Team Project

Introduction

You will be creating the slave side board using PSoC 4 for a game that flashes sequences of LEDs that the player needs to repeat. The game starts by flashing one of four LEDs. If the player repeats the sequence by pressing the button corresponding to the LED that was flashed, the game then flashes the same LED followed by a second LED. The player has to then press the correct 2 button sequence. Then the game will add a third LED to the sequence and so on until the player either doesn't repeat the sequence correctly or the maximum sequence length has been reached.

The maximum sequence length can be adjusted by the master while the speed at which the game flashes the LEDs can be adjusted by the slave.

As you complete various sections of the game you must test and demonstrate its operation to one of the course instructors. There will be prizes given out for those who complete various parts of the game first. You should get the basic features working and validated before moving on to the optional challenges.

Hint: most, if not all of the functionality can be tested using the Cypress bridge control panel to emulate the master rather than using the true master. The provided shield board allows for the master to be disconnected so that the built in Pioneer I2C bridge can be used to interface with a PC.

System Overview

The system will be built using two PSoC 4 devices as shown in the following block diagram. The slave is on the Pioneer board while the master is on the provided shield. The master PSoC 4 controls game play while the slave PSoC 4 interacts with the player.

A schematic of the shield board will be provided. Hint: all pins that connect from the Pioneer board can be found in the “Pioneer Kit Connectors” section of the schematic. That should tell you everything you need to know to assign pins for your project. The relevant pin assignments are also provided at the end of this document.

The connections between the Pioneer kit and the shield are ground, SCL (I2C serial clock) and SDA (I2C serial data).

The master PSoC 4 is pre-programmed – you do <u>not</u> have to write the firmware for the master.

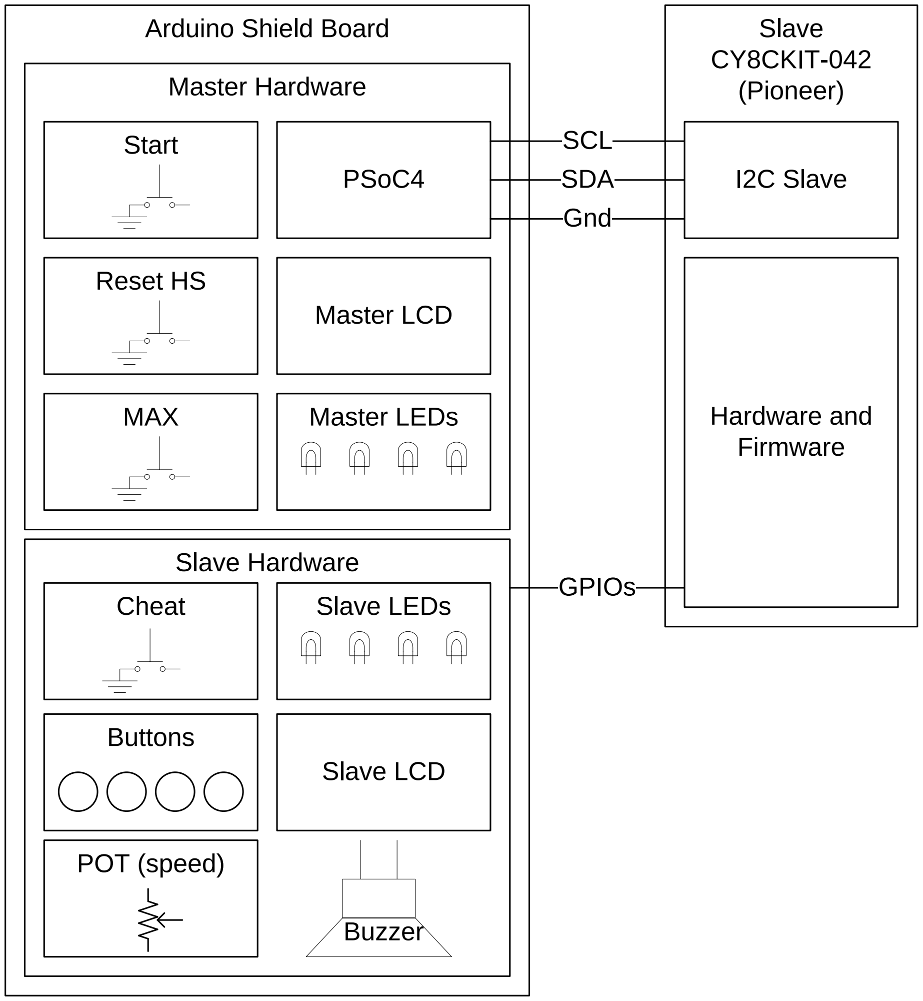

Figure 1: System Block Diagram

Shield Operation

The shield board is shown pictured here attached to a Pioneer kit:

