BOP BOP UPRISING

EE128 Group Project

EE128, Section 021
TA: Chinmay Raje

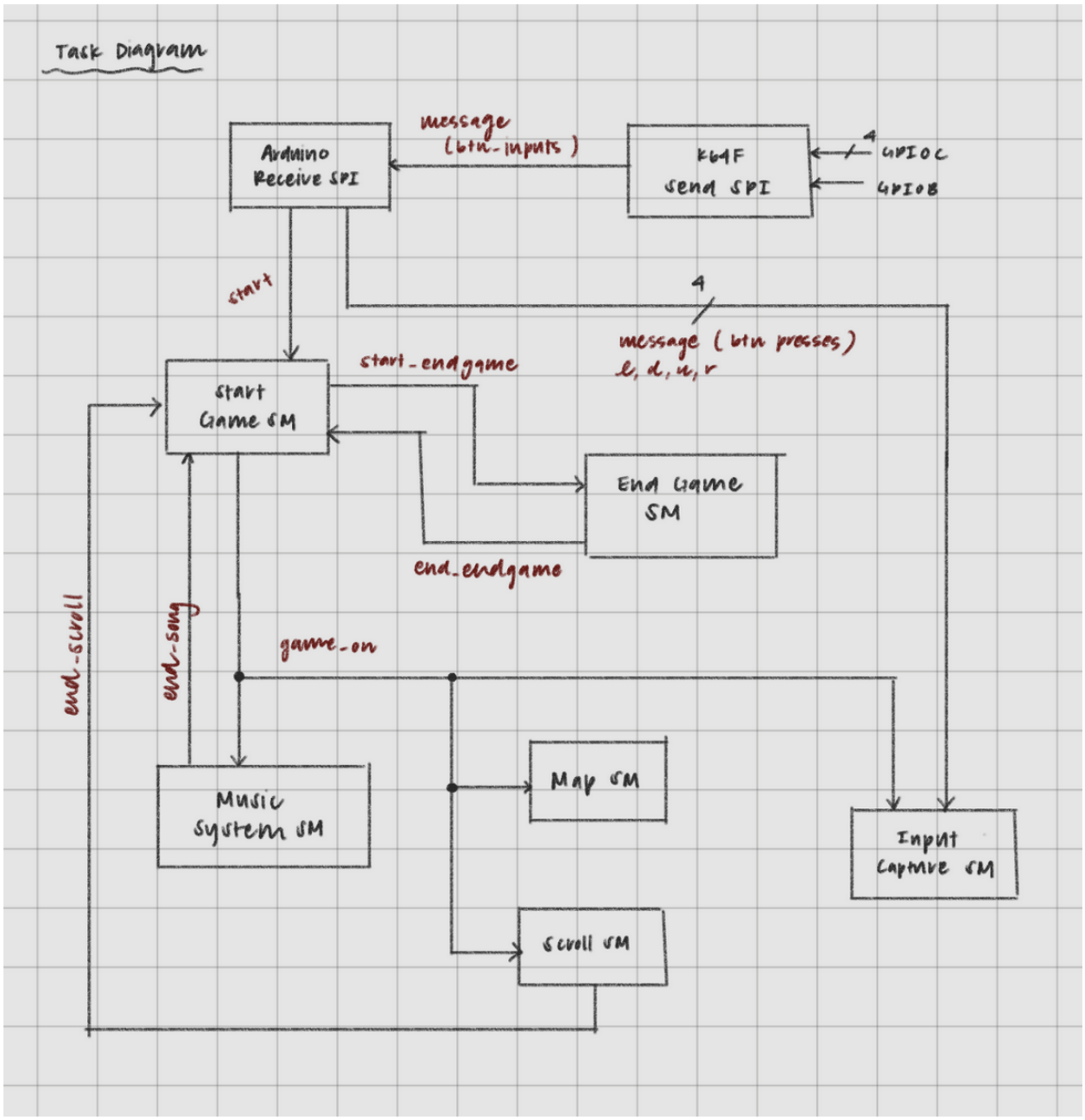
Kiana Dumdumaya
SID: 862199109
Vy Vo
SID: 862140774

Youtube Link
https://youtu.be/PtNjeA9cgvE

Project Description:

We created our own rendition of a rhythm game where a user must give particular inputs by pressing the correct buttons corresponding to the note tiles on screen. The user must time the presses with the tiles as they hit the white lines. Our must-have functionalities were to play a song of our choosing aloud, create a custom map/note sequence timed with the song, obtain user input and compare corresponding tiles pressed with the white line, and continuously keep track of score and accuracy of user input.

System Design:



F64F Send SPI

period = 10 ms (via timer interrupt)

LEFT = ~L & 0x08
UP = ~L & 0x04
DOWN = ~L & 0x02
RIGHT = ~L & 0x01

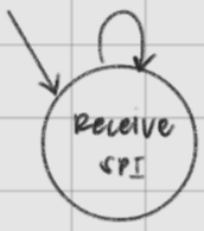
START = ~B & 0x04

SEND_MESSAGE = 0x00

```
if (LEFT)
    SEND_MESSAGE |= 0x80
else
    SEND_MESSAGE &= ~(0x80)

if (UP)
    SEND_MESSAGE |= 0x40
else
    SEND_MESSAGE &= ~(0x40)

if (DOWN)
    SEND_MESSAGE |= 0x20
else
    SEND_MESSAGE &= ~(0x20)

if (RIGHT)
    SEND_MESSAGE |= 0x10
else
    SEND_MESSAGE &= ~(0x10)

if (START)
    SEND_MESSAGE |= 0x01
else
    SEND_MESSAGE &= ~(0x01)

len = sprintf(write, "BINARY\n", SEND_MESSAGE)
sendBlock(SM1, &write, len)
```

Arduino Receive SPI

period = 1ms




```
if (process)
    process = false
    indx = 0

if (buf[0])
    e = true
else
    e = false

if (buf[1])
    d = true
else
    d = false
```

SPI ISR // interrupt routine to get message

```
if (buf[2])
    u = true
else
    u = false

if (buf[3])
    r = true
else
    r = false

if (buf[4])
    start = true
else
    start = false
```

Start Game SM

period = 1 ms

End Game Display SM

period = 1 ms

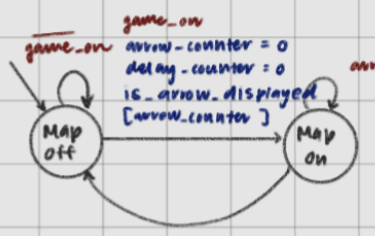
Music System SM

period : 1 ms


```
curr_time = millis()
if ( curr_time - prev_time >= note_duration)
    this_note += 2
    note_duration_counter ++
    calculate_duration( note_duration)

    prev_time = curr_time
else if ( curr_time - prev_time >= note_duration * 0.9
     &&  curr_time - prev_time < note_duration)
    tone( buzzer, melody [this_note]

else
    noTone (buzzer)


note_duration_counter ++
```

Map SM

period = 1 ms



game_on
arrow_counter = 0
delay_counter = 0
is_arrow_displayed [arrow_counter]

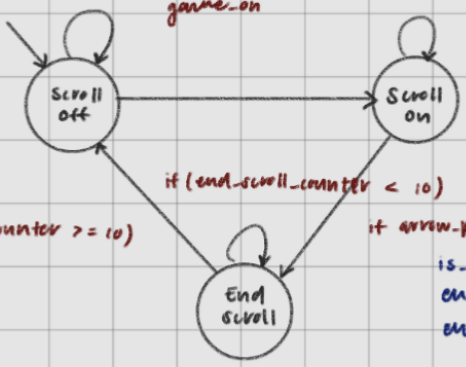
arrow_counter < map_sequence_size

if (delay_counter < 10)
    delay_counter ++
else
    delay_counter = 0
    arrow_counter ++
    is_arrow_displayed [arrow_counter] = true

arrow_counter >= map_sequence_size

Scroll Map SM

period = 1 ms



i >= arrow_position [map_sequence_size - 1] >= -8

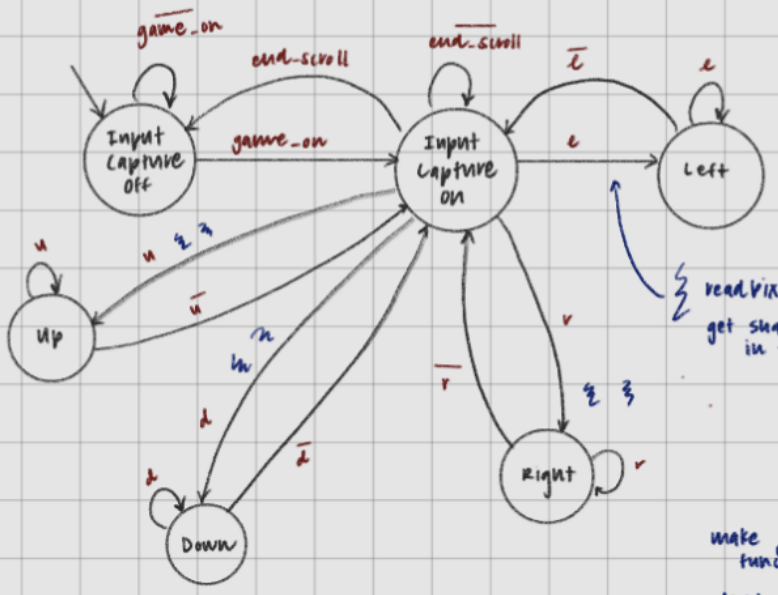
if (end_scroll_counter < 10)

if (end_scroll_counter >= 10)

