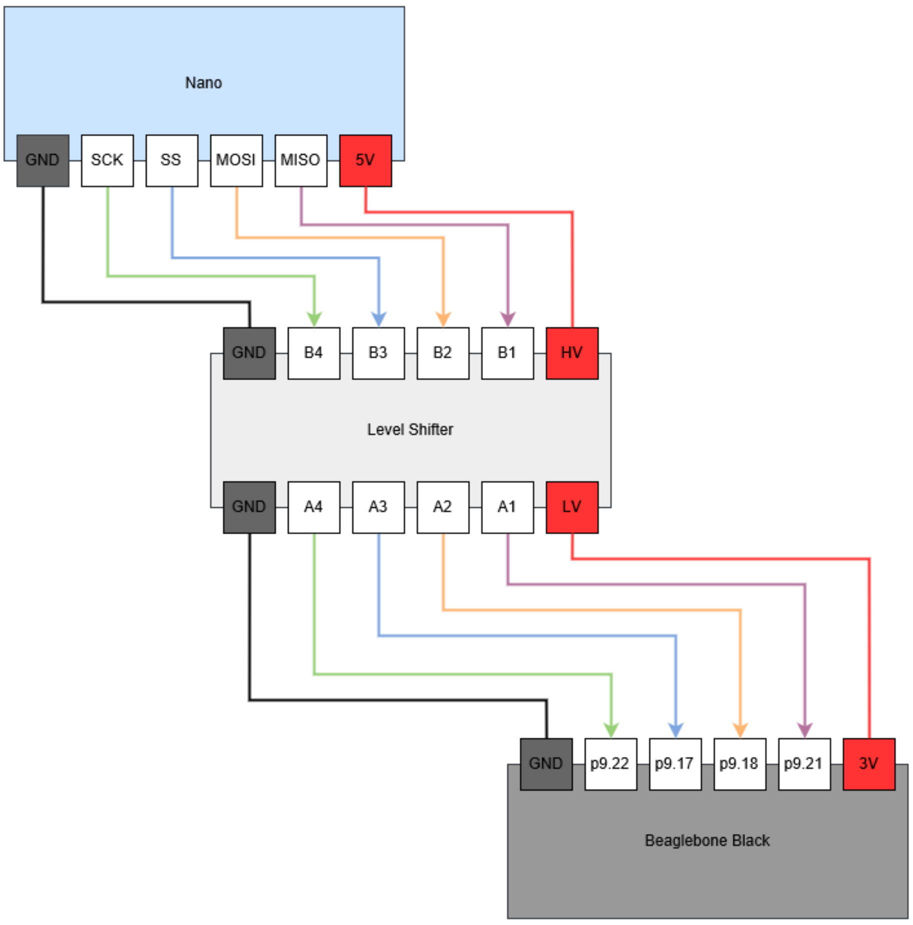


Figure 5 SPI Wiring

3.2.4 SPI Communication

On the master side, the following pins are outputs: SCLK, MOSI, CS. On the slave side, only MISO is an output. The effect is that when the CS/SS line is pulled from high to low, it tells the slave board to listen for incoming data. The SCLK pulses to give rhythm to the communication exchange. Both boards will exchange data that exists within the data buffer of each SPDR.

4. Software Design

For each microcontroller, we have different running code. The following section explains how both boards communicate with each other to accomplish the goal.

4.1 System Software Design

The system features two main software components working in tandem. The BeagleBone Black has an AM335x processor on an ARM architecture, and runs a full-fledged OS in the form of Debian 4.4. This allows it to be in control of the other peripherals as well as the Nano, sending them commands as well as receiving data. The Nano has an ATMega328p microcontroller running an 8-bit AVR architecture. Being much less powerful, the latter is only responsible for transmitting user commands coming from the BBB to the H-bridge, which will drive the motors.

4.1.1 Arduino System Software

The Nano software is written in C, and serves two main purposes. Firstly, it communicates with the BBB through a serial protocol. Secondly, it drives the motors by sending the appropriate PWM signals to the H-bridge.

