07

0129320285

Mechanical Pong

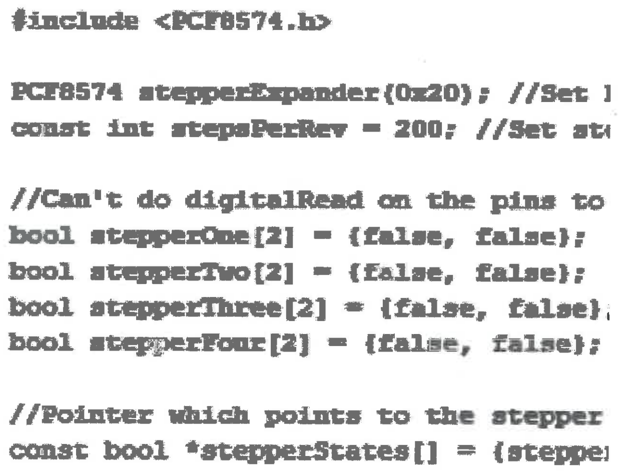
I have developed a Mechanical Pong game based off the computer Pong game from the 70's for my electronics project. For the creation of this product many complex concepts from software and hardware were used to guide the outcome that I have made. Using my Mechanical Pong game, I will be discussing these software and hardware concepts. The outcome I have created used Linear potentiometers for the players paddle input, limit switches to provide more feedback to the program that runs the game to say where the stepper motors are, stepper motors for the XY movement of the Pong ball and movement of the paddles, and seven segment displays to show the players what the score is. The complex concepts that were used in this system were for the hardware side Suitability of the Microcontroller, H-Bridge and Stepper Motor Control, and Multiple Actuators. For the software side Structuring Complex Programs Logically, Flags, and Interrupts.

Software Concepts

Structuring Complex Programs Logically

For creating complex programs such as the one I have created to drive the Mechanical Pong game it is very important to have them structured logically. This is because without logical structuring maintaining and troubleshooting code becomes an extremely daunting task. This is why I have used declaring variables at beginning of code, declaring Arduino pins as variables, using functions, and giving variables descriptive names to help with the creation of my outcome.

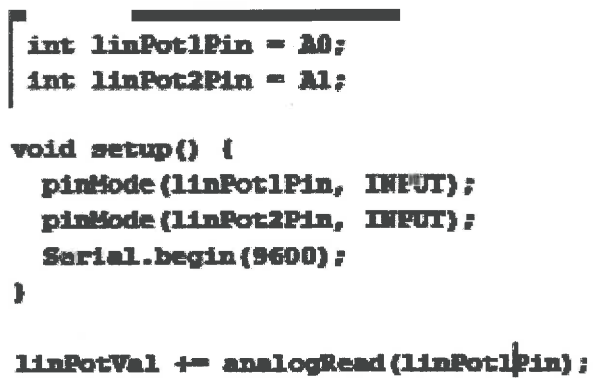
Declaring Variables at Beginning of Code



```
#include <PCF8574.h>

PCF8574 stepperExpander(0x20); //Set
const int stepsPerRev = 200; //Set st

//Can't do digitalRead on the pins to
bool stepperOne[2] = {false, false};
bool stepperTwo[2] = {false, false};
bool stepperThree[2] = {false, false};
bool stepperFour[2] = {false, false};

//Pointer which points to the stepper
const bool *stepperStates[] = {steppe
```

Declaring variables at beginning of code is very straight forward, when creating the variables that will be used throughout the program, they will be placed at the top of the code alongside all of the other variables. For mine I have grouped them at the top of the code as well as organising them in groups of what their function relates to, for the case of the stepper motors states, they are grouped together. This has been used because by having all of the variables at the beginning of the code it helps with troubleshooting as instead of looking through the entire code to find variables and change their values they can easily be located, along side the grouping of variables, finding the specific variable you were looking for would be much easier therefore cutting down on time spent programming, and cutting down on maintenance of the code. When declaring the variable at the beginning of the program they become global variables. Global variables are variables that can be accessed and modified at every point in the code and the variable will be updated everywhere. These global variables compared to local, which can only be referenced inside the function it was created in, tend to take more memory from the microcontroller as it is having to put aside memory for the variable even when it is empty. However, the project I have made uses mostly global variables as the upsides of having the variables accessed and change anywhere in the code outweighs the downside of less memory.

