Laboratory No. 10:
BZFlag Controller

By: Derek Hildreth

Instructor: Brother Karl

BYU-Idaho CompE 360

December 9, 2008

I. INTRODUCTION

A. Purpose

The objectives in this laboratory are to

1. Utilize the hardware as well as the drivers and software developed in the previous labs to control and play a networked game called BZFlag.

B. Equipment

- Level 0 Drivers for Parallel Port
- Level 1 Drivers for MicroDigital Board
- ADC hardware and software
- Controller components: 1 joystick, 3 buttons, 1 rumble pack, 3 10KΩ Resistors

C. Procedure

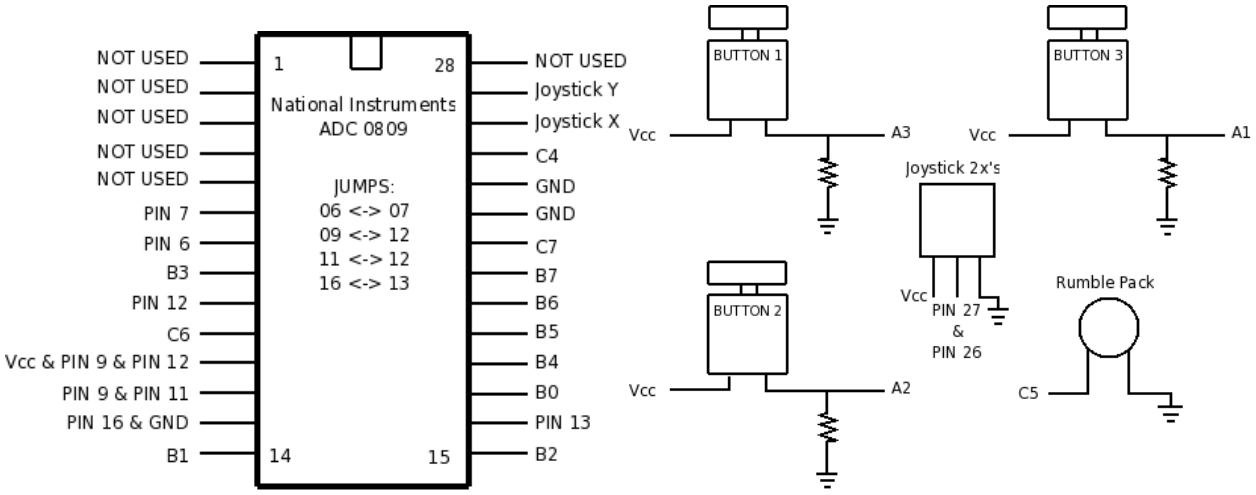
1. Use all the drivers, software, and hardware to produce a working ADC which gets the two voltages of the joystick (X & Y Axis).
2. Integrate those drivers, software, and hardware into the BZFlag game source code to control the tank's movement, canon, jumping, and exit the game.

D. Requirements

1. The requirements are to have control of the tank using the joystick, to use the buttons to 1. fire the canon, 2. jump the tank, and 3. exit the game, and to get feedback from the rumble pack when the tank fires (1 second) and when you are destroyed (2 seconds).

II. SCHEMATICS

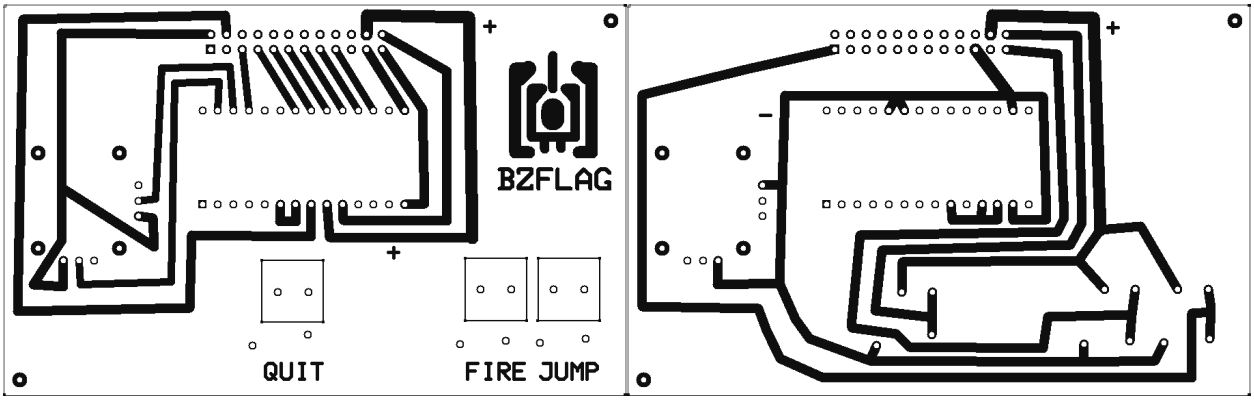
This page will consist of schematics which were used in this laboratory.