4.1.2 Beaglebone Black System Software

The BeagleBone black runs Debian, a Linux-based OS, allowing it to compile and run the Python code required to operate the rest of the system. It also features a rather unique aspect of software management that occurs mainly in the world of embedded systems: capes, or more specifically, device tree overlays. These are what allow the kernel, and the rest of the software, to properly interface with the hardware and peripherals present on the board. In this case, the SPIDEV module, which uses the built-in SPI facilities of the BBB, was enabled by first disabling the universal cape, and then activating its respective cape.

4.2 Software Subsystem Design

SPIDEV is a Python module that enables SPI to be properly initialized, and for commands to be subsequently sent and received on the BBB. It is built in to the BBB, where it can be enabled by activating its cape from within Debian’s uEnv file. Its structure is simple enough, being able to be run from a user-made Python script. Then, with the proper wire connections set up, the BBB will be able to interface over SPI with the Nano, allowing it to communicate byte-by-byte.

4.3 Arduino Functions

4.3.1 Motor Commands

The commands for the motor were written using bare-metal methods, using the provided motor circuit as base. All functions for motor controls are defined as follows:

- motor_init(): used to initialize all the pins involved in sending signals to the H-bridge, as well as configuring prescaler, timer, and pwm mode.
- stop_moving(): Clears all motor control pins so they can be set by all move functions. All move function call this function before setting their pins.
- Move_forward(): Set all the pins to rotate all the wheels to propel the car forward.
- Move_back(): Set all the pins to rotate all the wheels to propel the car in reverse
- Move_right():Rotate the wheels on the right and left in opposite directions, in such a way to direct the car towards the right.
- Move_left(): Rotate the wheels on the right and left in opposite directions, in such a way to direct the car towards the left.

4.3.2 SPI USART

The SPI functions were coded in bare-metal, as covered and done so in the earlier portions of the course. As well, USART was used to interface with the laptop itself. All functions used are explained in the following list:

- SPI_SLAVE_init(): Used to initialize SPI on the Nano side by configuring MISO, MOSI, SCLK and SS on the correct pins
- SPI_transceiver(uint8_t data): Pre-loads data into the SPI data register, to be sent to the master upon start of an SPI transmission
- SPI_receiver(): Used to receive data from the master
- USART_init(): Used to initialize USART for communication between the Nano and the laptop
- USART_transmit(unsigned char data): Print out prompts and information to the serial monitor, to be seen by the user
- USART_receive(): Capture input from the user through the serial monitor
- USART_println(): Print multiple prompts out in a row, by looping through uses of USART_transmit
- USART_printnum(): Similar to USART_transmit, but takes only the first character and discards the rest
- USART_flush(): Flush and clear the USART data register

4.4 System Software Communication

Communication happens between several of the system's components. These communication channels are detailed as follows:

- Beaglebone-Laptop: The connection between the Beaglebone itself and the laptop is over the USB protocol, which conveniently provides both power and data, through a USB A-to-micro USB cable. This allows the user to access the Beaglebone through SSH. From there, the user can configure the Beaglebone capes, write and run custom scripts, e.g. Python scripts needed to operate the SPIDEV module.

- Beaglebone-Nano: For the purposes of setting up an SPI communication channel, the required 4 wires connecting the Beaglebone to the Nano pass through a level shifter to accommodate the difference in operating voltage, i.e. 5V for the Nano, 3.3V for the Beaglebone.

- Nano-Laptop: While the Nano doesn't require a continuous link to the laptop, an initial connection is required to compile and upload the SPI slave code to it. This is also done through the USB protocol, through a USB A-to-micro USB cable just like in the case of the Beaglebone-laptop connection.