Declaring Arduino Pins as Variables

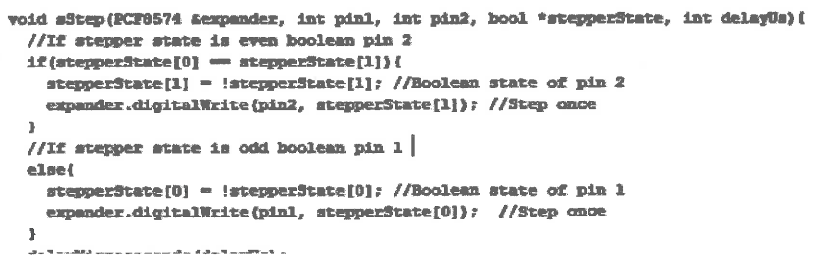


```
int linPot1Pin = A0;
int linPot2Pin = A1;

void setup() {
  pinMode(linPot1Pin, INPUT);
  pinMode(linPot2Pin, INPUT);
  Serial.begin(9600);
}

linPotVal += analogRead(linPot1Pin);
```

The declaration of variables that store the Arduino pin numbers are also a vital part in structuring complex programs logically. This is because pins are used very often in the program and because of this having ambiguous numbers such as analogRead(A0) makes programming very hard to keep track of which numbers/letters correspond to which component. To solve this creation of variables at the start of the program which store the pin letters/numbers for example int linPot = A0 is much easier to recognise what this is referencing. Another issue that this solves is troubleshooting, when not using declared pin variables if pins in the program needed changing every instance of that pin being mentioned must be removed, compared to having a variable which only one instance in that case must be changed. The most common reason for changing pins is the component that was connected to said pin has either been moved to another pin or replaced with another component. The risks when missing a single instance when changing pins in the program could lead to components not working as intended or in the worst case, damaging the components outright

Using functions

Because of such a large program for this, it is inevitable to have repeating code, however, this is always bad. This is because of for one the size constraints of the program for it to fit onto the Arduino's storage. Repeating code is also very hard to maintain as repeating code tends to carry the same function, but for a different component such as multiple stepper motors. Finally, for repeating code, it is also very hard to make changes to the way it functions as every time a change needs to be made it needs to be made multiple times throughout the code to cover whenever it was used this causes an increase in the likelihood that an error will occur. Functions allow the repeated code to be only written once in the program and to run that piece of code the program just needs to run the function. One of the main uses of functions in my program was with the stepper motors. Every motor has it's own stepper states (the states in which coils are active) and it's own pins which are used to control the stepper motor, without functions every time a



```
void sStep(PCF8574 &expander, int pin1, int pin2, bool *stepperState, int delayUs){
  //If stepper state is even boolean pin 2
  if(stepperState[0] == stepperState[1]){
    stepperState[1] = !stepperState[1]; //Boolean state of pin 2
    expander.digitalWrite(pin2, stepperState[1]); //Step once
  }
  //If stepper state is odd boolean pin 1
  else{
    stepperState[0] = !stepperState[0]; //Boolean state of pin 1
    expander.digitalWrite(pin1, stepperState[0]);  //Step once
  }
```

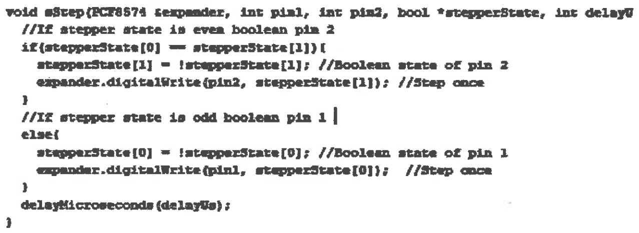
stepper motor were to step a large chunk of code was devoted to that one step, and for every step it needed to be repeat, this was replaced by one function which when called takes in the parameters of which pins were used, and which states the motor is in. The names of functions are also an important part of them as having ambiguous names can lead to confusion as to what the specific task the function carries out in the program.

Giving Variables Descriptive Names

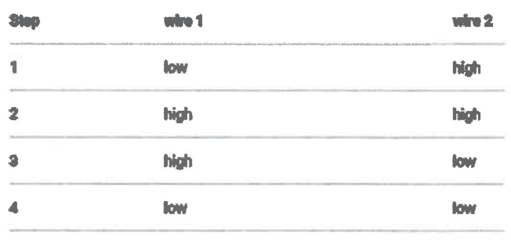
This is the last part of structuring complex programs logically which heavily guided the way the program was written. With the declaration of pins, variables and function names in general it is very important that the names of these reflect what their function is. This is because without proper naming the troubleshooting and maintenance of the code would be incredibly difficult as with random or ambiguous names for variables without commenting, or even with commenting, it is incredibly difficult for telling which part of the program does what task even for the creator of the program, and especially others who would want to work on it. Therefore, using descriptive names such as stepperStates, or stepsPerRev help with the flow of the program and help the program be structured logically. Camel case is also an important part of descriptive naming as stepsperrev is much harder to read than stepsPerRev. The order in which camel case is used can also be used to differentiate between functions and variables with functions normally starting with a capital and variables starting in lowercase.

