JoyStick to USB interface

2016-08-15

Contents

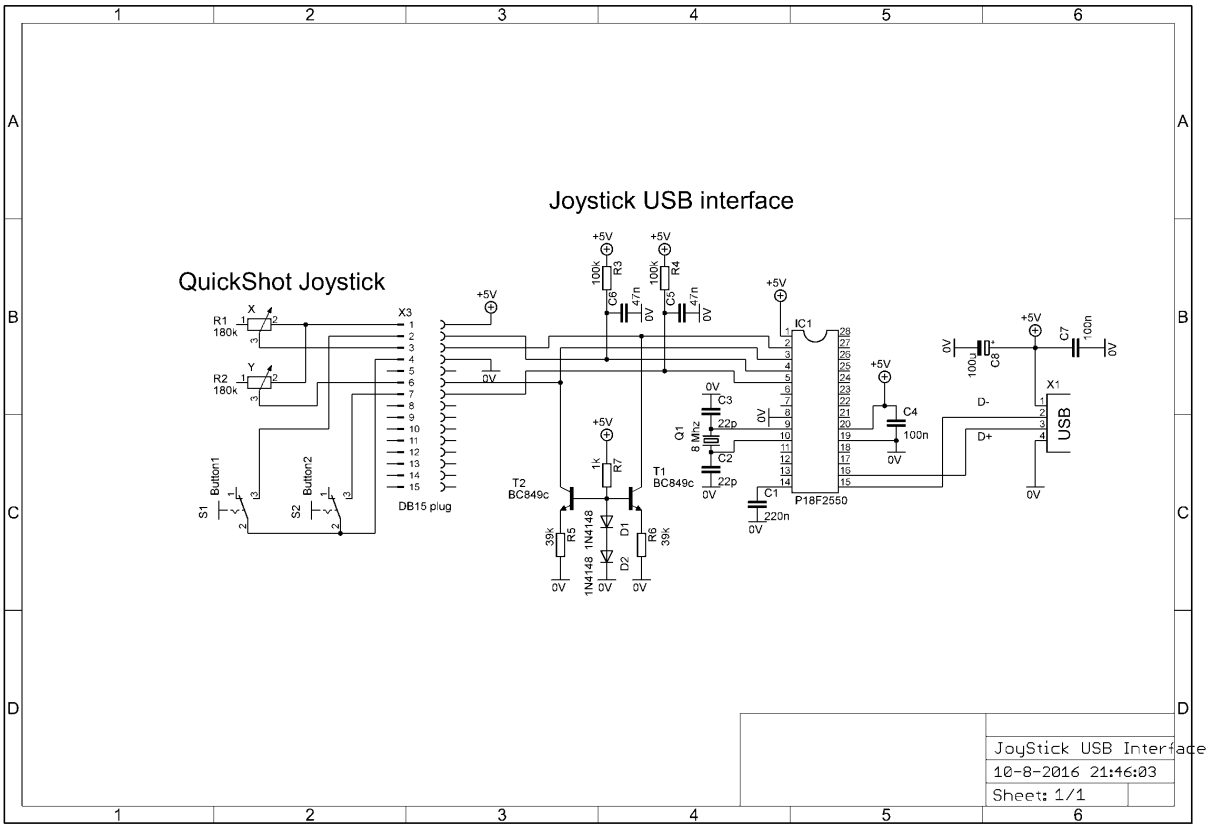
1 Purpose

The purpose of this project is to provide an interface between an “old” analog joystick (in this case a joystick from Quickshot with 2 buttons and an X and Y positioning), connected via a game port (DB15 connector) and a PC via USB.

The analog joystick uses as X and Y positioning 2 potmeters with a value of approximately 180 KOhm. See the schematic below for its internals (left side of the schematic).

The hardware and software should be easily be adaptable to other joysticks.

2 Schematic



The left side of the diagram shows the internal circuit of the joystick, the right hand side shows the interface between the joystick and the USB port to be connected to the PC.

The output of the joystick to its DB15 connector is:

- Pin1: +5V
- Pin2: Button 1
- Pin3: X-axis potmeter
- Pin4: Ground for Button1 and Button 2
- Pin6: Y-axis potmeter

The rest of the DB15 pins are unused by this type of joystick.

2.1 Details

The debouncing of the 2 buttons is done with a resistor of 100K and a capacitor of 47nF, e.g. R3 and C6. Both debounced buttons are connected to PIC inputs acting as digital inputs.

Both X-axis and Y-axis potmeters are connected to analog inputs of the PIC, their values will be converted to digital values with the built in ADC convertor of the PIC.
To make the voltage on the analog inputs linear with respect to the joystick axis potmeter values each potmeter is sent a constant (very small) current through it.
The constant current is chosen such that the minimal voltage (when the resistance of the axis

potmeter is maximum) on the analog PIC inputs is approximately 1V. This means that a voltage range of 1V to 5V is available for actual analog processing by the PIC.
The constant current source is made of a transistor with a fixed base voltage and a fixed emitter resistor, e.g. T1 and R6. Here the emitter resistor is 39K. The latter has to be adapted if the potmeter value deviates from 180K. The base voltage of both constant current transistors (T1 and T2) is made by R7, D1 and D2.

