Wireless Mouse

Project Introduction

The proposed project is a prototype of a wireless mouse using a FPGA and Pmods, JSTK2 and BT2. This is interesting because it explores wireless HID (Human Interface Device), various integrations of rand, integrating multiple Pmods together, and communicating via bluetooth using RN-42. The lack of clear documentation for RN-42, figuring out the state machine for proper interaction of the JSTK2 and Basy buttons to occur, and researching the integrations of rand makes this project engaging and challenging despite how simple it sounds at first.

This project's purpose is to develop a working prototype for a Bluetooth-based wireless mouse which includes real-time HID communication with an external device, allowing adjustable sensitivity for the mouse, including randomness to allow for a self-destruct mode, and interacting the two Pmods together, BT2 and JSTK2. In the end, we developed the project using a finite state machine for the different buttons and moving states, Basys3 buttons and LEDS, Pmod JSTK2 and BT2 (which uses RN-42), and LFSR and LCG (Linear congruential generator) for rand.

Real-World Constraints

Some constraints are user satisfaction, producing a user friendly product that promotes users to want to use our mouse over other mouses. Our current prototype serves as a rather horrible mouse in terms of ergodynamics which can be improved with the final product's manufacturing.

Along with this, since we use a Basys3 Board and multiple Pmods, the energy consumption is rather high. If we were to transition to manufacturing and an actual usable product, an ASIC is a must. Along with this comes the rather large size of the prototype that also can be fixed with an ASIC and integrated chips for the Bluetooth and X,Y, click sensors.

The prototype is quite expensive compared to what a mouse should be in a relative range of for the general public. However, transitioning to an ASIC will solve this.

Manufacturing costs will also cost a bit, but due to the rather simple nature of the mouse and the few things that we need, such as an ASIC chip, a bluetooth module, sensors, and microcontrollers.

The prototype does take quite a bit of energy, however the final product will have a lower energy consumption and proper sleep modes.

Mouses are generally made with a lot of plastic but it is possible to use recycled plastics in order to make them.

While a mouse that utilizes a Basys3 board is overkill, transitioning to a ASIC would serve to make this product very manufacturable. We can include a PCB in order to connect the different modules (Bluetooth, sensors, etc.) and the ASIC. Along with this we would need a casing in order to make it more ergonomic and user friendly.

A mouse doesn't really have any social or political constraints. However, it does have some privacy concerns if the bluetooth is transferring data unsafely.

In regards to sustainability, the casing of a mouse can be made out of recycled materials to be more sustainable.

Industry Standards

Bluetooth (IEEE 802.15.1) – Our project uses the RN-42 Bluetooth module to wirelessly communicate mouse movement and button clicks. Bluetooth is governed by the Bluetooth SIG and follows the IEEE 802.15.1 standard for low-power wireless communication.

ISO 9241-110 – This standard covers ergonomic principles for interaction design, ensuring users can easily interpret LED status indicators for scrolling mode, scaling factor, and other functions.

System-on-Chip (SoC) and FPGA Design Standards – The use of a Basys3 FPGA means we need to follow FPGA development best practices, which is specified in IEEE 1801 (Unified Power Format - UPF) for power-aware design (important for manufacturability).

Embedded Software Standards (MISRA C/C++) – Following best practices for embedded C programming on MicroBlaze, it is important to see MISRA C, which ensures reliable, maintainable embedded software.

SO 9241-9 – This standard covers the ergonomics of pointing devices, ensuring accessible and easy usability of the mouse.

FCC Part 15 – Since Bluetooth operates in the 2.4 GHz ISM band, it must comply with FCC Part 15 regulations, which limit interference with other wireless devices.