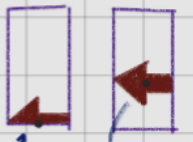
if arrow_position [map_sequence_size - 1] < 8
    is_arrow_displayed [size - 1] = false
    end_scroll = true
    end_scroll_counter = 0

end_scroll_counter ++

Input Capture SM period = 1 ms



readPixel (range for left arrow)
get shape input of the arrow in the range



make judgement function

BAD

GREAT

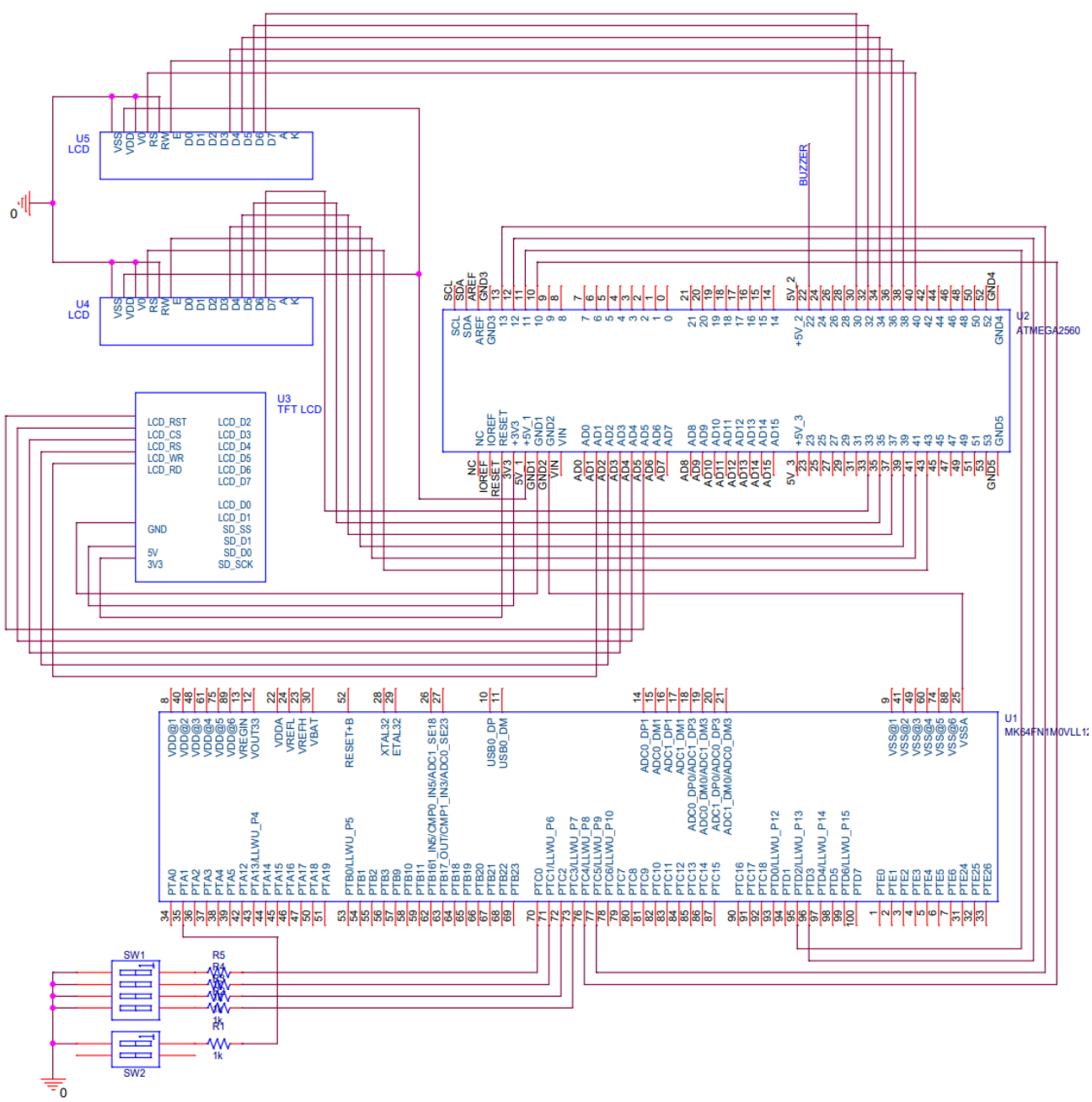
display judgement

```
switch (judgement)
    MISS
    BAD :
        combo = 0
        great_streak = 0
        perfect_streak = 0

    GOOD :
        score += 10
        combo = 0
        great_streak = 0
        perfect_streak = 0

    GREAT :
        combo ++
        great_streak ++
        if (combo >= 10)
            score += great_streak * 30
        else
            score += 30

    PERFECT :
        combo ++
        perfect_streak ++
        if (combo >= 10)
            score += perfect_streak * 50
        else
            score += 50
```

create a function calculate_score

display judgement
display score
display combo
update max combo

Implementation Details:



NOTE: We used processor expert for timer interrupt (ISR) and SPI communication to send messages to Arduino

K64F main.c

```
/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "Pins1.h"
#include "FX1.h"
#include "GI2C1.h"
#include "WAIT1.h"
#include "CI2C1.h"
#include "CsIO1.h"
#include "IO1.h"
#include "SM1.h"
#include "TU1.h"
#include "TI1.h"
#include "TimerIntLdd1.h"
#include "MCUC1.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) */

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */

/*library to configure GPIO*/
#include "MK64F12.h"

#define BYTE_TO_BINARY_PATTERN "%c%c%c%c%c%c%c%c"
#define BYTE_TO_BINARY(byte)  \
  (byte & 0x80 ? '1' : '0'), \
  (byte & 0x40 ? '1' : '0'), \
  (byte & 0x20 ? '1' : '0'), \
  (byte & 0x10 ? '1' : '0'), \
  (byte & 0x08 ? '1' : '0'), \
  (byte & 0x04 ? '1' : '0'), \
  (byte & 0x02 ? '1' : '0'), \
  (byte & 0x01 ? '1' : '0')


unsigned long delay = 300000;
void software_delay(unsigned long delay)
{
    while (delay > 0) delay--;
}

extern unsigned char write[512];
extern int len;
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
    /* Write your local variable definition here */
    /*Enable Clock Gating for PORTS A, B, C, D*/
    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*/
    SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK; /*Enable Port D Clock Gate Control*/

    /*Configure PORTS for GPIO*/
    PORTB_GPCLR = 0x00040100; /*Configure PORTB[2] for GPIO*/
    PORTC_GPCLR = 0x0FFF0100; /*Configure PORTC[11:0] for GPIO*/
    PORTD_GPCLR = 0x00FF0100; /*Configure PORTD[7:0] for GPIO*/

    PORTA_PCR1 = 0xA0100; /*Configures PORTA[1] for GPIO and to trigger interrupt on falling
edge */
    PORTB_PCR10 = 0x100; /*Configure PORTB[10] for GPIO and to generate clock*/
```

```
    /*PDDR: configures direction of PORT PINS for INPUT/OUTPUT     */
    GPIOA_PDDR = (0 << 1); /*Set PORTA[1] as INPUT*/
    GPIOB_PDDR = 0x0; /*Set PORTB[2] as INPUTS*/
    GPIOB_PDDR |= (1 << 10); /*Set PORTB[10] as OUTPUT*/
    GPIOC_PDDR = 0x00; /*Set PORTC[3:0] as INPUTS*/
    GPIOD_PDDR = 0xFF; /*Set PORTD[7:0] as OUTPUTS*/

    /*Initialize PORTS*/
    GPIOD_PDOR = 0x00; /*Initialize PORT D to 0*/

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

    //PORTA_ISFR = (1 << 1); /* Clear interrupt status flag register for PORTA[1]*/
    //NVIC_EnableIRQ(INT_PORTA);

    /*Initialization to send SPI message*/
    LDD_TDeviceData *SM1_DeviceData;
    SM1_DeviceData = SM1_Init(NULL);

    /* Write your code here */

    while(1)
    {
        software_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 10.4 [05.11]
 **     for the Freescale Kinetis series of microcontrollers.
 **
 ** ###################################################################
 */
```

K64F event.h

```
/* ###################################################################
**     Filename       : Events.h
**     Project        : SPI
**     Processor   : MK64FN1M0VLL12
**     Component   : Events
**     Version        : Driver 01.00
```

```
**      Compiler       : GNU C Compiler
**      Date/Time   : 2022-11-19, 17:28, # CodeGen: 0
**      Abstract       :
**      This is user's event module.
**      Put your event handler code here.
**      Contents       :
**      Cpu_OnNMI - void Cpu_OnNMI(void);
**
** ###################################################################*/
/*!
** @file Events.h
** @version 01.00
** @brief
**      This is user's event module.
**      Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/

#ifndef __Events_H
#define __Events_H
/* MODULE Events */

#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "Pins1.h"
#include "FX1.h"
#include "GI2C1.h"
#include "WAIT1.h"
#include "MCUC1.h"
#include "CI2C1.h"
#include "CsIO1.h"
#include "IO1.h"
#include "SM1.h"
#include "TU1.h"
#include "TI1.h"
#include "TimerIntLdd1.h"

#ifdef __cplusplus
extern "C" {
#endif

#define BYTE_TO_BINARY_PATTERN "%c%c%c%c%c%c%c%c"
#define BYTE_TO_BINARY(byte)  \
  (byte & 0x80 ? '1' : '0'), \
  (byte & 0x40 ? '1' : '0'), \
  (byte & 0x20 ? '1' : '0'), \
  (byte & 0x10 ? '1' : '0'), \
  (byte & 0x08 ? '1' : '0'), \
  (byte & 0x04 ? '1' : '0'), \
  (byte & 0x02 ? '1' : '0'), \
  (byte & 0x01 ? '1' : '0')


/*
** ===================================================================
**      Event          :  Cpu_OnNMI (module Events)
**
**      Component   :  Cpu [MK64FN1M0LL12]
*/
/*!
**      @brief
**      This event is called when the Non maskable interrupt had
**      occurred. This event is automatically enabled when the [NMI
**      interrupt] property is set to 'Enabled'.
*/
```

```
/* ===================================================================*/
void Cpu_OnNMI(void);


/*
** ===================================================================
**     Event          :  SM1_OnBlockSent (module Events)
**
**     Component   :  SM1 [SPIMaster_LDD]
*/
/*!
**     @brief
**     This event is called after the last character from the
**     output buffer is moved to the transmitter. This event is
**     available only if the SendBlock method is enabled.
**     @param
**     UserDataPtr    - Pointer to the user or
**                           RTOS specific data. The pointer is passed
**                           as the parameter of Init method.
*/
/* ===================================================================*/
void SM1_OnBlockSent(LDD_TUserData *UserDataPtr);

/*
** ===================================================================
**     Event          :  SM1_OnBlockReceived (module Events)
**
**     Component   :  SM1 [SPIMaster_LDD]
*/
/*!
**     @brief
**     This event is called when the requested number of data is
**     moved to the input buffer. This method is available only if
**     the ReceiveBlock method is enabled.
**     @param
**     UserDataPtr    - Pointer to the user or
**                           RTOS specific data. The pointer is passed
**                           as the parameter of Init method.
*/
/* ===================================================================*/
void SM1_OnBlockReceived(LDD_TUserData *UserDataPtr);

/*
** ===================================================================
**     Event          :  TI1_OnInterrupt (module Events)
**
**     Component   :  TI1 [TimerInt]
**     Description :
**     When a timer interrupt occurs this event is called (only
**     when the component is enabled - <Enable> and the events are
**     enabled - <EnableEvent>). This event is enabled only if a
**     <interrupt service/event> is enabled.
**     Parameters  : None
**     Returns        : Nothing
** ===================================================================
*/
void TI1_OnInterrupt(void);

/* END Events */

#ifdef __cplusplus
}  /* extern "C" */
#endif

#endif
/* ifndef __Events_H*/
/*!
** @}
*/
/
```

