Final Project Report: Bop It!: MicroPs Edition

Nick Tan and Andreas Roeseler

Fall 2020

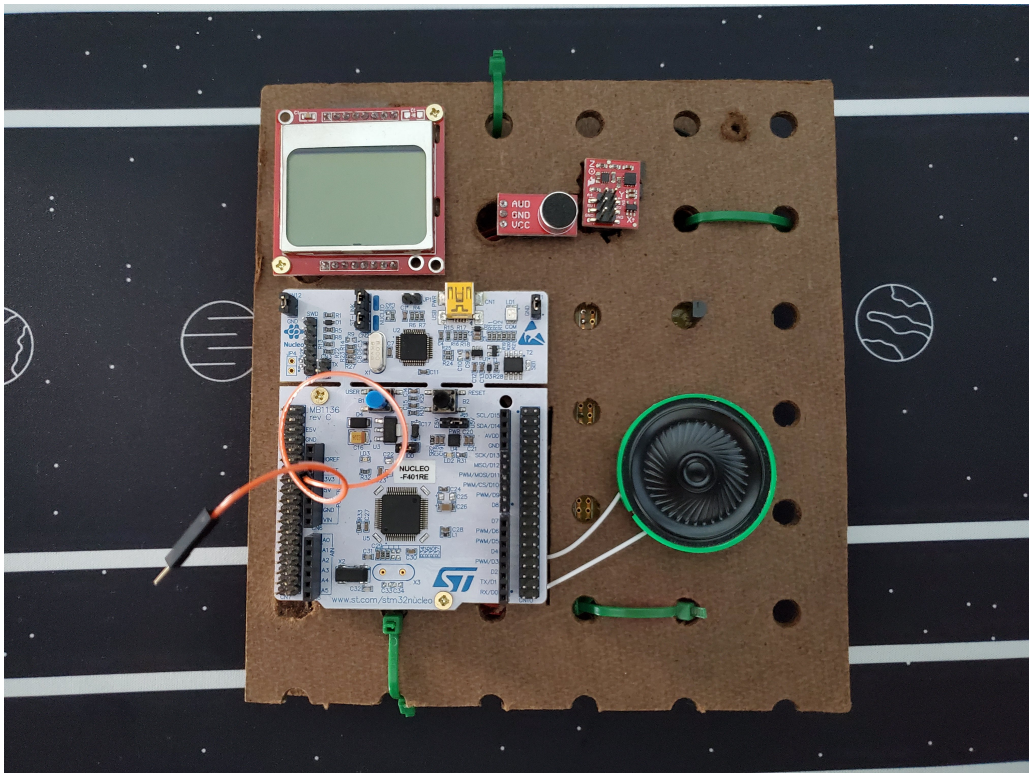

Figure 1: Bop It!: MicroPs Edition

Abstract

Since it was released in 1996, "Bop It!" has become a popular toy recognized all across the US. In our final project, we recreated a version of this game using our Nucleo Board and numerous peripherals. Our version of the game can choose between 5 different commands to send the user via audio and visual queues. It keeps track of the score and speeds up as the game progresses, forcing the user to respond faster with each successful input. If the user input does not match the command, the game also fails. This report contains details and schematics of our game, as well as the source code as an attached appendix.

1 Introduction

"Bop it!", a toy first released in 1996, is a popular toy recognized by people all over the US. The toy offers various tasks for the player to do within some amount of time: "Twist It!", "Bop It!", "Pull It!", etc. As the player successfully does their assigned task, their score increases, and their time to do their next task decreases. Since 1996, various iterations of the game have been released, incorporating more commands with different aesthetics.

For our final project, we decided to make "Bop It!: MicroPs Edition". We used components and libraries that we had built in labs over the course of the semester, as well as several new peripherals and hardware. Our "Bop It!: MicroPs Edition" offers the following commands: "Push It!", "Wire It!", "Heat It!", "Shout It!", and "Shake It!". The microcontroller uses the on-board button B1, a GPIO port, the temperature sensor from Lab 6, a microphone, and an accelerometer to gather input from the user for each command respectively. To issue those commands, we made use of a LCD Display and the speaker-amplifier circuit from Lab 3 to give visual and audio cues to the player. Finally, we used several timers and interrupts to handle key game functionality such as gathering input from the player and keeping track of the amount of time a user has to respond to a command.

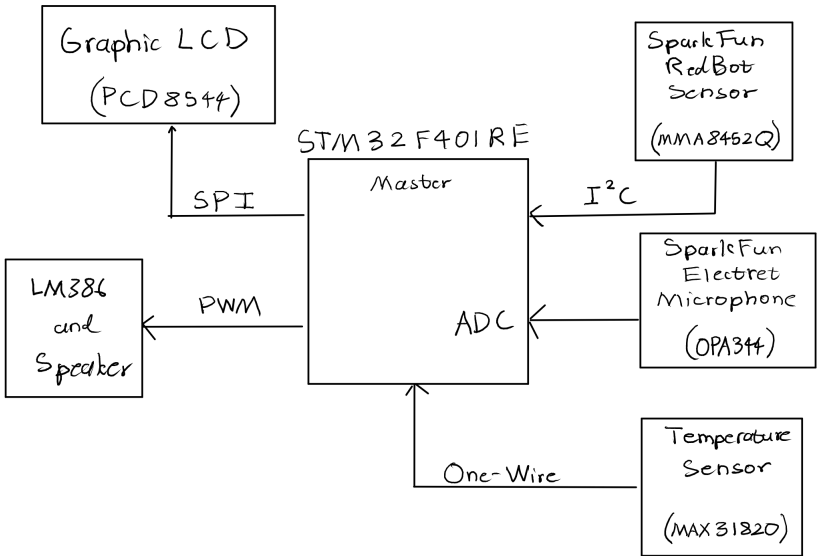

Figure 2: System block diagram showing showing all external peripherals

2 New Hardware

This project uses three pieces of hardware that has not been used in E155 before: accelerometer, microphone, and a LCD display. Using the accelerometer allows our Nucleo to obtain information regarding movement, allowing "Shake I!t" to be implemented. The microphone gives the microcontroller access to ambient noise, which will be used for "Shout It!". Lastly, the LCD display will be used to give interpretable instructions to the user. The specifics of each of these new peripherals will be discussed later in the paper.

3 Game Controller

Our game is controlled by the STM32F401RE chip on our Nucleo Board, and can be described by the state machine in Figure 3. Upon booting up, the board initializes the peripherals as well as key game values, such as a baseline measurement for the temperature sensor, before writing a welcome message to the screen. It then waits for the user to press button B1 on the Nucleo board, before entering the core loop of the game in state S1. Here, the game controller randomly chooses a command from the above list. If Heat It! is chosen, the game controller will poll the peripheral to ensure the temperature has dropped to within a certain threshold of the baseline before continuing with the command. If the temperature sensor's reading is too high, it skips this command and randomly chooses another command from the remaining list. Next, the game enters S2 where it polls the peripherals and waits for a duration

specified by the game delay, which is decreased each iteration. If the user does not input a command before the time runs out, the game controller will transition to the game over state S5. We poll the accelerometer, temperature sensor and microphone while the other inputs are triggered by interrupts. In our experience, polling each of these peripherals can occur within the span of 5 ms, so we believe this is adequate to make the game feel responsive to the users commands. We will discuss the method of communicating with these devices in a later section. Upon receiving an input from the user, either by polling or via interrupt, the game controller checks the input to ensure it matches the issued command. During this time, interrupts are disabled and the other peripherals are no longer polled to simplify the game logic. If the user's input matches the command, the score is incremented and the game delay is reduced before returning to state S1 for the next iteration of the game. If the user input a wrong command, the game will go to the game over state, which displays a game over message and the user's score.

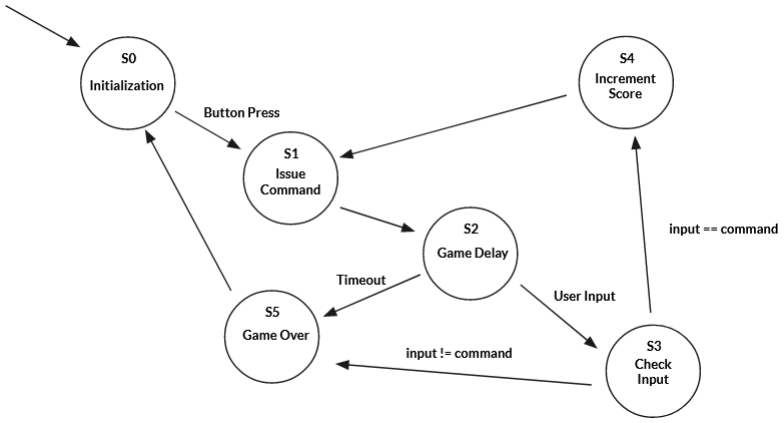

Figure 3: State machine of our game logic

3.1 Push It!

The Push It! command is connected to PB1 on the Nucleo Board and is triggered via an interrupt on EXTI15_10. This button handles starting the game if the game has not already begun and triggering the Push It! input when the game is running.

3.2 Shout It!

The Shout It! command recieves input from a separate microphone peripheral that sends an analog signal to our microprocessor. We use the on-board ADC to read the microphone signal and we have set up the Analog Watchdog of the

ADC to trigger an interupt after a conversion is complete, if the reported value deviates too far from the baseline.

3.3 Shake It!

Shake It! reads input an input signal from our accelerometer. When the accelerometer receives a force greater than $1.5g$, it sets a flag in the accelerometer's internal registers. In our Game Delay state (S3), we poll the status registers of the accelerometer to determine if the device has been shaken.

3.4 Heat It!

Heat It! consists of a temperature sensor communicating with our Nucleo Board via a One-Wire communication scheme. We poll this peripheral in our core game loop to read in values and determine if the user has input this command. We have set up the temperature sensor to 9 bits of precision to speed up the calculation. Additionally, our game is built around a fast response by the user, so more frequent polling is more beneficial than extra bits of precision.

3.5 Wire It!

Wire It! recieves input on PA5 and triggers an interrupt on the rising edge of the pin. We chose pin PA5 so the on-board LED LD2 would light up if the pin is high, providing an additional visual queue. However, we later discovered that PA5 is right below a ground pin on the board, and if this pin is accidentally connected to power, the board resets. As a result, we have decided to have a pin sticking out of PA5, making the task easier to complete without accidentally shorting the board.

4 Peripherals

4.1 Equipment

Below is a table of the exact equipment used in our final project. For simplicity, we will refer to each piece of equipment by its generic name in column 1, rather than the part name or part number.

Generic Name	Part Name	Part Number
LCD	Nokia 511 Graphic LCD 84x48	LCD-10168
Microphone	SparkFun Electret Microphone Breakout	BOB-12758
Accelerometer	SparkFun RedBot Sensor - Accelerometer	SEN-12589
Speaker	Speaker .5 W 9 ohm ± 15% at 1500 HZ	1891

Table 1: Equipment List

4.2 LED Display

We communicate with the LCD Display through the Nucleo's SPI peripheral. We were able to utilize code from previous labs to configure our board's SPI peripheral which helped us quickly establish a framework to communicate. The LCD Display has its own set of instructions that we used to write each pixel to the display in groups of 8[5]. The Nucleo board contains an array of bits that correspond to pixels on the display. This gives us the ability to set pixels individually before updating the entire board at the once. We also have defined preset messages in memory that are copied over to our display array before updating the board. We were able to generate the preset messages using a combination of Paint 3D and LCDAssistant, which allowed us to quickly create any image we wished to display and convert it to a C array.

