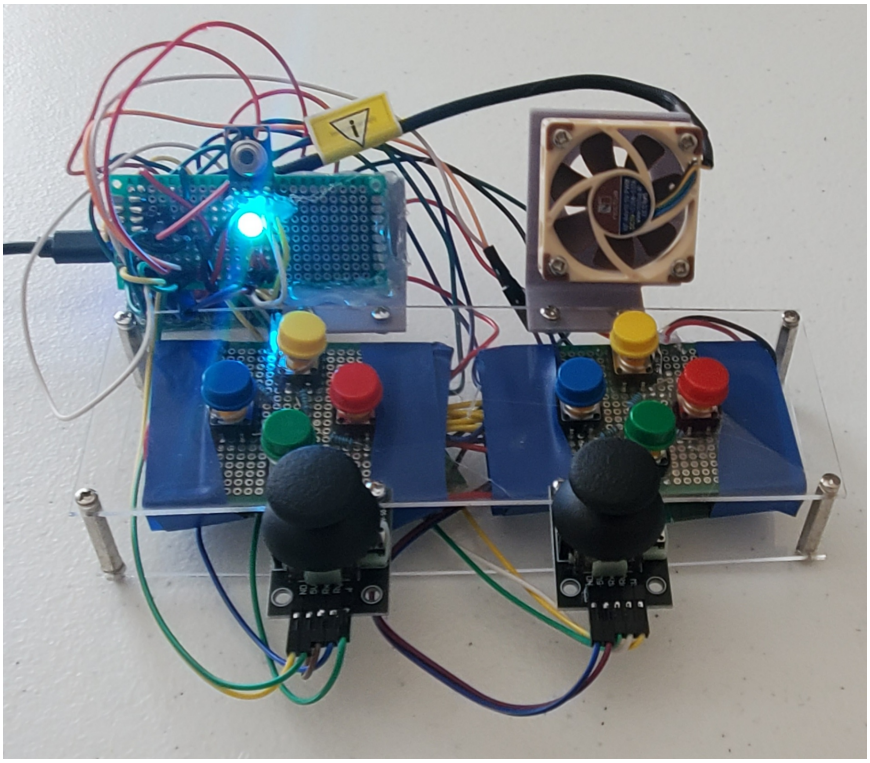
Video Game Controller



ECE3301L

Ryan Gutierrez
Kyler Martinez
Anongnat Suebsak
Derrick Varnes

ABSTRACT

The purpose of this project was to explore the potential applications of microcontrollers. Our project is a simple study on the use of a microcontroller for controlling PWM-based hardware, for reading/writing with digital input signals and two-dimensional analog input signals, and for reading and translating precise analog measurement data. Our investigation of potential microcontroller applications began with the intention of creating a system that can respond to a user's environment as well as their spontaneous controlled inputs. The Arduino platform allowed our team to develop a gaming controller that measures the user's skin temperature and outputs a signal to a fan system for moderated cooling. We incorporated digital I/O pins, analog pins, SCL/SDA pins, digital logic circuitry, and PWM to accomplish our goal. C-language programming forms the foundation of the project's main concepts, while simple wiring techniques and affordable hardware provide its structure. The Arduino microcontroller, when paired with sensors, can relay precise data quickly and efficiently and is a viable tool for driving PWM-based hardware. It also works well as a simple controller, although design precautions are necessary for avoiding false reads in larger networks of spontaneous digital/analog user inputs.

INDEX TERMS

Analog and digital circuits, feedback circuits, microcontrollers, pulse width modulation, analog to digital conversion, and sensor systems.

TABLE OF CONTENTS

INTRODUCTION

In this project, we use an Arduino Micro to program our video game controller by using the C programming language. By doing this project, we are able to understand how the digital input/output and analog input works with the 10 push buttons, the 2 LEDs, and the two thumbsticks. We also found out how to use the Pulse Width Modulation (PWM) and have the fan's speed be dependent on the temperature reading.

METHODOLOGY

Controller Buttons

Hardware

To implement the controller buttons, two-button modules were created, depicted in figure 1, with 4 buttons connected to 3.3V, each with a 150 Ohm resistor connected to the button and ground with a button at the node connecting the switch and resistor. All the button outputs were taken to be used in the digital logic circuit.