K64F event.c

```
/* ###################################################################
**     Filename    : Events.c
**     Project     : SPI
**     Processor   : MK64FN1M0VLL12
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2022-11-19, 17:28, # CodeGen: 0
**     Abstract    :
**     This is user's event module.
**     Put your event handler code here.
**     Contents    :
**     Cpu_OnNMI - void Cpu_OnNMI(void);
**
** ###################################################################*/
/*!
** @file Events.c
** @version 01.00
** @brief
**     This is user's event module.
**     Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"
#include "Init_Config.h"
#include "PDD_Includes.h"

#ifdef __cplusplus
extern "C" {
#endif


/* User includes (#include below this line is not maintained by Processor Expert) */

/*
** ===================================================================
**     Event          :  Cpu_OnNMI (module Events)
**
**     Component   :  Cpu [MK64FN1M0LL12]
*/
/*!
**     @brief
**     This event is called when the Non maskable interrupt had
**     occurred. This event is automatically enabled when the [NMI
**     interrupt] property is set to 'Enabled'.
*/
/* ===================================================================*/
void Cpu_OnNMI(void)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event          :  SM1_OnBlockSent (module Events)
**
**     Component   :  SM1 [SPIMaster_LDD]
*/
/*!
**     @brief
```

```
**      This event is called after the last character from the
**      output buffer is moved to the transmitter. This event is
**      available only if the SendBlock method is enabled.
**      @param
**      UserDataPtr    - Pointer to the user or
**                       RTOS specific data. The pointer is passed
**                       as the parameter of Init method.
*/
/* ===================================================================*/
void SM1_OnBlockSent(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**      Event          :  SM1_OnBlockReceived (module Events)
**
**      Component   :  SM1 [SPIMaster_LDD]
*/
/*!
**      @brief
**      This event is called when the requested number of data is
**      moved to the input buffer. This method is available only if
**      the ReceiveBlock method is enabled.
**      @param
**      UserDataPtr    - Pointer to the user or
**                       RTOS specific data. The pointer is passed
**                       as the parameter of Init method.
*/
/* ===================================================================*/
void SM1_OnBlockReceived(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
}


/*
** ===================================================================
**      Event          :  TI1_OnInterrupt (module Events)
**
**      Component   :  TI1 [TimerInt]
**      Description :
**      When a timer interrupt occurs this event is called (only
**      when the component is enabled - <Enable> and the events are
**      enabled - <EnableEvent>). This event is enabled only if a
**      <interrupt service/event> is enabled.
**      Parameters  : None
**      Returns        : Nothing
** ===================================================================
*/

unsigned char write[512];
int len;
void TI1_OnInterrupt(void)
{
  /* Write your code here ... */
    /*Initialization to send SPI message*/


        LDD_TDeviceData *SM1_DeviceData;
        SM1_DeviceData = SM1_Init(NULL);

        unsigned short LEFT = ~GPIOC_PDIR & 0x08;
        unsigned short UP = ~GPIOC_PDIR & 0x04;
        unsigned short DOWN = ~GPIOC_PDIR & 0x02;
        unsigned short RIGHT = ~GPIOC_PDIR & 0x01;

        unsigned short START = ~GPIOB_PDIR & 0x04;
```

```
        unsigned short LED_OUTPUT = 0x00;
        unsigned short SEND_MESSAGE = 0x00;

        //printf("Hello Arduino! \n");
        //len = sprintf(write, "Hello Arduino! Sending from K64F\n");
        //SM1_SendBlock(SM1_DeviceData, &write, len);

        if (LEFT == 0x08) {
               LED_OUTPUT |= 0x80;
               SEND_MESSAGE |= 0x80;
        }
        else {
               LED_OUTPUT &= ~(0x80);
               SEND_MESSAGE &= ~(0x80);
        }

        if (UP == 0x04) {
               LED_OUTPUT |= 0x40;
               SEND_MESSAGE |= 0x40;

        }
        else {
               LED_OUTPUT &= ~(0x40);
               SEND_MESSAGE &= ~(0x40);
        }

        if (DOWN == 0x02) {
               LED_OUTPUT |= 0x02;
               SEND_MESSAGE |= 0x20;

        }
        else {
               LED_OUTPUT &= ~(0x02);
               SEND_MESSAGE &= ~(0x20);
        }

        if (RIGHT == 0x01) {
               LED_OUTPUT |= 0x10;
               SEND_MESSAGE |= 0x10;

        }
        else {
               LED_OUTPUT &= ~(0x10);
               SEND_MESSAGE &= ~(0x10);
        }

        if (START == 0x04) {
               LED_OUTPUT |= 0x01;
               SEND_MESSAGE |= 0x08;

        }
        else {
               LED_OUTPUT &= ~(0x01);
               SEND_MESSAGE &= ~(0x08);
        }

        //printf("SEND MESSAGE\n");
        len = sprintf(write, ""BYTE_TO_BINARY_PATTERN"\n", BYTE_TO_BINARY(SEND_MESSAGE));
        SM1_SendBlock(SM1_DeviceData, &write, len);



        //PORT D OUTPUT
        GPIOD_PDOR = LED_OUTPUT;


        //PORTA_ISFR = (1 << 1); /* Clear ISFR*/

}
```

```
/* END Events */

#ifdef __cplusplus
}  /* extern "C" */
#endif
```

Testing/Evaluation:

Equipment:

- FRDM-K64F
- Arduino UNO
- TFT LCD Screen
- LCD Screen (x2)
- Buzzer
- Buttons (x5)
- Breadboard
- Resistors
- Jumper Wires

Test Environment: LED bar, LEDs, Serial Monitor

Test Scenarios:

(1) We tested button presses on K64F by connecting the LED bar and setting LEDs to be on for the corresponding button.
(2) We tested if the message is sent via SPI by printing the buf message on the Arduino side on the serial monitor.
(3) We tested each state machine on the arduino by looking at the display on the LCD display and TFT display to see that the correct view/text is displayed.
(4) We also tested each state machine by printing the state to the serial monitor to know that the system is in the correct state.
(5) We also used LEDs to show that we transitioned from one state to the next state correctly.
(6) We also used the serial monitor to see the counters increment or the incorrect index of an array in this case the map_sequence array and music array.

Discussions:

Challenges:

(1) *Arrow Display on TFT LCD Screen*: Originally, we wanted to use arrow shapes instead of a fixed line and rectangles. Unfortunately, due to the complicated shape, when the moving arrow passes the fixed arrow, the only way to refresh the fixed arrow was to reveal it row by row which caused noticeable delays in the display. Due to this, we opted for a fixed line and moving rectangles as through testing we found that there was little to no delay especially in comparison to arrows.

Limitations:

(1) *Multiple LCD Screens:* Though we wanted score, combo, and accuracy judgements to be all in the same screen as the map, there would have been too many elements on a single screen causing significant lag for the same reasons as Challenge (1) above.
(2) *Read-Only on Button Press:* Unlike any typical rhythm game, where if the user gives no input while the map plays, a "miss" will be detected, we do not have that functionality. Input is only taken when a button is pressed and then determined at that moment if the right button was pressed.

Possible Improvements:

(1) Detect "misses" even while buttons are not pressed
(2) Create an in-built leaderboard: most likely use eeprom to save scores
(3) More songs and map difficulties available
(4) Upgrade button setup and screen casings

Roles and Responsibilities:

Kiana designed the circuit schematics, wrote the Arduino code to create the start menu, gameplay display, and the end screen on the TFT LCD screen, and wrote the Arduino code for input capture of button presses to accuracy on the map. Vy designed the state machine diagrams, wrote the SPI and ISR code for the K64F, wrote the Arduino code to play a song through the buzzer, wrote the Arduino code to calculate scores (score multipliers and combos), and wrote the Arduino code to display score, judgment results, and combo on LCD screens. Both worked on the report, circuit hardware/wiring the board, tested their codes' implementation, and debugged their programs.

Conclusion:

In this personal project, we were able to apply the K64F to something that may be used in day-to-day life. By using SPI to transfer data between the K64F and Arduino Mega, we were able to use time interrupts (ISR) in the K64F to send a message to the Arduino to let the Arduino know if a button was pressed or not. We also use the Arduino Mega to make connections to multiple LCD displays and TFT Display to represent the rhythm game of pressing the corresponding button to each different note. In addition, we had to use the mega instead of the uno because the mega had more memory and more input/output pins to make connections to the TFT display, LCD displays, and SPI connection.

Appendices.

