Microcontrollers programming

Jordi Bueno Domínguez and Abdelrahman Rayyan

June 2014

Abstract

The bluetooth controller has been designed with an ATmega328 micro-controller. It makes use of four potentiometers using analog to digital conversion for the two analog sticks, and also one push button for each stick using interrupts. The controller also uses the data from a 3-axis accelerometer and gyroscope conected with I2C. All the user input is then transmited using UART over bluetooth to the controlled device. Our setup includes a second micro-controller acting as a receiver that uses the user input to modify the color of a RGB led using pulse width modulation.

Contents

1. Introduction

Wireless controllers are used everywhere to interface humans with machines. There are popular new ways of interaction, like touch screens, movement and gesture detection, and voice recognition. But when you need speed and precision, like in videogames or wireless vehicle control, what you need is a joystick.

That's why we choosed a controller similar to the ones used by Playstation or Xbox. This project requieres using most of the features in a micro-controller.

- ADC to convert the analog data from the sticks to digital
- I2C to communicate with the accelerometer
- UART for the communication between micro-controllers
- PWM is used for the controlled RGB led
- Interrupts are used to read the push buttons

As well as using the already mentioned peripherals.

- Potentiometers x4
- Buttons x2
- Accelerometer and gyroscope
- RGB led
- Bluetooth devices x2
- LCD and several leds for debugging

1.1 Objectives

The objective of the project is to have a functional wireless controller. There's no need to have as many buttons or features as a comercial one, nor be compatible with them. The purpose is to learn how to use all the featues of the micro-controller in a real example.

The wirless controller relies on the analog sticks as the main user input. The acceleremoter and gyroscope data is used as a secondary way of control. The buttons, the number of which is easly modified depending on the application, are the last input the user can give. All the data is transfered using wireless communication, which for now, is Bluetooth using the UART protocol. Further revisions of the project may use other wirless devices.

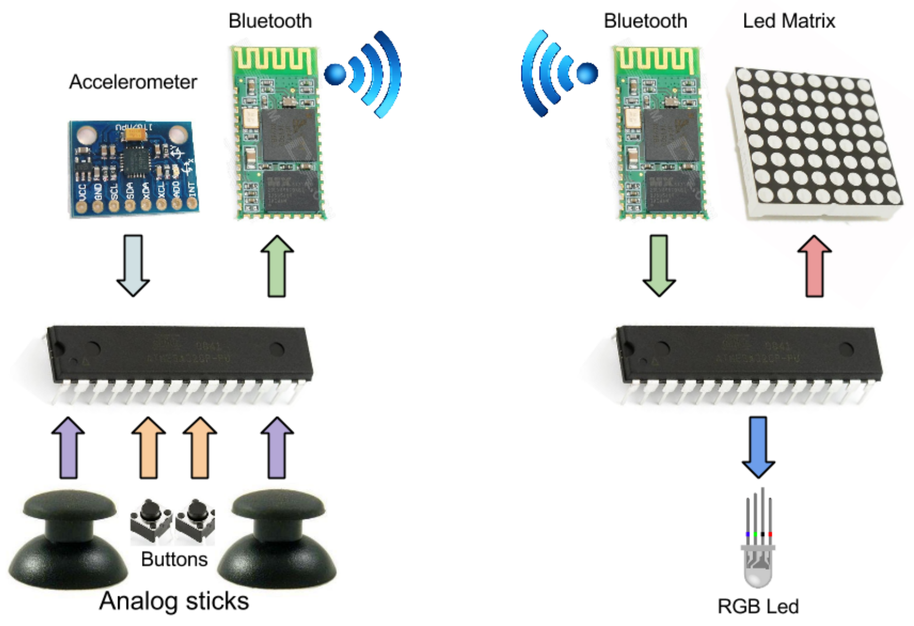

Figure 1.1: Project diagram

2. Implementation

2.1 Hardware

For this project we are using the Atmel [1] micro-controller ATmega328, which comes in a 28 pin package.