4.3 Accelerometer

The accelerometer we used for our project communicates to the Nucleo through the I2C protocol, which we had written for this project. Since we wanted our accelerometer to check if the player has shaken the "Bop It!", we configured the accelerometer to detect motion in the x, y, and z axis that above the set threshold of $1.5g$. After testing with different values for the acceleration threshold, we found that $1.5g$ was a good balance for the accelerometer's sensitivity. Within the accelerometer itself, the Freefall/Motion Source Register contains the EA bit that indicates if motion is detected in the x, y, or z axis, which is configured to be always be set until our Nucleo reads the respective register through I2C. Since the Nucleo had to repeatedly read a register in the accelerometer, we decided to have the microcontroller poll the peripheral, which did not noticeably slow down the overall game. And with how the accelerometer is configured to keep the EA bit set until the register is read, polling the accelerometer will not miss user input.

4.4 Microphone

The microphone sends an analog signal through a single wire to our Nucleo board. In order to read out the data from the microphone, we wrote a library to configure our on-board ADC. Before any command is issued, we take a baseline reading from the microphone to calibrate the ambient noise in the room. The ADC is set to sample over the longest number of cycles possible (480 cycles at 84 MHz) to attempt to minimize any errors caused by undersampling. While this does not conform to the Shannon-Nyquist Theorem, this is the highest sampling time allowed by our ADC and in practice we have not noticed any issues.
Once a command is issued, we set up the watchdog interrupt of our ADC to trigger if the microphone reading is above or below the baseline reading by a certain threshold. We set it to both above and below the ambient voltage reading of our microphone because the signal rests at around 3.3V in a quiet room. In state S2 of our game logic, the device is polled by simply telling the

ADC to do a single conversion. Since the Analog Watchdog is enabled, the ADC will trigger an interrupt after the conversion is complete if the signal deviates from the baseline, so this is all that is required by our game controller in the polling cycle of S2. This helps keep our polling cycle low and our game feeling responsive to the user's inputs.

4.5 Speaker and Amplifier

We send audio queues to the user through a speaker and amplifier setup described in Figure 4. We send the output signal through a LM386N-4/NOPB amplifier to increase the volume of our speaker. We also have a potentiometer so the user can control the volume while the game is playing. This set up is based off of our work in Lab 3 of this class, and the library to send signals to the speaker is built upon the code for that lab. As a result, we use a PWM signal generated by the Nucleo board to control the frequency of the speaker's output. Similarly to our LCD display, we have also defined frequency and duration tuples in the board's memory that correspond to the different sound queues.

4.6 Temperature Sensor

We communicate with the temperature sensor via a One-Wire control scheme. The library to read data from our temperature sensor is based off our work in Lab 6 to configure the One-Wire communication and send commands to the sensor. However, the temperature sensor has some drawbacks that do not make it ideal for an input in a fast-paced game such as Bop It! First, the sensor has significant latency before its value is updated, and as a result we have had to adjust the delay to give users more time to respond when the game chooses the Heat It! command. We have also set the temperature sensor to 9 bits of precision, which allows the sensor to take measurements much more quickly, and speed is much more important than precision past the tenths of a degree. Additionally, the sensor does not drop back to a baseline as quickly as the other peripherals. This can cause the disastrous gameplay experience of attempting to heat up the sensor that is already at its maximum temperature. As a result, we have built in functionality to check that the sensor has returned to within a threshold of the baseline measurement before it is allowed to be sent as a command. This avoids the issue entirely and allows our game to still offer Heat It! as a command.

5 Conclusion

In our work this semester, we have completed every deliverable described in our Final Project Proposal[6]. We have created libraries to interface with all of our hardware via 1Wire, I2C, and SPI communications, and have configured the on-board ADC, timers, and PWM functionality. The game randomly chooses between five commands to send to the user (Push It!, Shout It!, Shake It!, Heat It!, and Wire It!), and requires the user to respond faster with each successful input. It also keeps track of the score and displays the final score at the end of the game. We also have distinct audio and visual queues for each command, as well as special welcome and game over messages.

A Schematics

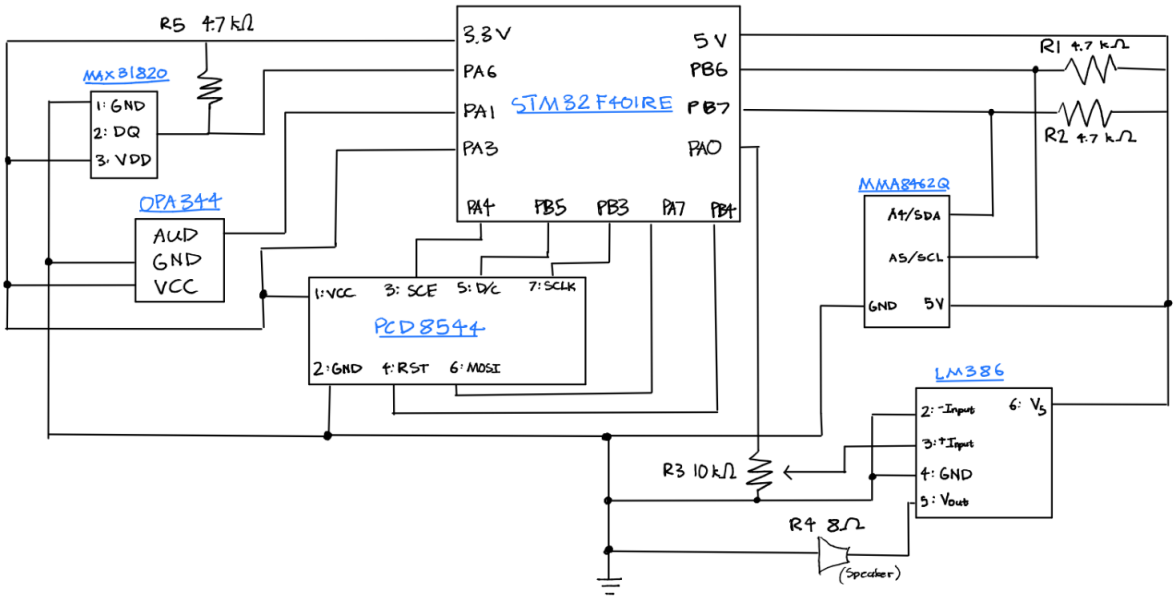

Figure 4: Full schematics of our "Bop It!"