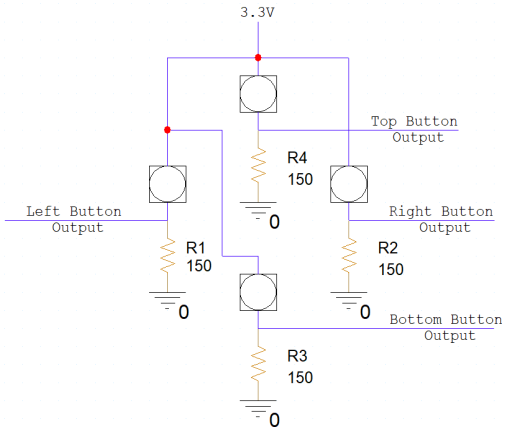


Figure 1: PSPICE Schematic For Button Module

The controller is designed such that when one button is pressed, the Arduino will then read the output of each of the buttons and update the controller status. To implement this, see figure 2, all the button inputs from each module were passed into OR gates to have one OR gate that outputs high if one button on the left side is pressed, and one for the right. These two outputs were then passed into an OR gate for our input for our interrupt at pin D7.

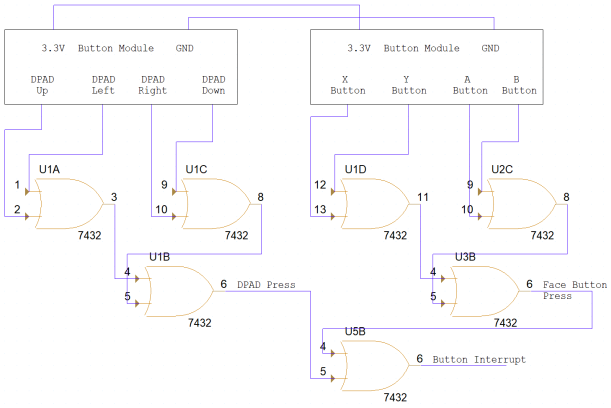


Figure 2: PSPICE Schematic For Button Interrupt Circuit

To reduce the number of used pins for the Arduino, two 4-1 multiplexers were used to select the button to read based on the combination of S_0 and S_1. An inverter was used to disable one multiplexer while the other was enabled, this was done to prevent inaccurate button measurements. The Arduino will output the logic for S_0, S_1, and Mux Enable and read the logic at the DPAD and Face Button outputs.

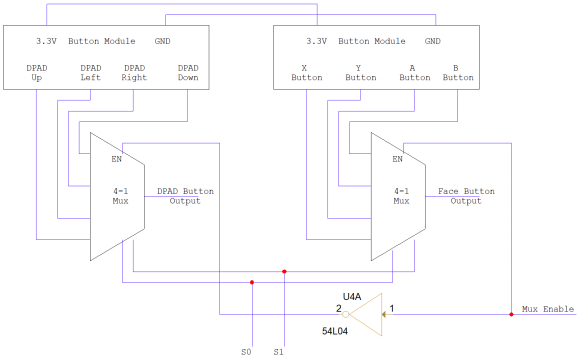


Figure 3: PSPICE Schematic For Button Read Circuit

Software

In the program, an interrupt at D7 is used connected to the Button Interrupt. The interrupt is designed to be triggered twice per button press, a)when a button is pressed and no buttons were recently pressed and b)when no buttons

are pressed after a button was recently pressed. To accomplish this, three modes are used to model this behavior:

1) Idle - buttons were not recently pressed
2) Active - buttons are pressed
3) Over - buttons are not pressed and were previously Active

The ISR for the interrupt is *ButtonDetect()* where it will disable the interrupt and determine if the controller is transiting into the over state by checking the value of *Active* and setting the interrupt to trigger when the input transitions to high or the buttons are pressed. If the previous state was not active, the interrupt will be changed to falling. At the end of the ISR, *Active* is inverted.

```
void ButtonDetect()
{
  detachInterrupt(digitalPinToInterrupt(BIP));
  if (Active)
  {
    attachInterrupt(digitalPinToInterrupt(BIP), ButtonDetect, RISING);
    Over = 1;
  }
  else attachInterrupt(digitalPinToInterrupt(BIP), ButtonDetect, FALLING);
  Active = ~Active;
}
```

Listing 1: C Code for the function *ButtonDetect()*

In the main loop, the value of *Active* will be OR'ed with *Over* and checked if either is 1 and if so, execute the needed code to read and set the buttons. If *Over* is 1, it is cleared to signify the end of the over state and return to idle.