Flags

Flags are variables which usually have two or more states, but in most cases are either true or false. These flags are used for the program to determine which action should be executed next, be it the same function it just carried out, or a new function. The most common use for a flag is a while loop, for example they could be used to keep running a while loop while the stepper has not completed a certain number of steps and once it has, go on to the next task in the program. The main example of flags in my program are for storing the states of the stepper motors. These flags are in an array of two where each item in the array is one of the states of the coils in the stepper motor, be it active or inactive. For the stepper motor to successfully step it must go to the next combination of coil on/off for it to turn in the correct direction. This is where the flags come in, since they store the states of the stepper motors coils it compares it to a truth table of which step should come after the next (see image for truth table[1]) and from that changes which coil is active/inactive on the stepper motor. Because the states of the variables affect how the program behaves these variables are flags for the stepper motor. The



```
void sStep(PCF8574 &expander, int pin1, int pin2, bool *stepperState, int delayU
  //If stepper state is even boolean pin 2
  if(stepperState[0] == stepperState[1]){
    stepperState[1] = !stepperState[1]; //Boolean state of pin 2
    expander.digitalWrite(pin2, stepperState[1]); //Step once
  }
  //If stepper state is odd boolean pin 1 |
  else{
    stepperState[0] = !stepperState[0]; //Boolean state of pin 1
    expander.digitalWrite(pin1, stepperState[0]);  //Step once
  }
  delayMicroseconds(delayUs);
}
```



Step	wire 1	wire 2
1	low	high
2	high	high
3	high	low
4	low	low

[1] https://itp.nyu.edu/physcomp/lessons/dc-motors/stepper-motors/

reason for using these flags is for one, using an I2C port expander and the library used to control it, there is no way to digitalRead a pin which is set to be an output, and because of this there must be variables to keep track of what step the stepper motor is on. The other reason for using flags is, as mention a few times above, the need for the program to keep track of what step the stepper motor is on. Without the ability to know what the current step is, attempting to do the next step is near impossible as if the stepper motor goes to the wrong step it can either go backwards or go one step too far. In both of these cases the stepper motor will "miss" a step resulting in jittering of the stepper motor as it moves, or if it misses steps enough, it will stay in one spot twitching. This is why I have used flags as they give the ability of keeping track of which coils are active and from that in the program it can find the appropriate next step for the stepper motor.

Interrupts

Interrupts are a signal to the microprocessor that tells it to address this new information immediately. In doing this, it pauses the task it was currently doing to handle the new task. This can be very useful for time sensitive operations such as gathering user inputs, or in the case of my project, being used for informing the program immediately that the stepper motor has reached the end of the axis it is turning on. Interrupts are used in my project with the limit switches so that when the limit switches detect a change the Arduino will instantaneously pick the change up and deal with it, this will be used for one calibrating the program so it knows where its bounds for the stepper motors are, and for detecting if the stepper motor has left it's bound while people are playing the game. These interrupts are used to tell that the stepper motors need to stop but it's only when used in conjunction with flags that it allows the system to respond to problems on its own. This second use for the interrupts is very important as if the stepper motors turn so that it's out of the bound of the game due to missed steps or incorrect calibration, it will be needed to be stopped as soon as possible so that there is no damage caused to the machine as forcing stepper motors against a wall will result in them breaking quickly. Limitations with the interrupts regarding my project is that they can only detect a digital change in state so either high to low or low to high, this means interrupts will not be able to be used with the linear potentiometer for the users input as that takes an analogue value which is then mapped to the stepper motor to where it should be in relation to what the player has set their paddle to.