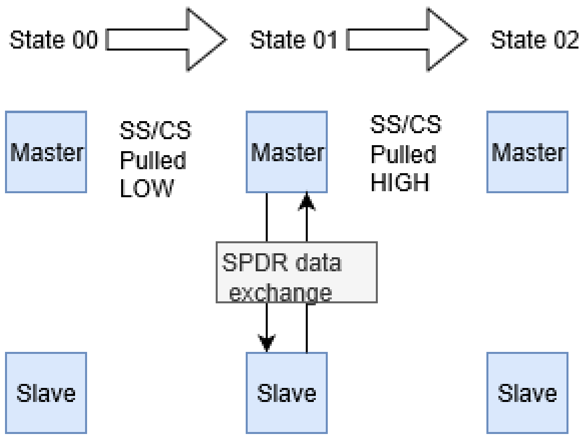


Figure 6 SPI Communication

5. Development Software

Both of the microcontrollers in our setup are developed in their own environments. This section details the programming languages and specific software tools used by the different boards.

5.1 Arduino Nano Unit Software

Given that the Nano was used exclusively as a slave to control the motors, no additional libraries needed to be installed. Moreover, the innate SPI and PWM libraries of the Nano weren't used either, as all the code was written with bare-metal methods, i.e. the appropriate registers and pins were all individually and manually set. This was done through the Arduino IDE, which also simplified the process of compiling and uploading the code to the Nano.

5.2 Beagleboard Software

5.2.1 Linux Debian

Debian 4.4 is the default OS present on the Beaglebone. Being Linux-based, all configuration required was done by going through its file structure. Namely, enabling the SPI cape involved going through Debian's uEnv.txt located in the /boot/ folder and enabling it on boot. The Python script used to implement the imported SPIDEV library was also done directly on the Beaglebone. These changes were made through SSH with a text editor, in our case Nano.

5.2.2 Python

The SPIDEV module that was used to achieve SPI functionality on the Beaglebone side is written in Python, and was therefore installed through the use of the built-in Python package manager, *pip*. Debian itself comes pre-installed with Python 2, which is what was used for this process.

6. System Performance

The completion of this project was no reached without adversity. Through all the different stages of development, we discuss these difficulties and how we reached a resolution.

6.1 Component Testing

Arduino nano

With the arduino nano we had to set up the SPI communication protocol as a slave to read the commands sent from the BeagleBone. For testing the SPI first we used another Arduino Nano as a master. Once we defined the SPI in the Nano we tested its functionality with the BeagleBone by sending the commands and verifying the input received through the serial monitor. In parallel, we set up the PWM control for the motors through the H-Bridge. We first tested the functionality of the motor control without the SPI with the use of a power supply of 7.5v.

BeagleBone Black

Our tests for the BeagleBone were primarily on the proper functioning of the SPI communication with the Arduino Nano. Setting the SPI in the BeagleBone was the most challenging part of the process, so through multiple trials and errors in sending data to the Nano we were able to successfully implement it.

6.2 System Testing

Once we tested each component in separate, we connected them all and tested the system as a whole. Throughout the process we encountered many problems where some components did not work properly, to which we needed to adjust either the code or the hardware pieces.

7. System Delivery

Embedded systems are no simple matter and we have not abstracted most of the set up for the user yet, so here is our step by step guide for our project.

7.1 System Initialization

7.1.1 Arduino

The slave code needs to be present on the Nano. If it is not on the board, proj_slave_spi.c needs to be flashed. It is ready to be used with the BBB. Follow the steps to connect the board to the level shifter:

1. Connect CLK to the level shifter
2. Connect MOSI to the level shifter
3. Connect MISO to the level shifter
4. Connect SS to the level shifter
5. Connect GND to the level shifter
6. Connect 5V to the level shifter (HV)

7.1.2 Beaglebone Black

First, we download SPIDEV from https://pypi.org/project/spidev/. The board needs to disable the universal cape and enable the SPIDEV cape. We would just need to run the python script for SPI communication. Follow the steps to connect the BBB to the level shifter:

1. Connect SCLK to the level shifter
2. Connect D0 to the level shifter
3. Connect D1 to the level shifter
4. Connect CS to the level shifter
5. Connect GND to the level shifter
6. Connect 3V to the level shifter (LV)

7.2 System Operation

7.2.1 Running the Arduino SPI Slave

After flashing the slave code, we just need to connect the arduino to the beagleboard.