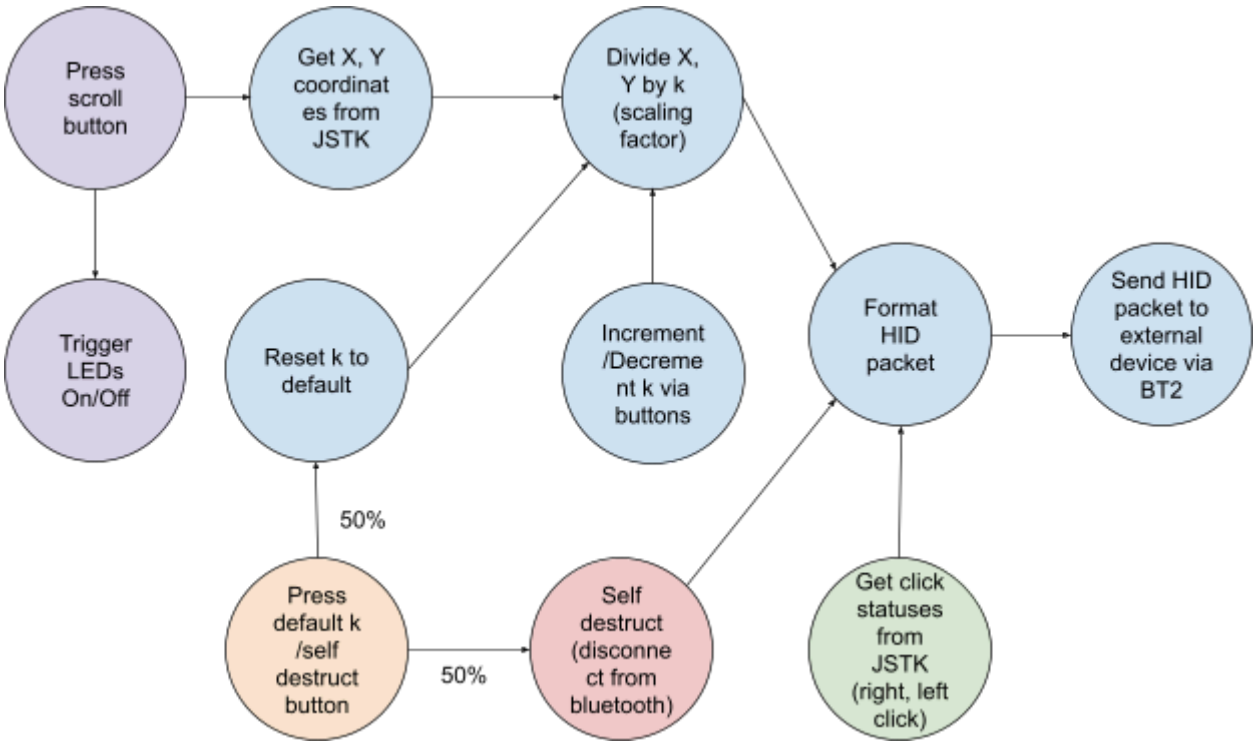
SPI/I2C Communication (JSTK2 Module) – The PMOD JSTK2 module communicates via SPI (Serial Peripheral Interface), which follows industry standards set by IEEE 1149.1.