Hardware Concepts

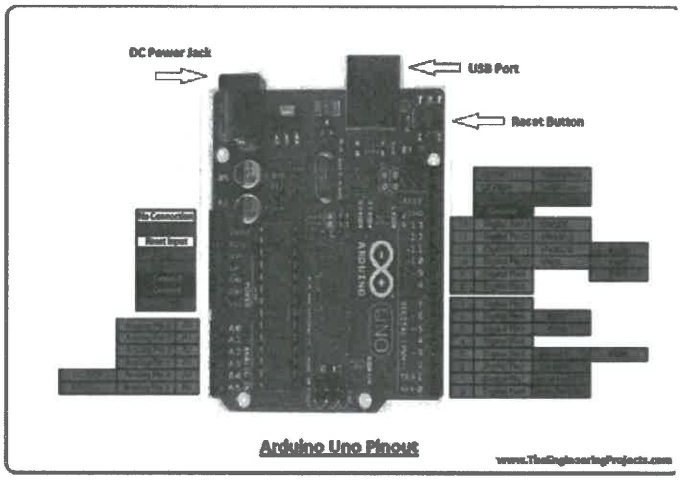
Suitability of Microcontroller

For this project I needed to assess the suitability of the microcontroller as a logic-based circuit would not be able to complete the task of running stepper motors, seven segment displays, and other ICs. There are a few microcontrollers what could have been used in this project such as the picaxe microcontroller, but in the end, I decided on the Arduino microcontroller.

Picaxe

The picaxe microcontroller is a low-cost single chip solution which is quite small, for this project this would be ideal, however there are a few problems with this microcontroller. One of the larger ones being I have to wire the microcontroller up myself which is very time consuming as to get everything connected right to even get it running. A second problem with the picaxe is the programming of it, the installation of the programming applications is problem enough coupled with the unfriendly interface and programming language made programming very painful, and with the size of the program required for this project, it would make using this microcontroller very impractical. Other issues with the picaxe are that there are not a lot of support forums for using this, combined with a lack of libraries for controlling ICs such as a shift register, or a I2C port expander leaves this microcontroller unsuitable for this project.

Arduino



[2]The Arduino microcontroller on the other hand while still low-cost, is more expensive than the picaxe alternative and is larger than the picaxe. This microcontroller does have the benefits of being pre-assembled making it a minimal setup solution. Compared to the picaxe the programming experience is much better and due to the popularity of this microcontroller there are many support forms and libraries which help with the programming of this microcontroller for the task of this Mechanical Pong game. The Arduino does have some limitations however, these limitations are due the amount of output/input pins that this microcontroller has. The specific Arduino that will be used for this project is an Arduino Uno which has 12 digital pins. The stepper motors each use two pins to drive and there will be four meaning a total of 8 of the digital pins are being used for driving stepper motors, the two displays for showing the players score has 4 seven segment displays, with using Binary to Digital converter ICs to run the displays this still requires four pins per IC which results in 16 pins to drive all of the displays. While two shift registers could be used this still requires 6 pins to drive all displays which brings the total amount of pins required so far to 14 which is more than the Arduino supports, and this is only for the stepper motors and the score displays. This could be solved by using another Arduino such as the Arduino Mega which allows for far more digital pins, the problem with this solution is that I do not have access to this microcontroller so instead an I2C expander is going to be used which gives 8 digital pins per IC which runs of the SDA and SCL lines (A4 and A5 pins), and as many as 8 of these ICs can be used running off only two pins of the Arduino. Because of the ability for the Arduino Uno to utilise

[2] https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-uno.html

libraries, the ease of programming, support forums, and the expansion of IO ports I have chosen this microcontroller as the microcontroller of choice for this project.

Multiple Actuators

For this project the selection of the right actuators for movement of the XY position of the ball and the players paddles is one of the most important decisions as the wrong actuator would cause more difficulty in the creation of the project. This is why I have considered stepper motors, DC motors, and servo motors for the actuators of this project where I ultimately decided on using stepper motors.

DC Motor

DC motors are one of the simplest actuators that could be used for this project where only an H-Bridge would be required to run it and allow the motor to go backwards and forwards, by switching the direction of current using the H-Bridge, with the code for running this being very simple as well as the ability to change speed. The problem with the motor, however, is that the torque supplied by the DC motors which I have is abysmal making any slight touch to the motor while in operation would affect its operation either slowing it down or speeding it up. Coupled with this, there is no way to tell where the motor is positioned at all so if there was interference during operation it would easily lose it's simulated position in the code causing the Mechanical Pong game to not operate properly at all, be it trying to place the Pong ball out of the bound, or when it "bounces" off walls it could do so prematurely. Because of these issues I have decided not to use DC motors as the actuator of choice for this project.

