Nathan Nguyen

Steven Ryan Leonido

EE128 SEC: 021
Final Project: Bop-It Game

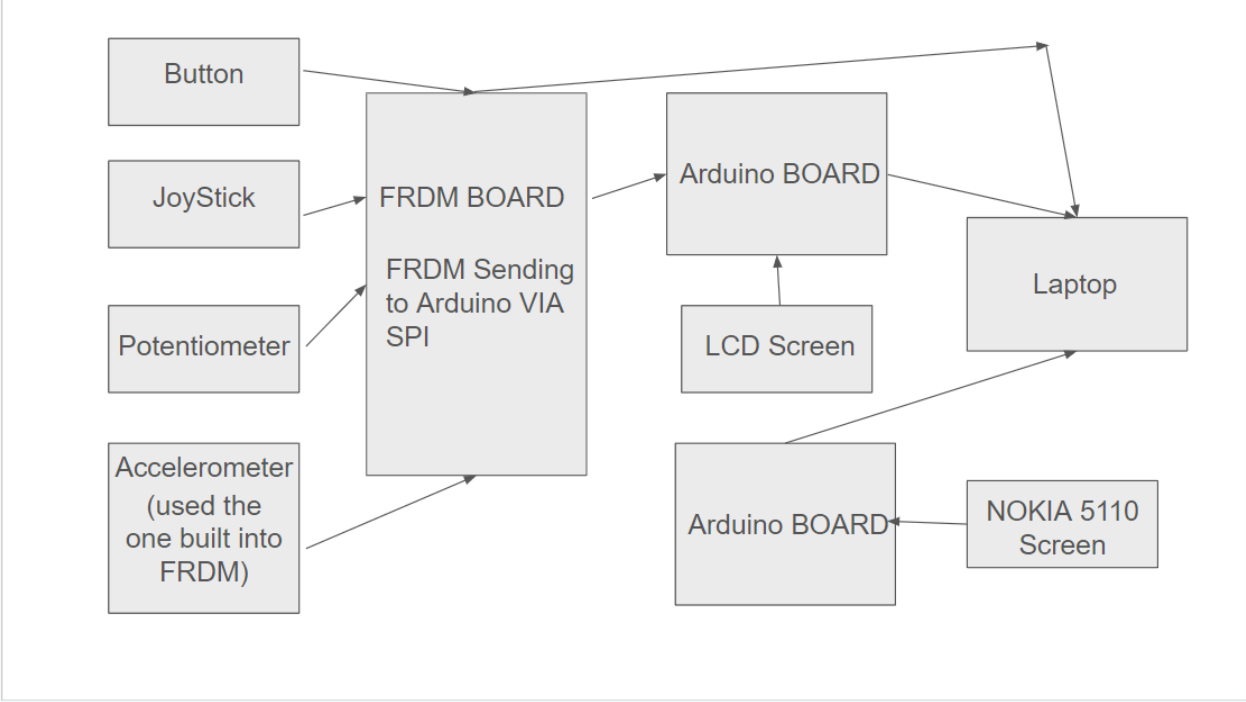
-Project Description

-Summary, requirements/goals

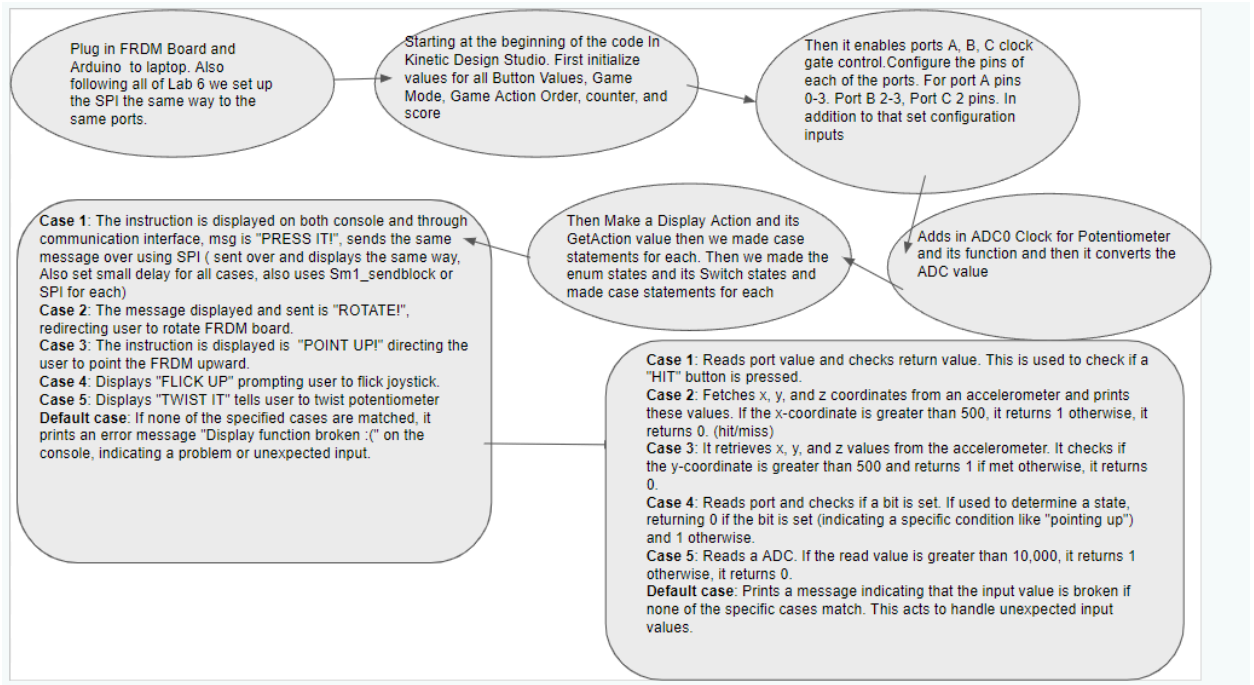
Our EE128 project is a Bop-it game that utilizes all that we have learned from Lab 1-6 of EE128. It uses both a FRDM and an Arduino board in addition to that we used its dedicated coding softwares. With them being Arduino IDE and Kinetic Studios Software. As the name suggests our goal was to make a Bop-It style game which is a game that tells you an action and followed by that do that specific action in a time frame if not then it will register it as a miss and then overall calculate your end score. In terms of the actions we used all arduino components for them so we used the button, joystick, FRDM accelerometer, and potentiometer, then displayed it all using SPI into the LCD screen. Some requirements that we had were to use the K64 board and then add some other complex operations to it, in addition to that we also wanted to add as many given arduino components in as possible for the complexity.