B Source Code

```
// main.h
// Standard library includes
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

// Vendor-provided device header file
#include "stm32f4xx.h"
#include "STM32F401RE.h"


////////////////////////////
// Definitions
////////////////////////////
```

```
#ifndef max
    #define max(a, b) ((a) > (b) ? (a) : (b))
#endif


#define PUSH_BUTTON 13
#define WIRE_IT_PIN 5
#define DELAY_TIM TIM2
#define SOUND_TIM TIM5

#define PUSH_IT 1
#define SHOUT_IT 2
#define WIRE_IT 3
#define HEAT_IT 4
#define SHAKE_IT 5

#define NUM_COMMANDS 5

#define MESSAGE_DELAY 1500
#define GAME_DELAY 5000
#define GAME_DELAY_CHANGE -15
#define MIN_GAME_DELAY 500


////////////////////////////
// Function Prototypes
////////////////////////////

void initPushButton();
void waitForInput(uint32_t gameDelay);
void displayInit(void);
void EXTI15_10_IRQHandler(void);
void ADC_IRQHandler(void);
```

```
#include "main.h"

#define VOP_VAL 80

uint8_t gameStarted;                // Tracks if the game has started or not
uint8_t gameOver;                   // Tracks Game over state
uint32_t gameDelay;                 // Tracks the amount of game will delay after each turn
uint32_t score;                     // Tracks score of player
uint8_t task;                       // Stores the task given to the player
uint8_t input;                      // Stores the user's registered input
```

```
uint16_t ambientTemp;               // Stores the ambient room temperature when the game sta
uint16_t currTemp;      // Stores current temperature when "Heat It" is issued


// All commands that the game will support
uint8_t commands[] = {PUSH_IT, SHOUT_IT, WIRE_IT, SHAKE_IT, HEAT_IT};


void initPushButton() {
// Set EXTICR4 for PC13
SYSCFG->EXTICR[3] &= ~(SYSCFG_EXTICR4_EXTI13_Msk);
SYSCFG->EXTICR[3] |= (0b0010 << SYSCFG_EXTICR4_EXTI13_Pos);

EXTI->IMR |= EXTI_IMR_IM13;             // Configure IM13 mask bit
EXTI->RTSR &= ~(EXTI_RTSR_TR13_Msk);    // Disable rising edge trigger
EXTI->FTSR |= EXTI_FTSR_TR13;           // Enable falling edge trigger
__NVIC_EnableIRQ(EXTI15_10_IRQn);       // Enable External Line[15:10] Interrupts
}

void initWireIt() {
pinMode(GPIOA, WIRE_IT_PIN, GPIO_INPUT);

// Set EXTICR2 for PA5
SYSCFG->EXTICR[1] &= ~(SYSCFG_EXTICR2_EXTI5_Msk);

EXTI->IMR |= EXTI_IMR_IM5; // Configure IM5 mask bit
EXTI->RTSR |= EXTI_RTSR_TR5_Msk; // Enable rising edge trigger
EXTI->FTSR &= ~(EXTI_FTSR_TR5); // Disable falling edge trigger
__NVIC_EnableIRQ(EXTI9_5_IRQn); // Enable External Line[9:5] Interrupts
}

void initADCInterrupt() {
    __NVIC_EnableIRQ(ADC_IRQn);
}

// Waits for gameDelay time to get input from user
void waitForInput(uint32_t gameDelay) {
while (gameDelay > 0 && input == 0) {
// Initializes inputs from user
uint8_t temp = 0;
uint8_t shake = SHAKE_IT * detectMotion(I2C1);

// Reads for any input
read_ADC(ADC1);
```

```
// Read temperature if needed
if (task == HEAT_IT) {
temp = (getTemperature() > (currTemp + 2));
temp *= HEAT_IT;
}

// If either temp or shake is triggered, update input if input is 0
if ((temp || shake) || input) {
if (input) return;

// Choose shake if both are non-zero, otherwise choose the non-zero var as input
input = max(temp, shake);
return;
}

// Delay for 5 milliseconds and poll again
gameDelay -= 5;
delay_millis(DELAY_TIM, 5);
}
}

// Initialize the LCD Display
void displayInit(void) {
// Power on the display
digitalWrite(GPIOA, 3, 1);

// Send reset pulse
digitalWrite(GPIOB, DISPLAY_RESET, 0);
delay_micros(TIM2, 100);
digitalWrite(GPIOB, DISPLAY_RESET, 1);

digitalWrite(GPIOA, DISPLAY_CS, 1);

// Set Power Down = 0, V to 0, and H to 1 (extended instruction set)
displaySend(0, 0b00100001);
// Set Vop
displaySend(0, 0b10000000 + VOP_VAL);
// Set Power Down = 0, V to 0 (horizontal addressing), and H to 0 (normal instruction)
displaySend(0, 0b00100000);
// Set display configuration to normal mode
displaySend(0, 0b00001100);
}

/**
 * Main program.
 */
```

```
int main(void) {
uint8_t msg[64];
uint16_t adc_val;
configureFlash();
configureClock();

//////////////////////////
// Enable Clock Domains
//////////////////////////

// Enable GPIOA, GPIOB, GPIOC
RCC->AHB1ENR  |= (RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIOCEN);
// Enable TIM2, TIM5, and I2C1
RCC->APB1ENR  |= (RCC_APB1ENR_TIM2EN | RCC_APB1ENR_TIM5EN |RCC_APB1ENR_I2C1EN);
// Enable SYSCFG clock domain and SPI1
RCC->APB2ENR  |= (RCC_APB2ENR_SYSCFGEN | RCC_APB2ENR_SPI1EN);

////////////////////////////////
// Init TIM2/TIM5
////////////////////////////////

initTIM(DELAY_TIM);                          // Initialize TIM2
configMusicTIM(SOUND_TIM); // Initialize TIM5

// Enable channel 1 for TIM5
pinMode(GPIOA, 0, GPIO_ALT);
GPIOA->AFR[0] |= (2 << GPIO_AFRL_AFSEL0_Pos);

////////////////////
// ADC Set Up
////////////////////

initADC(ADC1, GPIOA, 1, 7);                  // Initialize ADC to pin A1
pinMode(GPIOA, 1, GPIO_ANALOG);              // PA1 is input for ADC
initADCInterrupt();

/////////////////////
// Push Button Set Up
/////////////////////

initPushButton();
pinMode(GPIOC, PUSH_BUTTON, GPIO_INPUT);             // PC13 is Nucleo Push Button

////////////////////
// I2C1 Set up
////////////////////
```

```
// Set PB6 and PB7 to alternate function 4 and to open drain configuration
GPIOB->AFR[0] |= (4 << GPIO_AFRL_AFSEL6_Pos | 4 << GPIO_AFRL_AFSEL7_Pos);
pinMode(GPIOB, 6, GPIO_ALT);
pinMode(GPIOB, 7, GPIO_ALT);
GPIOB->OTYPER |= (1 << GPIO_OTYPER_OT6_Pos | 1 << GPIO_OTYPER_OT7_Pos);

initI2C(I2C1);
setUpAccelerometer(I2C1);

//////////////////////
// Temp Sensor Set Up
//////////////////////

setupOneWire(); // Sets up OneWire communication over PA6
initTempSensor(); // Configure temp sensor to be 9 bits resolution

//////////////////////
// Display SPI Set Up
//////////////////////

spiInit(4, 1, 1);
digitalWrite(GPIOA, 3, GPIO_OUTPUT);
delay_millis(TIM2, MESSAGE_DELAY);
displayInit();

//////////////////////
// Wire it setup
//////////////////////

initWireIt();


// Enable interrupts globally and initialize random seed
__enable_irq();
srand(time(NULL));

while (1) {
// Reset game state
gameStarted = 0;
delay_millis(DELAY_TIM, MESSAGE_DELAY);
writeMessage("WELCOME");
updateDisplay();

// Wait for user to start the game
while(!gameStarted){
```

```
delay_millis(DELAY_TIM, MESSAGE_DELAY);
}

// Initialize a new random seed
srand(rand());

// Reinitializes game elements
score = 0;
gameOver = 0;
gameDelay = GAME_DELAY;
ambientTemp = getTemperature();
detectMotion(I2C1);

// Send visual and audio cue that game has started
playMusic(START);
writeMessage("READY");
updateDisplay();
delay_millis(DELAY_TIM, MESSAGE_DELAY);
writeMessage("START");
updateDisplay();

// Main game functionality
while(1) {

// Only add "Heat It" to pool of commands if temp is low enough
currTemp = getTemperature();
if (currTemp <= ambientTemp + 2) {
task = commands[rand() % NUM_COMMANDS];
} else {
task = commands[rand() % (NUM_COMMANDS - 1)];
}

// Write command to screen and play corresponding music
writeCommand(task);
updateDisplay();
playMusic(task);

__enable_irq();

// Clear user input and selects a random command for task
input = 0;
detectMotion(I2C1);

// Handles different tasks
switch (task)
{
```

```
case SHOUT_IT:
calibrate_ADC(ADC1);
waitForInput(gameDelay);
break;
case HEAT_IT:
// Decrease delay since heat it takes longer
waitForInput(gameDelay/4);
break;
default:
waitForInput(gameDelay);
break;
}

// Processes user input
if(task != input) goto game_over;
__disable_irq();

// Play victory tone, update game state
playMusic(SUCCESS);
++score;
if(gameDelay > MIN_GAME_DELAY) gameDelay += GAME_DELAY_CHANGE;

// Clear flag in motion sensor
delay_millis(DELAY_TIM, MESSAGE_DELAY);
}

game_over:
// Game Over Handler
writeMessage("GAME OVER");
writeScore(score, DISPLAY_WIDTH - 12, DISPLAY_HEIGHT - 8);
updateDisplay();

playMusic(GAME_OVER);
delay_millis(DELAY_TIM, MESSAGE_DELAY);
}
}

/*
 * Interrupts
 */

// Push button interrupt handler
void EXTI15_10_IRQHandler(void) {
// Check that the button EXTI_13 was what triggered our interrupt
if (EXTI->PR & (1 << PUSH_BUTTON)){
// Clear Interrupt
```

```
EXTI->PR |= (1 << PUSH_BUTTON);
if(!gameStarted) {
gameStarted = 1;
}
else {
input = PUSH_IT;
DELAY_TIM->EGR |= TIM_EGR_UG;    // Write bit to clear the delay timer
}
}
}

// Wire it interrupt handler
void EXTI9_5_IRQHandler(void) {
// Check that the button EXTI_5 was what triggered our interrupt
if(EXTI->PR & (1 << WIRE_IT_PIN)) {
// Clear Interrupt
EXTI->PR |= (1 << WIRE_IT_PIN);
input = WIRE_IT;
DELAY_TIM->EGR |= TIM_EGR_UG; // Write bit to clear the delay timer
}
}

// ADC interrupt handler
void ADC_IRQHandler(void) {
if(ADC1->SR & ADC_SR_AWD) {
// Disable watchdog interrupt enable
     ADC1->CR1 &= ~ADC_CR1_AWDIE;
// Clear interrupt
ADC1->SR &= ~(ADC_SR_AWD);

input = SHOUT_IT;
DELAY_TIM->EGR |= TIM_EGR_UG; // Write bit to clear the delay timer
}
}
```

```
// STM32F401RE.h
// Header to include all other STM32F401RE libraries.

#ifndef STM32F4_H
#define STM32F4_H

#include <stdint.h>

// Include other peripheral libraries
```

```
#include "STM32F401RE_GPIO.h"
#include "STM32F401RE_FLASH.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_SPI.h"
#include "STM32F401RE_TIM.h"
#include "STM32F401RE_1WIRE.h"
#include "STM32F401RE_USART.h"
#include "STM32F401RE_I2C.h"
#include "STM32F401RE_ADC.h"


#endif
```

```
// STM32F401RE_1WIRE.h
#ifndef STM32F4_1WIRE_H
#define STM32F4_1WIRE_H

#include <stdint.h>
#include "stm32f4xx.h"
// #include "STM32F401RE_TIM.h"
// #include "STM32F401RE_GPIO.h"

////////////////////////////////////////////////////////////////////////////////
// Definitions
////////////////////////////////////////////////////////////////////////////////