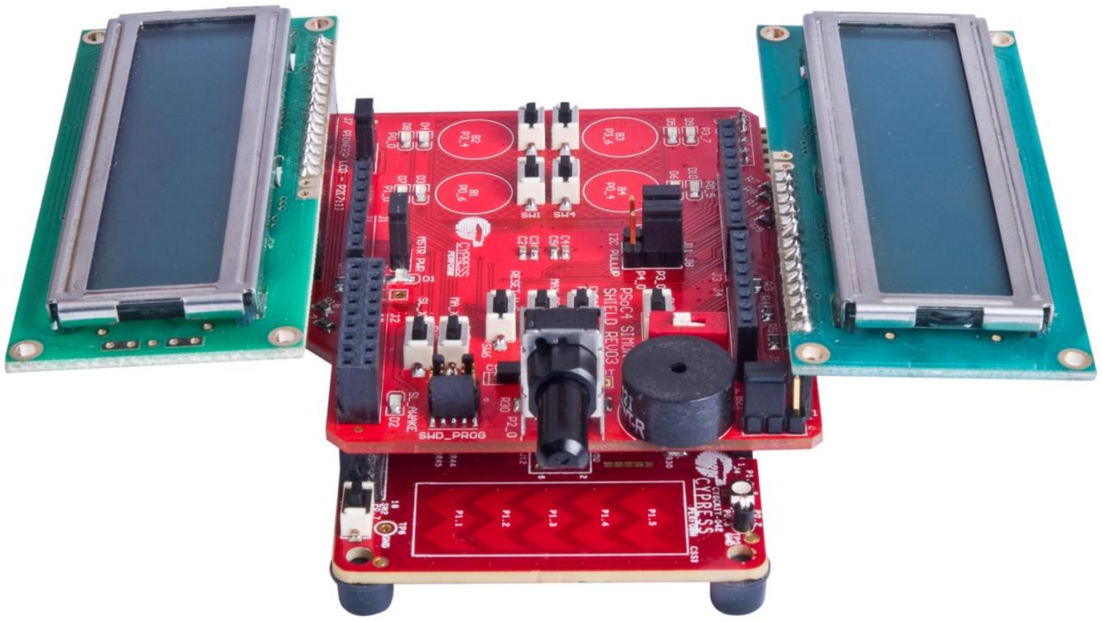

Figure 2: Shield Board Connected to Pioneer Kit

In addition to the connections to the Pioneer kit, the shield contains the following components:

- 2 Character LCDs
    - One LCD is connected to the master while the second LCD connects to pins on the slave.
    - The master LCD will show the following display.

```
RUN    #:  2  HS
SLOW MAX: 10  15
```

Figure 3: Master LCD Display Example

    The first line shows the game state (INIT, RUN, PASS, FAIL) and the current game's sequence length (2 in this case).
    The second line shows the speed (SLOW, MED, FAST), the sequence required to pass the game (10 in this case), and the high score (15 in this case).

- 10 LEDs
    - 1 LED is used to indicate that the master is powered. A jumper is provided to remove power from the master.
    - 4 LEDs are lit by the master during game play.