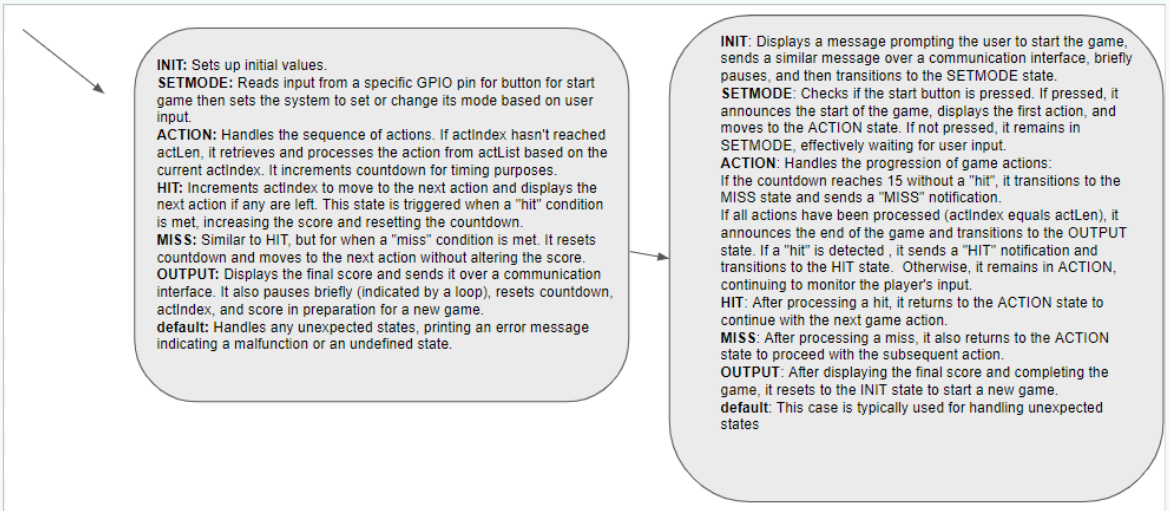
-System Design

-Hardware Block Diagram



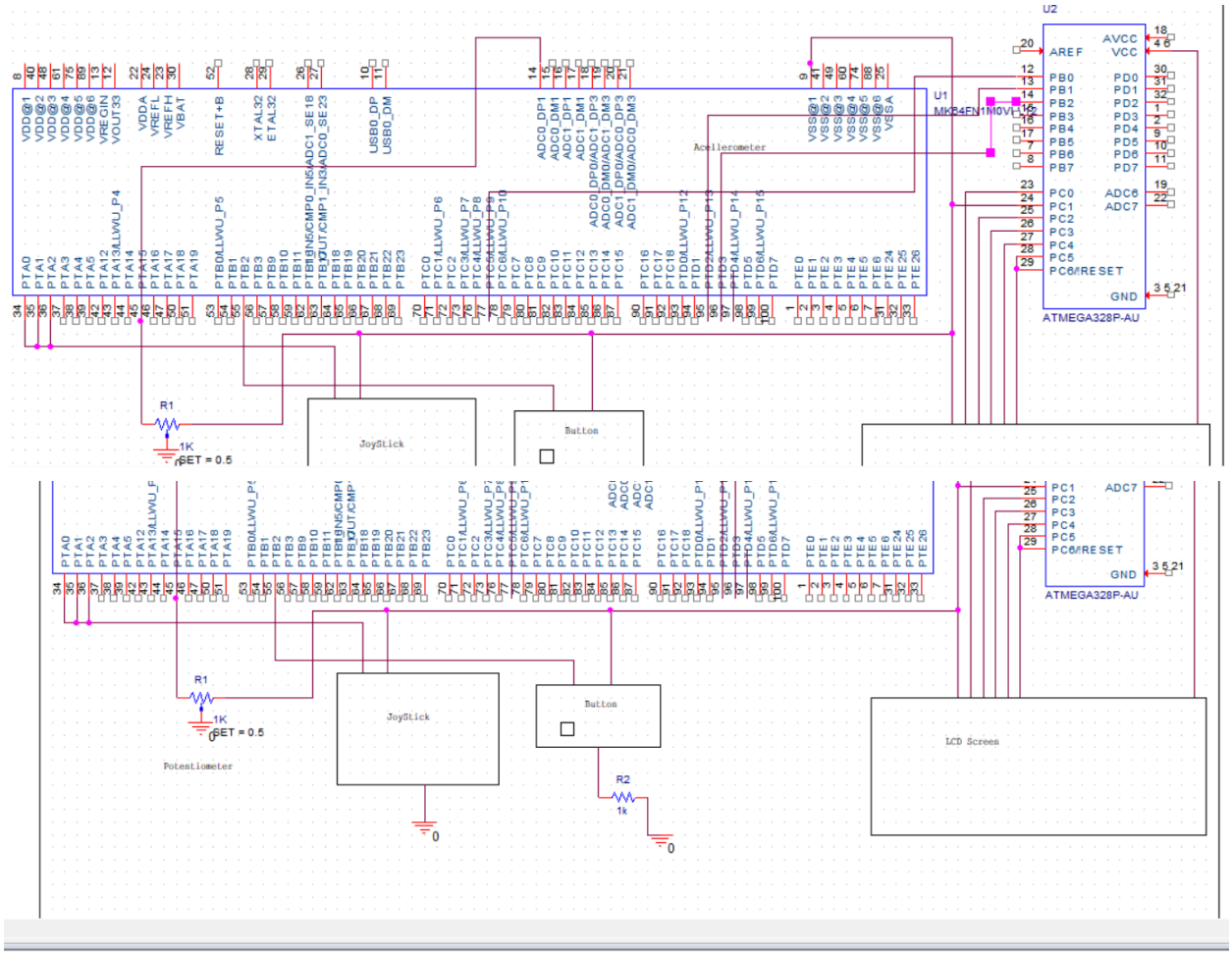
-Software Flowchart



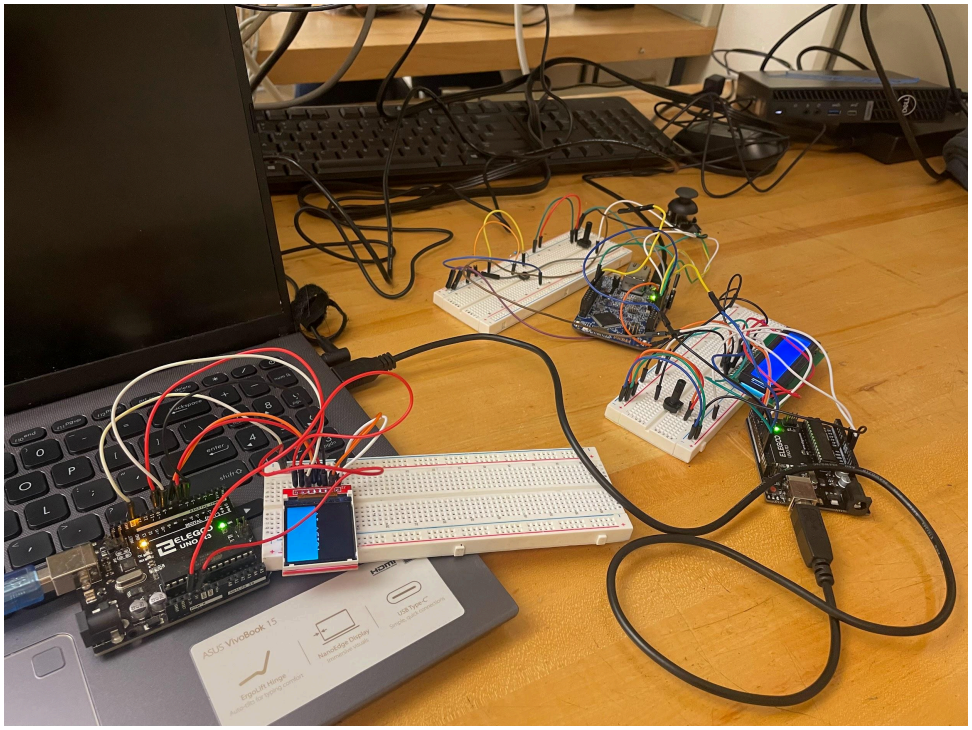


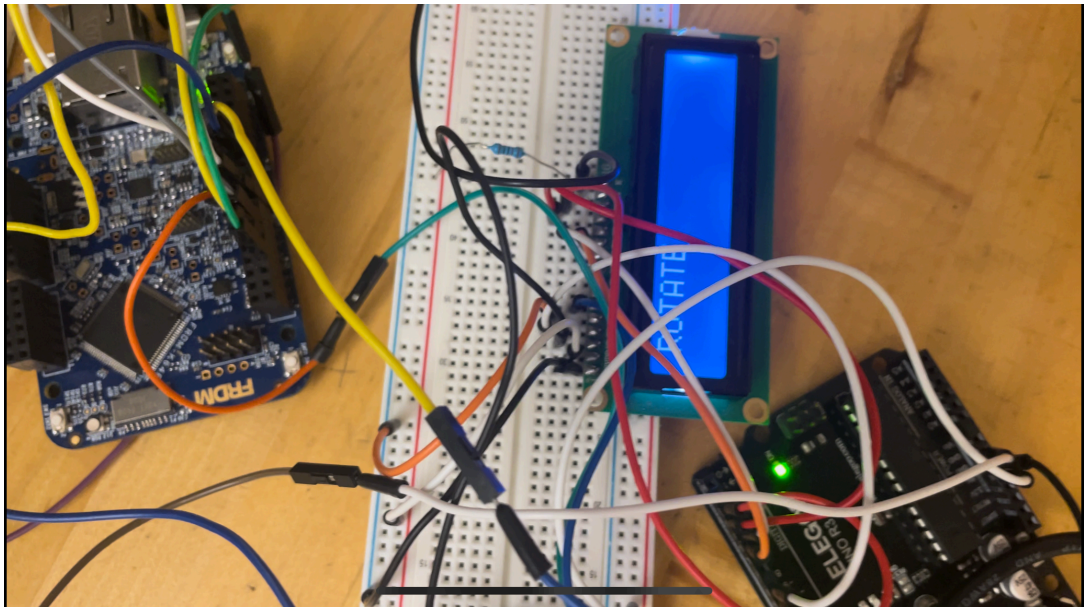
-Implementations Details

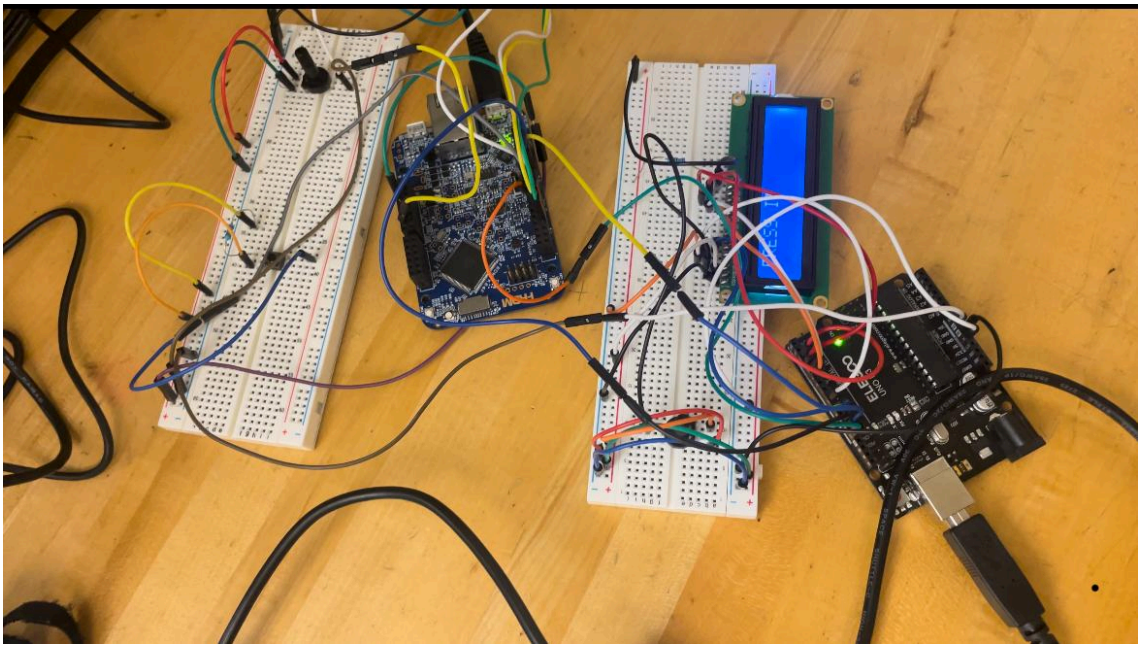
-Cadence schematic

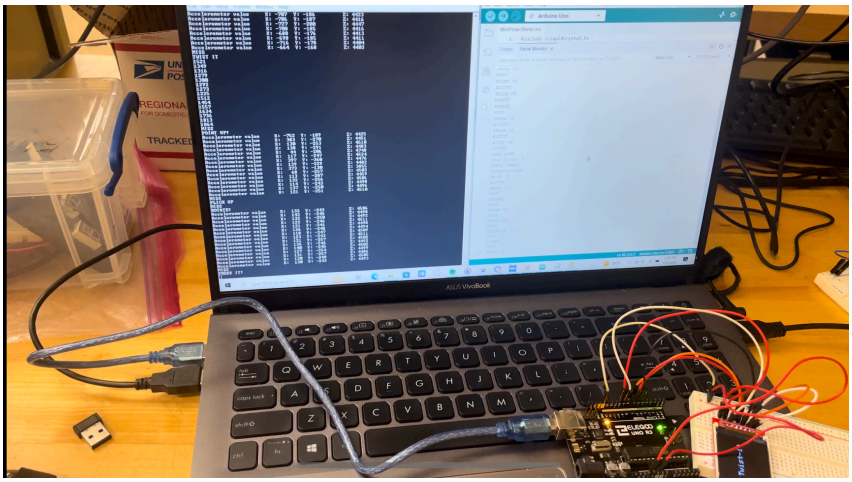


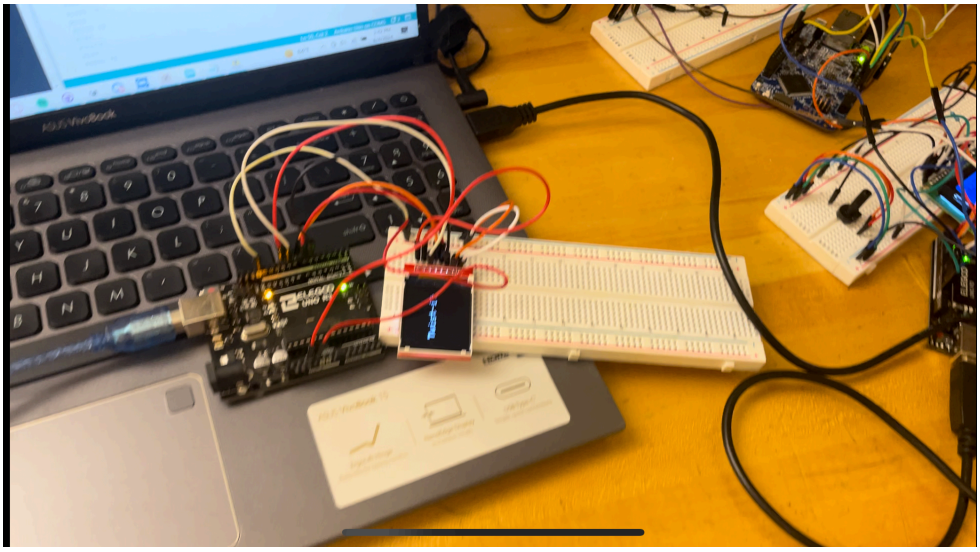
-Setup Photos











- Functional Code

Kinetic Code:

```
/*
###################################################################
**     Filename    : main.c
**     Project     : FinalProject
**     Processor   : MK64FN1M0VLL12
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2024-05-21, 14:41, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
**
###################################################################*/
/*!
** @file main.c
** @version 01.01
** @brief
**         Main module.
**         This module contains user's application code.
*/
/*!
**  @addtogroup main_module main module documentation
**  @{
*/
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "Pins1.h"
#include "CsIO1.h"
#include "IO1.h"
#include "FX1.h"
#include "GI2C1.h"
#include "WAIT1.h"
#include "MCUC1.h"
#include "CI2C1.h"
#include "SM1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
```

```
      #include "PE_Error.h"
      #include "PE_Const.h"
      #include "IO_Map.h"
      #include "PDD_Includes.h"
      #include "Init_Config.h"
      /* User includes (#include below this line is not maintained by
Processor Expert) */
      #include <stdbool.h>
      /*lint -save  -e970 Disable MISRA rule (6.3) checking. */
      int main(void)
      /*lint -restore Enable MISRA rule (6.3) checking. */
      {
        /* Write your local variable definition here */
        uint32_t inval = 0;
        uint32_t B1 = 0;
        uint32_t B2 = 0;
        uint32_t B3 = 0;
        bool modeFlag = 0;
        int actLen = 10;
        int actList [10] = {5, 1, 2, 5, 3, 4 , 2, 1, 1, 3};
        int countdown = 0;
        int actIndex = 0;
        int score = 0;

        unsigned char write[512];
        int len;
        LDD_TDeviceData *SM1_DeviceData;
        SM1_DeviceData = SM1_Init(NULL);

        /*** Processor Expert internal initialization. DON'T REMOVE THIS
CODE!!! ***/
        PE_low_level_init();
        /*** End of Processor Expert internal initialization.
***/

        /* Write your code here */
        SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; /*Enable Port A Clock Gate
Control*/
      //  SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; /*Enable Port B Clock Gate
Control*/
        SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK; /*Enable Port C Clock Gate
Control*/

        PORTA_GPCLR = 0x00030100; /* Configure pins 0  - 3 port A to GPIO
*/
      //  PORTB_GPCLR = 0x000C0100; /* Configure pins 2 and 3 port B to
GPIO */
```

```
        PORTC_GPCLR = 0x00040100; /* Configure pins 2 port C to GPIO */

        GPIOA_PDDR = 0x00000000; /* Configure for input*/
      //  GPIOB_PDDR = 0x00000000; /* Configure for input*/
        GPIOC_PDDR = 0x00000000; /* Configure for input*/

        SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK; // 0x8000000u; Enable ADC0 Clock
        ADC0_CFG1 = 0x0C; // 16bits ADC; Bus Clock
        ADC0_SC1A = 0x1F; // Disable the module, ADCH = 11111

        unsigned short ADC_read16b(void)
        {
            ADC0_SC1A = 0x00; //Write to SC1A to start conversion from
ADC_0
            while(ADC0_SC2 & ADC_SC2_ADACT_MASK); // Conversion in
progress
            while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); // Until conversion
complete
            return ADC0_RA;
        }

        void DisplayAction(int i) {
              switch (i) {
              case 1:
                    printf("PRESS IT!\n");
                    len = sprintf(write, "PRESS IT\n");
                    SM1_SendBlock(SM1_DeviceData, &write, len);
                    for (int i = 0; i < 100000; ++i);
                    break;
              case 2:
                    printf("ROTATE!\n");
                    len = sprintf(write, "ROTATE\n");
                    SM1_SendBlock(SM1_DeviceData, &write, len);
                    for (int i = 0; i < 100000; ++i);
                    break;
              case 3:
                    printf("POINT UP!\n");
                    len = sprintf(write, "POINT UP\n");
                    SM1_SendBlock(SM1_DeviceData, &write, len);
                    for (int i = 0; i < 100000; ++i);
                    break;
              case 4:
                    printf("FLICK UP\n");
                    len = sprintf(write, "FLICK UP\n");
                    SM1_SendBlock(SM1_DeviceData, &write, len);
                    for (int i = 0; i < 100000; ++i);
                    break;
```