7.2.2 Running the BBB SPI Master

After the setup, we need to move to the working directory and run the script.
The following code would be used to run the script:

```
cd /testing          #move to the working directory
python test.py       #to run the python script called test.py
```

Just follow the on-screen instructions to send commands to the Nano

8. Project Process Discussion

This project came with rewards as well as a plethora of challenges. This section outlines every difficulty we came across.

8.1 Phases of development

8.1.1 Planning

It was difficult to come up with the initial idea. The group had little to no experience with embedded systems so we were not aware of what was possible. We decided to come up with a basic idea and build upon it. We settled on the remote controlled car because it came naturally in relation to the material that was being taught at that moment.

8.1.2 Developing the code

8.1.2.a SPI

We ran into problems with the SPI code for the BBB. We tried to use the Adafruit library but we were unsuccessful, we changed tactics and used the SPIDEV libraries instead.

8.1.2.b USART

We also had issues with reading the data received by the slave since the USART code we developed could not determine what character was sent. It took some time figure out the proper USART code by sending messages between two arduino Nanos.

8.1.3 Building the rig

We had some issues with wire management and organizing space on the rig. We ran into space issues when we tried to mount all the components on the same platform. We had to add more platforms to get extra space.

8.1.4 Testing and fine-tuning

We were successful in getting the car to move forward, backwards and to turn at right angles. We noticed that one side of the car had stronger DC motors and we surmised this was caused by different levels of voltage.

8.1.5 USB Camera

We tried to follow the Derek Molloy site to run video streaming through UDP, but we were not familiar with data protocols and dropped the idea.

8.2 The Learning Experience

Learning new microcontrollers as a group was an awkward experience. Everyone had different levels of expertise and comfort with both boards. The biggest challenge was learning how each component works.

The whole group was familiar with programming on the Nano but has not worked exclusively with motors yet. One half of the group dedicated themselves to learn the intricacies of BBB and the other half developed on the Nano.

Learning the BBB was difficult since we were not sure if the learning material provided in class would work. We know how to start and what the end product should be. However, when we were stuck on the intermediate steps, we were not certain if we should try another approach or try again.

We ran into issues with getting the car to drive in a straight line and we believe this is due to the difference in voltage level between the left and right side of the H bridge. One side is 5V from the Arduino Nano and the other is from the the 7.9V battery pack.

It was fairly late into the development of the project when we tried to implement video streaming through UDP. We wanted to make sure mobility was completed before working on the camera since we thought it would be wise to have one feature available at least. We could not get Derek Molloy's video streaming library to work and we decided to leave that for future endeavors.

Working together as a group was fairly difficult. As the difficulty of the project rose, the ones who had trouble learning the basics fell behind as they tried to catch up. There were those who kept on track with the material and were able to contribute more to the project. It was not too difficult to divide the work so that everyone worked on something they were fairly comfortable with.

We can see why there was such a strong emphasis on having hands-on experience for this class. Most of the time, we were looking up data sheets and external sources to use in the project. There does not seem to be enough time during the regular semester to fully cover every topic required for the project. It does however show us the true depth of embedded system programming. For some members, it does encourage them to continue on and start their own projects with microcontrollers.