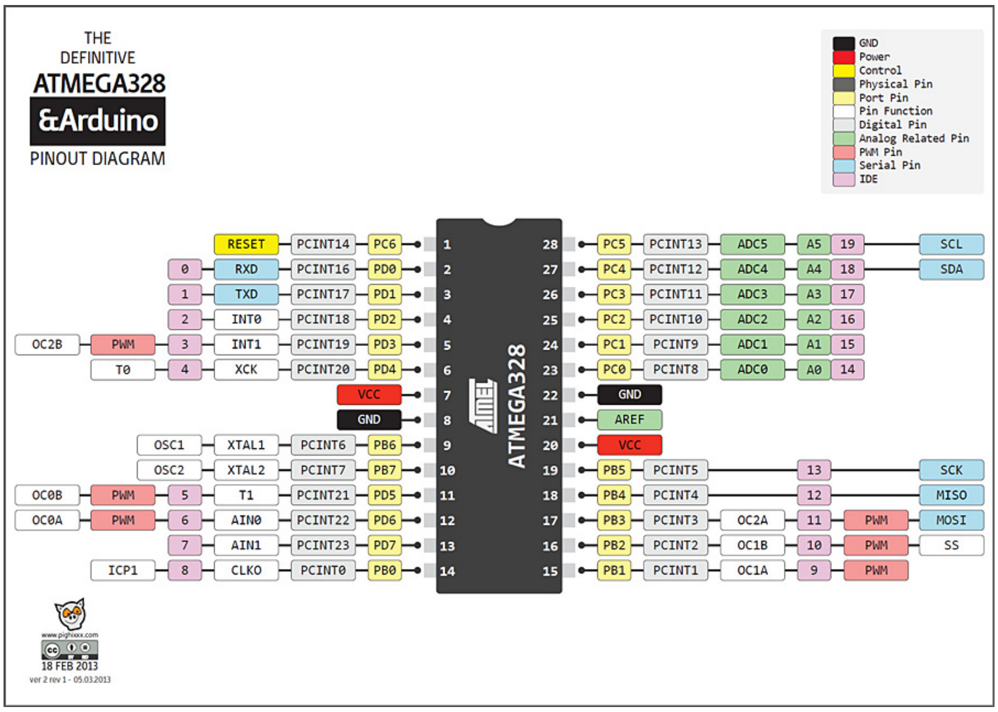

Figure 2.1: ATmega328

The high-performance Atmel 8-bit AVR RISC-based microcontroller combines 32 KB ISP flash memory with read-while-write capabilities, 1 KB EEPROM, 2 KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8-5.5 volts. By executing powerful instructions in a single clock cycle, the device achieves throughputs approaching 1 MIPS per MHz, balancing power consumption and processing speed.

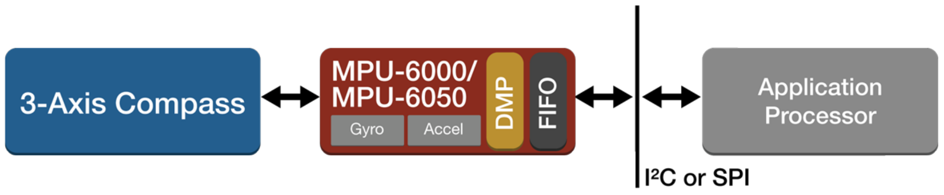

Figure 2.2: MPU-6000/6050 System Diagram

The MPU-6050 packages an accelerometer and a gyroscope. It communicates as a slave with the micro-controller and provides the necessary data to get the roll, pitch and yaw. It is very useful to get the orientation and movements of the device, and as such is one of the best sensors for user interaction.

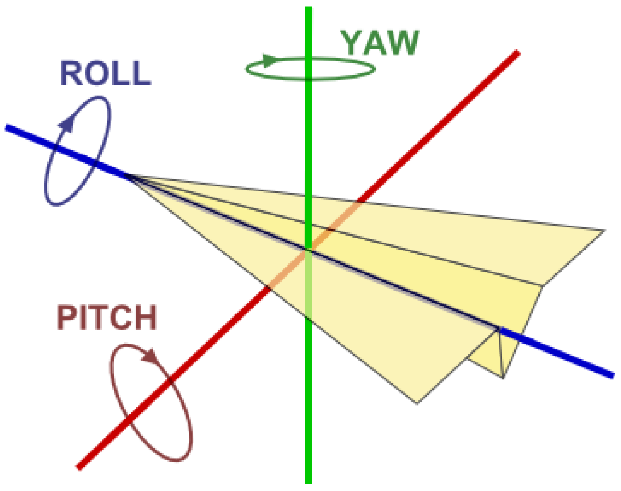

Figure 2.3: Roll, pitch, yaw

The analog sticks consists of two potentiometers build up together. They are packaged in a certain way that allows the user to control all the directions in a plane. It works by using the variation on the resistance depending on the position. Because of these, it requieres the usage of ADC to get the data into the micro-controller. Each analog stick also includes a press button that works by pushing down the stick.