A1. Arduino Code

```
//====================== TASK SCHEDULER ======================
typedef struct task {
  int state;
  unsigned long period;
  unsigned long prev_time;
  int (*TickFct)(int );
} task;

const unsigned char task_num = 7;
const unsigned char period = 1;

task tasks[task_num]; // array of tasks

//====================== GLOBAL VARIABLES ======================
bool game_on = false;
bool start_endgame = false;
bool end_song = false;
bool end_scroll = false;
bool end_endgame = true;
bool end_game = false;

//scores
unsigned long score = 0;
unsigned long max_combo = 0;
//unsigned long current_combo = 0;
//unsigned long great_combo = 0;
//unsigned long perfect_combo = 0;

unsigned long total_miss = 0;
unsigned long total_bad = 0;
unsigned long total_great = 0;
unsigned long total_perfect = 0;


//====================== NOTE FREQUENCY ======================
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
```

```
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
#define REST 0

//====================== MUSIC SYSTEM SM VARIABLES ======================
// change this to make the song slower or faster
const char tempo = 114;

// change this to whichever pin you want to use
const char buzzer = 22;

// notes of the melody followed by the duration.
// a 4 means a quarter note, 8 an eighteenth , 16 sixteenth, so on
// !!negative numbers are used to represent dotted notes,
// so -4 means a dotted quarter note, that is, a quarter plus an eighteenth!!
int melody[] = {
  REST,4,       REST,4,       REST,4,       REST,4,       REST,4,       REST,4,       REST,4,
  REST,4,       REST,4,       REST,4,       REST,4,       REST,4,       REST,4,       REST,4,
  REST,4,       REST,4,       REST,4,       REST,4,

  REST,8,       NOTE_GS4,8,   NOTE_GS4,8,   NOTE_A4,8,    NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,4,
  REST,8,       NOTE_GS4,8,   NOTE_GS4,8,   NOTE_A4,8,    NOTE_B4,8,    NOTE_GS4,8,   NOTE_E4,4,
  REST,8,       NOTE_FS4,8,   NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_DS4,8,   NOTE_B3,4,
  NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,8,    NOTE_CS4,4,   REST,4,

  REST,4,       REST,4,       NOTE_GS5,8,   NOTE_CS5,8,   NOTE_GS5,8,   NOTE_CS5,16,  NOTE_GS5,16,  REST,16,      NOTE_CS5,16,
NOTE_GS5,8,   NOTE_CS5,8,   NOTE_GS5,8,
  NOTE_GS5,8,   NOTE_B4,8,    NOTE_GS5,8,   NOTE_B5,16,   NOTE_GS5,16,  REST,16,      NOTE_B4,16,   NOTE_GS5,8,   NOTE_B5,8,
NOTE_GS5,8,
  NOTE_FS5,8,   NOTE_B5,8,    NOTE_FS5,8,   NOTE_B5,16,   NOTE_FS5,16,  REST,16,      NOTE_B4,16,   NOTE_FS5,8,   NOTE_B5,8,
NOTE_FS5,8,
  NOTE_GS5,8,   NOTE_CS5,8,   NOTE_GS5,8,   NOTE_CS5,16,  NOTE_GS5,16,  REST,16,      NOTE_CS4,16,  NOTE_GS5,8,   NOTE_GS3,8,
NOTE_GS3,8,

  REST,4,       REST,4,       NOTE_CS4,-8,  NOTE_CS4,16,  NOTE_CS4,8,   NOTE_DS4,8,   NOTE_E4,4,    REST,4,
  REST,4,       NOTE_E4,-8,   NOTE_E4,16,   NOTE_E4,8,    NOTE_FS4,8,   NOTE_GS4,4,   REST,4,
  NOTE_FS4,4,   NOTE_FS4,8,   NOTE_DS4,8,   NOTE_B3,4,    NOTE_DS4,4,
  NOTE_E4,8,    NOTE_FS4,8,   NOTE_E4,8,    NOTE_DS4,8,   NOTE_CS4,4,   REST,4,

  NOTE_CS4,-8,  NOTE_CS4,16,  NOTE_CS4,8,   NOTE_DS4,8,   NOTE_E4,4,    REST,4,
  NOTE_DS4,16,  NOTE_E4,8,    NOTE_DS4,16,  NOTE_E4,8,    NOTE_FS4,8,   NOTE_GS4,4,   REST,4,
  NOTE_FS4,4,   NOTE_FS4,8,   NOTE_DS4,8,   NOTE_B3,4,    NOTE_DS4,4,
  NOTE_E4,8,    NOTE_FS4,8,   NOTE_E4,8,    NOTE_DS4,8,   NOTE_CS4,4,   REST,4,

  NOTE_GS4,4,   NOTE_GS4,8,   NOTE_B4,8,    NOTE_CS5,4,   NOTE_GS4,8,   NOTE_B4,8,
  NOTE_CS5,8,   NOTE_E5,8,    NOTE_CS5,8,   NOTE_B4,8,    NOTE_GS4,4,   REST,4,
  NOTE_FS4,4,   NOTE_FS4,8,   NOTE_GS4,8,    NOTE_B4,4,    NOTE_GS4,8,  NOTE_E4,8,
  NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,8,    NOTE_CS4,4,   REST, 4,

  NOTE_GS4,4,   NOTE_GS4,8,   NOTE_B4,8,    NOTE_CS5,4,   NOTE_GS4,8,   NOTE_B4,8,
  NOTE_CS5,8,   NOTE_E5,8,    NOTE_CS5,8,   NOTE_B4,8,    NOTE_GS4,4,   REST,4,
  NOTE_FS4,4,   NOTE_FS4,8,   NOTE_GS4,8,   NOTE_B4,4,    NOTE_GS4,8,   NOTE_E4,8,
  NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,8,    NOTE_CS4,4,   REST, 4,

  REST,4,       REST,4,       NOTE_GS5,8,   NOTE_CS5,8,   NOTE_GS5,8,   NOTE_CS5,16,  NOTE_GS5,16,  REST,16,      NOTE_CS5,16,
NOTE_GS5,8,   NOTE_CS5,8,   NOTE_GS5,8,
  NOTE_GS5,8,   NOTE_B4,8,    NOTE_GS5,8,   NOTE_B5,16,   NOTE_GS5,16,  REST,16,      NOTE_B4,16,   NOTE_GS5,8,   NOTE_B5,8,
NOTE_GS5,8,
  NOTE_FS5,8,   NOTE_B5,8,    NOTE_FS5,8,   NOTE_B5,16,   NOTE_FS5,16,  REST,16,      NOTE_B4,16,   NOTE_FS5,8,   NOTE_B5,8,
```

```
NOTE_FS5,8,
  NOTE_GS5,8,   NOTE_CS5,8,   NOTE_GS5,8,   NOTE_CS5,16,  NOTE_GS5,16,  REST,16,      NOTE_CS4,16,  NOTE_GS5,8,   NOTE_GS3,8,
NOTE_GS3,8,

  NOTE_CS4,-8,  NOTE_CS4,16,  NOTE_CS4,8,   NOTE_DS4,8,   NOTE_E4,4,    REST,4,
  NOTE_DS4,16,  NOTE_E4,8,    NOTE_DS4,16,  NOTE_E4,8,    NOTE_FS4,8,   NOTE_GS4,4,   REST,4,
  NOTE_FS4,4,   NOTE_FS4,8,   NOTE_DS4,8,   NOTE_B3,4,    NOTE_DS4,4,
  NOTE_E4,8,    NOTE_FS4,8,   NOTE_E4,8,    NOTE_DS4,8,   NOTE_CS4,4,   REST,4,

  NOTE_GS4,4,   NOTE_GS4,8,   NOTE_B4,8,    NOTE_CS5,4,   NOTE_GS4,8,   NOTE_B4,8,
  NOTE_CS5,8,   NOTE_E5,8,    NOTE_CS5,8,   NOTE_B4,8,    NOTE_GS4,4,   REST,4,
  NOTE_FS4,4,   NOTE_FS4,8,   NOTE_GS4,8,    NOTE_B4,4,    NOTE_GS4,8,  NOTE_E4,8,
  NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,8,    NOTE_CS4,4,   REST, 4,

  NOTE_GS4,4,   NOTE_GS4,8,   NOTE_B4,8,    NOTE_CS5,4,   NOTE_GS4,8,   NOTE_B4,8,
  NOTE_CS5,8,   NOTE_E5,8,    NOTE_CS5,8,   NOTE_B4,8,    NOTE_GS4,4,   REST,4,
  NOTE_FS4,4,   NOTE_FS4,8,   NOTE_GS4,8,   NOTE_B4,4,    NOTE_GS4,8,   NOTE_E4,8,
  NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,8,    NOTE_CS4,4,   REST, 4,

  REST,8,       NOTE_GS4,8,   NOTE_GS4,8,   NOTE_A4,8,    NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,4,
  REST,8,       NOTE_GS4,8,   NOTE_GS4,8,   NOTE_A4,8,    NOTE_B4,8,    NOTE_GS4,8,   NOTE_E4,4,
  REST,8,       NOTE_FS4,8,   NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_DS4,8,   NOTE_B3,4,
  NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,8,    NOTE_CS4,4,   REST,4,

  REST,8,       NOTE_GS4,8,   NOTE_GS4,8,   NOTE_A4,8,    NOTE_GS4,8,   NOTE_FS4,8,   NOTE_E4,4,
  REST,8,       NOTE_GS4,8,   NOTE_GS4,8,   NOTE_A4,8,    NOTE_B4,8,    NOTE_GS4,8,   NOTE_E4,4,
  REST,8,       NOTE_FS4,8,   NOTE_FS4,8,   NOTE_GS4,8,   NOTE_FS4,8,   NOTE_DS4,8,   NOTE_B3,4,
  NOTE_FS3,8,   NOTE_GS3,8,   NOTE_FS3,8,   NOTE_E3,8,    NOTE_CS3,4,   REST,4,

  REST,4,       REST,4,
};

// sizeof gives the number of bytes, each int value is composed of two bytes (16 bits)
// there are two values per note (pitch and duration), so for each note there are four bytes
int notes = sizeof(melody)/sizeof(melody[0])/2;

// this calculates the duration of a whole note in ms (60s/tempo)*4 beats
int wholenote = (60000 * 4) / tempo;
int divider = 0;

//====================== SPI VARIABLES ======================
#include <SPI.h>

//buff for received message from K64F
char buff [255];
volatile byte indx;
volatile boolean process;

//initialize button presses
bool l = false;
bool d = false;
bool u = false;
bool r = false;
bool start = false;

//====================== LCD SCREEN ======================
#include <LiquidCrystal.h>

// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs1 = 43, en1 = 41, d4_1 = 39, d5_1 = 37, d6_1 = 35, d7_1 = 33;
LiquidCrystal lcd1(rs1, en1, d4_1, d5_1, d6_1, d7_1);

const int rs2 = 42, en2 = 40, d4_2 = 38, d5_2 = 36, d6_2 = 34, d7_2 = 32;
LiquidCrystal lcd2(rs2, en2, d4_2, d5_2, d6_2, d7_2);

//====================== START GAME SM VARIABLES ======================
//button input
const char end_display_led_pin = 23;
const char game_led_pin = 25;
const char start_btn = 28;

//temprary
const char l_btn = 24;
const char d_btn = 26;
const char u_btn = 48;
const char r_btn = 46;

//====================== TFT DISPLAY ======================
#include <Elegoo_GFX.h>                 // Core graphics library
#include <Elegoo_TFTLCD.h>              // Hardware-specific library
#include <TouchScreen.h>                // Touch Support
//#include "Adafruit_GFX.h"
//#include "MCUFRIEND_kbv.h"

#define TS_MINX 920
#define TS_MINY 120
#define TS_MAXX 150
#define TS_MAXY 940
#define YP A3                           // must be an analog pin, use "An" notation!
#define XM A2                           // must be an analog pin, use "An" notation!
#define YM 9                            // can be a digital pin
#define XP 8                            // can be a digital pin
```

```
TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);

// macros for color (16 bit)
#define BLACK   0x0000
#define BLUE    0x001F
#define RED     0xF800
#define GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF

#define LCD_CS A3                          // Chip Select goes to Analog 3
#define LCD_CD A2                          // Command/Data goes to Analog 2