(a) ADC Pin Out

(b) Misc parts and their pin outs.

FIG. 1: Overall schematics and pin outs for interfacing the ADC and MicroDigital Board.



(a) Front connections of PCB Board.

(b) Back connections of PCB Board.

FIG. 2: The front and back connections of the two-layer PCB Board.

III. PROGRAMMING

This section will consist of the code that made the integration of the controller possible with the BZFlag game.

```
/* bzflag
 * Copyright (c) 1993 - 2007 Tim Riker
 *
 * This package is free software;  you can redistribute it and/or
 * modify it under the terms of the license found in the file
 * named COPYING that should have accompanied this file.
 *
 * THIS PACKAGE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

// interface header
#include "playing.h"
/*SNIP--SNIP--SNIP----------------------------------------------------------------------*/
/*****************************************************************************
* Most of the code below this line has been deleted from the original "playing.
* cxx" file with only the changed portion for the controller still in tact.
* Look for "SNIP" to indicate where a good chunk of code was deleted.
*****************************************************************************/

/*****************************************************************************
* Initialize and Functions
*****************************************************************************/
const int BASE = 0x378;
static BOOL bPrivException = FALSE;
static bool shootRumble = false;  // The start of the rumble that was never fin.

/*SNIP*************************************************************************
* This is where the Level 0 and Level 1 drivers are located.
*****************************************************************************/

// getVoltageX - accepts the address of the port (0x378), gets the voltage from
// the ADC and then returns the voltage to the calling variable.

int getVoltageX(int port)
{
   int mx;
   int in;

   // Toggle ALE on ADC
   outPortC(BASE, 0x00);
   outPortC(BASE, 0x80);
   outPortC(BASE, 0x00);
   // END TOGGLE

   // CLOCK IT IN
   for (int i=0; i < 150; i++)
   {
```

```
        outPortC(BASE, 0x40);
        outPortC(BASE, 0x00);
    }
    // END CLOCK

    in = inPortB(BASE) & 0xff; // Get only needed bits from PortB (NEEDED)
    mx = in * 5 - 680;         // * 5 - 680 needed to get scale right for game.

    return(mx);
}

// getVoltageY - accepts the address of the port (0x378), gets the voltage from
// the ADC and then returns the voltage to the calling variable.

int getVoltageY(int port)
{
    int my;
    int in;

    // Toggle ALE on ADC
    outPortC(BASE, 0x10);
    outPortC(BASE, 0x90);
    outPortC(BASE, 0x10);
    // END TOGGLE


     // CLOCK IT IN
     for (int i=0; i < 150; i++)
     {
    outPortC(BASE, 0x40);
    outPortC(BASE, 0x00);
     }
    // END CLOCK

    in = inPortB(BASE) & 0xff; // Get only needed bits from PortB (NEEDED)
    my = -(in * 4 - 528);      // * 4 - 528 needed to get scale right for game.

    return(my);
}

/*SNIP--SNIP--SNIP----------------------------------------------------------------------*/

/*****************************************************************************
* doMotion() - This function was a part of the original playing.cxx code.  It
* is where the code to control the tank with the joystick is contained.
*****************************************************************************/
static void doMotion()
{
  float rotation = 0.0f, speed = 1.0f;
  const int noMotionSize = hud->getNoMotionSize();
  const int maxMotionSize = hud->getMaxMotionSize();

  int keyboardRotation = myTank->getRotation();
  int keyboardSpeed    = myTank->getSpeed();
```

```
  /* see if controls are reversed */
  if (myTank->getFlag() == Flags::ReverseControls) {
    keyboardRotation = -keyboardRotation;
    keyboardSpeed    = -keyboardSpeed;
  }

  // mouse is default steering method; query mouse pos always, not doing so
  // can lead to stuttering movement with X and software rendering (uncertain why)
  int mx = 0, my = 0;
  mainWindow->getMousePosition(mx, my);

   mx = getVoltageX(BASE);
   my = getVoltageY(BASE);

   // This block of code was used to output messages to the display to help get
   // the scale correct for the joystick
   //  std::string message;
   //  char msg[5];
   //  message += "this is mx ";
   //  sprintf( msg, "%d", mx );
   //  message += msg;
   //  message += "  my ";
   //  sprintf( msg, "%d", my);
   //  message += msg;
   //  addMessage(myTank, message);
   // END BLOCK

/*SNIP--SNIP--SNIP----------------------------------------------------------------------*/

/*****************************************************************************
* playingLoop() - The controls for firing, jumping, and exiting are in here.
*****************************************************************************/
static void playingLoop()
{
  int i;
  int fireTimer = 0;
  int deadTimer = 0;
  static bool shotRumble = false; // Begining of the RumblePack that wasn't fin.

/*****************************************************************************
* OPEN AND CONFIGURE PORTS
*****************************************************************************/
OpenPortDriver(BASE);
OpenPortDriver(BASE + 1);
OpenPortDriver(BASE + 2);

configPort(BASE, 0x83);
/*****************************************************************************/

/*SNIP--SNIP--SNIP----------------------------------------------------------------------*/

   // FIRE THE CANON WITH BUTTON A (Notice the magic: inPortA(BASE) == 4).
   if (myTank->getTeam() != ObserverTeam &&
       ((fireButton && myTank->getFlag() == Flags::MachineGun) ||
```

```
        (myTank->getFlag() == Flags::TriggerHappy)) || (inPortA(BASE) == 4))
    {
      myTank->fireShot();
    // outPortC(BASE, 0x20);  // Begining of the RumblePack that wasn't fin.
        }
        } else {
    int mx, my;

    outPortC(BASE, 0x20); // Begining of the RumblePack that wasn't finished.

    mainWindow->getMousePosition(mx, my);
        }
        myTank->update();


          // JUMP THE TANK WITH BUTTON B
    if(inPortA(BASE) == 2)
      myTank->doJump();

/* Begining of the RumblePack that wasn't finished
      if(inPortA(BASE) == 4)
    {
        outPortC(BASE, 0x20);
        shootRumble = true;
    } */

    // EXIT THE GAME WITH BUTTON C
    if(inPortA(BASE) == 1)
      CommandsStandard::quit();  // Several different commands would've worked.
     }

/*SNIP--SNIP--SNIP----------------------------------------------------------------------*/
//EOF
```