Appendix 1

```
#define F_CPU 16000000UL
#define BAUD 9600
#define MYUBBR F_CPU/16/BAUD-1

int main(void)
{
  USART_init(MYUBBR);
  SPI_SLAVE_init();
  motor_init();
  USART_println( "Initiated." );

  unsigned int greet = 49;
  unsigned long result;
  unsigned int max_speed = 64;

  USART_println( greet );
while(1)
  {
    USART_println( "Loop started. " );

    for( int i = 0; i < 4; i++)
    {
      result = SPI_tranceiver(greet);
      _delay_us(20);
      USART_printnum( result );
      if(result == '1')
        move_forward(max_speed);
      if(result == '2')
        move_left(max_speed);
      if(result == '3')
        move_right(max_speed);
      if(result == '4')
        move_back(max_speed);
      if(result == '0')
        stop_moving();
    }

    USART_println( "Loop ended." );

  }
  return 0;
}


void USART_init(unsigned int ubrr)
{
  //shift out first 8 bits to clear it out ?
  UBRR0H = (unsigned char)(ubrr >> 8);
  UBRR0L = (unsigned char)(ubrr);
  #if USE_2X
    UCSR0A |= (_BV(U2X0);
  #else
    UCSR0A &= ~(_BV(U2X0));
  #endif

  UCSR0B = ((1 << RXEN0) | (1 << TXEN0)); //enable rx and tx
  UCSR0C = ((1 << USBS0) | (3 << UCSZ00));  //set frame to be 8 bits and
two stop bits

}
void USART_transmit(unsigned char data)
{
  while(!(UCSR0A & ( 1<<UDRE0)));
  UDR0 = data;
}
uint8_t USART_receive(void)
{
  unsigned char stat, resh, resl;

  while(!(UCSR0A&(1<<RXC0)));
  stat = UCSR0A;
  resh = UCSR0B;
  resl = UDR0;
  if(stat & (1<FE0)|(1<<DOR0)|(1<<UPE0))
  return -1;
  resh = (resh >> 1) & 0x01;
  return ((resh << 8 ) | resl);
}

void USART_println(const char *s)
{
    while(*s)
    {
      USART_transmit(*s);
```

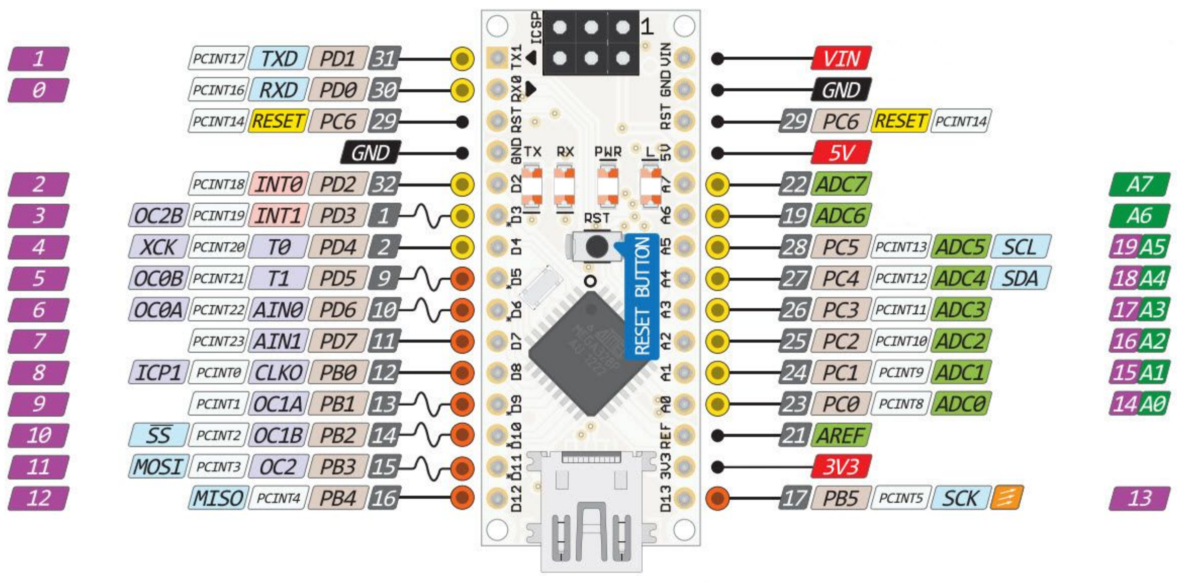
```
       s++;
     }
     USART_transmit('\n');
}

void USART_printnum(const char *s)
{
  USART_transmit(*s);

   USART_transmit('\n');
}

void USART_flush(void)
{
  unsigned char dummy;
  while( UCSR0A & ( 1<<RXC0))
  dummy = UDR0;
}
void SPI_SLAVE_init(void)
{
  DDRB = (1<<4);
  DDRB &= ~((1<<3)|(1<<5)|(1<<2));
  SPCR = (1<<SPE);
}


/* spi usart */

uint8_t SPI_tranceiver(uint8_t data)
{
  SPDR = data;
  while(!(SPSR&(1<<SPIF)));
  return SPDR;
}

uint8_t SPI_receiver()
{
  //SPDR = data;
  while(!(SPSR&(1<<SPIF)));
  return SPDR;
}
void LED_init()
{
  DDRD |= (1<<4)|(1<<5)|(1<<6);
}
```

```
void LED_on()
{
    PORTD |= (1<<4)|(1<<5)|(1<<6);
}

void motor_init(void){
  TCCR0A |= (1 << COM0A1)|(1 << COM0B1)|(1 << WGM00);//wgm for pwm phase
correct
  TCCR0B |= (1 << WGM02)|(0 << CS02)|(1 << CS01)|(1 << CS00);//CS0n
control prescaler, current 64
  TCNT0 |= 0;

  //H-bridge 1-8 set IO
  DDRD |= (1 << 5);//pwm pin, 12en
  DDRC |= (1 << 0)|(1 << 1);//H bridge inputs, A0=1A,A1=2A

  //H-bridge 9-16 set IO
  DDRD |= (1 << 6);//pwm pin, 34en
  DDRC |= (1 << 2)|(1 << 3);//H bridge inputs, A2=3A,A3=4A
}


//motor controls
void move_forward(unsigned int spd){
  stop_moving();
  PORTC |= (1 << 0)|(0 << 1);//left wheel
  PORTC |= (0 << 2)|(1 << 3);//right wheel
  OCR0A = spd;
  OCR0B = spd;
}

void move_back(unsigned int spd){
  stop_moving();
  PORTC |= (0 << 0)|(1 << 1);//left wheel
  PORTC |= (1 << 2)|(0 << 3);//right wheel
  OCR0A = spd;
  OCR0B = spd;
}

void move_right(unsigned int spd){
  stop_moving();
  PORTC |= (1 << 0)|(0 << 1);//left wheel
  PORTC |= (1 << 2)|(0 << 3);//right wheel
  OCR0A = spd;
  OCR0B = spd;
}

void move_left(unsigned int spd){
  stop_moving();
  PORTC |= (0 << 0)|(1 << 1);//left wheel
  PORTC |= (0 << 2)|(1 << 3);//right wheel
  OCR0A = spd;
  OCR0B = spd;
}

void stop_moving(void){///reset all pins for next move function
  PORTC &= ~((1 << 0)|(1 << 1)|(1 << 2)|(1 << 3));//clear pins
  OCR0A = 0;
  OCR0B = 0;
}
```