#define LCD_WR A1                          // LCD Write goes to Analog 1
#define LCD_RD A0                          // LCD Read goes to Analog 0
#define LCD_RESET A4                       // Can alternately just connect to Arduino's reset pin

Elegoo_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);

enum ARROWS{L, D, U, R} ARROW;
int map_sequence[] = {
  U, R, U, R, U, L, U, L,
  U, R, U, L, U, R, R, R,
  R, L, L, L, R, D, L, U,
  R, L, R, R, R, L, L, L,

  R, R, D, L, U, R, L, R,
  U, U, R, R, U, L, D, U,
  D, L, R, L, D, U, U, U,
  R, R, U, L, D, U, D, R,

  L, R, U, D, D, R, D, R,
  D, U, D, U, D, R, D, R,
  D, L, R, D, R, L, U, U,
  R, L, U, U, R, R, U, L,

  D, U, D, L, R, L, D, U,
  U, U, R, R, U, L, D, U,
  D, R, D, R, U, D, D, R,
  D, L, D, R, D, L, U, D,

  R, D, L, D, R, D, L, D,
  R, U, D, R, U, U, U, R

};

//delay between each arrow is 30 ms
int arrow_position[] = {
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,

  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,

  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,

  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320,

  320, 320, 320, 320, 320, 320, 320, 320,
  320, 320, 320, 320, 320, 320, 320, 320
};

bool is_arrow_displayed[] = {
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,

  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,

  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,

  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false,
```

```
  false, false, false, false, false, false, false, false,

  false, false, false, false, false, false, false, false,
  false, false, false, false, false, false, false, false
};

unsigned char map_sequence_size = sizeof(map_sequence)/sizeof(map_sequence[0]);

void line() {
  tft.drawFastHLine(0, 50, 240, WHITE);
  tft.drawFastHLine(0, 49, 240, WHITE);
}
// map components
void l_rect(int y) {                              // X O O O
  tft.fillRect(24, y - 1, 30, 1, BLACK);
  tft.fillRect(24, y, 30, 6, BLUE);
  tft.fillRect(24, y + 6, 30, 1, BLACK);

  if (y == 60) {line();}
  if (y == 59) {line();}
  if (y == 58) {line();}
  if (y == 57) {line();}
  if (y == 56) {line();}
  if (y == 55) {line();}
  if (y == 54) {line();}
  if (y == 53) {line();}
  if (y == 52) {line();}
  if (y == 51) {line();}
  if (y == 50) {line();}
  if (y == 49) {line();}
  if (y == 48) {line();}
  if (y == 47) {line();}
  if (y == 46) {line();}
  if (y == 45) {line();}
  if (y == 44) {line();}
  if (y == 43) {line();}
  if (y == 42) {line();}
  if (y == 41) {line();}
  if (y == 40) {line();}
  if (y == 39) {line();}
  if (y == 38) {line();}
}

void d_rect(int y) {                              // O X O O
  tft.fillRect(78, y - 1, 30, 1, BLACK);
  tft.fillRect(78, y, 30, 6, RED);
  tft.fillRect(78, y + 6, 30, 1, BLACK);

  if (y == 65) {line();}
  if (y == 64) {line();}
  if (y == 63) {line();}
  if (y == 62) {line();}
  if (y == 61) {line();}
  if (y == 60) {line();}
  if (y == 59) {line();}
  if (y == 58) {line();}
  if (y == 57) {line();}
  if (y == 56) {line();}
  if (y == 55) {line();}
  if (y == 54) {line();}
  if (y == 53) {line();}
  if (y == 52) {line();}
  if (y == 51) {line();}
  if (y == 50) {line();}
  if (y == 49) {line();}
  if (y == 48) {line();}
  if (y == 47) {line();}
  if (y == 46) {line();}
  if (y == 45) {line();}
  if (y == 44) {line();}
  if (y == 43) {line();}
  if (y == 42) {line();}
  if (y == 41) {line();}
  if (y == 40) {line();}
  if (y == 39) {line();}
  if (y == 38) {line();}
}

void u_rect(int y) {                              // O O X O
  tft.fillRect(132, y - 1, 30, 1, BLACK);
  tft.fillRect(132, y, 30, 6, GREEN);
  tft.fillRect(132, y + 6, 30, 1, BLACK);

  if (y == 60) {line();}
  if (y == 59) {line();}
  if (y == 58) {line();}
  if (y == 57) {line();}
  if (y == 56) {line();}
  if (y == 55) {line();}
  if (y == 54) {line();}
  if (y == 53) {line();}
  if (y == 52) {line();}
  if (y == 51) {line();}
```

```
  if (y == 50) {line();}
  if (y == 49) {line();}
  if (y == 48) {line();}
  if (y == 47) {line();}
  if (y == 46) {line();}
  if (y == 45) {line();}
  if (y == 44) {line();}
  if (y == 43) {line();}
  if (y == 42) {line();}
  if (y == 41) {line();}
  if (y == 40) {line();}
  if (y == 39) {line();}
  if (y == 38) {line();}
}

void r_rect(int y) {                              // O O O X
  tft.fillRect(186, y - 1, 30, 1, BLACK);
  tft.fillRect(186, y, 30, 6, MAGENTA);
  tft.fillRect(186, y + 6, 30, 1, BLACK);

  if (y == 60) {line();}
  if (y == 59) {line();}
  if (y == 58) {line();}
  if (y == 57) {line();}
  if (y == 56) {line();}
  if (y == 55) {line();}
  if (y == 54) {line();}
  if (y == 53) {line();}
  if (y == 52) {line();}
  if (y == 51) {line();}
  if (y == 50) {line();}
  if (y == 49) {line();}
  if (y == 48) {line();}
  if (y == 47) {line();}
  if (y == 46) {line();}
  if (y == 45) {line();}
  if (y == 44) {line();}
  if (y == 43) {line();}
  if (y == 42) {line();}
  if (y == 41) {line();}
  if (y == 40) {line();}
  if (y == 39) {line();}
  if (y == 38) {line();}
}

//====================== INPUT CAPTURE FUNCTIONS ======================
// blue, red, green, magenta
enum JUDGEMENTS {NONE, MISS, BAD, GREAT, PERFECT} JUDGEMENT;

int readPixelBlue() {
  // BLUE = 0x001F = 31
  // WHITE = 0xFFFF = 65535
  // BLACK = 0x0000 = 0
  uint16_t color = BLUE; //BLUE

  uint16_t miss3_top = tft.readPixel(39, 44);
  uint16_t miss2_top = tft.readPixel(39, 45);
  uint16_t miss1_top = tft.readPixel(39, 46);
  uint16_t bad_top = tft.readPixel(39, 47);
  uint16_t great_top = tft.readPixel(39, 48);
  uint16_t perfect_top = tft.readPixel(39, 49);
  uint16_t perfect_bot = tft.readPixel(39, 50);
  uint16_t great_bot = tft.readPixel(39, 51);
  uint16_t bad_bot = tft.readPixel(39, 52);
  uint16_t miss1_bot = tft.readPixel(39, 53);
  uint16_t miss2_bot = tft.readPixel(39, 54);
  uint16_t miss3_bot = tft.readPixel(39, 55);

  if (bad_top == color && bad_bot == color) {
    return PERFECT;
  }
  else if (great_top == color && miss1_bot == color && bad_top == BLACK) {
    return GREAT;
  }
  else if (great_bot == color && miss1_top == color && bad_bot == BLACK) {
    return GREAT;
  }
  else if (great_top == color && miss2_top == color && miss3_top == BLACK) {
    return BAD;
  }
  else if (great_bot == color && miss2_bot == color && miss3_bot == BLACK) {
    return BAD;
  }
  else {
    return MISS;
  }
}

int readPixelRed() {
  // RED = 0xF800 = 63488
  // WHITE = 0xFFFF = 65535
  // BLACK = 0x0000 = 0
  uint16_t color = RED;
```

```
  uint16_t miss3_top = tft.readPixel(93, 44);
  uint16_t miss2_top = tft.readPixel(93, 45);
  uint16_t miss1_top = tft.readPixel(93, 46);
  uint16_t bad_top = tft.readPixel(93, 47);
  uint16_t great_top = tft.readPixel(93, 48);
  uint16_t perfect_top = tft.readPixel(93, 49);
  uint16_t perfect_bot = tft.readPixel(93, 50);
  uint16_t great_bot = tft.readPixel(93, 51);
  uint16_t bad_bot = tft.readPixel(93, 52);
  uint16_t miss1_bot = tft.readPixel(93, 53);
  uint16_t miss2_bot = tft.readPixel(93, 54);
  uint16_t miss3_bot = tft.readPixel(93, 55);

  if (bad_top == color && bad_bot == color) {
    return PERFECT;
  }
  else if (great_top == color && miss1_bot == color && bad_top == BLACK) {
    return GREAT;
  }
  else if (great_bot == color && miss1_top == color && bad_bot == BLACK) {
    return GREAT;
  }
  else if (great_top == color && miss2_top == color && miss3_top == BLACK) {
    return BAD;
  }
  else if (great_bot == color && miss2_bot == color && miss3_bot == BLACK) {
    return BAD;
  }
  else {
    return MISS;
  }
}

int readPixelGreen() {
  // GREEN = 0x07E0 = 2016
  // WHITE = 0xFFFF = 65535
  // BLACK = 0x0000 = 0
  uint16_t color = GREEN;

  uint16_t miss3_top = tft.readPixel(147, 44);
  uint16_t miss2_top = tft.readPixel(147, 45);
  uint16_t miss1_top = tft.readPixel(147, 46);
  uint16_t bad_top = tft.readPixel(147, 47);
  uint16_t great_top = tft.readPixel(147, 48);
  uint16_t perfect_top = tft.readPixel(147, 49);
  uint16_t perfect_bot = tft.readPixel(147, 50);
  uint16_t great_bot = tft.readPixel(147, 51);
  uint16_t bad_bot = tft.readPixel(147, 52);
  uint16_t miss1_bot = tft.readPixel(147, 53);
  uint16_t miss2_bot = tft.readPixel(147, 54);
  uint16_t miss3_bot = tft.readPixel(147, 55);

  if (bad_top == color && bad_bot == color) {
    return PERFECT;
  }
  else if (great_top == color && miss1_bot == color && bad_top == BLACK) {
    return GREAT;
  }
  else if (great_bot == color && miss1_top == color && bad_bot == BLACK) {
    return GREAT;
  }
  else if (great_top == color && miss2_top == color && miss3_top == BLACK) {
    return BAD;
  }
  else if (great_bot == color && miss2_bot == color && miss3_bot == BLACK) {
    return BAD;
  }
  else {
    return MISS;
  }
}

int readPixelMagenta() {
  // MAGENTA = 0xF81F = 63519
  // WHITE = 0xFFFF = 65535
  // BLACK = 0x0000 = 0
  uint16_t color = MAGENTA;