```
		case 5:
			printf("TWIST IT\n");
			len = sprintf(write, "TWIST IT\n");
			SM1_SendBlock(SM1_DeviceData, &write, len);
			for (int i = 0; i < 100000; ++i);
			break;
		default:
			printf("Display function broken :(\n");
			break;
		}
	}

	uint32_t GetActionValue(int i) {
		uint32_t input = 0;
		int16_t accX, accY, accZ;
		unsigned short data = 0;


		switch (i) {
		case 1:
			input = GPIOC_PDIR & 0x04; // HIT Button
			return input & 0x4;
			break;

		case 2:
			accX = FX1_GetX();
			accY = FX1_GetY();
			accZ = FX1_GetZ();
			printf("Accelerometer value \tX: %4d\t Y: %4d\t Z:
%4d\n", accX, accY, accZ);
			if (accX > 500) { // Rotate Right;
				return 1;
			}
			else {
				return 0;
			}
			break;

		case 3:
			accX = FX1_GetX();
			accY = FX1_GetY();
			accZ = FX1_GetZ();
			printf("Accelerometer value \tX: %4d\t Y: %4d\t Z:
%4d\n", accX, accY, accZ);
			if (accY > 500) { // if point up, check accY  > 0;
				return 1;
			}
```

```
                else {
                        return 0;
                }
                break;

        case 4:
                input = GPIOA_PDIR & 0x2;
//              printf("%u\n", B3);
                if (input != 0) { // if point up, check accY  > 0;
                    return 0;
                }
                else {
                    return 1;
                }
                break;

        case 5:
                data = ADC_read16b();
                printf("%d\n", data);
                if (data > 10000) {
                        return 1;
                }
                else {
                        return 0;
                }
                break;

        default:
                printf("INPUT VALUE BROKEN\n");
                break;
        }
    }

    enum States {INIT, SETMODE, ACTION, MISS, HIT, OUTPUT} state =
INIT;

    void Tick() {
        // Actions
        switch (state) {
                case INIT:
                        // Do any initialization here
                        actIndex = 0;
                        score = 0;
                        countdown = 0;
                        break;

                case SETMODE:
```

```
                        // Get input from buttons
                        inval = GPIOC_PDIR & 0x04; // HIT Button
                        B1 = inval & 0x4;
                        break;

                  case ACTION:
                        if (actIndex >= actLen) break;