IV. CONCLUSION

This laboratory's purpose was to get the drivers, software, and hardware to work with the BZFlag game. Students were to control the tank using the ADC converter, joystick, buttons, and MicroDigital board as well as provide feedback through the rumble pack.

The most difficult part of this laboratory was integrating the code into the existing BZFlag source code. Once the important file and the important lines within that file were located, things became a little easier. The most difficult part that I personally struggled with was the joystick movement. I was able to overcome these by thinking things through and making sure that the code made sense. The other part that I struggled with is the Rumble Pack. I never did get it to work without a pause in the game. In other words, using the *Sleep(1000)* or *for (int i = 0; i < 1000; i++)* commands resulted in a literal pause in the game when the tank was fired or killed. The work around that I thought of was to utilize the *While* loop that was already in the *playingLoop()* function to do the rumble for me, this way there wouldn't be any pause. Unfortunately, time was a factor in getting this completed and I ran out of time.

The PCB Board layout (as seen in Figure 2a and 2b) was hard getting started with because of the many connections and the space to do them in. It was almost like a puzzle trying to route the wires to a spot on the header without crossing another line. The other difficult part was getting the dimensions of the parts spot-on. After making a few print outs and testing the pins of the parts, the PCB was ready to go into production. Unfortunately, there was a "conversion" process that it had to go through in order to get the final, ready to print and iron version. This "conversion" was to take a screenshot of the PCB within ExpressPCB, crop it in MS Paint, copy that into MS Word, and then re-size the images to the board size. I had accidentally assumed 4" length when it was actually 3.8". This really scaled the board so that when I had tried to place the parts, the parts were too small. I tried to bend some pins to fit, but it was hopeless. Another thing that I realized after-the-fact, is that some of my lead were going underneath the chip and joystick which was impossible to solder once they were mounted. So, I redesigned the whole thing and made absolute sure what size I was working with and where my solder points were going to be. After this, the board was easy to assemble. One other thing to note is the absence of the rumble pack. Since I was unable to get the code working with the rumble feature, I made a decision to exclude the rumble pack from the final design.

All in all, this project was excellent and quite possible one of the funnest I've had in a long time. I really enjoyed developing drivers and working with parallel and serial connections in order to produce a product such as this. It has sparked an interest in electronic design and what I can accomplish with programming and hardware know-how.