  uint16_t miss3_top = tft.readPixel(201, 44);
  uint16_t miss2_top = tft.readPixel(201, 45);
  uint16_t miss1_top = tft.readPixel(201, 46);
  uint16_t bad_top = tft.readPixel(201, 47);
  uint16_t great_top = tft.readPixel(201, 48);
  uint16_t perfect_top = tft.readPixel(201, 49);
  uint16_t perfect_bot = tft.readPixel(201, 50);
  uint16_t great_bot = tft.readPixel(201, 51);
  uint16_t bad_bot = tft.readPixel(201, 52);
  uint16_t miss1_bot = tft.readPixel(201, 53);
  uint16_t miss2_bot = tft.readPixel(201, 54);
  uint16_t miss3_bot = tft.readPixel(201, 55);

  if (bad_top == color && bad_bot == color) {
```

```
    return PERFECT;
  }
  else if (great_top == color && miss1_bot == color && bad_top == BLACK) {
    return GREAT;
  }
  else if (great_bot == color && miss1_top == color && bad_bot == BLACK) {
    return GREAT;
  }
  else if (great_top == color && miss2_top == color && miss3_top == BLACK) {
    return BAD;
  }
  else if (great_bot == color && miss2_bot == color && miss3_bot == BLACK) {
    return BAD;
  }
  else {
    return MISS;
  }
}

//====================== SPI INTERRUPT ROUTINE ======================

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;
      }
    }
}

//====================== SPI SM ======================
//continuously print buff and assign message to correspondiing buttons presses from K64F

int SPISM_Tick(int state) {
  if (process) {
      process = false; //reset the process
      //Serial.println(buff); //print the array on serial monitor
      indx=0; //reset button to zero

   }

   if (buff[0] == '1') {
      //Serial.println("ARDUINO L");
      l = true;
    }
    else {
      l = false;
    }

    if (buff[1] == '1') {
      //Serial.println("ARDUINO D");
      d = true;
    }
    else {
      d = false;
    }

    if (buff[2] == '1') {
      //Serial.println("ARDUINO U");
      u = true;
    }
    else {
      u = false;
    }

    if (buff[3] == '1') {
      //Serial.println("ARDUINO R");
      r = true;
    }
    else {
      r = false;
    }

    if (buff[4] == '1') {
      //Serial.println("ARDUINO START");
      start = true;
    }
    else {
      start = false;
    }

  return 0;
}


//====================== START GAME SM ======================
//look for start btn press to start game
enum START_GAME_SM_STATES {SG_START, SG_GAME_OFF, SG_GAME_ON, SG_ENDGAME_DISPLAY} START_GAME_SM_STATE;
```

```
int StartGameSM_Tick(int state) {
  //get inputs
  //bool start = (digitalRead(start_btn) == LOW);
  static unsigned char flash = 0;

  //transition
  switch(state) {
    case SG_START:
      state = SG_GAME_OFF;
      game_on = false;
      start_endgame = false;

      //Print start menu
      lcd1.setCursor(0,0);
      lcd1.print("BOP BOP UPRISING");
      lcd1.setCursor(0, 1);
      lcd1.print("* * * * * * * *");

      lcd2.setCursor(2,0);
      lcd2.print("PRESS START");
      lcd2.setCursor(4, 1);
      lcd2.print("TO PLAY");

      tft.fillScreen(BLACK);
      tft.setRotation(2);

      tft.setTextColor(GREEN);
      tft.setTextSize(5);
      tft.setCursor(80, 60);
      tft.println("BOP");
      tft.setTextColor(BLUE);
      tft.setCursor(80, 100);
      tft.println("BOP");
      tft.setTextColor(RED);
      tft.setCursor(0, 140);
      tft.println("UPRISING");

      break;

    case SG_GAME_OFF:
      if (start) {
        state = SG_GAME_ON;
        game_on = true;

        // Print a message to the LCD1.
        lcd1.clear();
        lcd1.setCursor(0,0);
        lcd1.print("SCOREBOARD");
        lcd1.setCursor(0, 1);
        lcd1.print("SCORE:");

        // Print a message to the LCD2.
        lcd2.clear();
        lcd2.setCursor(0,0);
        lcd2.print("COMBO:100");
        lcd2.setCursor(0, 1);
        lcd2.print("PERFECT");

        tft.setRotation(0);
        tft.fillScreen(BLACK);
        tft.drawFastHLine(0, 49, 240, WHITE);
        tft.drawFastHLine(0, 50, 240, WHITE);

        //set score to 0
        score = 0;
        max_combo = 0;
        //current_combo = 0;
        //great_combo = 0;
        //perfect_combo = 0;

        total_miss = 0;
        total_bad = 0;
        total_great = 0;
        total_perfect = 0;

      }
      else {
        state = SG_GAME_OFF;

      }
      break;

    case SG_GAME_ON:
      if (end_song && end_scroll) {
        state = SG_ENDGAME_DISPLAY;
        start_endgame = true;
        game_on = false;

        // Print a message to the LCD1.
        lcd1.clear();
        lcd1.setCursor(0,0);
        lcd1.print("RESULTS");
        lcd1.setCursor(0, 1);
```

```
      lcd1.print("SCORE:");
      lcd1.setCursor(6,0);
      lcd1.print(score);

      // Print a message to the LCD2.
      lcd2.clear();
      lcd2.setCursor(0,0);
      lcd2.print("MAXCOMBO:");
      lcd2.setCursor(0, 1);
      lcd2.print(max_combo);

      //Print Results on TFT
      tft.fillScreen(BLACK);
      tft.setRotation(2);

      /*
      tft.setTextColor(WHITE);
      tft.setTextSize(4);
      tft.setCursor(40, 10);
      tft.println("RESULTS");

      tft.setTextSize(2);
      tft.setCursor(20, 80);
      tft.println("SCORE:");
      tft.setCursor(100, 80);
      tft.println(score, DEC);

      tft.setCursor(20, 110);
      tft.println("MAX COMBO:");
      tft.setCursor(145, 110);
      tft.println(max_combo, DEC);

      tft.setTextColor(BLUE);
      tft.setCursor(20, 170);
      tft.println("PERFECT:");
      tft.setCursor(125, 170);
      tft.println(total_perfect, DEC);

      tft.setTextColor(GREEN);
      tft.setCursor(20, 210);
      tft.println("GREAT:");
      tft.setCursor(100, 210);
      tft.println(total_great);

      tft.setTextColor(YELLOW);
      tft.setCursor(20, 250);
      tft.println("BAD:");
      tft.setCursor(80, 250);
      tft.println(total_bad, DEC);

      tft.setTextColor(RED);
      tft.setCursor(20, 290);
      tft.println("MISS:");
      tft.setCursor(90, 290);
      tft.println(total_miss, DEC); */

      tft.setTextColor(WHITE);
      tft.setTextSize(5);
      tft.setCursor(60, 120);
      tft.println("GAME");
      tft.setCursor(60, 160);
      tft.println("OVER");


    }
    else {
      state = SG_GAME_ON;
    }
    break;

  case SG_ENDGAME_DISPLAY:
    if (end_endgame) {
      state = SG_GAME_OFF;
      end_game = true;
      start_endgame = false;

      //reset game
      //Print start menu
      lcd1.clear();
      lcd1.setCursor(0,0);
      lcd1.print("BOP BOP UPRISING");
      lcd1.setCursor(0, 1);
      lcd1.print("* * * * * * * *");

      lcd2.clear();
      lcd2.setCursor(2,0);
      lcd2.print("PRESS START");
      lcd2.setCursor(4, 1);
      lcd2.print("TO PLAY");

      //Print start menu
      lcd1.setCursor(0,0);
      lcd1.print("BOP BOP UPRISING");
```

```
        lcd1.setCursor(0, 1);
        lcd1.print("* * * * * * * *");

        lcd2.setCursor(2,0);
        lcd2.print("PRESS START");
        lcd2.setCursor(4, 1);
        lcd2.print("TO PLAY");

        tft.fillScreen(BLACK);
        tft.setRotation(2);

        tft.setTextColor(GREEN);
        tft.setTextSize(5);
        tft.setCursor(80, 60);
        tft.println("BOP");
        tft.setTextColor(BLUE);
        tft.setCursor(80, 100);
        tft.println("BOP");
        tft.setTextColor(RED);
        tft.setCursor(0, 140);
        tft.println("UPRISING");

        unsigned char i;
        for (i=0; i<map_sequence_size;i++) {
          arrow_position[i] = 320;
        }
      }
      else {
        state = SG_ENDGAME_DISPLAY;
      }
      break;

    default:
      break;
  }

  //action
  switch(state) {
    case SG_GAME_OFF:
      //Serial.println("SG GAME OFF");
      digitalWrite(game_led_pin, LOW);
      digitalWrite(end_display_led_pin, LOW);

      if (flash < 8) {
        tft.setTextColor(WHITE);
        tft.setTextSize(1);
        tft.setCursor(40, 240);
        tft.println("PRESS START BUTTON TO PLAY");
      }
      else if (flash >= 8 && flash < 16){
        tft.fillRect(40, 240, 240, 20, BLACK);
      }else {
        flash = 0;
      }

      flash++;

      break;

    case SG_GAME_ON:
      //Serial.println("SG GAME ON");
      digitalWrite(game_led_pin, HIGH);
      digitalWrite(end_display_led_pin, LOW);

      break;

    case SG_ENDGAME_DISPLAY:
      //Serial.println("SG ENDGAME DISPLAY");

      digitalWrite(game_led_pin, LOW);
      digitalWrite(end_display_led_pin, HIGH);

      break;

    default:
      break;

  }

  return state;
}

//====================== MUSIC SYSTEM SM ======================
//play butterfly song when game is on
enum MUSIC_SYSTEM_SM_STATES {MS_START, MS_MUSIC_OFF, MS_MUSIC_ON, MS_END_SONG} MUSIC_SYSTEM_SM_STATE;

int MusicSystemSM_Tick(int state) {

  static unsigned short this_note = 0;
  static unsigned short note_duration = 0;
  static unsigned short note_duration_counter = 0;

  static unsigned long curr_time = 0;
```

```
static unsigned long prev_time = 0;