  - 4 LEDs are connected to the slave. These LEDs should be lit by the slave both when the master sends a sequence and when the player repeats the sequence. If the game is working properly, the two sets of LEDs (master and slave) should always match.
  - 1 additional LED called “SL_Awake” is connected to the slave. This can be used to indicate when the slave PSoC 4 is awake or asleep (if low power modes are implemented).
- 10 Switches (six connect to the PSoC 4 on the Pioneer board and four connect to the master)
  - There are 4 switches connected to the slave that are used during game play to repeat the pattern.
  - There are 2 reset switches – one for the master and one for the slave.
  - The “Cheat” switch connects to the slave and is used to toggle cheat mode on/off (this optional feature is described later).
  - The “Start” switch connects to the master and is used to tell the master to start a game.
  - The “Max Seq” switch connects to the master and is used to set the maximum sequence length needed for the user to “Win” the game. It will sequence between values of 5, 10, and 99.
  - The “Reset H/S” switch connects to the master and will reset the high score stored in the master. It must be pressed 4 times to reset the high score.
- 4 Capsense buttons
  - The capsense buttons connect to the same pins on the slave as the 4 game play switches. Therefore, either mechanical switches or capsense buttons may be implemented in the slave (but not both at the same time).
- 1 Buzzer
  - The buzzer connects to the slave and can be used for creating game sound effects.
- 1 Potentiometer
  - The potentiometer (POT) connects to the slave and can be used to set one of three possible speed values (if implemented).

Master and Slave Functional Overview

The master is already provided on the shield board and it handles the following functions. You do NOT need to implement the master.

- Starts game play.
- Creates the random LED sequences and flashes the LEDs at the specified speed.
- Checks to see if the sequence returned from the slave is correct or not.
- Keeps track of the game state (INIT, RUN, PASS, FAIL).
- Keeps track of the length of sequence correctly returned by the player.
- Stores the high score in non-volatile memory.
- Adjusts the maximum sequence required to pass the game.

→ The slave must provide the following **REQUIRED** functions. You need to implement the slave.

- Monitor button presses and store their state in the appropriate I2C register. The I2C register should always reflect the current state of the button presses. The button register should retain a button value as long as a given button is being held but must be 0x00 whenever no button is being pressed. If more than one button is pressed at the same time, the register bit for each button pressed should be set (e.g. if the first two buttons are pressed, the register should contain 0x03). Some sort of debounce should be used on the buttons – you can use either firmware or hardware debounce (or experiment with both).
- Light LEDs on the shield board when buttons are pressed.
- Light LEDs on the shield board corresponding to the master LED sequence. That is, you must monitor the info register and light the LED corresponding to any bit set in the register.

  Note: It must be possible to restart the game after a pass or fail condition without having to reset the kit.

→ The slave should provide the following **OPTIONAL** functions:

- Sound a buzzer for different button presses and when the player fails. See “Optional Slave Features” section for more details.
- Display game status information on a character LCD. What you display and the format is up to you. See “Optional Slave Features” section for more details.
- Send game speed control information to the master by reading the voltage provided by the POT or by using the CapSense slider on the Pioneer board. See “Optional Slave Features” section for more details.
- Use CapSense buttons instead of mechanical buttons. See “Optional Slave Features” section for more details.
- Provide a low power mode (e.g. deep sleep) that goes to sleep when the game is not running and wakes up on an I2C address match. See “Optional Slave Features” section for more details.
- Provide a “cheat” mode in which the slave sends back the correct sequence on its own. See “Optional Slave Features” section for more details.
- Convert the project to run on a PSoC 5LP using a CY8CKIT-050B kit. See “Optional Slave Features” section for more details.

Project Objective

Build the slave side system as described. The shield board and master firmware will be provided for you to test your design. The optional tasks may be done in any order.

Detailed Requirements

Power supply = 3.3V.

I2C interface: 100kbps, address: 0x36, EZI2C protocol, 8-bit sub-address.

The I2C interface should consist of a 3-byte buffer as described here:

Register Offset	Name	Purpose	Master Access
0x00	Info	Info from master	Write/Read
0x01	Button	Button press info	Read Only
0x02	Speed	Speed info	Read Only

Table 1: I2C Register Map

Each of the three registers is an 8-bit value. The maps for each of the three registers are shown below:

Info Register

Bit	7	6	5	4	3	2	1	0
Purpose	0	Init	Pass	Fail	LED4	LED3	LED2	LED1

Table 2: Info Register Values (input to slave)

Init:	This bit is set high at initial powerup of the master. It is cleared when a game is started and will then remain low until the master is reset. The slave should set this bit at powerup (when initializing the I2C buffer array) so that it will start in the Init state even if it is powered up after the master.
Pass:	This bit is high after the game sequence passes (i.e. the player has completed all of the sequences correctly). It is cleared when a new game is started. Note that you can change the max sequence required to pass the game by pressing the “Max Seq” button. Make sure the master LCD is installed so that you can see the passing sequence length.
Fail:	This bit is high after the game sequence fails (i.e. the player has entered an incorrect sequence). It is cleared when a new game is started.
LED4:	This bit is high when the master is sending LED4 as a part of a sequence.
LED3:	This bit is high when the master is sending LED3 as a part of a sequence.
LED2:	This bit is high when the master is sending LED2 as a part of a sequence.
LED1:	This bit is high when the master is sending LED1 as a part of a sequence.