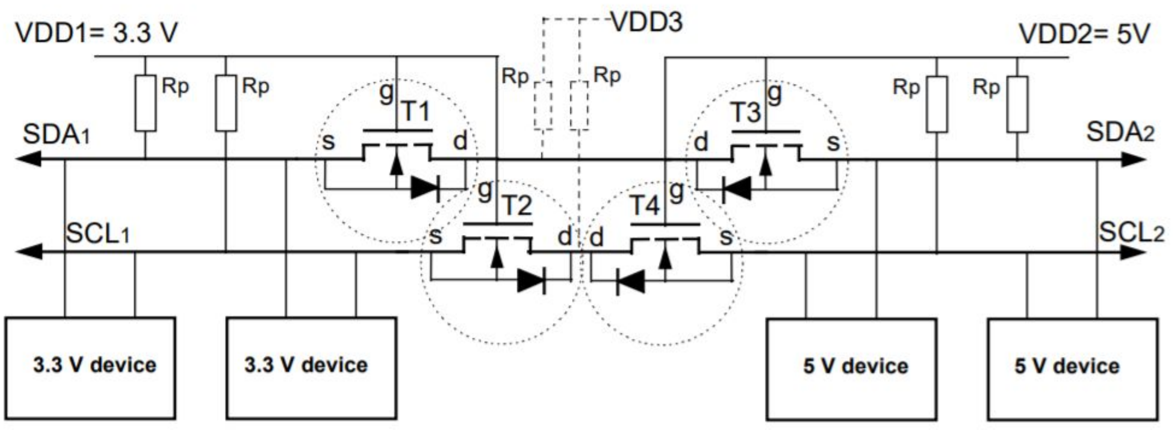
Python Script, *test.py*

```
import spidev
spi = spidev.spiDev()
spi.open(1,0)
spi.max_speed_hz = 125000
running = True
while(running):
      text = raw_input("Enter a command")
      if text == 'w':
            print(spi.xfer( [0x31] ))
      elif text == 'a':
            print(spi.xfer( [0x32] ))
      elif text =='d':
            print(spi.xfer( [0x33] ))
      elif text =='s':
            print(spi.xfer( [0x34] ))
      elif text =='q':
            print(spi.xfer( [0x30] ))
            running = False
      else
            print(spi.xfer([0x30] ))
            running = False
spi.close()
```