#define GPIOx GPIOA
#define ONEWIRE_PIN 6
#define ONEWIRE_TIMx TIM2

////////////////////////////////////////////////////////////////////////////////
// Function prototypes
////////////////////////////////////////////////////////////////////////////////

/**
    Setups up a 1-Wire interface on ONEWIRE_PIN.
    @return The area of the circle.
*/
void setupOneWire(void);

/**
    Sets up up a 1-Wire interface on ONEWIRE_PIN.
    @return 1 if a device is detected on the line and a 0 otherwise
```

```
*/
uint8_t init(void);

/**
    Writes a bit to the 1-Wire data line on ONEWIRE_PIN
    @param Value to be written (1 or 0)
*/
void write_bit(uint8_t val);

/**
    Reads a bit to the 1-Wire data line on ONEWIRE_PIN
    @return Value read (1 or 0)
*/
uint8_t read_bit(void);

/**
    Writes a byte (8 bits) to the 1-Wire data line on ONEWIRE_PIN
    @param Byte to be written
*/
void write_byte(uint8_t val);

/**
    Reads a byte from the 1-Wire data line on ONEWIRE_PIN
    @return Byte read
*/
uint8_t read_byte(void);

/**
    Reads the ROM code of the MAXIM 31820 device on the bus.
    @param Array where rom_code should be written
*/
void read_rom(uint8_t rom_code[8]);

/**
    Reads the scratchpad code of the MAXIM 31820 device on the bus.
    @param Array where rom_code should be written
*/
void read_scratchpad(uint8_t scratchpad[8]);

/**
    Returns the 2 byte register for temperature in the MAXIM 31820
*/
int getTemperature();

/**
    Runs init command, followed by a skip rom command
```

```
*/
void skip_rom();

void initTempSensor();

#endif
```

```
// STM32F401RE_1WIRE.c
#include "STM32F401RE_1WIRE.h"


void setupOneWire(void){
    initTIM(ONEWIRE_TIMx);
    digitalWrite(GPIOx, ONEWIRE_PIN, 1);        // Set PA6 to output
    pinMode(GPIOx, ONEWIRE_PIN, 1);
    GPIOx->OTYPER |= (1 << ONEWIRE_PIN);
    GPIOx->OSPEEDR |= (0b11 << ONEWIRE_PIN * 2);
}

uint8_t init(void){
    // Reset pulse
    digitalWrite(GPIOx, ONEWIRE_PIN, 0);
    delay_micros(ONEWIRE_TIMx, 480);
    digitalWrite(GPIOx, ONEWIRE_PIN, 1);


    // Read the bus every 60 microsec and delay for at least 480 us
    int time = 0;
    for (int i = 0; i < 4; ++i) {
        delay_micros(ONEWIRE_TIMx, 60);
        time += 60;
        if (digitalRead(GPIOx, ONEWIRE_PIN) == 0) {
            delay_micros(ONEWIRE_TIMx, 480 - time);
            return 1;
        }
    }
    return 0;
}

void write_bit(uint8_t val){
    // Set the bus low
    digitalWrite(GPIOx, ONEWIRE_PIN, 0);

    if (val == 0) {                             // Write-zero slot
```

```
        delay_micros(ONEWIRE_TIMx, 60);
        digitalWrite(GPIOx, ONEWIRE_PIN, 1);
    } else {                                    // Write-one slot
        delay_micros(ONEWIRE_TIMx, 5);
        digitalWrite(GPIOx, ONEWIRE_PIN, 1);
        delay_micros(ONEWIRE_TIMx, 60);
    }
}

uint8_t read_bit(void){
    // Initiate read process
    digitalWrite(GPIOx, ONEWIRE_PIN, 0);
    delay_micros(ONEWIRE_TIMx, 2);
    digitalWrite(GPIOx, ONEWIRE_PIN, 1);

    // Read the bus at about 12 us
    delay_micros(ONEWIRE_TIMx, 10);
    uint8_t output = digitalRead(GPIOx, ONEWIRE_PIN);

    // Delay for at least 60 us
    delay_micros(ONEWIRE_TIMx, 70);
    return output;
}

void write_byte(uint8_t val){
    // Write bit by bit starting from the LSB
    for (int i = 0; i < 8; i++) {
        uint8_t bit = val % 2;
        val = val >> 1;
        write_bit(bit);
    }
}

uint8_t read_byte(void){
    uint8_t output = 0;
    uint8_t shifter = 0;
    // Read the bit on the bus and add amount to a counter
    for (int i = 0; i < 8; i++) {
        uint8_t bit = read_bit();
        output +=  bit << shifter;
        shifter += 1;
    }
    return output;
}

void read_rom(uint8_t rom_code[8]){
```

```
    // Send initialization sequence
    while(init() == 0);
    // Send SKIP ROM command code
    write_byte(0xCC);
}

void skip_rom() {
    // Send initialization sequence
    while(init() == 0);
    // Send SKIP ROM command code
    write_byte(0xCC);
}

void read_scratchpad(uint8_t scratchpad[8]){
    // Send Read Scratchpad command
    write_byte(0xBE);

    // Read out scratchpad
    for (int i = 0; i < 8; i++) {
        uint8_t byte = read_byte();
        scratchpad[i] = byte;
    }

    volatile uint8_t crc = read_byte();
}

int getTemperature() {
    uint8_t scratchpad[8];

    // Init, skip rom, convert T
    skip_rom();
    write_byte(0x44);

    // Wait until conversion is done
    while(read_bit() == 0);

    // Init, skip rom, read scratchpad
    skip_rom();
    read_scratchpad(scratchpad);

    // Return the first two temperatures as one integer
    int temp = (scratchpad[1] << 8) + scratchpad[0];

    // Remove all decimals in the temperature
    temp = temp >> 4;
```

```
    return temp;
}

void initTempSensor() {
    // Init, skip rom, write scratchpad
    skip_rom();
    write_byte(0x4e);

    write_byte(0b01111111);     // Set Th to max
    write_byte(0b11111111);     // Set Tl to min
    write_byte(0b00011111);     // Configure sensor to 9 bit resolution
}
```

```
// STM32F401RE_ADC.h

#ifndef STM32F4_I2C_H
#define STM32F4_I2C_H

#include <stdint.h>
#include "stm32f4xx.h"

#define ADC_THRESHOLD_CHANGE 1500
/* Initialize ADC A2D to take input from a GPIO pin
 * And sample for a set time
 * Values for sample:
 *      0 = 3 cycles
 *      1 = 15 cycles
 *      2 = 28 cycles
 *      3 = 56 cycles
 *      4 = 84 cycles
 *      5 = 112 cycles
 *      6 = 144 cycles
 *      7 = 480 cycles
 */
void initADC(ADC_TypeDef *A2D, GPIO_TypeDef *GPIO, uint8_t pin, uint8_t sample);
void calibrateADC(ADC_TypeDef *A2D);
uint16_t read_ADC(ADC_TypeDef *A2D);

#endif
```

```
// STM32F401RE_ADC.c
#include "STM32F401RE_ADC.h"
```

```
void initADC(ADC_TypeDef *A2D, GPIO_TypeDef *GPIO, uint8_t pin, uint8_t sample) {
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // Enable ADC

    A2D->CR2 |= ADC_CR2_ADON;   // Power on ADC
    A2D->CR2 |= ADC_CR2_EOCS;   // Set software conversion management

    A2D->SQR1 &= ~(ADC_SQR1_L_Msk); // Only do conversion on one channel

    // Enable watchdog
    A2D->CR1 |= ADC_CR1_AWDEN;  // Enable watchdog on regular channels

    // Set watchdog on input channel 1
    A2D->CR1 &= ~(ADC_CR1_AWDCH_Msk);
    A2D->CR1 |= (1 << ADC_CR1_AWDCH_Pos);

    // Calculate ADC IN based on GPIO and pin
    if(GPIO == GPIOB) pin += 8;
    else if(GPIO == GPIOC) pin += 10;
    A2D->SQR3 |= (pin << ADC_SQR3_SQ1_Pos);
    A2D->SMPR2 |= (sample << ADC_SMPR2_SMP0_Pos);
}

void calibrate_ADC(ADC_TypeDef *A2D) {
    uint16_t baseline = read_ADC(A2D);
    A2D->HTR &= ~(ADC_HTR_HT_Msk);
    A2D->HTR |= ((baseline + ADC_THRESHOLD_CHANGE) & ADC_HTR_HT_Msk);
    A2D->LTR &= ~(ADC_LTR_LT_Msk);
    A2D->LTR |= ((baseline - ADC_THRESHOLD_CHANGE) & ADC_LTR_LT_Msk);
    // Clear Analog watchdog Flag
    A2D->SR &= ~(ADC_SR_AWD_Msk);
    // Enable watchdog interrupt enable
    A2D->CR1 |= ADC_CR1_AWDIE;
}


uint16_t read_ADC(ADC_TypeDef *A2D) {
    A2D->SR &= ~(ADC_SR_EOC_Msk);       // Clear end of conversion flag
    A2D->CR2 &= ~(ADC_CR2_CONT_Msk);    // Disable continuous conversion
    A2D->CR2 |= ADC_CR2_SWSTART;        // Begin conversion of ADC
    while(!(A2D->SR & ADC_SR_EOC));     // Wait until conversion is complete
    return (A2D->DR & ADC_DR_DATA);
}
```

```
// STM32F401RE_FLASH.h
```

```
// Header for FLASH functions

#ifndef STM32F4_FLASH_H
#define STM32F4_FLASH_H

#include <stdint.h>
#include "stm32f4xx.h"

////////////////////////////////////////////////////////////////////////////////
// Function prototypes
////////////////////////////////////////////////////////////////////////////////

void configureFlash();

#endif
```

```
// STM32F401RE_FLASH.c
// Source code for FLASH functions

#include "STM32F401RE_FLASH.h"

void configureFlash() {
    // Set waitstates and turn on the ART
    FLASH->ACR &= ~(FLASH_ACR_LATENCY_Msk | FLASH_ACR_PRFTEN_Msk);
    FLASH->ACR |= (2 << FLASH_ACR_LATENCY_Pos | 1 << FLASH_ACR_PRFTEN_Pos);
}
```

```
// STM32F401RE_GPIO.h
// Header for GPIO functions

#ifndef STM32F4_GPIO_H
#define STM32F4_GPIO_H

#include <stdint.h> // Include stdint header
#include "stm32f4xx.h"

////////////////////////////////////////////////////////////////////////////////
// Definitions
////////////////////////////////////////////////////////////////////////////////

// Values for GPIO pins ("val" arguments)
#define GPIO_LOW    0
#define GPIO_HIGH   1
```

```
// Arbitrary GPIO functions for pinMode()
#define GPIO_INPUT  0
#define GPIO_OUTPUT 1
#define GPIO_ALT    2
#define GPIO_ANALOG 3

// Pin definitions for every GPIO pin
#define GPIO_PA0    0
#define GPIO_PA1    1
#define GPIO_PA2    2
#define GPIO_PA3    3
#define GPIO_PA4    4
#define GPIO_PA5    5
#define GPIO_PA6    6
#define GPIO_PA7    7
#define GPIO_PA8    8
#define GPIO_PA9    9
#define GPIO_PA10   10
#define GPIO_PA11   11
#define GPIO_PA12   12
#define GPIO_PA13   13
#define GPIO_PA14   14
#define GPIO_PA15   15

////////////////////////////////////////////////////////////////////////////////
// Function prototypes
////////////////////////////////////////////////////////////////////////////////

void pinMode(GPIO_TypeDef *, int pin, int function);

int digitalRead(GPIO_TypeDef *, int pin);

void digitalWrite(GPIO_TypeDef *, int pin, int val);

void togglePin(GPIO_TypeDef *, int pin);

#endif
```

```
// STM32F401RE_GPIO.c
// Source code for GPIO functions

#include "STM32F401RE_GPIO.h"
```

```
void pinMode(GPIO_TypeDef* GPIO_PORT_PTR, int pin, int function) {
    switch(function) {
        case GPIO_INPUT:
            GPIO_PORT_PTR->MODER &= ~(0b11 << 2*pin);
            break;