      //                printf("You are on action number: %d\n",
actIndex);
      //                len = sprintf(write, "ACTION\n");
      //                SM1_SendBlock(SM1_DeviceData, &write, len);
                        B3 = GetActionValue(actList[actIndex]);
                        ++countdown;
                        break;

                  case HIT:
                        ++actIndex;
                        if (actIndex < actLen)
DisplayAction(actList[actIndex]);
                        for (int i = 0; i < 10000; ++i);
                        ++score;
                        countdown = 0;
      //                printf("HIT STATE\n");
                        break;

                  case MISS:
                        countdown = 0;
                        ++actIndex;
                        if (actIndex < actLen)
DisplayAction(actList[actIndex]);
                        break;

                  case OUTPUT:
                        printf("||||||||||||||||\n");
                        printf("YOUR SCORE IS: %d\n", score);
                        printf("||||||||||||||||\n\n");
                        len = sprintf(write, "Your Score: %d\n", score);
                        SM1_SendBlock(SM1_DeviceData, &write, len);
                        for (int i = 0; i < 3000000; ++i);
                        countdown = 0;
                        actIndex = 0;
                        score = 0;
                        break;

                  default:
                        printf("BROKENNNNN\n");
                        break;
```

```
	}

	//Transitions
	switch(state) {
		case INIT:
			printf("PRESS BUTTON TO START GAME\n");
			len = sprintf(write, "PRESS START\n");
			SM1_SendBlock(SM1_DeviceData, &write, len);
			for (int i = 0; i < 10000; ++i);
			state = SETMODE;
			break;

		case SETMODE:
			if (B1 != 0) {
				printf("STARTING GAME\n\n");
				printf("||||||||||||||||||\n\n");
				len = sprintf(write, "STARTING GAME\n");
			    SM1_SendBlock(SM1_DeviceData, &write, len);
				for (int i = 0; i < 1000000; ++i);
				DisplayAction(actList[actIndex]);
				modeFlag = FALSE;
				inval = 0;
				B1 = 0;
				state = ACTION;
			}
			else {
//				printf("Waiting...\n");
				state = SETMODE;
			}
			break;

		case ACTION:
			if (countdown == 15) {
				state = MISS;
				printf("MISS\n");
				len = sprintf(write, "MISS\n");
				SM1_SendBlock(SM1_DeviceData, &write, len);