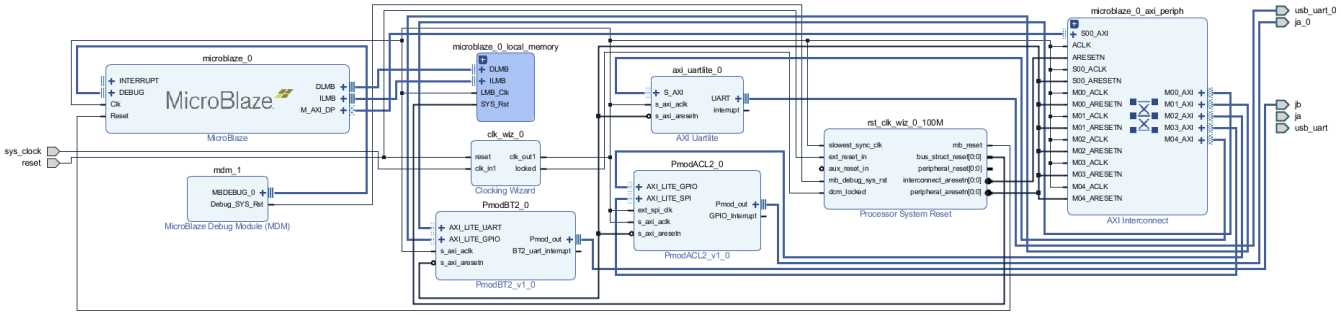
Costs

In regards to development costs, we still would need engineers in order to transition to an ASIC design, along with the manufacturing, labor, and warranty costs. We would also need to market this mouse and in order to do that we would need a marketing research team to figure out why our mouse is better than those already on the market. Physically the prototype costs $211.99 in total including the Basys3 board, Pmod JSTK2, and Pmod BT2. Human costs would include the marketing team, the engineers, the shipping, and the warranty/customer service handling.

Project Description

We ended up getting a fully functional wireless mouse, but it obviously was not able to satisfy all the constraints. The mouse was very large due to the size of the Basys3 board. It was not fully wireless because the board needed to constantly be connected to power. And the mouse was controlled with a joystick, which isn't very ergonomic compared to standard mouse designs. However, we were able to develop in a fast period of time (two weeks), serving as a speedy time to create a foundational prototype for a better wireless mouse.





Implementation Details

Algorithms

In order to enable our self-destruct mode to have a roughly 50/50 chance, we needed to implement some sort of manner to generate the random numbers. We thus followed this stackoverflow post in order to implement C stdlib srand() and rand() using a Linear congruential generator. https://stackoverflow.com/questions/4768180/rand-implementation

However, this is a rather rudimentary algorithm that will fail to produce a good distribution over multiple iterations if just rand() was used. Therefore, we needed to produce some sort of random generator for the seed in order to get a better distribution. Generally, seed() is set by taking the internal time of the computer, but due to the nature of the FPGA, we are unable to recreate this. Therefore, for the seed we decided to use a Linear-feedback shift register in order to generate a random number. The first number is set deterministically, then we shift it in certain ways to create a pseudo random number. It will repeat at some point, but it has a longer cycle than our previous method.

In order to implement our multiple actions, we had a state machine that was basically in a while loop and we would check the state of each button in order to determine which action we are taking.

Hardware

JSTK2 Hardware: Served as an interface to gather user input (coordinates and clicks of the mouse)

BT2 Hardware: Served as the bluetooth module that allows bluetooth devices to connect to the Basys3 board.

Basys3 FPGA: Runs the MicroBlaze soft processor. The GPIO inputs from the buttons and inputs from JSTK2 were sent to the processor and were used to generate HID reports (in the software), where data was then transmitted using the Bluetooth module.

Software

JSTK2 Software: Read joystick data in the software so that they can be sent via HID packets to the device connected via bluetooth. JSTK2 communicates via SPI using the Xilinx SPI driver.

BT2 Software: Uses UART in order to communicate serially. We customize the hardware so that the byte data we send in the software is dedicated for mouse HID packets. This is supported via Xilinx UART drivers.

HID Report: Generated in software and sent serially through BT2 and UART.

State Machine: Used to keep track of the states of each button (left click, right click, DPI increase and decrease, scroll mode). This is made completely in software because it needed to be used synchronously with BT2 data transmission

Random Number Generator: Used to generate random numbers for the self_destruct() function (50% chance of disconnecting if DPI reset is clicked). This is was fully implemented in software because we were able to make use of Linear Congruential Generator and Linear-feedback shift register algorithms.