3 Software

The software is written in MikroPascal from MikroElektronika (see http://www.mikroe.com/).

The project consists mainly of 2 code files:

- The main file “USB_JoyStick.mpas”
- The USB descriptor file “USB_HID_ProjectItems.mpas”

Additionally there are 2 project related files:

- The Project file “USB_JoyStick.mpppi”
- The PIC configuration file “USB_JoyStick.cfg”

The project also uses one library: USB_HID_Library (not added here, because specific for mikroPascal).

The software should be easily adaptable to e.g. C and the usage of the USB library at hand.

3.1 Details

The USB library together with the USB descriptor (see below) make that the USB device created here will be shown by the PC as an joystick.

3.1.1 The USB descriptor

The USB descriptor tells the PC the details about the USB device we are making:

```
const ReportDescriptor: array[ReportDescriptorLen] of byte =
(
    0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
    0x09, 0x04,                    // USAGE (Joystick)
    0xa1, 0x01,                    // COLLECTION (Application)
    0x16, 0x00, 0x00,              // LOGICAL_MINIMUM (0)
    0x26, 0xFF, 0x03,              // LOGICAL_MAXIMUM (1023)
    0x05, 0x01,                    //   USAGE_PAGE (Generic Desktop)
    0x09, 0x01,                    //   USAGE (Pointer)
    0xa1, 0x00,                    //   COLLECTION (Physical)
    0x09, 0x30,                    //     USAGE (X)
    0x09, 0x31,                    //     USAGE (Y)
    0x75, 0x10,                    //     REPORT_SIZE (16)
    0x95, 0x02,                    //     REPORT_COUNT (2)
    0x81, 0x02,                    //     INPUT (Data,Var,Abs)
    0xc0,                          //   END_COLLECTION
    0x05, 0x09,                    //   USAGE_PAGE (Button)
    0x19, 0x01,                    //   USAGE_MINIMUM (Button 1)
```

```
    0x29, 0x02,                    //   USAGE_MAXIMUM (Button 2)
    0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
    0x25, 0x01,                    //   LOGICAL_MAXIMUM (1)
    0x75, 0x01,                    //   REPORT_SIZE (1)
    0x95, 0x08,                    //   REPORT_COUNT (8)
    0x55, 0x00,                    //   UNIT_EXPONENT (0)
    0x65, 0x00,                    //   UNIT (None)
    0x81, 0x02,                    //   INPUT (Data,Var,Abs)
    0xc0                           // END_COLLECTION
);
```

As one can see the X-axis and Y-axis both use 2 bytes (16 bits), the 2 buttons use 2 bits in the last byte. This makes a total of 5 bytes (numbered 0..4) in the report, with this layout:

```
// USB Data Map
//   |----|----|----|----|----|----|----|----|
// 0 | X Axis lo byte                        |
//   |----|----|----|----|----|----|----|----|
// 1 | X Axis hi byte                        |
//   |----|----|----|----|----|----|----|----|
// 2 | Y Axis lo byte                        |
//   |----|----|----|----|----|----|----|----|
// 3 | Y Axis hi byte                        |
//   |----|----|----|----|----|----|----|----|
// 4 |    |    |    |    |    |    |SW2 |SW1 |
//   |----|----|----|----|----|----|----|----|
```

3.1.2 X and Y-axis processing

The code performing this (code for X-axis only here):

```
MyValX := ADC_Read(0);  // (1)
if MyValX >= 200  // offset (actual values range from 200 to 1023) (2)
then MyValX := MyValX - 200 (3)
else MyValX := 0; (4)
MyValX := (MyValX * 5) shr 2; (5)
if MyValX > 1023 then MyValX := 1023; (6)
MyValX := 1023 - MYValX; (7)
SendBuffer[0] := lo(MyValX); (8)
SendBuffer[1] := hi(MyValX); (9)
```

Explanation:

- Line 1: Get the x-axis SDC value (a word ranging from 0 to 1023)
- Line 2, 3, 4: Decrement the measured value with 200. The latter corresponds to the minimal ADC voltage of 1V (see section 3.1). This makes the value range 0..823.
- Line 5,6: Make the value range again 0..1023 again: multiply by 5 and subsequent divide by 4 (actually gives as range 0..1028, limited again to 1023 in line 6).
- Line 7: Mirrors the value: the measured value is low when the resistance is high, so it has to be mirrored.

- Line 8,9: Fill the USB sendbuffer with the Low order byte and the high order bytes of the X-axis value.

3.1.3 Button processing

The code for this is:

```
SendBuffer[4] := 0; (1)
SendBuffer[4].0 := Not(PortA.2); // switch 1 (2)
SendBuffer[4].1 := Not(PortA.3); // switch 2 (3)
```

Explanation:

- Line 1: Clear all bits in the last byte of the USB sendbuffer
- Line 2: Place the inverted (low = active) button 1 state in bit 0
- Line 2: Place the inverted button 2 state into bit 1.

3.1.4 Send via USB to the PC

The code doing this:

```
repeat
until
USB_HID_Write(@SendBuffer, 5); // send 5 bytes
```

The above statement does the actual sending of 5 bytes to the PC.

[end of document]