				for (int i = 0; i < 10000; ++i);
				break;
			}
			if (actIndex >= actLen) {
			    state = OUTPUT;
			    printf("GAME DONE\n\n");
			    len = sprintf(write, "GAME DONE\n");
			    SM1_SendBlock(SM1_DeviceData, &write, len);
			  for (int i = 0; i < 10000; ++i);
			  break;
```

```
                        }
                        if (B3 != 0) {
                              printf("HITTTTTTTTTTTT\n\n\n");
                              len = sprintf(write, "HITTTT\n");
                              SM1_SendBlock(SM1_DeviceData, &write, len);
                              for (int i = 0; i < 10000; ++i);
                              state = HIT;
                        }
                        else {
                              state = ACTION;
                        }
                        break;

                  case HIT:
                        state = ACTION;
                        break;

                  case MISS:
                        state = ACTION;
                        break;

                  case OUTPUT:
                        state = INIT;
                        break;

                  default:
                        break;
            }


      }

      while(1) {
            Tick();
            for(int delay = 0; delay < 1000000; delay++); //delay
      }
      /* For example: for(;;) { } */

      /*** Don't write any code pass this line, or it will be deleted
during code generation. ***/
      /*** RTOS startup code. Macro PEX_RTOS_START is defined by the
RTOS component. DON'T MODIFY THIS CODE!!! ***/
      #ifdef PEX_RTOS_START
        PEX_RTOS_START();                  /* Startup of the selected
RTOS. Macro is defined by the RTOS component. */
      #endif
      /*** End of RTOS startup code.  ***/
```

```
        /*** Processor Expert end of main routine. DON'T MODIFY THIS
CODE!!! ***/
        for(;;){}
        /*** Processor Expert end of main routine. DON'T WRITE CODE
BELOW!!! ***/
      } /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

      /* END main */
      /*!
      ** @}
      */
      /*
      **
##################################################################
      **
      **     This file was created by Processor Expert 10.5 [05.21]
      **     for the Freescale Kinetis series of microcontrollers.
      **
      **
##################################################################
      */
```