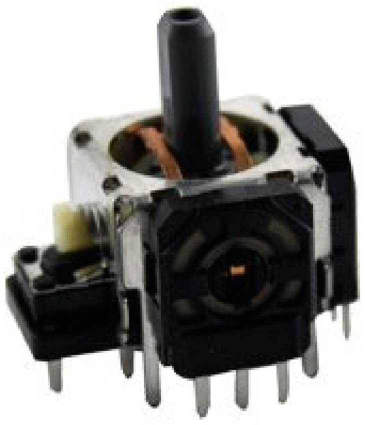

Figure 2.4: Analog stick without the rubber

The easiest way to communicate wirelessly is using one of these Bluetooth modules, which work the same as a pair of UART cables.

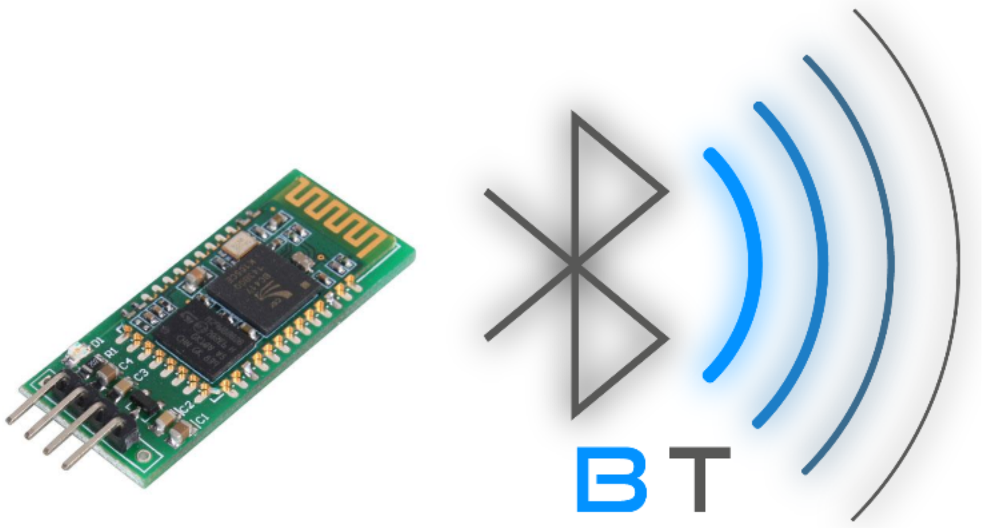

Figure 2.5: Bluetooth module

2.2 Connections

The schematics for the connections look like this. The UART connection is represented here with cables, but it can be easly accomodated to the bluetooth modules. The usb connections for programming, like the lcd and other debugging elements, are not included.