        case GPIO_OUTPUT:
            GPIO_PORT_PTR->MODER |= (0b11 << 2*pin);
            GPIO_PORT_PTR->MODER &= ~(0b10 << (2*pin));
            break;
        case GPIO_ALT:
            GPIO_PORT_PTR->MODER &= ~(0b11 << 2*pin);
            GPIO_PORT_PTR->MODER |= (0b10 << (2*pin));
            break;
        case GPIO_ANALOG:
            GPIO_PORT_PTR->MODER |= (0b11 << 2*pin);
            break;
    }
}

int digitalRead(GPIO_TypeDef* GPIO_PORT_PTR, int pin) {
    return ((GPIO_PORT_PTR->IDR) >> pin) & 1;
}

void digitalWrite(GPIO_TypeDef* GPIO_PORT_PTR, int pin, int val) {
    if(val == 1){
        GPIO_PORT_PTR->ODR |= (1 << pin);
    }
    else if(val == 0){
        GPIO_PORT_PTR->ODR &= ~(1 << pin);
    }

}

void togglePin(GPIO_TypeDef* GPIO_PORT_PTR,int pin) {
    // Use XOR to toggle
    GPIO_PORT_PTR->ODR ^= (1 << pin);
}
```

```
// STM32F401RE_I2C.h
#ifndef STM32F4_I2C_H
#define STM32F4_I2C_H

#include <stdint.h>
#include "stm32f4xx.h"
```

```
////////////////////////////////////////////////////////////////////////////////
// Definitions
////////////////////////////////////////////////////////////////////////////////
#define DEVICE 0b0011101        // Address for MMA8452Q

////////////////////////////////////////////////////////////////////////////////
// Function prototypes
////////////////////////////////////////////////////////////////////////////////

// Initializes I2C Peripheral
void initI2C(I2C_TypeDef * I2Cx);

// Writes a byte (data) to the reg address
void writeI2C(I2C_TypeDef * I2Cx, uint8_t reg, uint8_t data);

// Reads and returns a byte from reg address
uint8_t readI2C(I2C_TypeDef * I2Cx, uint8_t reg);

// Configures the acceleromter to
void setUpAccelerometer(I2C_TypeDef * I2Cx);

// Returns 1 if motion detected on accelerometer; 0 otherwise
uint8_t detectMotion(I2C_TypeDef * I2Cx);

#endif
```

```
// STM32F401RE_I2C.c
#include "STM32F401RE_I2C.h"

void initI2C(I2C_TypeDef * I2Cx) {

    // Set FREQ to 10 MHz
    I2Cx->CR2 |= 10 << I2C_CR2_FREQ_Pos;

    // Configure Clock control register to Fast Mode with 400 kHz as SCL
    I2Cx->CCR |= (1 << I2C_CCR_FS_Pos | 120 << I2C_CCR_CCR_Pos);

    // Using max rise time to be 1000 ns, configure TRISE to be 10 + 1
    I2Cx->TRISE &= ~(I2C_TRISE_TRISE_Msk);
    I2Cx->TRISE |= 11 << I2C_TRISE_TRISE_Pos;

    // Enable I2C peripheral
    I2Cx->CR1 |= (1 << I2C_CR1_PE_Pos);
```

```
}

void writeI2C(I2C_TypeDef * I2Cx, uint8_t reg, uint8_t data) {
    // Set START bit
    I2Cx->CR1 |= 1 << I2C_CR1_START_Pos;

    // Wait until SB bit is set by hardware
    while (!(I2Cx->SR1 & I2C_SR1_SB_Msk));

    // Send device address with LSB as 0, indicating a write
    uint8_t address = DEVICE << 1;
    I2Cx->DR = address;

    // Wait until ADDR bit is set
    while (!(I2Cx->SR1 & I2C_SR1_ADDR_Msk));
    // Read SR2 to check if bus is in transmission mode
    if (!(I2Cx->SR2 & I2C_SR2_TRA)) {
        return writeI2C(I2Cx, reg, data);    // If TRA indicates receiver, start write over
    }

    // Push device register address into DR register
    I2Cx->DR = reg;

    // Wait until data register is empty
    while (!(I2Cx->SR1 & I2C_SR1_TXE_Msk));

    // Push data to register
    I2Cx->DR = data;

    // Wait until data register is empty
    while (!(I2Cx->SR1 & I2C_SR1_TXE_Msk));

    // Set the STOP bit
    I2Cx->CR1 |= 1 << I2C_CR1_STOP_Pos;
}

uint8_t readI2C(I2C_TypeDef * I2Cx, uint8_t reg) {
    // Set START bit
    I2Cx->CR1 |= 1 << I2C_CR1_START_Pos;

    // Wait until SB bit is set by hardware
    while (!(I2Cx->SR1 & I2C_SR1_SB_Msk));

    // Send device address with LSB as 0, indicating a write
    uint8_t address = DEVICE << 1;
    I2Cx->DR = address;
```

```
    // Wait until ADDR bit is set
    while (!(I2Cx->SR1 & I2C_SR1_ADDR_Msk));
    // Read SR2 to check if bus is in transmission mode
    if (!(I2Cx->SR2 & I2C_SR2_TRA)) {
        return readI2C(I2Cx, reg);    // If TRA indicates receiver, start write over
    }

    // Push device register address into DR register
    I2Cx->DR = reg;

    // Wait until data register is empty
    while (!(I2Cx->SR1 & I2C_SR1_TXE_Msk));

    ////////////////////////////////////////////////////////////

    // Initiate another start pulse
    I2Cx->CR1 |= 1 << I2C_CR1_START_Pos;

    // Wait until SB bit is set by hardware
    while (!(I2Cx->SR1 & I2C_SR1_SB_Msk));

    // Send device address with LSB as 1, indicating a read
    address = DEVICE << 1;
    I2Cx->DR = address + 1;

    // Wait until ADDR bit is set
    while (!(I2Cx->SR1 & I2C_SR1_ADDR_Msk));
    // Read SR2 to check if bus is in receiver mode
    if (I2Cx->SR2 & I2C_SR2_TRA) {
        return readI2C(I2Cx, reg);    // If TRA indicates transmitter, start write over
    }

    // Generates nonacknowledge pulse and stop pulse
    I2Cx->CR1 |= 1 << I2C_CR1_STOP_Pos;

    // Wait until full byte received
    while (!(I2Cx->SR1 & I2C_SR1_RXNE_Msk));

    volatile uint8_t result = I2Cx->DR & I2C_DR_DR_Msk;

    return result;
}

void setUpAccelerometer(I2C_TypeDef * I2Cx) {
    writeI2C(I2Cx, 0x2a, 0x18);     // Place device in Standby mode at 100Hz ODR
```

```
    writeI2C(I2Cx, 0x15, 0xf8);     // Configure motion detection with ELE latch
    writeI2C(I2Cx, 0x17, 0x18);     // Set threshold value to be > 1.5g
    writeI2C(I2Cx, 0x18, 0x0a);     // Set debounce counter to 10 counts
    writeI2C(I2Cx, 0x2a, 0x19);     // Put device in Active mode
}

uint8_t detectMotion(I2C_TypeDef * I2Cx) {
    volatile uint8_t reg = readI2C(I2Cx, 0x16);
    return reg >> 7;
}
```

```
// STM32F401RE_RCC.h
// Header for RCC functions

#ifndef STM32F4_RCC_H
#define STM32F4_RCC_H

#include <stdint.h>
#include "stm32f4xx.h"

////////////////////////////////////////////////////////////////////////////////
// Definitions
////////////////////////////////////////////////////////////////////////////////

// Global defines related to clock
uint32_t SystemCoreClock;    // Updated by configureClock()
#define HSE_VALUE 8000000    // Value of external input to OSC from ST-LINK

// PLL
#define PLLSRC_HSI 0
#define PLLSRC_HSE 1

// Clock configuration
#define SW_HSI  0
#define SW_HSE  1
#define SW_PLL  2

////////////////////////////////////////////////////////////////////////////////
// Function prototypes
////////////////////////////////////////////////////////////////////////////////

void configurePLL();
void configureClock();

#endif
```

```
// STM32F401RE_RCC.c
// Source code for RCC functions

#include "STM32F401RE_RCC.h"

void configurePLL() {
    // Set clock to 84 MHz
    // Output freq = (src_clk) * (N/M) / P
    // (8 MHz) * (336/8) / 4 = 84 MHz
    // M:8, N:336, P:4, Q:7
    // Use HSE as PLLSRC

    RCC->CR &= ~(RCC_CR_PLLON_Msk);     // Turn off PLL

    while ((RCC->CR & RCC_CR_PLLRDY_Msk) != 0);   // Wait till PLL is unlocked

    // Load configuration
    RCC->PLLCFGR &= ~(RCC_PLLCFGR_PLLSRC_Msk |
                    RCC_PLLCFGR_PLLM_Msk | RCC_PLLCFGR_PLLN_Msk |
                    RCC_PLLCFGR_PLLP_Msk | RCC_PLLCFGR_PLLQ_Msk);

    RCC->PLLCFGR |= (PLLSRC_HSE << RCC_PLLCFGR_PLLSRC_Pos |
                    8 << RCC_PLLCFGR_PLLM_Pos | 336 << RCC_PLLCFGR_PLLN_Pos |
                    0b01 << RCC_PLLCFGR_PLLP_Pos | 4 << RCC_PLLCFGR_PLLQ_Pos);


    // Enable PLL and wait until it's locked
    RCC->CR |= (1 << RCC_CR_PLLON_Pos);
    while((RCC->CR & RCC_CR_PLLRDY_Msk) == 0);
}

void configureClock(){
    /* Configure APB prescalers
    1. Set APB2 (high-speed bus) prescaler to no division
    2. Set APB1 (low-speed bus) to divide by 2.
    */

    RCC->CFGR &= ~(RCC_CFGR_PPRE1_Msk | RCC_CFGR_PPRE2_Msk);
    RCC->CFGR |= (0b100 << RCC_CFGR_PPRE1_Pos | 0b000 << RCC_CFGR_PPRE2_Pos);

    // Turn on and bypass for HSE from ST-LINK
    RCC->CR |= (1 << RCC_CR_HSEBYP_Pos | 1 << RCC_CR_HSEON_Pos);
    while(!(RCC->CR & RCC_CR_HSERDY_Msk));
```

```
    // Configure and turn on PLL for 84 MHz
    configurePLL();

    // Select PLL as clock source
    RCC->CFGR &= ~(RCC_CFGR_SW_Msk);
    RCC->CFGR |= (SW_PLL << RCC_CFGR_SW_Pos);
    while((RCC->CFGR & RCC_CFGR_SWS_Msk) != (0b10 << RCC_CFGR_SWS_Pos));

    SystemCoreClock = 84000000;
}
```

```
// STM32F401RE_SPI.h
// Header for SPI functions

#ifndef STM32F4_SPI_H
#define STM32F4_SPI_H

#include <stdint.h> // Include stdint header
#include "stm32f4xx.h"

#define DISPLAY_RESET   4
#define DISPLAY_DC      5
#define DISPLAY_CS      4


#define DISPLAY_WIDTH       84
#define DISPLAY_HEIGHT      48

#define PUSH_IT 1
#define SHOUT_IT 2
#define WIRE_IT 3
#define HEAT_IT 4
#define SHAKE_IT 5

//////////////////////////////////////////////////////////////////////////////
// Function prototypes
//////////////////////////////////////////////////////////////////////////////

/* Enables the SPI peripheral and intializes its clock speed (baud rate), polarity, and phas
 *    -- clkdivide: (0x01 to 0xFF). The SPI clk will be the master clock / clkdivide.
 *    -- cpol: clock polarity (0: inactive state is logical 0, 1: inactive state is logical
 *    -- cpha: clock phase (1: data changed on leading edge of clk and captured on next edge
 *          0: data captured on leading edge of clk and changed on next edge)
 * Note: the SPI mode register is set with the following unadjustable settings:
```

```
 *    -- Master mode
 *    -- Fixed peripheral select
 *    -- Chip select lines directly connected to peripheral device
 *    -- Mode fault detection enabled
 *    -- WDRBT disabled
 *    -- LLB disabled
 *    -- PCS = 0000 (Peripheral 0 selected), means NPCS[3:0] = 1110
 * Refer to the datasheet for more low-level details. */
// Display clk freq - between 0-4 MHz
void spiInit(uint32_t clkdivide, uint32_t cpol, uint32_t ncpha);