Atmega328p Pinout Diagram

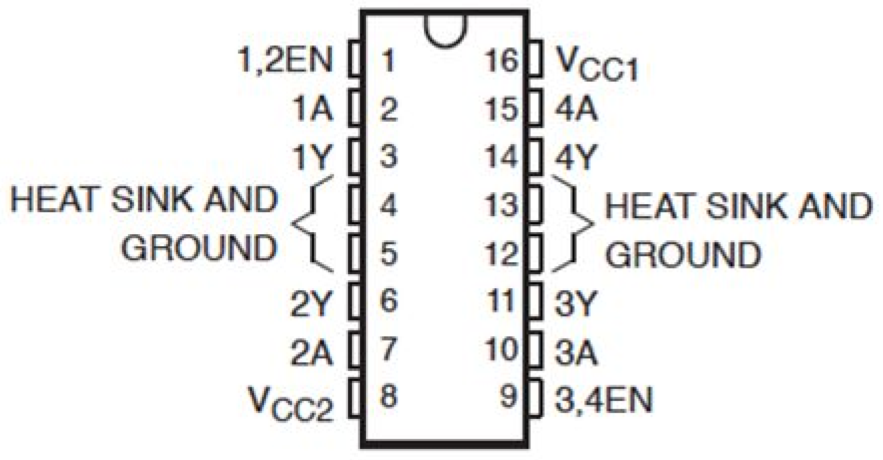

1 PCINT17 TXD PD1 31 TX1
0 PCINT16 RXD PD0 30 RX0
PCINT14 RESET PC6 29 RST
GND
2 PCINT18 INT0 PD2 32 D2
3 OC2B PCINT19 INT1 PD3 1 D3
4 XCK PCINT20 T0 PD4 2 D4
5 OC0B PCINT21 T1 PD5 9 D5
6 OC0A PCINT22 AIN0 PD6 10 D6
7 PCINT23 AIN1 PD7 11 D7
8 ICP1 PCINT0 CLKO PB0 12 D8
9 PCINT1 OC1A PB1 13 D9
10 SS PCINT2 OC1B PB2 14 D10
11 MOSI PCINT3 OC2 PB3 15 D11
12 MISO PCINT4 PB4 16 D12
ICSP
VIN
GND
29 PC6 RESET PCINT14
5V
22 ADC7 A7
19 ADC6 A6
28 PC5 PCINT13 ADC5 SCL 19 A5
27 PC4 PCINT12 ADC4 SDA 18 A4
26 PC3 PCINT11 ADC3 17 A3
25 PC2 PCINT10 ADC2 16 A2
24 PC1 PCINT9 ADC1 15 A1
23 PC0 PCINT8 ADC0 14 A0
21 AREF
3V3
17 PB5 PCINT5 SCK 13 D13
TX RX PWR L
RST
RESET BUTTON

Level Shifter Circuit Diagram



Source: Philips Conductors

H-Bridge Pinout


1,2EN 1
1A 2
1Y 3
HEAT SINK AND GROUND 4
5
2Y 6
2A 7
V_{CC2} 8
16 V_{CC1}
15 4A
14 4Y
13 HEAT SINK AND GROUND
12
11 3Y
10 3A
9 3,4EN