SANTA CLARA UNIVERSITY
Electrical and Computer Engineering Department

Real-Time Embedded Systems - ECEN 121
Lab 6

Dylan Thornburg & Kai Hoshide

Consumer Infra-Red Remote Control Receiver

Andrew Wolfe

The objective of this project is to be able to receive and process signals from a consumer IR remote control. Some of the preparatory materials are in the class lecture notes.

Prelab:

1. Read this lab handout
2. Review the TSOP38238 datasheet
3. Review your class notes and the documents in the Lab Handouts directory related the NEC protocol
4. Draw a schematic indicating how to connect the TSOP38238 to the DISCO board for this lab.
5. Determine the total transmit time of an initial button-press message frame using the NEC IR protocol.
6. Review the material on repeat codes and think about how you will distinguish them. 7. When the B button is pressed on the remote, the IR remote sends out the 16-bit address 0xFF and 16-bit data 0xFD using a NEC protocol as discussed in class.
    a. Draw a signal diagram showing the output of the remote control (light amplitude on Y, time on X). Label key timing parameters.
    b. Draw a signal diagram showing the output of the TSOP38238 in response to this signal (light amplitude on Y, time on X). Label key timing parameters.

Part 1:

Step 1:

Create a new directory for Lab 6 and a subdirectory for Lab6p1.

Copy the LCD TImer.ioc file from Lab 4 into the Lab6p1 directory and rename it

IR1.ioc Open IR1.ioc.

Disable Timer TIM16.

Configure Timer TIM7 so that it is activated and interrupts every 100 µs.

Change the configuration of pin PD0 so that it is configured as a GPIO_Input and named

IR_IN. Save the project and generate code.

Step 2:

Open the Keil project. Check the debugger and compiler settings and make sure they are what you want.

Add the code to start Timer TIM7.

Build the project. You should have no errors.

Find the private defines section of main.h. Define SAMPLE_COUNT as 700.

Create a global volatile array of integers called irdat[] of size SAMPLE_COUNT. This is the IR input data sample buffer and can hold up to SAMPLE_COUNT consecutive measurements from the input pin.

Next you need to write the interrupt handler for Timer TIM7. You can add whatever variables you want and share variables between main.c and stm32l4xx_it.c as needed. Pay attention to what needs to be volatile.

The interrupt handler should do the following:

- Read the current input value at pin PD0 (i.e. IR_IN).
- The first time that the input pin value is low, begin capturing data and placing it in the irdat[] buffer. Capture one IR_IN value per interrupt until the irdat[] buffer is full.
- Don't overwrite the buffer until you are certain that the data in the buffer has been processed by main.

Step 3:

Connect up the Vishay TSOP38238 IR receiver so that it provides a signal to pin PD0. The TSOP38238 is described in the datasheet (tsop382.pdf) in the class Lab Handouts directory. You should be able to read that data sheet and properly connect that component. **Use 3.3 V as the power supply. Do not connect it to 5V.**

Perform an initial test. Run your code in the debugger and use the remote to send in an IR code by pressing button B.

You should be able to view the captured data using the debugger. It should be a combination of 0x00000000 and 0x0000001 values. You should be able to convince yourself that the pulses match the

NEC protocol.

Step 4:

Write a function:

```
Uint32_t parseIRCode() {}
```

Using your knowledge of the NEC IR protocol, this function should be able to extract a 32-bit code from the irdat[] buffer that matches the 32 bits transmitted by the remote and return that value. Remember that your parsing routine must not lock up or fail to return a value when a repeat code is detected. You do not need to do anything for a repeat code, so as long as the returned value does not match any of the codes below, nothing will happen.

The IR remote transmits the following 32-bit codes:

```
#define IR_A 0x00ff22dd
#define IR_B 0x00ff02fd
#define IR_C 0x00ffc23d
#define IR_POWER 0x00ff629d
#define IR_UP 0x00ff9867
#define IR_DOWN 0x00ff38c7
#define IR_LEFT 0x00ff30cf
#define IR_RIGHT 0x00ff7a85
#define IR_CIRCLE 0x00ff18e7
```

I pasted these into my main.c file so that I could use these names.