/* Send reset pulse to board */
// void displayInit();

/* Write a single byte to the board */
void displaySend(uint8_t command, uint8_t send);

/* Write a single bit on the display */
void writePixel(uint8_t x, uint8_t y, uint8_t val);

/* Use bitmaps defined in bitmaps.h to write
 * commands to the display
 */
void writeCommand(uint8_t command);

/* Takes in a string and writes the corresponding message to the display
 * Valid commands are:
 *      WELCOME
 *      READY
 *      START
 *      GAME OVER
 */
void writeMessage(char *message);

/* Set all pixels to 0 */
void clearDisplay();

/* Update all pixels on the display */
void updateDisplay();

/* Writes a single digit to the location specified by x y */
void writeDigit(int val, int x, int y);

/* Writes a maximum 3 digit number to the location specified by x y */
void writeScore(int val, int x, int y);
```

```
/* Transmits a character (1 byte) over SPI and returns the received character.
 *    -- send: the character to send over SPI
 *    -- return: the character received over SPI */
uint8_t spiSendReceive(uint8_t send);

/* Transmits a short (2 bytes) over SPI and returns the received short.
 *    -- send: the short to send over SPI
 *    -- return: the short received over SPI */
uint16_t spiSendReceive16(uint16_t send);

#endif
```

```
// bitmaps.h
// Contains definitions for the LCD Messages
#ifndef BITMAPS_H
#define    BITMAPS_H

#include <stdint.h>
#include "stm32f4xx.h"

const char Shake_it_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0x20, 0x20, 0x20, 0x20, 0x
0x00, 0x00, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x20, 0x60, 0x80, 0x00, 0x00, 0x00, 0x00, 0xE0, 0xE0, 0x00, 0x00, 0x80, 0x40, 0x20, 0x00, 0x
0x00, 0xE0, 0x20, 0x20, 0x20, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x00, 0x
0x20, 0x20, 0x20, 0xE0, 0xE0, 0x20, 0x20, 0x20, 0x00, 0x00, 0xE0, 0x60, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x41, 0x43, 0x
0x46, 0x44, 0x38, 0x00, 0x00, 0x00, 0x7F, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x7F, 0x00, 0x
0x00, 0x70, 0x1E, 0x11, 0x10, 0x10, 0x1F, 0x3C, 0x60, 0x00, 0x00, 0x7F, 0x7F, 0x00, 0x06, 0x
0x30, 0x60, 0x00, 0x00, 0x00, 0x7F, 0x42, 0x42, 0x42, 0x42, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4F, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xE0, 0xF8, 0xFC, 0x3E, 0x
0x02, 0x00, 0x00, 0xC0, 0xE0, 0xE0, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x
0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x
0xE0, 0xC0, 0x00, 0x00, 0x00, 0x0C, 0x3C, 0xFC, 0xF8, 0xE0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0xFF, 0xFF, 0x
0x03, 0x00, 0x00, 0x00, 0xE0, 0xFC, 0xFF, 0x7F, 0x0F, 0x01, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x
0x00, 0x00, 0x00, 0x01, 0x07, 0xFF, 0xFF, 0xF8, 0xA0, 0x00, 0x00, 0x00, 0x07, 0xFF, 0xFF, 0x
0xB0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x7F, 0xFF, 0xFF, 0xE0, 0x00, 0x00, 0x00, 0x04, 0xFF, 0xFF, 0xFF, 0x80, 0x00, 0x00, 0x00, 0x
```

```
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFB, 0xFF, 0xFF, 0x3F, 0x00, 0x00, 0x
0xC0, 0xFF, 0xFF, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x07, 0x07, 0x00, 0x00, 0x00, 0x00, 0x01, 0x07, 0x1F, 0x
0x1C, 0x00, 0x00, 0x00, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x
0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x18, 0x3F, 0x1F, 0x1F, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char Shout_it_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xE0, 0x10, 0x10, 0x10, 0x10, 0x00, 0x00, 0x
0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x00, 0x00, 0x00, 0xC0, 0x60, 0x10, 0x10, 0x
0x10, 0x20, 0xE0, 0x80, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x
0x10, 0x10, 0x10, 0xF0, 0x10, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0x
0x00, 0x10, 0x10, 0x10, 0xF0, 0xF0, 0x10, 0x10, 0x10, 0x00, 0x00, 0xF0, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x21, 0x21, 0x23, 0x
0x1C, 0x00, 0x00, 0x00, 0x3F, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x3F, 0x00, 0x00, 0x00, 0x
0x18, 0x20, 0x20, 0x20, 0x20, 0x10, 0x0F, 0x07, 0x00, 0x00, 0x0F, 0x1F, 0x20, 0x20, 0x20, 0x
0x1F, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x3F, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0xF8, 0x
0xC0, 0x00, 0x00, 0x00, 0x00, 0x0E, 0x7E, 0xFE, 0xF8, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0E, 0xFE, 0xFC, 0xF8, 0x80, 0x
0x00, 0x00, 0x03, 0xFF, 0xFF, 0xFF, 0x90, 0x00, 0x00, 0x00, 0x00, 0x03, 0xFF, 0xFF, 0xFF, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x60, 0x
0xFF, 0x7F, 0x0B, 0x00, 0x00, 0x00, 0xE0, 0xFF, 0xFF, 0xFF, 0x07, 0x00, 0x00, 0x00, 0x00, 0x
0xFF, 0xFF, 0xFF, 0x13, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x7E, 0x7F, 0x3F, 0x07, 0x00, 0x00, 0x
0x00, 0xE0, 0xFC, 0xFF, 0x3F, 0x0F, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
```

```
const char Heat_it_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0xF0, 0x30, 0x30, 0x30, 0x30, 0x
0x00, 0x00, 0x00, 0x80, 0xF0, 0x30, 0xF0, 0x80, 0x00, 0x00, 0x30, 0x30, 0x30, 0xF0, 0xF0, 0x
0x30, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x00, 0x00, 0x00, 0x30, 0x30, 0x30, 0x
0x30, 0x30, 0x30, 0x00, 0x00, 0xF0, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x7F, 0x03, 0x03, 0x03, 0x03, 0x03, 0x7F, 0x7F, 0x00, 0x00, 0x00, 0x7F, 0x
0x63, 0x63, 0x63, 0x00, 0x00, 0x60, 0x7C, 0x1F, 0x18, 0x18, 0x18, 0x1F, 0x7C, 0x60, 0x00, 0x
0x00, 0x7F, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x6F, 0x60, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0xC0, 0xE0, 0xF0, 0xF8, 0x7C, 0x1C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x70, 0xF0, 0x
0xC0, 0x00, 0x00, 0x1F, 0x7F, 0xFF, 0xF1, 0xE0, 0xC0, 0x00, 0x00, 0x00, 0xC0, 0xE0, 0xF0, 0x
0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0xF8, 0xFF, 0xFF, 0xEF, 0xF1, 0x70, 0x78, 0x7F, 0x7F, 0x7F, 0x1F, 0x0F, 0x0F, 0x00, 0x
0xFF, 0xFF, 0xFD, 0xFE, 0xCE, 0x0E, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x07, 0x0F, 0x1E, 0x1C, 0x00, 0x00, 0x07, 0x1F, 0x
0x3C, 0x38, 0x38, 0x3F, 0x1F, 0x1F, 0x1F, 0x07, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char Push_it_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x
0x10, 0x10, 0x10, 0xE0, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x00, 0x
0x00, 0xE0, 0xF0, 0x10, 0x10, 0x10, 0x10, 0x00, 0x00, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x
0xF0, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF0, 0x00, 0x00, 0x10, 0x10, 0x10, 0x
0xF0, 0x10, 0x10, 0x10, 0x00, 0x00, 0xF0, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x3F, 0x02, 0x02, 0x02, 0x02, 0x01, 0x00, 0x00, 0x01, 0x1F, 0x30, 0x20, 0x20, 0x
0x10, 0x0F, 0x00, 0x00, 0x00, 0x20, 0x21, 0x21, 0x22, 0x3E, 0x1C, 0x00, 0x00, 0x3F, 0x3F, 0x
0x01, 0x01, 0x01, 0x01, 0x3F, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x00, 0x
0x00, 0x00, 0x00, 0x3F, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x27, 0x20, 0x00, 0x00, 0x00, 0x
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0x
0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0x
0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0xC0, 0xE0, 0xE0, 0xE8, 0xFC, 0xFC, 0xFC, 0xFC, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC, 0xFC, 0x
0xEC, 0xE0, 0xE0, 0xE0, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};


const char Wire_it_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0xF0, 0x
0x00, 0x00, 0xE0, 0x38, 0xF8, 0x80, 0x00, 0x00, 0xE0, 0x78, 0x00, 0x00, 0xF8, 0xF8, 0x00, 0x
0x00, 0xF8, 0x18, 0x18, 0x18, 0x18, 0xF0, 0x60, 0x00, 0x00, 0x00, 0xF8, 0x98, 0x98, 0x98, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x00, 0x00, 0x18, 0x18, 0x18, 0xF8, 0xF8, 0x18, 0x
0x18, 0x00, 0x00, 0xF8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x07, 0x3E, 0x38, 0x1F, 0x01, 0x00, 0x00, 0x0F, 0x30, 0x38, 0x07, 0x00, 0x00, 0x
0x3F, 0x3F, 0x00, 0x00, 0x00, 0x3F, 0x03, 0x03, 0x03, 0x07, 0x1D, 0x30, 0x20, 0x00, 0x00, 0x
0x31, 0x31, 0x31, 0x31, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x
0x3F, 0x3F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x37, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xF8, 0x
0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0xFE, 0x3E, 0x0E, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFC, 0xFC, 0xFC, 0x7C, 0x3C, 0x1C, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xF8, 0xFF, 0x7F, 0x3F, 0x0F, 0x07, 0x03, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x08, 0x07, 0x03, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char Welcome_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0xF8, 0x08, 0x08, 0x08, 0x08, 0x08, 0xF0, 0x40, 0x00, 0x00, 0x00, 0xF8, 0x08, 0x
0x08, 0x08, 0x08, 0x10, 0x60, 0x00, 0x00, 0x00, 0x00, 0xF8, 0x08, 0x08, 0x08, 0x08, 0x08, 0x
0x00, 0x00, 0x00, 0xF0, 0x80, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00, 0x00, 0x00, 0x60, 0x90, 0x
0x08, 0x08, 0x08, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x02, 0x02, 0x02, 0x02, 0x00, 0x01, 0x00, 0x00, 0x
0x00, 0x7F, 0x02, 0x02, 0x02, 0x02, 0x06, 0x18, 0x20, 0x40, 0x00, 0x00, 0x00, 0x7F, 0x41, 0x
0x41, 0x41, 0x41, 0x00, 0x00, 0x00, 0x00, 0x40, 0x41, 0x41, 0x43, 0x42, 0x24, 0x18, 0x00, 0x
0x00, 0x40, 0x40, 0x41, 0x43, 0x42, 0x02, 0x3C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xC6, 0xC6, 0xC6, 0xC6, 0x
0xBC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x0C, 0x0C, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x
0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x
0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x9F, 0x
0x98, 0x18, 0x18, 0x08, 0x8F, 0x82, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x1F, 0x00, 0x
0x00, 0x00, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x