Note that during a game, this register will be 0x00 when the master is not sending an LED as part of the sequence. This will be the case between LEDs in a given sequence and also when the master is waiting for a response from the slave.

Button Register

Bit	7	6	5	4	3	2	1	0
Purpose	0	0	0	0	Button4	Button3	Button2	Button1

Table 3: Button Register Values (output from slave)

Button4: This slave must set this bit whenever Button4 is pressed and must clear this bit whenever Button4 is released. As long as the button is held, this bit must remain set.

Button3: This slave must set this bit whenever Button3 is pressed and must clear this bit whenever Button3 is released. As long as the button is held, this bit must remain set.

Button2: This slave must set this bit whenever Button2 is pressed and must clear this bit whenever Button2is released. As long as the button is held, this bit must remain set.

Button1: This slave must set this bit whenever Button1 is pressed and must clear this bit whenever Button1 is released. As long as the button is held, this bit must remain set.

Speed Register

Bit	7	6	5	4	3	2	1	0
Purpose	0	0	0	0	0	Speed 3	Speed 2	Speed 1

Table 4: Speed Register Values (output from slave)

Speed3: This slave sets this bit for speed value 3 (fast).

Speed2: This slave sets this bit for speed value 2 (medium).

Speed1: This slave sets this bit for speed value 1 (slow).

Only 1 speed bit should be set at a time (i.e. 1-hot encoding).

Pinout (Pioneer)

PSoC 4 Pin	Shield Connection	Function
P0[6]	SW1 and Capsense 1	Button 1
P3[4]	SW2 and Capsense 2	Button 2
P3[6]	SW3 and Capsense 3	Button 3
P0[4]	SW4 and Capsense 4	Button 4
P0[7]	SW10	Button “Cheat”
P1[0]	D7	LED 1
P0[0]	D8	LED 2
P3[7]	D9	LED 3
P0[5]	D10	LED 4
P0[1]	D2	LED “Awake”
P2[0]	R30	Potentiometer (POT)
P3[5]	BZ1	Buzzer
P3[0]	I2C SCL	I2C Clock
P3[1]	I2C SDA	I2C Data
P2[7:1]	LCD	LCD
P4[2]	None	Capsense CMOD

In addition to the pinout above, the following jumper settings are required on the shield board:

Jumper	Setting
J6	Installed
J8	P3_0
J11	P3_1
J12	Select appropriate voltage for the LCD
J13	Select appropriate voltage for the LCD

Optional Slave Features

Buzzer

Make the slave sound a buzzer when a button is pressed. Use a different sound for each button.

Try to make the buzzer function operate in the hardware as much as possible. That is, there should be little firmware interaction to control the buzzer.

Make the slave sound a buzzer when the master sends the sequence to the slave. The tones should match the button press tones. Depending on the implementation, this may require some interaction between the firmware and hardware (e.g. a control register). Try to minimize the firmware required.

Add a buzzer sound when the player fails to enter the correct sequence. Implement the fail buzzer so that it stops after 2 seconds even though the Fail bit in the Info register will stay set until the game is restarted.

The tones for the game are: 415Hz, 310Hz, 252Hz, and 209Hz. The end of game (fail) sound is 42Hz.

Extra Credit: For the pass condition, play a song. Note that you can change the maximum sequence length expected by the master by pressing the “MAX SEQ” button on the shield board.

Game Status on LCD

Print the game status to the character LCD on the slave (INIT, RUN, PASS, FAIL) as well as the speed (if implemented).

Game Speed

Send a speed command to the master based on a POT value that is measured by the PSoC (3 speed bins – slow, medium, fast). The desired speed should be displayed on the LCD on the slave.

Extra Credit: Use the CapSense slider on the Pioneer board to control the speed of the game instead of the POT.