Test this routine.

Step 5:

In the while loop in main.c:

Check to see if the irdat[] buffer is full. If so:

- Call `parseIRCode()` to extract the IR code.
- Mark the irdat[] buffer as empty again.
- If the B button has been pressed, toggle the red LED.
- If the center circle button has been pressed, toggle the green LED.

Test this code.

DEMO for the TA.

CODE:

```
//in main.c

/* USER CODE BEGIN PV */
volatile unsigned int irdat[SAMPLE_COUNT];
unsigned int z;
extern int flag;
#define IR_A 0x00ff22dd
#define IR_B 0x00ff02fd
#define IR_C 0x00ffc23d
#define IR_POWER 0x00ff629d
#define IR_UP 0x00ff9867
#define IR_DOWN 0x00ff38c7
#define IR_LEFT 0x00ff30cf
#define IR_RIGHT 0x00ff7a85
#define IR_CIRCLE 0x00ff18e7
/* USER CODE END PV */



uint32_t parseIRCode()
        {
  uint32_t num = 0;
                int count = 0;
                int arrayCount = 0;
                uint32_t mask = 0x80000000;
                /*skips header */
                while(irdat[arrayCount] !=0)
                {
                        arrayCount++;
                }
                while(irdat[arrayCount] !=1)
                {
                        arrayCount++;
                }
                /* skips header */
                while(arrayCount != SAMPLE_COUNT)
                {
                        if(irdat[arrayCount] ==0)
                        {
                                count++;
                        }
                        else
                        {
                                if(count < 8 && count !=0)
                                {
```

```
                        mask/=2;
                    }
                    else if(count > 8 && count != 0)
                    {
                        num |= mask;
                        mask/=2;
                    }
                    count = 0;
                }
                arrayCount++;
            }
            return num;
        }



int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RTC_Init();
  MX_LCD_Init();
  MX_TIM7_Init();
  /* USER CODE BEGIN 2 */
        HAL_TIM_Base_Start_IT(&htim7);
  /* USER CODE END 2 */

  /* Infinite loop */
```

```
/* USER CODE BEGIN WHILE */
while (1)
{
		if(flag == 1)
		{
			uint32_t num = parseIRCode();
			if(num == IR_A)
			{
				break;
			}
			if(num == IR_B)
			{
				HAL_GPIO_TogglePin(LD_R_GPIO_Port, LD_R_Pin);
			}
			if(num == IR_C)
			{
				break;
			}
			if(num == IR_POWER)
			{
				break;
			}
			if(num == IR_UP)
			{
				break;
			}
			if(num == IR_DOWN)
			{
				break;
			}
			if(num == IR_LEFT)
			{
				break;
			}
			if(num == IR_RIGHT)
			{
				break;
			}
			if(num == IR_CIRCLE)
			{
				HAL_GPIO_TogglePin(LD_G_GPIO_Port, LD_G_Pin);
			}
			flag=0;
		}
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

```
  /* USER CODE END 3 */
}


//in stm32l4xx_it.c
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
extern unsigned int irdat[];
uint32_t x;
int i=0;
int count = 0;
int data = 0;
int flag =0;
/* USER CODE END PV */