static unsigned short end_song_counter = 0;

//transition
switch(state) {
  case MS_START:
    state = MS_MUSIC_OFF;
    end_song = true;
    break;

  case MS_MUSIC_OFF:
    if (game_on) {
      state = MS_MUSIC_ON;
      end_song = false;
      this_note = 0;
      note_duration_counter = 0;

      // calculates the duration of first note
      divider = melody[this_note + 1];

      if (divider > 0) {
        // regular note, just proceed
        note_duration = (wholenote) / divider;
      }
      else if (divider < 0) {
        // dotted notes are represented with negative durations!!
        note_duration = (wholenote) / abs(divider);
        note_duration *= 1.5; // increases the duration in half for dotted notes
      }

      prev_time = millis();
    }
    else {
      state = MS_MUSIC_OFF;
    }
    break;

  case MS_MUSIC_ON:
    if (this_note < notes * 2) {
      state = MS_MUSIC_ON;
    }
    else if (this_note >= notes * 2) {
      state = MS_END_SONG;
      end_song = true;
      end_song_counter = 0;
    }

    //Serial.print("this note: " );
    //Serial.println(this_note);

    //Serial.print("notes size: " );
    //Serial.println(notes * 2);
    break;

  case MS_END_SONG:
    if (end_song_counter < 10) { //end_song is 1 for 10 ms -> so it is read by MS system
      state = MS_END_SONG;
    }
    else {
      state = MS_MUSIC_OFF;
      end_song = false;
    }
    break;

  default:
    break;
}

//action
switch(state) {
  case MS_MUSIC_OFF:
    //Serial.println("MS MUSIC OFF");

    break;

  case MS_MUSIC_ON:
    //Serial.println("MS MUSIC ON");

    curr_time = millis();
    if (curr_time - prev_time >= note_duration) {
      // stop the waveform generation before the next note.
      this_note += 2;

      //reset note_duration counter
      note_duration_counter = 0;

      // calculates the duration of next note
      divider = melody[this_note + 1];

      if (divider > 0) {
        // regular note, just proceed
        note_duration = (wholenote) / divider;
```

```
        }
        else if (divider < 0) {
          // dotted notes are represented with negative durations!!
          note_duration = (wholenote) / abs(divider);
          note_duration *= 1.5; // increases the duration in half for dotted notes
        }

        prev_time = curr_time;
      }
      else if(curr_time - prev_time >= note_duration*0.9 && curr_time - prev_time < note_duration) {
        noTone(buzzer);
        note_duration_counter++;
      }
      else if(curr_time - prev_time < note_duration*0.9){
        if (melody[this_note] != REST) {
          tone(buzzer, melody[this_note]);
        }
        else {
          noTone(buzzer);
        }
        note_duration_counter++;
      }

      break;

    case MS_END_SONG:
      //Serial.println("MS END SONG");

      end_song_counter++;
      //Serial.print("end_song_counter: ");
      //Serial.println(end_song_counter);
      break;

    default:
      break;
  }

  return state;

}

//====================== MAP SM ======================
//display next arrow once it is the arrow's turn to be displayed on the 320x240 screen
enum MAP_STATES{M_START, M_MAP_OFF, M_MAP_ON} MAP_STATE;

int MapSM_Tick(int state) {

  static unsigned short arrow_counter = 0;
  static unsigned short delay_counter = 0;

  //transition
  switch(state) {
    case M_START:
      state = M_MAP_OFF;
      break;

    case M_MAP_OFF:
      if (game_on) {
        state = M_MAP_ON;
        static unsigned short arrow_counter = 0;
        static unsigned short delay_counter = 0;
        is_arrow_displayed[arrow_counter] = true;
      }
      else {
        state = M_MAP_OFF;
      }
      break;

    case M_MAP_ON:
      if(arrow_counter < map_sequence_size) {
        state = M_MAP_ON;
      }
      else {
        state = M_MAP_OFF;
      }
      break;

    default:
      break;
  }

  //action
  switch(state) {
    case M_MAP_OFF:
      //Serial.println("M MAP OFF");
      break;

    case M_MAP_ON:
      //Serial.println("M MAP ON");
      //display a new arrow every 10 ms
      if(delay_counter < 10) {
        delay_counter++;
      }
```

```
      else {
        delay_counter = 0;
        arrow_counter++;
        is_arrow_displayed[arrow_counter] = true;
      }
      break;

    default:
      break;
  }

  return state;
}

//====================== SCROLL MAP SM ======================
//display next arrow once it is the arrow's turn to be displayed on the 320x240 screen
enum SCROLL_MAP_STATES{SM_START, SM_SCROLL_OFF, SM_SCROLL_ON, SM_END_SCROLL} SCROLL_MAP_STATE;

int ScrollMapSM_Tick(int state) {
  static unsigned short end_scroll_counter = 0;
  //transition
  switch(state) {
    case SM_START:
      state = SM_SCROLL_OFF;
      end_scroll = true;
      break;

    case SM_SCROLL_OFF:
      if(game_on) {
        state = SM_SCROLL_ON;
        end_scroll = false;
      }
      else {
        state = SM_SCROLL_OFF;
      }
      break;

    case SM_SCROLL_ON:
      if (arrow_position[map_sequence_size-1] >= -8) {
        state = SM_SCROLL_ON;
      }
      else {
        state = SM_END_SCROLL;
        end_scroll = true;
        end_scroll_counter = 0;
      }

      break;

    case SM_END_SCROLL:
      if (end_scroll_counter < 10) {
        state = SM_END_SCROLL;
      }
      else {
        state = SM_SCROLL_OFF;
        end_scroll = false;
      }

      break;

    default:
      break;
  }

  //action
  switch(state) {
    case SM_START:
      break;

    case SM_SCROLL_OFF:
      //Serial.println("SM SCROLL OFF");
      break;

    case SM_SCROLL_ON:
      //Serial.println("SM SCROLL ON");

      unsigned short i;
      for (i=0; i<map_sequence_size; i++) {
        if(is_arrow_displayed[i] == true) {
          if (map_sequence[i] == L) {
            l_rect(arrow_position[i]);
          }
          else if (map_sequence[i] == D) {
            d_rect(arrow_position[i]);
          }
          else if (map_sequence[i] == U) {
            u_rect(arrow_position[i]);
           }
          else if (map_sequence[i] == R) {
            r_rect(arrow_position[i]);
          }

          arrow_position[i]--; //decrement arrow position after displaying it
```

```
          //arrow does not need to be displayed anymore once the position reaches 0
          if (arrow_position[i] <= -8) {
            is_arrow_displayed[i] = false;
          }
        }
      }
      break;

    case SM_END_SCROLL:
      //Serial.println("SM END SCROLL");

      end_scroll_counter++;
      //Serial.print("end_scroll_counter: ");
      //Serial.println(end_scroll_counter);

      break;

    default:
      break;
  }

 return state;
}

//====================== INPUT CAPTURE SM ======================
enum INPUT_CAPTURE_SM_STATES{IC_START, IC_INPUT_CAPTURE_OFF, IC_INPUT_CAPTURE_ON, IC_LEFT, IC_DOWN, IC_UP, IC_RIGHT}
INPUT_CAPTURE_STATE;

int InputCaptureSM_Tick(int state) {
  /*
  bool l = (digitalRead(l_btn) == LOW);
  bool d = (digitalRead(d_btn) == LOW);
  bool u = (digitalRead(u_btn) == LOW);
  bool r = (digitalRead(r_btn) == LOW);
  */

  static int judgement = NONE;

  static unsigned long ic_score = 0;
  static unsigned long ic_max_combo = 0;
  static unsigned long ic_current_combo = 0;
  static unsigned long ic_great_combo = 0;
  static unsigned long ic_perfect_combo = 0;

  static unsigned long ic_total_miss = 0;
  static unsigned long ic_total_bad = 0;
  static unsigned long ic_total_great = 0;
  static unsigned long ic_total_perfect = 0;

  //transitions
  switch(state) {
    case IC_START:
      state = IC_INPUT_CAPTURE_OFF;
      break;

    case IC_INPUT_CAPTURE_OFF:
      if (game_on) {
        state = IC_INPUT_CAPTURE_ON;

        ic_score = 0;
        ic_max_combo = 0;
        ic_current_combo = 0;
        ic_great_combo = 0;
        ic_perfect_combo = 0;

        ic_total_miss = 0;
        ic_total_bad = 0;
        ic_total_great = 0;
        ic_total_perfect = 0;
      }
      else {
        state = IC_INPUT_CAPTURE_OFF;
      }
      break;

    case IC_INPUT_CAPTURE_ON:
      if (!game_on) {
        state = IC_INPUT_CAPTURE_OFF;

        //end of map

        //update max combo
        if (ic_current_combo > ic_max_combo) {
          ic_max_combo = ic_current_combo;
        }

        score = ic_score;
        max_combo = ic_max_combo;
        total_miss = ic_total_miss;
        total_bad = ic_total_bad;
        total_great = ic_total_great;
        total_perfect = ic_total_perfect;
```

```
  Serial.println("\nRESULTS:");

  Serial.print("SCORE:");
  Serial.println(score);

  Serial.print("MAX COMBO:");
  Serial.println(max_combo);

  Serial.print("PERFECT:");
  Serial.println(total_perfect);

  Serial.print("GREAT:");
  Serial.println(total_great);

  Serial.print("BAD:");
  Serial.println(total_bad);

  Serial.print("MISS:");
  Serial.println(total_miss);