CapSense Buttons

Use the CapSense buttons instead of the mechanical buttons. Note that they are on the same PSoC 4 pins so you will have to remove the pins connected to the mechanical buttons from your schematic (or put them on a separate page that you can comment out).

Cheat mode

Use a switch press on the board to enter cheat mode. When in cheat mode send back the correct combination to the master automatically so that the slave plays the game by itself. That is, in cheat mode the player does not have to press the buttons to repeat the sequence. Rather, the slave will record the correct sequence and will send it back to the master automatically each time the master completes sending a sequence.

Hints: After blinking the sequence, the master waits one blink delay (500ms on slow speed) before looking for button presses from the slave. The master expects at least 100ms for each button press and at least 100ms between button presses from the slave. After receiving a correct sequence, the master will send the next sequence after 300ms.

Low-Power Mode

Put the PSoC into deep sleep mode when the game is not in progress (after a timeout period of 10-20 seconds). An I2C address match should be used to wake the PSoC. You should light an LED on the shield board (use the LED labeled “AWAKE”) when awake so that the sleep/wake behavior can be observed.

Measure the current consumed by the PSoC in active mode and in deep sleep mode. How low can you get current in low power mode? (Hint: make sure you have pull-ups on the I2C lines and make sure that you have disconnected the POT or any other circuits which may draw additional power. You will also have to change the “Debug Select” setting to “GPIO” in the System tab of the design wide resources so that the SWD pins don’t float during deep sleep.)

PSoC 5LP FreeSoC2

Convert your project to run on a Sparkfun FreeSoC2 kit. Compare the differences between PSoC 4 and PSoC 5LP. How many of the features can you get to run in hardware on PSoC5 LP (i.e. without CPU intervention)?

FreeSoC2 Pinout

FreeSoC2	Shield Connection	Function
P6[7]	SW1	Button 1
P6[4]	SW2	Button 2
P12[5]	SW3	Button 3
P2[0]	SW4	Button 4
P2[2]	SW10	Button "Cheat"
P2[6]	D7	LED 1
P2[4]	D8	LED 2
P2[3]	D9	LED 3
P2[1]	D10	LED 4
N/A [1]	D2	LED "Awake"
P15[5]	R30	Potentiometer (POT)
P2[5]	BZ1	Buzzer
P6[5]	I2C SCL	I2C Clock
P6[6]	I2C SDA	I2C Data
RS: P12[4] [2] E: P6[0] DB4: P15[4] DB5: P6[3] DB6: P6[2] DB7 P6[1]	LCD	LCD
N/A [3]	None	CapSense CMOD

1. **The LED "Awake" pin is connected to Vdd on the FreeSoC2 board so it is always on.**
2. **The LCD requires a custom component which allows non-contiguous pin assignments. Search the Cypress Community Components library for "Character LCD across multiple ports".**
3. **There is no CMOD on the FreeSoC board so CapSense buttons are not supported.**

In addition to the pinout above, the following shield settings are required:

Jumper	Setting
J6	Installed
J8	P3_0
J11	P3_1
J12	Select appropriate voltage for the LCD
J13	Select appropriate voltage for the LCD

In order to use the Bridge Control Panel with a FreeSoC2 board, you will need to connect a Miniprog3 to the 5 pin I2C header provided on the shield board.

I2C Commands

The Bridge Control panel can be used to test the slave functionality without using the master. This is an extremely useful debugging tool since it allows you to control exactly what is being sent to the slave and monitor what the slave is sending back.

Some useful commands:

```
w 36 00 p      ; set write pointer to first register

r 36 x x x p   ; read register values - status, button, speed

; Master sends these sequences to the slave
w 36 00 00 p   ; game in progress, no LED sent by master (use between
LEDs and when waiting for slave response)
w 36 00 01 p   ; LED 1 being sent by master
w 36 00 02 p   ; LED 2 being sent by master
w 36 00 04 p   ; LED 3 being sent by master
w 36 00 08 p   ; LED 4 being sent by master
w 36 00 10 p   ; Game failed
w 36 00 20 p   ; Game passed
w 36 00 40 p   ; Initial powerup state
```

A bridge control panel file with these commands can be found in: “C:\Cypress Academy\PSoC 101 v<n>\Simon_I2C_Commands.iic” where <n> is the version number. This file can be loaded by selecting “File->Open File” from the Bridge Control Panel.