The *Buttons()* function takes an integer for the *BRB*, button read pin, *offsetNum*, and *EN*, the logic level outputted to the multiplexer enable. A nested for loop will output all the possible input combinations to the multiplexer and then read the value at the pin denoted by *BRB* and store it in an array called *results*. A second for loop will then run and check the value of the button reading and press the button if the voltage was high or release the button if not. *offsetNum* is the offset in the array for both the left and right side buttons which allow for one function to be used to activate and deactivate the buttons.

```
void Buttons(int BRB, int offsetNum, int EN)
{
  temp = 0;
  bitWrite(PORTC, 6, EN);
  delay(5);
  for (int i = 0; i < 2; i++)
  {
    for (int j = 0; j < 2; j++)
    {
      bitWrite(PORTD, 6, i);
      bitWrite(PORTB, 7, j);
      delay(5);
      results[temp] = bitRead(PINB, BRB);
      temp++;
    }
  }

  for (int k = 0; k < 4; k++)
  {
    if (results[k]) XInput.press(buttons[k + offsetNum]);
    else XInput.release(buttons[k + offsetNum]);
  }
}
```

Listing 2: C Code for the function *Buttons()*

Analog Sticks

Hardware

To implement the analog sticks, each analog stick uses 2 analog ports, 1 digital, and 2 pins for 5V and ground as seen in Figure 4. The analog ports are used to read the potentiometer analog value and convert it to a digital one. Since each analog stick is made up of 2 potentiometers for the full 360-degree motion, the y-axis and x-axis each use an analog port and the digital port is used for the button. The buttons are connected to analog ports but are configured to digital. The buttons are active low so the output is LOW when the button is pressed.

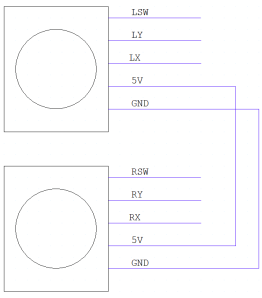


Figure 4: PSPICE Schematic For Analog Sticks

Software

In the program, the XInput library does most of the heavy lifting when it comes to setting up the analog sticks. The x-axis and y-axis are defined by the respective analog port that they are connected to. The XInput library then takes the value from the potentiometer and orients the direction. The *StartSelect()* function is used to set up the analog stick buttons as the "Start" and "Select" buttons of the controller. It is also used to set those buttons as a way to update the colors of the LEDs.

```
        void JoyStickRead()
        {

          leftJoyX = analogRead(Pin_LeftJoyX);
          leftJoyY = analogRead(Pin_LeftJoyY);
          XInput.setJoystickX(JOY_LEFT, leftJoyX);
          XInput.setJoystickY(JOY_LEFT, leftJoyY, invert);

          rightJoyX = analogRead(Pin_RightJoyX);
          rightJoyY = analogRead(Pin_RightJoyY);
          XInput.setJoystickX(JOY_RIGHT, rightJoyX);
          XInput.setJoystickY(JOY_RIGHT, rightJoyY, invert);
        }

void StartSelect()
{
  buttonBack  = !digitalRead(Pin_ButtonBack);
  buttonStart = !digitalRead(Pin_ButtonStart);
  XInput.setButton(BUTTON_BACK, buttonBack);
  XInput.setButton(BUTTON_START, buttonStart);
  if(buttonBack | buttonStart) updateLEDs(buttonBack, buttonStart);
}
```

Listing 3: C Code for the functions *JoyStickRead()* and *StartSelect()*

Temperature Control & Fan

Hardware

To implement the fan, we connected the 3 wires (PWM Signal, +5V, and Ground) to the Arduino as in Figure 5 below. In this case, we do not use the TACH wire or RPM speed signal wire since we do not intend to read the speed of the fan. We connected the +5V wire of the fan to the power on the Arduino Micro board. The ground pin of the fan goes into the drain of the MOSFET (2N7000) since we will use MOSFET to activate the fan on and off. The source of the MOSFET goes to ground, and the gate of the MOSFET is connected to port 8

(D8) of an Arduino Micro board. Lastly, the PWM signal wire is connected to port D6 or PD6.