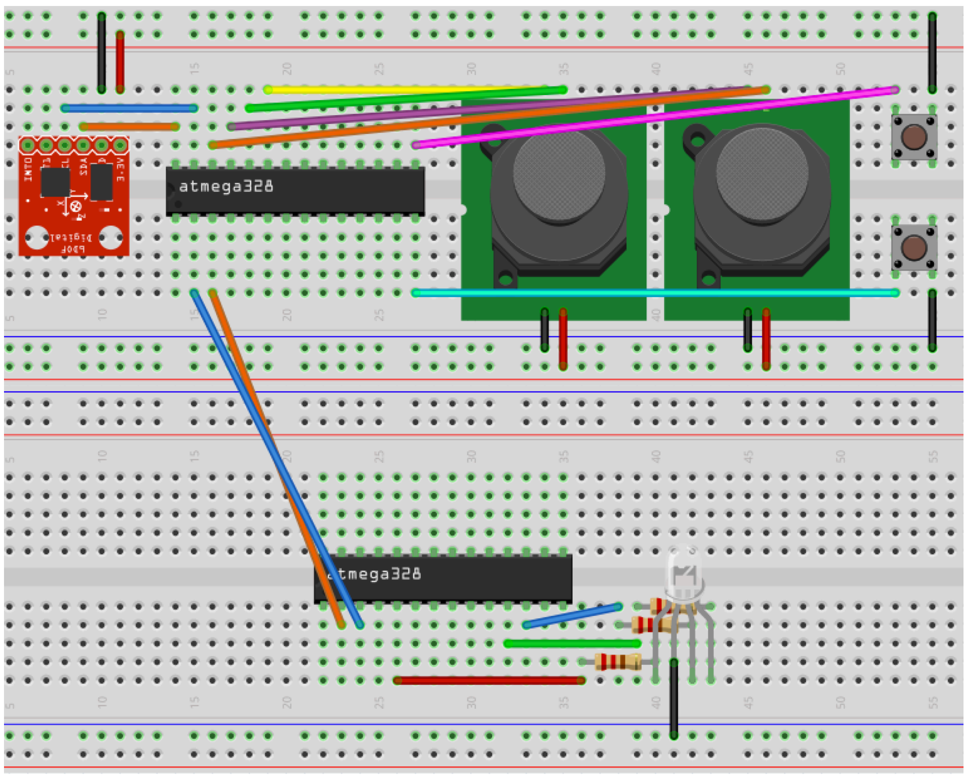

Figure 2.6: Connections

2.3 Software

The flow of both applications is very simple. The controller needs to initialize the interrupts for the buttons, the accelerometer, the analog to digital converter and the UART communication. The target initializes the RGB output, the PWM, UART communication, and optionally the LCD or Matrix of leds to output the data. The main loop for the controller consistis of reading the inputs and sending the data, while the target receives the data and outputs the results.

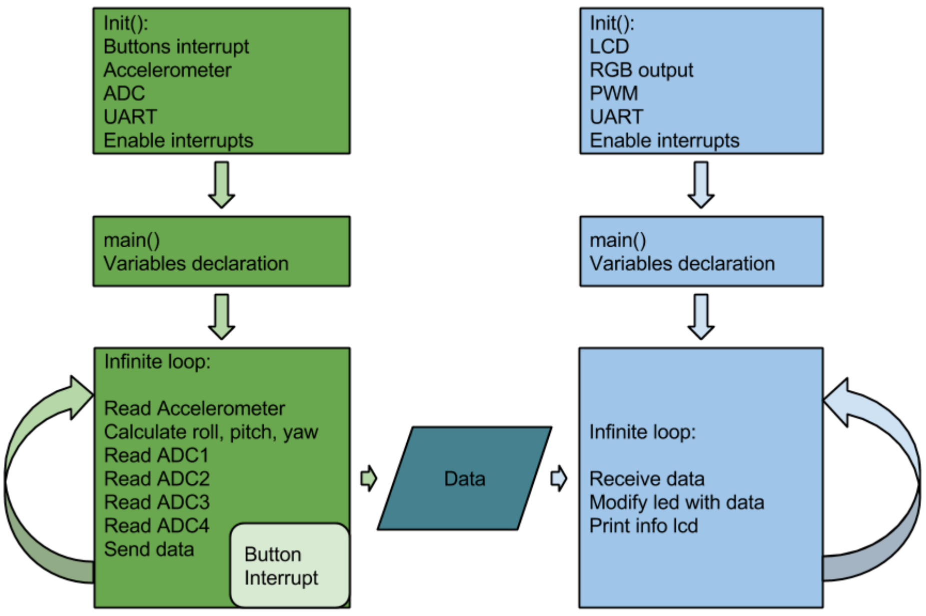

Figure 2.7: Flow chart

The PWM is used to control the bright and color of a RGB led, three channles are requiered, one for each color. Using this code, the pins PD3, PD5 and PD6 become the output for the led pins. PWM outputs the correct intensity by switching on and off the led fast enough.

```
        // initialize rgb output
        DDRD = 0b01101000;

        // initialize PWM 0A, 0B, 2B
        TCCR0A = (1 << COM0A1) | (0 << COM0A0) | (1 << COM0B1)
                      | (0 << COM0B0) | (0 << WGM01) | (1 << WGM00);
        TCCR0B = (0 << WGM02) | (0 << CS02) | (0 << CS01) | (1 <<
            CS00);
```

```
    OCR0A = 0;
    OCR0B = 0;

    TCCR2A = (1 << COM2B1) | (0 << COM2B0) | (0 << WGM21) | (1
        << WGM20);
    TCCR2B = (0 << WGM22) | (0 << CS22) | (0 << CS21) | (1 <<
        CS20);
    OCR2B = 0;
```

```
#define RED_LED      OCR2B
#define GREEN_LED    OCR0B
#define BLUE_LED     OCR0A
```