Code

```
#include "PmodBT2.h"
#include "PmodJSTK2.h"
#include "xil_cache.h"
#include "xgpio.h"
#include "xparameters.h"
#include "sleep.h"


// Required definitions for sending & receiving data over host board's UART
port
#ifdef __MICROBLAZE__
#include "xuartlite.h"
typedef XUartLite SysUart;
#define SysUart_Send            XUartLite_Send
#define SysUart_Recv            XUartLite_Recv
```

```
#define SYS_UART_DEVICE_ID      XPAR_AXI_UARTLITE_0_DEVICE_ID
#define BT2_UART_AXI_CLOCK_FREQ XPAR_CPU_M_AXI_DP_FREQ_HZ
#else
#include "xuartps.h"
typedef XUartPs SysUart;
#define SysUart_Send            XUartPs_Send
#define SysUart_Recv            XUartPs_Recv
#define SYS_UART_DEVICE_ID      XPAR_PS7_UART_1_DEVICE_ID
#define BT2_UART_AXI_CLOCK_FREQ 100000000
#endif

PmodBT2 myDevice;
PmodJSTK2 joystick;
SysUart myUart;
XGpio gpio;

void DemoInitialize();
void DemoRun();
void SysUartInit();
void EnableCaches();
void DisableCaches();

int main() {
   DemoInitialize();
   DemoRun();
   DisableCaches();
   return XST_SUCCESS;
}
void send_hid_mouse_report(uint8_t buttons, int8_t x, int8_t y, int8_t
wheel) {
      uint8_t report[7] = {0xFD, 0x05, 0x02, buttons, x, y, wheel};
      BT2_SendData(&myDevice, report, 7);
}

static unsigned long int next = 1;
static unsigned int lfsr = 0xACE1u;

unsigned int lsfr_rand() {
      lfsr ^= lfsr << 13;
      lfsr ^= lfsr >> 17;
      lfsr ^= lfsr << 5;
      return lfsr;
}
```

```
int rand(void) // RAND_MAX assumed to be 32767
{
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}

void srand(unsigned int seed)
{
    next = seed;
}

void init_rand() {
      srand(lsfr_rand());
}

void DemoInitialize() {
   EnableCaches();
   SysUartInit();
   BT2_Begin (
      &myDevice,
      XPAR_PMODBT2_0_AXI_LITE_GPIO_BASEADDR,
      XPAR_PMODBT2_0_AXI_LITE_UART_BASEADDR,
      BT2_UART_AXI_CLOCK_FREQ,
      115200
   );

   JSTK2_begin(
      &joystick,
      XPAR_PMODJSTK2_0_AXI_LITE_SPI_BASEADDR,
      XPAR_PMODJSTK2_0_AXI_LITE_GPIO_BASEADDR
   );
}

void self_destruct() {
      for (int i = 0; i < 25; i++) {
            send_hid_mouse_report(0, 0b1000000, 0b10000000, 0);
            usleep(20000);
      }
      send_hid_mouse_report(0, -128, 0, 0);
      usleep(20000);
      send_hid_mouse_report(0, -71, 0, 0);
      usleep(20000);
```

```
      send_hid_mouse_report(0x01, 0, 0, 0);
      usleep(20000);
      send_hid_mouse_report(0x00, 0, 0, 0);
      usleep(20000);
      send_hid_mouse_report(0, 0, 80, 0);
      usleep(800000);
      send_hid_mouse_report(0x01, 0, 0, 0);
      usleep(800000);
      send_hid_mouse_report(0x00, 0, 0, 0);
      usleep(800000);
      send_hid_mouse_report(0x01, 0, 0, 0);
      usleep(800000);
      send_hid_mouse_report(0x00, 0, 0, 0);
      usleep(800000);
}

void DemoRun() {
   u8 buf[7];
   u32 btn, led;

//   int n = 0;
   int smooth_factor = 3;
   int scroll_state = 0;
   unsigned int rand_num = 11727;

   int btn8_clicked = 0;

   JSTK2_Position position;
   JSTK2_DataPacket rawdata;

   XGpio_Initialize(&gpio, 0);

   XGpio_SetDataDirection(&gpio, 2, 0x00000000);
   XGpio_SetDataDirection(&gpio, 1, 0xFFFFFFFF);

   print("Initialized PmodBT2 Demo\n\r");

//   print("Received data will be echoed here, type to send data\r\n");

   while (1) {
      // Echo all characters received from both BT2 and terminal to
terminal
      // Forward all characters received from terminal to BT2
```

```
        position = JSTK2_getPosition(&joystick);