Servo Motor

The servo motor is by far the simplest actuator that could have been used for this project as it does not require any external circuitry and only requires the Arduino and the Servo.h library to run. Unlike the DC motor the servo motor can keep track of its position and has much more torque which are some of the most important deciding factors in using an actuator for this. The way the servo knows where it is, is by using a potentiometer attached to the gearing inside the servo motor used as a potential divider and this output voltage by the potential divider is used to tell where the servo motor is positioned. For the turning of the servo motor, the Arduino applies a PWM signal to the servo motor which ranges between 1 to 2 milliseconds. The IC inside of the servo motor then compares the current position of the servo motor to where it should be according to the 1-2ms pulses. If there is a difference the IC activates the motor using an H-Bridge (to spin in either direction) to turn to the correct angle. This process of comparing the current position is repeated until the servo motor is pointed in the right direction. However, this actuator did come with some downsides. The servo motor is only able to sweep between 0 and 180 degrees which means multiple rotations would not be viable which meant that this is unsuitable for use in this project as the size of this Mechanical Pong game play area would not be able to be mapped by a servo motor doing a 180 degree turn. Alongside the downside of not being able to do full rotations, there is no speed control of the servo motor which is a major downside as one of the key feature of the original Pong game is that the Pong ball would speed up or slow down depending on how the ball was hit.

Stepper Motor

The final actuator that was considered, and subsequently chosen for the project, was the stepper motor. The stepper motor is the most complex of the three to operate but offers all of the features which are required for this project. The stepper motor has the same requirements circuitry wise as the DC motor with the use of the H-Bridge, but the programming side for the Arduino is more complex than driving the DC motor as for the stepper motor to rotate it has to follow specific step orders so that it will smoothly rotate. The stepper motor doesn't have direct information on where it is, instead this is managed by the code that runs the stepper motor as one step is 1/200th of a rotation (for the stepper motors I have selected, others may vary) where by counting how many steps has been taken it can be figured out where the stepper motor currently is. Despite the fact that the stepper motor gives no positional feedback, this can still be made useful through the use of interrupts and flags to make sure the stepper motor is where the code thinks it is. The stepper motor is also able to rotate in both directions, and complete full rotations which is mandatory for this project as there will need to be multiple rotations to cover the entire play area. Finally, the stepper motors provide the most amount of torque out of all of the actuators being considered, because of this any light obstruction in the way of the stepper motor would be easily clearable with the stepper motor not losing its position. Because of all these features the stepper motors provides I have chosen this for my project.

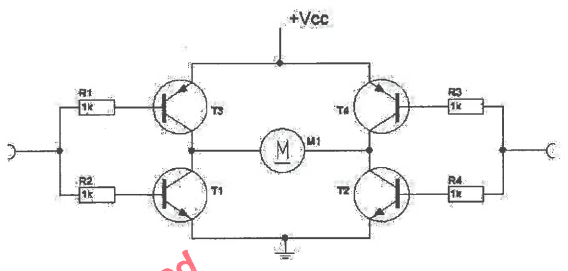
H-Bridge and Stepper Motor Control

Stepper motor

The stepper motor has two coils which are used to rotate the stepper motor. In order for the motor to rotate/step there is a drive pattern for the coils to follow, and the reverse of this order makes the stepper rotate/step in the opposite direction. This pattern follows by energizing the primary coil, then turning that coil off and energizing the second coil, after this that coil is then turned off and the first coil is energized once again but this time in the opposite polarity, for the final step in this loop, the primary coil is turned off and the secondary coil is energized in the opposite polarity. Due to the fact that the polarity of the coils must be switched the use of a Dual H-Bridge chip is required as the H-Bridge gives the ability to switch the direction of current.

H-Bridge

The H-Bridge is an integrated circuit which takes two inputs and depending on those inputs it switches the direction of current. The H-Bridge circuit uses both PNP and NPN transistor to accomplish this. When the no signal is low on either side of H-Bridge (see



diagram[3]) by default current from Vcc is allowed to flow through the PNP transistor and not allowing it to go through the NPN transistor, therefore the only path it has is to go through the load, in the case of the stepper motor the load being the coil but in the diagram it is the motor but there is no path to ground. The opposite is true when a high signal is applied, the PNP transistor won't allow current to flow through it and the NPN will taking the voltage to ground. Through the combination of this circuit a high signal on one side and low on the other will allow current to flow through in one direction. When these signals are reversed, low on one side high on the other, the direction of current will have been changed through the load and changed the direction of rotation.

[3] https://arduinodiy.wordpress.com/2014/03/28/the-h-bridge/