0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x
0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0xFF, 0x01, 0x01, 0x01, 0x01, 0x00, 0x00, 0xF8, 0x9E, 0x
0x01, 0x01, 0x01, 0x01, 0x01, 0x03, 0x06, 0xFC, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1E, 0x
0x31, 0x71, 0x61, 0xE1, 0xC1, 0x80, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0xFF, 0x01, 0x01, 0x
0x01, 0x00, 0x00, 0xC0, 0xF8, 0x8E, 0x81, 0x83, 0x8F, 0xF8, 0xE0, 0x00, 0x00, 0x00, 0x00, 0x
0xFF, 0x61, 0x61, 0x61, 0x61, 0xE1, 0x9F, 0x0E, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0xFF, 0x
0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x03, 0x06, 0x04, 0x0C, 0x0C, 0x0C, 0x0C, 0x06, 0x03, 0x01, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0x06, 0x07, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x0F, 0x00, 0x00, 0x00, 0x00, 0x08, 0x0E, 0x07, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x03, 0x
0x08, 0x00, 0x00, 0x00, 0x0F, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x0E, 0x0C, 0x00, 0x00, 0x
```

```
0x00, 0x00, 0x0F, 0x0F, 0x00, 0x00, 0x00, 0x00,
};


const char Ready_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0xFE, 0xFE, 0x86, 0x86, 0x86, 0x86, 0xFE, 0x7C, 0x38, 0x00, 0x00, 0x00, 0xFE, 0x
0xC6, 0xC6, 0xC6, 0xC6, 0xC6, 0x00, 0x00, 0x00, 0x00, 0x80, 0xF0, 0x7C, 0x0E, 0x06, 0x7E, 0x
0x80, 0x00, 0x00, 0x00, 0x00, 0xFE, 0xFE, 0xFE, 0x06, 0x06, 0x06, 0x06, 0x0E, 0x1C, 0xFC, 0x
0x00, 0x00, 0x06, 0x1E, 0x7E, 0xF0, 0xC0, 0x80, 0xF0, 0x3C, 0x1E, 0x06, 0x00, 0x00, 0x06, 0x
0x86, 0xC6, 0x7E, 0x3C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x3F, 0x01, 0x01, 0x01, 0x07, 0x0E, 0x3C, 0x38, 0x
0x00, 0x00, 0x3F, 0x3F, 0x30, 0x30, 0x30, 0x30, 0x30, 0x00, 0x00, 0x30, 0x3C, 0x1F, 0x07, 0x
0x06, 0x06, 0x06, 0x07, 0x1F, 0x3E, 0x30, 0x00, 0x00, 0x3F, 0x3F, 0x3F, 0x30, 0x30, 0x30, 0x
0x18, 0x1C, 0x0F, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x3F, 0x3F, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x13, 0x3B, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char Start_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x60, 0xF8, 0xF8, 0x8C, 0x8C, 0x0C, 0x0C, 0x0C, 0x00, 0x00, 0x0C, 0x
0x0C, 0x0C, 0xFC, 0xFC, 0x0C, 0x0C, 0x0C, 0x0C, 0x00, 0x00, 0x80, 0xE0, 0x7C, 0x0C, 0x3C, 0x
0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0xFC, 0x0C, 0x0C, 0x0C, 0x0C, 0x9C, 0xF8, 0x
0x00, 0x00, 0x0C, 0x0C, 0x0C, 0xFC, 0xFC, 0xFC, 0x0C, 0x0C, 0x0C, 0x00, 0x00, 0x00, 0xFC, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x60, 0x61, 0x61, 0x63, 0x63, 0x3F, 0x
0x0C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x60, 0x7C, 0x3F, 0x
0x0C, 0x0C, 0x0C, 0x0D, 0x0F, 0x7F, 0x78, 0x40, 0x00, 0x00, 0x00, 0x7F, 0x7F, 0x03, 0x03, 0x
0x07, 0x1F, 0x3C, 0x70, 0x60, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x7F, 0x7F, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x77, 0x77, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

const char Game_over_bitmap [] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0xC0, 0x30, 0x08, 0x00, 0x04, 0x04, 0x04, 0x04, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x20, 0x04, 0x04, 0x18, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0x0C, 0x0C, 0x30, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0xE4, 0xFC, 0x00, 0x00, 0x00, 0x00, 0xFC, 0x84, 0x
0x84, 0x84, 0x84, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x03, 0x0C, 0x10, 0x00, 0x20, 0x20, 0x21, 0x21, 0x21, 0x1F, 0x00, 0x
0x00, 0x20, 0x1C, 0x07, 0x04, 0x04, 0x04, 0x04, 0x04, 0x07, 0x38, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x01, 0x04, 0x38, 0x18, 0x02, 0x00, 0x00, 0x00, 0x3F, 0x3F, 0x00, 0x00, 0x
0x00, 0x3F, 0x20, 0x20, 0x20, 0x20, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE0, 0x30, 0x18, 0x
0x0C, 0x0C, 0x0C, 0x08, 0x18, 0x70, 0xC0, 0x00, 0x00, 0x04, 0x1C, 0xE0, 0x00, 0x00, 0x00, 0x
0x00, 0xC0, 0x38, 0x04, 0x00, 0x00, 0x00, 0xFC, 0x8C, 0x8C, 0x8C, 0x8C, 0x8C, 0x00, 0x00, 0x
0x00, 0xFC, 0x0C, 0x0C, 0x0C, 0x0C, 0x0C, 0xF8, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
```

```
0x0F, 0x18, 0x30, 0x20, 0x60, 0x60, 0x60, 0x20, 0x30, 0x1C, 0x07, 0x00, 0x00, 0x00, 0x00, 0x
0x07, 0x38, 0x70, 0x70, 0x0E, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x61, 0x61, 0x61, 0x
0x61, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x03, 0x03, 0x03, 0x03, 0x05, 0x18, 0x70, 0x40, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x
0x04, 0x04, 0x04, 0xC4, 0x24, 0x34, 0x34, 0x34, 0x04, 0x04, 0x04, 0x04, 0x84, 0x44, 0x24, 0x
0x34, 0x34, 0x34, 0x04, 0x04, 0x04, 0xC4, 0x64, 0x24, 0x34, 0x34, 0x34, 0x24, 0x44, 0x04, 0x
0x04, 0x04, 0xF4, 0x34, 0x34, 0x34, 0x34, 0x24, 0xC4, 0x04, 0x04, 0x04, 0x04, 0xF4, 0x34, 0x
0x34, 0x34, 0x04, 0x04, 0x04, 0x84, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x
0x04, 0x04, 0x04, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x61, 0x63, 0x63, 0x66, 0x26, 0x1C, 0x00, 0x00, 0x
0x0F, 0x30, 0x20, 0x60, 0x60, 0x60, 0x60, 0x00, 0x00, 0x00, 0x1D, 0x30, 0x20, 0x60, 0x60, 0x
0x20, 0x10, 0x07, 0x00, 0x00, 0x00, 0x7F, 0x06, 0x06, 0x06, 0x06, 0x1B, 0x20, 0x40, 0x00, 0x
0x00, 0x7F, 0x63, 0x63, 0x63, 0x60, 0x00, 0x00, 0x00, 0x73, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};


#endif
```

```
// STM32F401RE_SPI.c
// SPI function declarations

#include "STM32F401RE_SPI.h"
#include "STM32F401RE_RCC.h"
#include "STM32F401RE_GPIO.h"
#include "bitmaps.h"
// #include "main.h"


char DISPLAYMEM[DISPLAY_WIDTH * DISPLAY_HEIGHT] = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

/* Enables the SPI peripheral and intializes its clock speed (baud rate), polarity, and phas
 *    -- br: (0b000 - 0b111). The SPI clk will be the master clock / 2^(BR+1).
 *    -- cpol: clock polarity (0: inactive state is logical 0, 1: inactive state is logical
 *    -- ncpha: clock phase (0: data changed on leading edge of clk and captured on next edg
 *          1: data captured on leading edge of clk and changed on next edge)
 * Note: the SPI mode register is set with the following unadjustable settings:
 *    -- Master mode
 *    -- Fixed peripheral select
 *    -- Chip select lines directly connected to peripheral device
 *    -- Mode fault detection enabled
 *    -- WDRBT disabled
 *    -- LLB disabled
 *    -- PCS = 0000 (Peripheral 0 selected), means NPCS[3:0] = 1110
 * Refer to the datasheet for more low-level details. */
void spiInit(uint32_t br, uint32_t cpol, uint32_t cpha) {
    // Turn on GPIOA and GPIOB clock domains (GPIOAEN and GPIOBEN bits in AHB1ENR)
    RCC->AHB1ENR |= (RCC_AHB1ENR_GPIOAEN | RCC_AHB1ENR_GPIOBEN);
```

```
    RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // Turn on SPI1 clock domain (SPI1EN bit in APB2ENR)
    // Initially assigning SPI pins
    pinMode(GPIOB, 3, GPIO_ALT); // PB3, Arduino D13, SPI1_SCK
    pinMode(GPIOA, 7, GPIO_ALT); // PA7, Arduino D11, SPI1_MOSI
    pinMode(GPIOA, DISPLAY_CS, GPIO_OUTPUT); // Default: PA4, Arduino A2, Manual CS
    pinMode(GPIOB, DISPLAY_DC, GPIO_OUTPUT); // Default: PB5, Arduino D4, Manual D/C select
    pinMode(GPIOB, DISPLAY_RESET, GPIO_OUTPUT); // Default: PB4, Arduino D5, Manual Reset (I
    // pinMode(GPIOA, 0, GPIO_OUTPUT);

    // Set to AF05 for SPI alternate functions
    GPIOA->AFR[0] |= (5 << GPIO_AFRL_AFSEL7_Pos);
    GPIOB->AFR[0] |= (5 << GPIO_AFRL_AFSEL3_Pos);
    // Set all relevant bits to 0
    SPI1->CR1 &= ~(SPI_CR1_BR_Msk | SPI_CR1_CPOL_Msk | SPI_CR1_CPHA_Msk | SPI_CR1_LSBFIRST_M
                SPI_CR1_DFF_Msk | SPI_CR1_SSM_Msk | SPI_CR1_MSTR_Msk | SPI_CR1_SPE_Msk);

    // Set clock divisor, polarity, phase, 8 bit format, put SPI in master mode
    SPI1->CR1 |= (br << SPI_CR1_BR_Pos | cpol << SPI_CR1_CPOL_Pos | cpha << SPI_CR1_CPHA_Pos
                 0 << SPI_CR1_DFF_Pos | 1 << SPI_CR1_MSTR_Pos);
    // Set NSS pin to output mode
    SPI1->CR2 |= SPI_CR2_SSOE;
    // Turn on SSM
    SPI1->CR1 |= SPI_CR1_SSM;
    // Enable SPI
    SPI1->CR1 |= SPI_CR1_SPE;
}


void displaySend(uint8_t command, uint8_t send) {
    digitalWrite(GPIOB, DISPLAY_DC, command);
    digitalWrite(GPIOA, DISPLAY_CS, 0);
    // SPI1->CR1 |= SPI_CR1_SPE;
    SPI1->DR = send;

    // Wait until message has been transmitted
    while(!(SPI1->SR & SPI_SR_RXNE));

    // Read to clear the RXNE flag
    volatile uint16_t reg = SPI1->DR;

    // SPI1->CR1 &= ~(SPI_CR1_SPE_Msk);
    digitalWrite(GPIOA, DISPLAY_CS, 1);
}

void writePixel(uint8_t x, uint8_t y, uint8_t val) {
    if(val) DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] |= val << (y % 8);
```

```
    else DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] &= ~(1 << (y % 8));
}

void writeCommand(uint8_t command) {
    switch (command) {
        case PUSH_IT:
            memcpy(DISPLAYMEM, Push_it_bitmap, sizeof(DISPLAYMEM));
            break;
        case WIRE_IT:
            memcpy(DISPLAYMEM, Wire_it_bitmap, sizeof(DISPLAYMEM));
            break;
        case HEAT_IT:
            memcpy(DISPLAYMEM, Heat_it_bitmap, sizeof(DISPLAYMEM));
            break;
        case SHAKE_IT:
            memcpy(DISPLAYMEM, Shake_it_bitmap, sizeof(DISPLAYMEM));
            break;
        case SHOUT_IT:
            memcpy(DISPLAYMEM, Shout_it_bitmap, sizeof(DISPLAYMEM));
            break;
        default:
            break;
    }
}

void writeMessage(char *message) {
    if(strcmp(message, "WELCOME") == 0) memcpy(DISPLAYMEM, Welcome_bitmap, sizeof(DISPLAYMEN
    else if(strcmp(message, "START") == 0) memcpy(DISPLAYMEM, Start_bitmap, sizeof(DISPLAYME
    else if(strcmp(message, "READY") == 0) memcpy(DISPLAYMEM, Ready_bitmap, sizeof(DISPLAYME
    else if(strcmp(message, "GAME OVER") == 0) memcpy(DISPLAYMEM, Game_over_bitmap, sizeof(D
}

void clearDisplay() {
    int x, y;
    for (x = 0; x < DISPLAY_WIDTH; ++x) {