The MLX90614ESF is a contactless infrared temperature sensor we used in our project that features a precision of 0.02°C and a transmittable temperature range of [-20,120]°C. We connected the Vin to 3.3V, GND to ground, and the SDA and SCL pins to the SDA and SCL pins of the Arduino. The Arduino uses the serial data and clock pins to directly communicate with the sensor, providing a means of performing continuous serial readouts of the measured temperature.

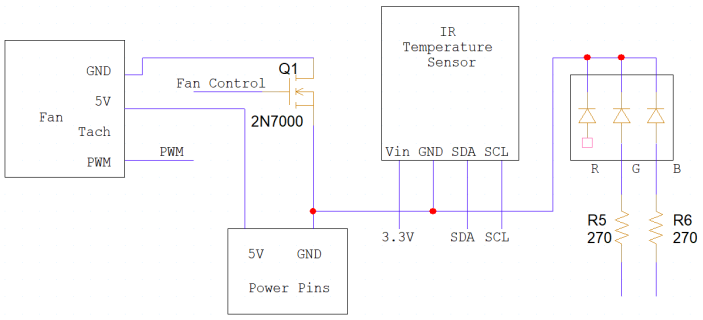


Figure 5: PSPICE Schematic For Fan & RGB LED Module.

Software

In this program, we use 25 kHz for the frequency of the fan as is stated on the fan datasheet. We use the TCNT1_TOP register to set up the frequency of the PWM fan as shown in Listing 4. The *analogWrite()* function is used to output a PWM signal based on the duty cycle specified as the second parameter. We also use the *digitalWrite()* function to set the pins to HIGH or LOW values. In this case, HIGH is to turn on the fan and LOW is to turn off the fan. Additionally, if and else statements have been implemented in this program to set the fan to be on, off, and to run at which duty cycle.

The Adafruit MLX90640 Library was used to handle the I^2C communication between the MLX90640 and Arduino. An instance of the MLX90640 was created and the *begin()* function started the I^2C communication. Whenever we wanted to read the temperature the function *readObjectTempF()* would be called and return a temperature reading in degrees Fahrenheit.

```
const word PWM_FREQ = 25000;
const word TCNT1_TOP = 16000000/(2*PWM_FREQ);
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
```

Listing 4: Fan & Temperature Sensor Configuration.

```
TempF = mlx.readObjectTempF();

//Nonie
if (TempF < 80.0)
{
  analogWrite (PWM, 0);
  digitalWrite (FC, LOW);
}
else if (TempF < 100)
{
  analogWrite (PWM, 255*(TempF-80)/20);
  digitalWrite (FC, HIGH);
}
else
{
  analogWrite (PWM, 255);
  digitalWrite (FC, HIGH);
}
```

Listing 5: C Code for controlling the fan.

RGB LED

The RGB LED is part of the fan module and is connected to the GND of the other pins and wires that lead from the green and blue components of the LED out. In the *updateLEDs()* function, there is an if statement that will check if the current running time is greater than that of the last reference time plus 750ms. If so the reference time will be reset and the logic values of the LEDs will be inverted if their respective button is pressed and then the LEDs will be updated.

```
void updateLEDs(int st, int sl)
{
  if (millis() > LEDTime + 750)
  {
    LEDTime = millis();
    if (st) LEDG = ~LEDG;
    if (sl) LEDB = ~LEDB;
    digitalWrite (GLED, LEDG);
    digitalWrite (BLED, LEDB);
  }
}
```

Listing 6: C Code for the function *updateLEDs()*

RESULTS

After assembling all the components together, we used an online gamepad tester to ensure the buttons read properly and signal the proper buttons. Each button read correctly and the analog sticks displayed the expected values. To be certain that the controller was compatible with real applications, the controller was tested using different computers and different games to ensure proper functionality. The temperature sensor accurately read the room and skin temperature and the fan was turned on and the speed adjusted as the room temperature increased. We activated the fan by changing the temperature surrounding it with items like a blow dryer and our hands. Overall, each of the functions implemented in the controller operated properly.