        rawdata = JSTK2_getDataPacket(&joystick);

        if (scroll_state == 0) {
              send_hid_mouse_report(0x00, (position.XData -
128)/smooth_factor, (position.YData - 128)/smooth_factor, 0);
              usleep(20000);
        } else {
              send_hid_mouse_report(0x00, 0, 0, (position.YData -
128)/smooth_factor);
              usleep(20000);
        }
        if (rawdata.Jstk != 0) {
              u8 prev_state = rawdata.Jstk; // prev_state is 1

              while (1) {
                    rawdata = JSTK2_getDataPacket(&joystick);
                    if (rawdata.Jstk != prev_state) {
                          send_hid_mouse_report(0x01, 0, 0, 0);
                          usleep(20000);
                          send_hid_mouse_report(0x00, 0, 0, 0);
                          break;
                    }
              }
        }

        if (rawdata.Trigger != 0) {
              u8 prev_state = rawdata.Trigger; // prev_state is 1

              while(1) {
                    rawdata = JSTK2_getDataPacket(&joystick);
                    if (rawdata.Trigger != prev_state) {
                          send_hid_mouse_report(0x02, 0, 0, 0);
                          usleep(20000);
                          send_hid_mouse_report(0x00, 0, 0, 0);
                          break;
                    }
              }
        }

        btn = XGpio_DiscreteRead(&gpio, 1);
//      xil_printf("%08x", btn);
```

```
        if (btn == 1) {
              init_rand();
              rand_num = rand();
              if (rand_num % 2) {
                    self_destruct();
              } else {
                    smooth_factor = 3;
              }
        } else if (btn == 2 && smooth_factor < 50) {
              smooth_factor = smooth_factor + 1;
        } else if (btn == 4 && smooth_factor > 1) {
              smooth_factor = smooth_factor - 1;
        } else if (btn == 8) {
              btn8_clicked = 1;
        } else if (btn8_clicked == 1 && btn != 8) {
              btn8_clicked = 0;
              if (scroll_state == 0) {
                    scroll_state = 1;
                    led = 0xFFFFFFFF;
              } else {
                    scroll_state = 0;
                    led = 0x00000000;
              }

              XGpio_DiscreteWrite(&gpio, 2, led);
        }

      // USED FOR BT2 SETUP (Entering command mode)
//      n = BT2_RecvData(&myDevice, buf, 7);
//      if (n != 0) {
//            SysUart_Send(&myUart, buf, 7);
//      }
//      n = SysUart_Recv(&myUart, buf, 1);
//      if (n != 0) {
//         SysUart_Send(&myUart, buf, 1);
//         BT2_SendData(&myDevice, buf, 1);
//
//      }
//
//      n = BT2_RecvData(&myDevice, buf, 1);
//      if (n != 0) {
//         SysUart_Send(&myUart, buf, 1);
//      }
```

```
   }
}

// Initialize the system UART device
void SysUartInit() {
#ifdef __MICROBLAZE__
   // AXI Uartlite for MicroBlaze
   XUartLite_Initialize(&myUart, SYS_UART_DEVICE_ID);
#else
   // Uartps for Zynq
   XUartPs_Config *myUartCfgPtr;
   myUartCfgPtr = XUartPs_LookupConfig(SYS_UART_DEVICE_ID);
   XUartPs_CfgInitialize(&myUart, myUartCfgPtr, myUartCfgPtr->BaseAddress);
#endif
}

void EnableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_ICACHE
   Xil_ICacheEnable();
#endif
#ifdef XPAR_MICROBLAZE_USE_DCACHE
   Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_DCACHE
   Xil_DCacheDisable();
#endif
#ifdef XPAR_MICROBLAZE_USE_ICACHE
   Xil_ICacheDisable();
#endif
#endif
}
```