Arduino Connected Via SPI:

```
#include <LiquidCrystal.h>
#include <SPI.h>

const int rs = 8, en = 9, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
char buff[255];
volatile byte indx;
volatile boolean process;

void setup(void)
{
   Serial.begin(115200);
   pinMode(MISO, OUTPUT); // have to send on master in so it set as
output
   SPCR |= _BV(SPE);      // turn on SPI in slave mode
   indx = 0;              // buffer empty
   process = false;
   SPI.attachInterrupt(); // turn on interrupt
   lcd.begin(16,2);
//   lcd.print("Hello");
}
```

```
ISR(SPI_STC_vect) // SPI interrupt routine
{
   byte c = SPDR; // read byte from SPI Data Register

   if (indx < sizeof(buff))
   {
      buff[indx++] = c; // save data in the next index in the array
buff
      if (c == '\n')
      {
         buff[indx - 1] = 0; // replace newline ('\n') with end of
string (0)
         process = true;
      }
   }
}

void loop(void)
{
   if (process)
   {
      process = false;      // reset the process
      Serial.println(buff); // print the array on serial monitor
      lcd.clear();
      delay(100);
      lcd.setCursor(0,0);
      lcd.print(buff);
      delay(100);
      indx = 0;             // reset button to zero
   }

     }
```

Arduino with just Nokia:

```
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library for ST7735
#include <SPI.h>

// Pin configuration
#define sclk 10
#define mosi 9
#define cs   6
#define dc   8
#define rst  7
```

```
Adafruit_ST7735 tft = Adafruit_ST7735(cs, dc, mosi, sclk, rst); //
Hardware SPI

//const char* messages[] = {"Bop-It", "Twist-It", "Pull-It", "Shake-It"};
const char* messages[] = {"Twist-It", "Press-It", "Rotate-It", "Twist-it",
"Point-Up", "Flick-up", "Rotate", "Press-it", "Press-it", "Point-up"};
int currentMessageIndex = 0;  // Index to keep track of the current
message

void setup() {
  tft.initR(INITR_BLACKTAB);  // Initialize display
  tft.fillScreen(ST7735_BLACK); // Clear the screen to black
  tft.setRotation(1);  // Set rotation
  tft.setTextColor(ST7735_WHITE);  // Set the text color to white
  tft.setTextSize(2);  // Set text size for better visibility
}

void loop() {
  // Function to display the color animation
  displayRainbowLaser();    // Function to display rainbow laser animation
    delay(5000);  // Pause between cycles
  updateMessage();
  displayColorCircles();    // Function to display color circles animation
    delay(5000);  // Pause between cycles
  updateMessage();
  displayLaserLine();       // Function to display laser line animation
    delay(5000);  // Pause between cycles
  updateMessage();
  displayColorAnimation();
  delay(5000);  // Pause between cycles
  updateMessage();  // Update the message to be displayed next
}

void displayColorAnimation() {
  uint16_t colors[] = {ST7735_RED, ST7735_GREEN, ST7735_BLUE,
ST7735_YELLOW, ST7735_CYAN, ST7735_MAGENTA};
  int numColors = sizeof(colors) / sizeof(colors[0]);

  for (int i = 0; i < numColors; i++) {
```

```
    tft.fillScreen(colors[i]);
    displayText();  // Display the current text
    delay(1000);
  }
}

void displayRainbowLaser() {
  int height = tft.height();
  for (int i = 0; i < height; i++) {
    uint16_t color = tft.color565(i * 8, 255 - i * 8, 255);
    tft.drawFastVLine(tft.width() / 2, i, 1, color);
  }
  displayText();  // Display the current text
}

void displayColorCircles() {
  int radiusStep = 10;
  int maxRadius = min(tft.width(), tft.height()) / 2;
  uint16_t colors[] = {ST7735_RED, ST7735_GREEN, ST7735_BLUE, ST7735_CYAN,
ST7735_MAGENTA, ST7735_YELLOW};
  int numColors = sizeof(colors) / sizeof(colors[0]);
  int colorIndex = 0;

  for (int radius = maxRadius; radius > 0; radius -= radiusStep) {
    tft.drawCircle(tft.width() / 2, tft.height() / 2, radius,
colors[colorIndex]);
    colorIndex = (colorIndex + 1) % numColors;
  }
  displayText();  // Display the current text
}

void displayLaserLine() {
  int width = tft.width();
  for (int i = 0; i < width; i++) {
    uint16_t color = tft.color565(255, i * 8, 255 - i * 8);
    tft.drawFastHLine(i, tft.height() / 2, 1, color); // Draw horizontal
line
    delay(10);
  }
  displayText();  // Display the current text
```

```
}

void displayText() {
  tft.fillScreen(ST7735_BLACK);  // Clear the screen to black for clear visibility
  tft.setCursor(20, 50);  // Position where the text will be displayed
  tft.setTextColor(ST7735_WHITE, ST7735_BLACK);  // Set text color to white with black background
  tft.println(messages[currentMessageIndex]);  // Print the current message to the screen
}

void updateMessage() {
  currentMessageIndex = (currentMessageIndex + 1) % 4;  // Cycle through messages
}
```