The ADC is requiered to read the values from the analog sticks. Each stick has two potentiometers and uses two signals. This code is used to transform the values to digital.

```
    // ADC init
    ADMUX = (0 << REFS1) | (1 << REFS0)
                  | (0 << ADLAR) // Left aligned
                  | (0 << MUX3) | (0 << MUX2) | (0 << MUX1) | (0
                      << MUX0);
    ADCSRA = (1 << ADEN)
                  | (1 << ADSC)
                  | (0 << ADATE) | (1 << ADIE)
                  | (0 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
```

```
            // Potentiometers readings
            ADMUX &= 0xF0;
            ADMUX |= (0 << MUX3) | (0 << MUX2) | (0 << MUX1) | (0
                << MUX0);
            ADCSRA |= (1 << ADSC);
            while (ADCSRA & (1 << ADSC));

            ADMUX &= 0xF0;
            ADMUX |= (0 << MUX3) | (0 << MUX2) | (0 << MUX1) | (1
                << MUX0);
            ADCSRA |= (1 << ADSC);
            while (ADCSRA & (1 << ADSC));

            ADMUX &= 0xF0;
            ADMUX |= (0 << MUX3) | (0 << MUX2) | (1 << MUX1) | (0
                << MUX0);
            ADCSRA |= (1 << ADSC);
            while (ADCSRA & (1 << ADSC));
```

```
              ADMUX &= 0xF0;
              ADMUX |= (0 << MUX3) | (0 << MUX2) | (1 << MUX1) | (1
                  << MUX0);
              ADCSRA |= (1 << ADSC);
              while (ADCSRA & (1 << ADSC));
```

```
// ADC interrupt, stores each value in the data structure
ISR(ADC_vect) {
       uint8_t mux_count;
       mux_count = ADMUX;
       mux_count &= 0x0F;
       switch(mux_count){
              case 0: data.adcAX = ADCW; break;
              case 1: data.adcAY = ADCW; break;
              case 2: data.adcBX = ADCW; break;
              case 3: data.adcBY = ADCW; break;
              default: break;
       }
}
```

The comunication with the MPU-6050 module is done using a dedicated library [2], our code makes uses of that library.

```
       // init accelerometer
       mpu6050_init();
       _delay_ms(50);
       mpu6050_dmpInitialize();
       mpu6050_dmpEnable();
```

```
       //Accelerometer variables
       double qw = 1.0f;
       double qx = 0.0f;
       double qy = 0.0f;
       double qz = 0.0f;
       double roll = 0.0f;
       double pitch = 0.0f;
       double yaw = 0.0f;

              // Accelerometer
              if(mpu6050_getQuaternionWait(&qw, &qx, &qy, &qz)) {
                     mpu6050_getRollPitchYaw(qw, qx, qy, qz, &roll,
                         &pitch, &yaw);
              }
              _delay_ms(10);

              data.roll = roll;
              data.pitch = pitch;
              data.yaw = yaw;
```

The buttons are managed using interrupts. In order to make PB0 and PB1 the input buttons, this code is required.

```
       //Init change pins interrupts
   DDRB &= ~((1 << DDB0) | (1 << DDB1));
   PORTB |= ((1 << PORTB0) | (1 << PORTB1));
   PCICR |= (1 << PCIE0);
   PCMSK0 |= (1 << PCINT0) | (1 << PCINT1);
```

```
ISR(PCINT0_vect){
       data.buttons = PINB & 3;
}
```

3. Conclusion

To sum up, we developed a controller for other projects. It can be wireless, and it sends the user input of analog sticks (joysticks), accelerometer and gyroscope, and buttons. User feedback like vibration or small leds could be easly added in future projects. This controller could be used to control a wireless controled vehicle, like a car, boat or quadcopter. It could also be used as input for games, like in a video game console.

Further projects should standardize the datagrams for the communications, to make it easier interfacing with new applications. The final packaging must also be improved to give the user a good holding of the device.

Bibliography

[1] Atmel.
Avr datasheet.
`http://www.atmel.com/Images/doc8161.pdf`.
last visited 01/06/2014.

[2] Davide Gironi.
Mpu-6050 library for avr.
`http://davidegironi.blogspot.se/2013/02/avr-atmega-mpu6050-gyroscope-and.html#.U4txAhZM7B0`.
last visited 01/06/2014.