  Serial.println();

}
else { //if game on -> press button keys to play
  if (l) {
    state = IC_LEFT;
    judgement = readPixelBlue();
  }
  else if (d) {
    state = IC_DOWN;
    judgement = readPixelRed();
  }
  else if(u) {
    state = IC_UP;
    judgement = readPixelGreen();
  }
  else if(r) {
    state = IC_RIGHT;
    judgement = readPixelMagenta();
  }
  else if (!l && !d && !u && !r) { //if no buttons are pressed
    state = IC_INPUT_CAPTURE_ON;
    judgement = NONE;
  }
}

//display judgement
lcd2.clear();
lcd2.setCursor(0, 1);

if (judgement == MISS) {
  lcd2.print("MISS");
  Serial.println("MISS");
}
else if (judgement == BAD) {
  lcd2.print("BAD");
  Serial.println("BAD");

}
else if (judgement == GREAT) {
  lcd2.print("GREAT");
  Serial.println("GREAT");

}
else if (judgement == PERFECT) {
  lcd2.print("PERFECT");
  Serial.println("PERFECT");
}

switch(judgement) {
  case MISS:
    ic_total_miss++;

    //update max combo and then reset current combo
    if (ic_current_combo > ic_max_combo) {
      ic_max_combo = ic_current_combo;
    }

    ic_current_combo = 0;
    ic_great_combo = 0;
    ic_perfect_combo = 0;

    break;

  case BAD:
    ic_total_bad++;

    ic_score += 10;

    //update max combo and then reset current combo
    if (ic_current_combo > ic_max_combo) {
      ic_max_combo = ic_current_combo;
    }
```

```
        ic_current_combo = 0;
        ic_great_combo = 0;
        ic_perfect_combo = 0;

        break;

      case GREAT:
        ic_total_great++;
        ic_current_combo++;
        ic_great_combo++;

        if(ic_current_combo >= 3) {
          ic_score += (ic_great_combo * 50);
        }
        else {
          ic_score += 30;
        }

        break;

      case PERFECT:
        ic_total_perfect++;
        ic_current_combo++;
        ic_perfect_combo++;

        if(ic_current_combo >= 3) {
          ic_score += (ic_perfect_combo * 100);
        }
        else {
          ic_score += 50;
        }

        break;

      default:
        break;
    }

    //display current combo
    lcd2.setCursor(0, 0);
    lcd2.print("Combo:");
    lcd2.setCursor(6, 0);
    lcd2.print(ic_current_combo, DEC);

    //display score
    lcd1.clear();
    lcd1.setCursor(0,0);
    lcd1.print("SCOREBOARD");

    lcd1.setCursor(0,1);
    lcd1.print("SCORE:");
    lcd1.setCursor(6, 1);
    lcd1.print(ic_score, DEC);

    break;

  case IC_LEFT:
    if (l) {
      state = IC_LEFT;
    }
    else {
      state = IC_INPUT_CAPTURE_ON;
    }

    break;

  case IC_DOWN:
    if (d) {
      state = IC_DOWN;
    }
    else {
      state = IC_INPUT_CAPTURE_ON;
    }

    break;

  case IC_UP:
    if (u) {
      state = IC_UP;
    }
    else {
      state = IC_INPUT_CAPTURE_ON;
    }

    break;

  case IC_RIGHT:
    if(r) {
      state = IC_RIGHT;
    }
    else {
      state = IC_INPUT_CAPTURE_ON;
```

```
      }

      break;

    default:
      break;
  }

  //actions
  switch(state) {
    case IC_INPUT_CAPTURE_OFF:
      //Serial.println("IC INPUT CAPTURE OFF");
      break;

    case IC_INPUT_CAPTURE_ON:
      //Serial.println("IC INPUT CAPTURE ON");
      break;

    case IC_LEFT:
      //Serial.println("IC_LEFT");
      break;

    case IC_DOWN:
      //Serial.println("IC DOWN");
      break;

    case IC_UP:
      //Serial.println("IC UP");
      break;

    case IC_RIGHT:
      //Serial.println("IC RIGHT");
      break;

    default:
      break;
  }

  return state;
}

//====================== ENDGAME DISPLAY SM ======================
//display the endscreen on LCD and TFT for a few seconds
//display score, max combo, judgements
enum ENDGAME_DISPLAY_SM_STATES{ED_START, ED_ENDGAME_DISPLAY_OFF, ED_ENDGAME_DISPLAY_ON};

int EndGameDisplaySM_Tick(int state) {

  static unsigned short endgame_counter = 0; //counter for end
  //transitions
  switch(state) {
    case ED_START:
      state = ED_ENDGAME_DISPLAY_OFF;
      endgame_counter = 0;
      break;

    case ED_ENDGAME_DISPLAY_OFF:
      if (start_endgame) {
        state = ED_ENDGAME_DISPLAY_ON;
        endgame_counter = 0;
        end_endgame = false;
      }
      else {
        state = ED_ENDGAME_DISPLAY_OFF;
      }
      break;


    case ED_ENDGAME_DISPLAY_ON:
      if (endgame_counter < 8000) {
        state = ED_ENDGAME_DISPLAY_ON;
      }
      else { //period is 500 ms
        state = ED_ENDGAME_DISPLAY_OFF;
        end_endgame = true;
      }

      break;

    default:
      break;
  }

  //actions
  switch(state) {
    case ED_ENDGAME_DISPLAY_OFF:
      //Serial.println("ED ENDGAME DISPLAY OFF");
      break;


    case ED_ENDGAME_DISPLAY_ON:
      //Serial.println("ED ENDGAME DISPLAY ON");
```

```
      //Serial.print("end_game_counter: ");
      //Serial.println(endgame_counter);

      endgame_counter++;
      break;

    default:
      break;
  }


  return state;
}


void setup() {
  //serial monitor - (for debugging)
  Serial.begin(9600);

  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

  //buzzer output
  pinMode(buzzer, OUTPUT);

  //led outputs
  pinMode(game_led_pin, OUTPUT);
  pinMode(end_display_led_pin, OUTPUT);

  //initialize led outputs
  digitalWrite(game_led_pin, LOW);
  digitalWrite(end_display_led_pin, LOW);

  //initialize btn
  pinMode(start_btn, INPUT_PULLUP);
  pinMode(l_btn, INPUT_PULLUP);
  pinMode(d_btn, INPUT_PULLUP);
  pinMode(u_btn, INPUT_PULLUP);
  pinMode(r_btn, INPUT_PULLUP);

  //initialize lcd
  lcd1.begin(16, 2);
  lcd2.begin(16, 2);

  //initialize tft lcd
  tft.reset();
  tft.begin(0x9341);
  tft.fillScreen(BLACK);

  //initialze tasks

  tasks[0].state = 0;
  tasks[0].period = period;
  tasks[0].prev_time = -(tasks[0].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[0].TickFct = &SPISM_Tick;

  tasks[1].state = SG_START;
  tasks[1].period = period;
  tasks[1].prev_time = -(tasks[1].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[1].TickFct = &StartGameSM_Tick;

  tasks[2].state = MS_START;
  tasks[2].period = period;
  tasks[2].prev_time = -(tasks[2].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[2].TickFct = &MusicSystemSM_Tick;

  tasks[3].state = M_START;
  tasks[3].period = period;
  tasks[3].prev_time = -(tasks[3].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[3].TickFct = &MapSM_Tick;

  tasks[4].state = SM_START;
  tasks[4].period = period;
  tasks[4].prev_time = -(tasks[4].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[4].TickFct = &ScrollMapSM_Tick;

  tasks[5].state = IC_START;
  tasks[5].period = period;
  tasks[5].prev_time = -(tasks[4].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[5].TickFct = &InputCaptureSM_Tick;

  tasks[6].state = ED_START;
  tasks[6].period = period;
  tasks[6].prev_time = -(tasks[5].period); //set to negative of the period to allow task to execute during starting up (
```

```
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[6].TickFct = &EndGameDisplaySM_Tick;


  /*
  tasks[0].state = SG_START;
  tasks[0].period = period;
  tasks[0].prev_time = -(tasks[0].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[0].TickFct = &StartGameSM_Tick;

  tasks[1].state = MS_START;
  tasks[1].period = period;
  tasks[1].prev_time = -(tasks[1].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[1].TickFct = &MusicSystemSM_Tick;

  tasks[2].state = M_START;
  tasks[2].period = period;
  tasks[2].prev_time = -(tasks[2].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[2].TickFct = &MapSM_Tick;

  tasks[3].state = SM_START;
  tasks[3].period = period;
  tasks[3].prev_time = -(tasks[3].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[3].TickFct = &ScrollMapSM_Tick;

  tasks[4].state = IC_START;
  tasks[4].period = period;
  tasks[4].prev_time = -(tasks[4].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[4].TickFct = &InputCaptureSM_Tick;

  tasks[5].state = ED_START;
  tasks[5].period = period;
  tasks[5].prev_time = -(tasks[5].period); //set to negative of the period to allow task to execute during starting up (
millis() - (-taskperiod) => 0 + taskperiod = elapsedtime
  tasks[5].TickFct = &EndGameDisplaySM_Tick;
  */
}

void loop() {
  unsigned char i;
  unsigned long current_time;
  for (i = 0; i < task_num; i++) {
    current_time = millis();
    if ( (current_time - tasks[i].prev_time) >= tasks[i].period ) { //check elasped time > period
      tasks[i].state = tasks[i].TickFct(tasks[i].state);
      tasks[i].prev_time = current_time;  //set previous time to current time
    }
  }
}
```

A2. K64F Code

Already available in the "Implementation Details" section above.

A3. Links to External Libraries Used

1. Elegoo_TFTLCD: https://github.com/Erutan409/Elegoo_TFTLCD
2. Elegoo_GFX: https://github.com/Erutan409/Elegoo_GFX
3. Touchscreen: https://github.com/adafruit/Adafruit_TouchScreen
4. LiquidCrystal: https://github.com/arduino-libraries/LiquidCrystal
5. Processor Expert: TimerInt, SPI