-Testing/Evaluation Description

-test environment, required equipment, test scenarios

The test environment for our particular project would need to be using the physical project. Required equipment would need to be the project, a laptop with USB, the Kinetis Software, Arduino IDE software, and console display software such as Tera Term. Test scenarios that can be run for the project once connected properly to the laptop and running involve each of the possible peripherals used for interaction, the test of making sure the game starts, and the testing of the LCD display.

For the game start testing scenario, one would need to just run the program, and wait for the LCD or Tera Term to display “Press to Start Game”. Once displayed, press the button and the console and LCD should display “Starting Game”, followed by actions to be done. If any part of this test fails, then the test scenario has failed.

For the peripheral testing scenarios, there are 5 different actions to test: pushing the button, twisting the potentiometer, pushing the joystick up, rotating the accelerometer on its side, and pointing the accelerometer to the ceiling. To test each of these scenarios, it would require you to start the game by pressing the button. Depending on which action is being tested, wait for the corresponding message to display on the LCD and Tera Term. Once displayed, you can test if the system can properly read the sensor by doing the said action, such as turning the potentiometer forward and back for “Twist It” command. The test will succeed if “HIT” is displayed when doing the action and “MISS” is displayed when waiting for about 10 seconds without doing the correct action.

The LCD test scenario can be done by playing the game as normal and seeing if you are able to successfully play the game using only the LCD screen and that screen displays the correct score at the end for the amount of times “HIT” is displayed.

-Discussions

-Technical Problems/Challenges

Some technical problems that we ran into were both hardware and software. In terms of hardware we found later on during the final stages of our project that plugging in components into the arduino in pins too close to the SPI pins will ruin the spi connection. So this leads into the challenge, it was hard to output all the serial monitor stuff into the LCD screen as everytime we tried to output into the LCD screen it would straight up end all values in the serial monitor and not output anything, but using more spaced out pins fixed the connection problem. Another problem we had was the gamemodes for some reason the code was not reading the different gamemode button inputs and would not change the states, so we ended up just gutting that idea. In terms of software issues as mentioned earlier the serial monitor was not outputting into the arduino via spi so we had to change the hardware as mentioned earlier and also add in some delay on both sides, this helped solve our main problems. A challenge that we had was linking the FRDM and the Arduino together. Also another challenge was getting all the components to work properly using a majority of the FRDM board. The main main challenge was that Steven’s laptop had died a day before demo day so time was of the essence to us, due to this we had to spend the remaining time reimplementing the SPI onto my laptop and to redevelop all the code there too. Which resulted in us not being able to add in the NOKIA screen.

-Limitations

A big limitation that we had was that to play the game it was a little bit finicky as you are restricted to only using small arduino components all wired to the breadboard and and in addition the accelerometer on the FRDM board. Using these components made it very easy to pull out wires from the project when not used gently. Another limitation that we had was time, as mentioned earlier due to our main laptop being broken last minute we did not have enough time to add in more game modes onto our project and our NOKIA screen via SPI. Also the one gamemode that we have is limited to a single order of actions and is only 10 actions long.

-Possible Improvements

Possible improvements for this project would involve re-doing the joystick to use ADC rather than GPIO in order to get more flexibility in using the joystick instead of one direction at the moment. Another improvement would be to switch the use of the 16 by 2 LCD screen with the Nokia 5110 LCD screen for a more vibrant screen that would be

more visually appealing to look at. One last improvement would be to add a second button and add functionality for a second “Endurance” game mode that would end once the player misses an action.

-Responsibilities

Nathan Nguyen- My responsibility was to build the project overall. I added and wired all the action components all using the arduino components. Minus the Nokia screen. I also created the Nokia screen code and implemented some of the Arduino spi code. Also I set up my laptop for our project after Steven’s laptop died.

Steven Ryan Leonido- My responsibility was to create the code for the project. I used Processor Expert to initialize I2C (Accelerometer), UART (Serial Console), and SPI (Arduino Communication) pins to use for the project, as well as create the finite state machine code to control the logic. I also created code surrounding the peripheral sensor inputs like the GPIO buttons and joystick, the ADC for the potentiometer, and the I2C accelerometer inputs.

-Conclusion

In conclusion, our Bop-It Style game project was able to successfully implement a game that mimics some of the same actions as the original game, along with a new tweak using an accelerometer. We were able to implement 5 possible actions to do, along with using an LCD display to show actions to do in lieu of the speaker in the actual Bop-It game. Despite the technical issues with the LCD and a broken laptop, we were able to create a usable product that has quite a few ways to be expanded with using more visually appealing aesthetics like better LCD screen, or adding more functionality with more sensors and a new game mode. Overall we were able to complete the goals of the project by getting new hands on experience with a microcontroller in using new peripherals, design and develop a working prototype for a real-world problem, and implement said product for demonstration.