void TIM7_IRQHandler(void)
{
  /* USER CODE BEGIN TIM7_IRQn 0 */

  /* USER CODE END TIM7_IRQn 0 */
  HAL_TIM_IRQHandler(&htim7);
  /* USER CODE BEGIN TIM7_IRQn 1 */
        x = HAL_GPIO_ReadPin(IR_IN_GPIO_Port, IR_IN_Pin);
        if(flag == 0)
        {
                if(x == 0 || data == 1)
                {
                        data = 1;
                        if(x==0)
                        {
                                irdat[count] = 1;
                                count = (count +1) %SAMPLE_COUNT;
                        }
                        else
                        {
                                irdat[count]=0;
                                count= (count +1) %SAMPLE_COUNT;
                        }
                        if(count == SAMPLE_COUNT-1)
                        {
                                flag =1;
                                data =0;
                        }
                }
        }
  /* USER CODE END TIM7_IRQn 1 */
```

}

Part 2:

Copy the project to a new directory called Lab6p2 using one of the procedures outlined in the instructions in the Lab Handouts directory. Do not forget any steps.

Add in the BSP files (.c and .h files) for the LCD in the same way you did in Lab

4. Edit main.c in this new project to initialize the LCD.

Add some code to main() to write something to the LCD to test it.

Once this is working, modify main() so that when you press each button on the remote, the string identified below is displayed for 1 second. You do not need to respond to additional remote control commands during that 1 second. The text should be right justified.

IR_POWER	POWER
IR_A	A
IR_B	B
IR_C	C
IR_UP	UP
IR_DOWN	DOWN
IR_LEFT	LEFT
IR_RIGHT	RIGHT
IR_CIRCLE	CIRCLE

Test this code.

DEMO for the TA.

CODE:

```
//The only code that changed was in main
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/
```

```
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RTC_Init();
  MX_LCD_Init();
  MX_TIM7_Init();
  /* USER CODE BEGIN 2 */
          HAL_TIM_Base_Start_IT(&htim7);
          BSP_LCD_GLASS_Init();
          BSP_LCD_GLASS_ClearBar(0xFFFFFFFF);
          BSP_LCD_GLASS_Clear();
          //the spaces are only added because text has to be right justified
          char *strings [9] = {" POWER", "    A", "    B", "    C", "   UP", "  DOWN", "  LEFT", " RIGHT",
"CIRCLE"};
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
                    if(flag == 1)
                    {
                              uint32_t num = parseIRCode();
                              if(num == IR_A)
                              {
                                        uint8_t * word = (uint8_t *) strings[1];
                                        BSP_LCD_GLASS_DisplayString(word);
                                        HAL_Delay(1000);
                                        BSP_LCD_GLASS_Clear();
                                        num=0;
                              }
                              if(num == IR_B)
                              {
                                        uint8_t * word = (uint8_t *) strings[2];
                                        BSP_LCD_GLASS_DisplayString(word);
                                        HAL_Delay(1000);
```

```
            BSP_LCD_GLASS_Clear();
            num=0;
    }
    if(num == IR_C)
    {
            uint8_t * word = (uint8_t *) strings[3];
            BSP_LCD_GLASS_DisplayString(word);
            HAL_Delay(1000);
            BSP_LCD_GLASS_Clear();
            num=0;
    }
    if(num == IR_POWER)
    {
            uint8_t * word = (uint8_t *) strings[0];
            BSP_LCD_GLASS_DisplayString(word);
            HAL_Delay(1000);
            BSP_LCD_GLASS_Clear();
            num=0;
    }
    if(num == IR_UP)
    {
            uint8_t * word = (uint8_t *) strings[4];
            BSP_LCD_GLASS_DisplayString(word);
            HAL_Delay(1000);
            BSP_LCD_GLASS_Clear();
            num=0;
    }
    if(num == IR_DOWN)
    {
            uint8_t * word = (uint8_t *) strings[5];
            BSP_LCD_GLASS_DisplayString(word);
            HAL_Delay(1000);
            BSP_LCD_GLASS_Clear();
            num=0;
    }
    if(num == IR_LEFT)
    {
            uint8_t * word = (uint8_t *) strings[6];
            BSP_LCD_GLASS_DisplayString(word);
            HAL_Delay(1000);
            BSP_LCD_GLASS_Clear();
            num=0;
    }
    if(num == IR_RIGHT)
    {
            uint8_t * word = (uint8_t *) strings[7];
            BSP_LCD_GLASS_DisplayString(word);
            HAL_Delay(1000);
            BSP_LCD_GLASS_Clear();
```

```
                        num=0;
                    }
                    if(num == IR_CIRCLE)
                    {
                        uint8_t * word = (uint8_t *) strings[8];
                        BSP_LCD_GLASS_DisplayString(word);
                        HAL_Delay(1000);
                        BSP_LCD_GLASS_Clear();
                        num=0;
                    }
                    flag=0;
                }
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
 }

 /* USER CODE END 3 */
}
```