        for (y = 0; y < DISPLAY_HEIGHT; ++y) {
            writePixel(x, y, 0);
        }
    }
    updateDisplay();
}

void updateDisplay() {
    // Set X address of RAM to 0
    displaySend(0, 0b10000000);
    // Set Y address of RAM to 0
```

```
    displaySend(0, 0b01000000);
    // We divide by 8 since we write 8 bits at a time
    for(int i = 0; i < DISPLAY_HEIGHT * DISPLAY_WIDTH / 8; ++i) {
        displaySend(1, DISPLAYMEM[i]);
    }
}

void writeDigit(int val,int x,int y) {
    switch(val) {
        case 0:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0x81;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            break;
        case 1:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0x00;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            break;
        case 2:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0xFB;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0xDB;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xDF;
            break;
        case 3:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0xDB;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0xDB;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            break;
        case 4:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0x1F;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0x18;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            break;
        case 5:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0xDF;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0xDB;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFB;
            break;
        case 6:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0xDB;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFB;
            break;
        case 7:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0x03;
```

```
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0x03;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            break;
        case 8:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0x99;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            break;
        case 9:
            DISPLAYMEM[x + ((y / 8)*DISPLAY_WIDTH)] = 0xDF;
            DISPLAYMEM[x+1 + ((y / 8)*DISPLAY_WIDTH)] = 0xDB;
            DISPLAYMEM[x+2 + ((y / 8)*DISPLAY_WIDTH)] = 0xFF;
            break;
        default:
            break;
    }
}

void writeScore(int val, int x, int y) {
    displaySend(0, 0b10000000 + x);         // Set X address of Ram
    displaySend(0, 0b01000000 + y);         // Set Y address of Ram
    if(val / 100 > 0) {
        writeDigit(val / 100, x, y);
        x += 4;
    }
    if(val / 10 > 0) {
        writeDigit((val % 100) / 10, x, y);
        x += 4;
    }
    writeDigit(val % 10, x, y);
}

// /* Transmits a character (1 byte) over SPI and returns the received character.
//  *    -- send: the character to send over SPI
//  *    -- return: the character received over SPI */
// uint8_t spiSendReceive(uint8_t send) {
//     SPI1->DR.DR = send; // Transmit the character over SPI
//     while (!(SPI->SPI_SR.RDRF)); // Wait until data has been received
//     return (char) (SPI->SPI_RDR.RD); // Return received character

//     SPI1->DR = send;
//     while (!(SPI1->))
// }

/* Transmits a short (2 bytes) over SPI and returns the received short.
 *    -- send: the short to send over SPI
```

```
 *    -- return: the short received over SPI */
uint16_t spiSendReceive16(uint16_t send) {
    digitalWrite(GPIOB, 6, 0);
    SPI1->CR1 |= SPI_CR1_SPE;
    SPI1->DR = send;

    while(!(SPI1->SR & SPI_SR_RXNE));
    uint16_t rec = SPI1->DR & SPI_DR_DR_Msk;

    SPI1->CR1 &= ~(SPI_CR1_SPE_Msk);
    digitalWrite(GPIOB, 6, 1);

    return rec;
}
```

```
// STM32F401RE_TIM.h
// Header for TIM functions

#ifndef STM32F4_TIM_H
#define STM32F4_TIM_H

#include <stdint.h> // Include stdint header
#include "STM32F401RE_GPIO.h"
#include "stm32f4xx.h"

//////////////////////////////////////////////////////////////////////////////
// Definitions
//////////////////////////////////////////////////////////////////////////////
#define PUSH_IT 1
#define SHOUT_IT 2
#define WIRE_IT 3
#define HEAT_IT 4
#define SHAKE_IT 5
#define GAME_OVER 6
#define SUCCESS 7
#define START 8

//////////////////////////////////////////////////////////////////////////////
// Function prototypes
//////////////////////////////////////////////////////////////////////////////

void initTIM(TIM_TypeDef * TIMx);
void configMusicTIM(TIM_TypeDef * TIMx);
void setFreq(TIM_TypeDef * TIMx, int freq);
```

```
void delay_millis(TIM_TypeDef * TIMx, uint32_t ms);
void delay_micros(TIM_TypeDef * TIMx, uint32_t us);
void playMusic(int mode);

#endif
```

```
// STM32F401RE_TIM.c
// TIM functions

#include "STM32F401RE_TIM.h"
// #include "STM32F401RE_RCC.h"

const int success[][2] = {
  {2349, 125}, {2489, 125}, {3135, 125}, {0, 10}
};

const int gameOver[][2] = {
  {155, 375}, {0, 125}, {146, 125}, {0, 125},
  {138, 125}, {0, 125}, {155, 125}, {0, 125},
  {110, 125}, {0, 125}, {98, 500}, {0, 10}
};

const int start[][2] = {
  {262, 250}, {0, 125}, {330, 250}, {349, 125},
  {0, 125}, {330, 125}, {277, 250}, {0, 125},
  {330, 125}, {0, 125}, {392, 125}, {0, 125},
  {523, 250}, {0, 10}
};

const int heat[][2] = {
  {740, 125}, {0, 125}, {740, 125}, {0, 125},
  {740, 125}, {0, 125}, {740, 125}, {0, 125},
  {698, 500}, {784, 500}, {0, 10}
};

const int push[][2] = {
  {123, 125}, {0, 125}, {117, 125}, {131, 125},
  {0, 125}, {110, 250}, {0, 10}
};

const int shout[][2] = {
  {587, 500}, {1480, 125}, {1397, 125}, {1480, 125},
  {1397, 125}, {1480, 125}, {1397, 375}, {0, 10}
};
```

```
const int shake[][2] = {
  {392, 50}, {0, 100}, {392, 50}, {0, 100},
  {392, 50}, {0, 100}, {392, 50}, {0, 100},
  {392, 50}, {0, 100}, {392, 50}, {0, 100},
  {392, 50}, {0, 100}, {392, 50}, {0, 100},
};

const int wire[][2] = {
  {831, 125}, {0, 125}, {831, 125}, {0, 125},
  {466, 125}, {0, 125}, {523, 125}, {0, 375},
  {349, 125}, {0, 125}, {1047, 125}, {0, 125},
};

void initTIM(TIM_TypeDef * TIMx){
  // Set prescaler to give 1 µs time base
  uint32_t psc_div = (uint32_t) ((SystemCoreClock/1e6)-1);

  // Set prescaler division factor
  TIMx->PSC = (psc_div - 1);
  // Generate an update event to update prescaler value
  TIMx->EGR |= TIM_EGR_UG;
  // Enable counter
  TIMx->CR1 |= TIM_CR1_CEN; // Set CEN = 1
}

void delay_millis(TIM_TypeDef * TIMx, uint32_t ms){
  TIMx->ARR = ms*1000;// Set timer max count
  TIMx->EGR |= 1 << TIM_EGR_UG_Pos;     // Force update
  TIMx->SR &= ~(TIM_SR_UIF_Msk); // Clear UIF
  TIMx->CNT = 0;      // Reset count

  while(!(TIMx->SR & 1)); // Wait for UIF to go high
}

void delay_micros(TIM_TypeDef * TIMx, uint32_t us){
  TIMx->ARR = us;// Set timer max count
  TIMx->EGR |= 1 << TIM_EGR_UG_Pos;     // Force update
  TIMx->SR &= ~(TIM_SR_UIF_Msk); // Clear UIF
  TIMx->CNT = 0;      // Reset count

  while(!(TIMx->SR & 1)); // Wait for UIF to go high
}

void configMusicTIM(TIM_TypeDef * TIMx) {
  // Enable TIM2 counter and select CLK_INT as timer clock
```

```
    TIMx->CR1 |= 1;

    // Initialize TIM2 attributes
    TIMx->CCR1 = 5;
    TIMx->ARR = 4;
    TIMx->PSC = 2;

    // Config CH1
    TIMx->CCMR1 |= (0b111 << 4);       // Set PWM mode 2 -> (ARR >= CCR1) then PWM is on
    TIMx->CCER |= 1;                   // Enable CH1

    TIMx->CR1 |= (1 << 7);             // Allow ARR reg to be buffered
    TIMx->EGR |= 1;               // Initialize preloaded registers
}

void setFreq(TIM_TypeDef * TIMx, int freq) {
  // If freq is 0, set CCR1 to be greater than ARR
    if (freq == 0) {
        TIMx->CCR1 = TIMx->ARR + 1;
    } else {
        // Adjust ARR and CCR1 to create the intended freq
        int clk = 28000000;
        int divider = clk / freq;
        TIMx->ARR = divider;
        TIMx->CCR1 = divider / 2;
    }
}

// Play music through the TIM5 timer based on which mode
void playMusic(int mode) {
  switch (mode) {
  case GAME_OVER:
    for (int i = 0; i < 12; ++i) {
      setFreq(TIM5, gameOver[i][0]);
      delay_millis(TIM2, gameOver[i][1]);
    }
    break;
  case SUCCESS:
    for (int i = 0; i < 4; ++i) {
      setFreq(TIM5, success[i][0]);
      delay_millis(TIM2, success[i][1]);
    }
    break;
  case START:
    for (int i = 0; i < 14; ++i) {
      setFreq(TIM5, start[i][0]);
```

```
      delay_millis(TIM2, start[i][1]);
    }
    break;
  case HEAT_IT:
    for (int i = 0; i < 11; ++i) {
      setFreq(TIM5, heat[i][0]);
      delay_millis(TIM2, heat[i][1]);
    }
    break;
  case PUSH_IT:
    for (int i = 0; i < 7; ++i) {
      setFreq(TIM5, push[i][0]);
      delay_millis(TIM2, push[i][1]);
    }
    break;
  case SHOUT_IT:
    for (int i = 0; i < 8; ++i) {
      setFreq(TIM5, shout[i][0]);
      delay_millis(TIM2, shout[i][1]);
    }
    break;
  case SHAKE_IT:
    for (int i = 0; i < 16; ++i) {
      setFreq(TIM5, shake[i][0]);
      delay_millis(TIM2, shake[i][1]);
    }
    break;
  case WIRE_IT:
    for (int i = 0; i < 12; ++i) {
      setFreq(TIM5, wire[i][0]);
      delay_millis(TIM2, wire[i][1]);
    }
    break;
  default:
    break;
  }
}
```

References

[1] SparkFun Electronics. Electret Microphone Breakout. https://www.sparkfun.com/products/12758.

[2] SparkFun Electronics. Graphic LCD 84x48 - Nokia 5110. https://www.sparkfun.com/products/10168.

[3] SparkFun Electronics. Redbot Sensor - Accelerometer. https://www.sparkfun.com/products/12589.

[4] Hasbro. Bop It! https://www.youtube.com/watch?v=q__gxzbbJns.

[5] Philips Semiconductors. Pcd85544 Data Sheet. https://www.sparkfun.com/datasheets/LCD/Monochrome/Nokia5110.pdf.

[6] Nick Tan and Andreas Roeseler. Final Project Proposal.