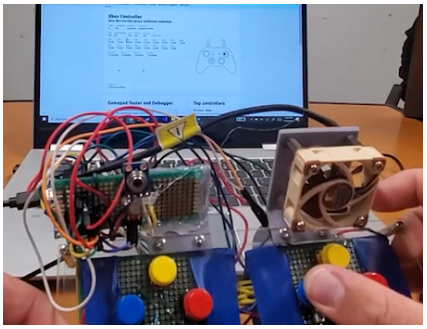


Figure 6: Controller Testing

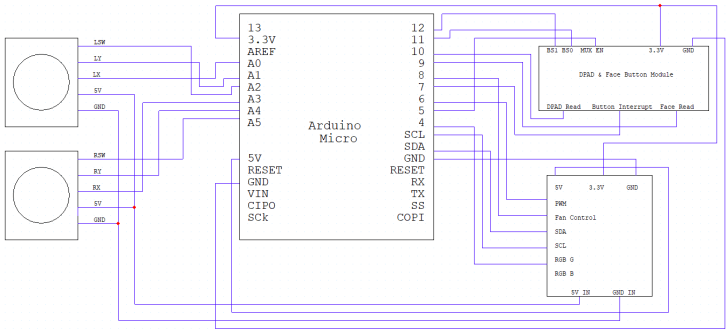


Figure 7: PSPICE Schematic Of All Modules

CHALLENGES

Controller Buttons

While testing the controller, it was noticed that the button readings would occasionally report inaccurate readings. Occasionally a change in output would not be recorded when the selection pins were updated. To resolve this, a delay was added after the select pins were changed which improved the controller's accuracy. Originally both of the multiplexers were enabled but after testing it was found that the inputs would short together and that would result in the controller thinking that a button was pressed on both sides of the controller. In the original design of the controller, this was not noticed but when altered the interrupts, the left side of the controller would be activated when only the right side was pressed. We resolved this issue by creating the inverter circuit for the enable pins.

PWM Fan

While trying to implement the code of the fan, it was noticed that the frequency that we used before we come down to 25 kHz would not work on the fan. The code that was written to control the fan would not work because the fan would not turn off once it was on and the duty cycle of the fan could not be controlled either. To fix these, the research has been done by going back to the fan datasheet and it was specified to 25 kHz frequency. Moreover, the new code has been implemented to be able to get the fan working by fixing the value in the

second arguments in *analogWrite()* functions and adding the *digitalWrite()* to program the MOSFET to turn on and off the fan.

CONCLUSION

From doing this project, we gained a greater understanding of microcontrollers by applying different types of applications to them. The PWM-based hardware was used to control the speed of the fan and turn the fan on and off. We also make use of MOSFET's ability to use the PWM fan. Moreover, digital input/output was applied to the PWM fan as well. The analog sticks were also successful since we used ADC to make the two sticks to be able to roll 360 degrees and be able to change the LEDs each time when it is pressed. SCL/SDA and I2C functions were used with the temperature sensor and get temperature data from it, so we can adjust the duty cycle of the fan. The GPIO and interrupts were used with the buttons; therefore, we were able to press a button on the control and translate that button press to one in the game.

In conclusion, We have reached our goal in terms of building the video game controller by using the C programming language and learning all the new materials. We have got to learn about different kinds of functions and program libraries that we have never seen or heard of before. Our video controller works properly without any features being unusable.

REFERENCES

- California State Polytechnic University, Pomona. (2021). Intro to Microcontrollers.[Online].
- Arduino XInput Library. (1.2.5), Dave Madison. Accessed: October 2021. [C++ Arduino Library]. Available: https://github.com/dmadison/ArduinoXInput
- Adafruit MLX90640 Library. (1.0.2), Adafruit. Accessed: October 2021. [C++ Arduino Library]. Available: GitHub - adafruit/Adafruit_MLX90640: MLX90640 library functions
- "*Noctua PWM specifications white paper*," 2021. Accessed on: October 2021[Online]. Available: https://noctua.at/pub/media/wysiwyg/Noctua_PWM_specifications_white_paper.pdf
- T. Youngblood, *How To Use Arduino's Analog and Digital Input/Output (I/O)*, June 10, 2015. Accessed on: October 2021. [Online]. Available: https://www.allaboutcircuits.com/projects/using-the-arduinos-analog-io/