Lab 6

Tues 2:15pm

Henry Fang Jack Landers

Introduction

Build circuit that reads IR signal from IR remoter.

Procedure

Describe what you did during the lab. The way you wired up the board, what code you wrote (don't paste your actual code here), etc…

Part 1

Step1

Copy the LCD TImer.ioc and renames it as IR1.ioc. Disable the Timer TIM16. Set TIM7 to 10KHz. Set PD0 as input pin and rename it as IR_IN.

Step2

Open the Keil project. Add the "HAL_TIM_Base_Start_IT(&htim7);" to start the timer. Build with no error report. Define SAMPLE_COUNT as 700, 10 frames per millisecond for 70ms. Write the interrupt handler for TIM7.

Step3

Connect the IR receiver to the board. Inspect memory to confirm the interrupt is reading input pin and write result to buffer.

Step4

Add IR code dictionary to the defines. Write parseIRCode() to extract NEC IR protocol codes.

Step5

Add a switch to respond to different IR codes. Toggle the red LED when "B" is pressed. Toggle the green LED when "O" is pressed.

Part 2

Copy part 1 project and rename it as Lab6p2. Add BSP files for LCD. Initialize LCD in main. Modify switch of response to IR codes. Use BSP_LCD_GLASS_DisplayString(uint8_t *) to display the strings corresponding to the button pressed on the remote.

Results

Place the results of your experiments and tasks here. Tables, charts, screenshots, etc.. Answer any questions from the lab document here.

Part 1

Demoed.

Part 2

Demoed.

Conclusion

Short paragraph talking about the takeaways from this lab.
In this Lab, we use the board to read the NEC IR protocol codes from the remote and respond to different codes by printing on LCD. Also practiced buffer management and concurrency. In this lab, we only have 1 interrupt producing data, and the main is consuming data.

Appendix

Place code and other data here
USER PV

```
/* USER CODE BEGIN PV */
volatile int irdat[SAMPLE_COUNT];
volatile int lock = 0;
volatile int *head = irdat;
volatile int IRCode[32];
unsigned int ExtractCode;
//volatile int *tail = irdat;
volatile int flag = 0;
volatile int counter=0;
uint32_t IRcodeHolder;
char
dictionary[9][7]={"POWER","A","B","C","UP","DOWN","LEFT","RIGHT","CIR
CLE"};
/* USER CODE END PV */
```

Main While True:

```
while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
          if(*head==0 && flag==1)//if the input value is low and
array is full
          {
               IRcodeHolder=parseIRCode();
          }


          switch(IRcodeHolder)
          {
               case IR_POWER:
                    BSP_LCD_GLASS_Clear ();
```

```
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[0]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_A:
                    BSP_LCD_GLASS_Clear ();
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[1]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_B:
                    BSP_LCD_GLASS_Clear ();
                    HAL_GPIO_TogglePin(LD_R_GPIO_Port,LD_R_Pin);
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[2]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_C:
                    BSP_LCD_GLASS_Clear ();
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[3]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_UP:
                    BSP_LCD_GLASS_Clear ();
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[4]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_DOWN:
                    BSP_LCD_GLASS_Clear ();
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[5]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_LEFT:
                    BSP_LCD_GLASS_Clear ();
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[6]);
```

```
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_RIGHT:
                    BSP_LCD_GLASS_Clear ();
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[7]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               case IR_CIRCLE:
                    BSP_LCD_GLASS_Clear ();
                    HAL_GPIO_TogglePin(LD_G_GPIO_Port,LD_G_Pin);
                    BSP_LCD_GLASS_DisplayString((uint8_t
*)dictionary[8]);
                    HAL_Delay(1000);
                    BSP_LCD_GLASS_Clear ();
                    break;
               default:
                    break;
          }


          if(IRcodeHolder!=0u)//resetting
          {
               head = irdat;//clean buffer
               flag=0;//set empty
               IRcodeHolder=0u;//set clear

               //wipe the buffer
               lock=1;
               for(int j=0;j<SAMPLE_COUNT;j++)
               {
                    irdat[j]=1;
               }
               lock=0;
          }
  }
```

PraseCode:

```
uint32_t parseIRCode()
{

     volatile int * temp;
     temp=head;
     int i;
```

```
counter=0;
while(*temp==0)//checking initial frame
{
      counter++;
      if(temp == &irdat[SAMPLE_COUNT-1])//increament of pointer
            {
                  temp=irdat;
            }
            else
            {
                  temp++;
            }
}
if(counter<=85||counter>=95)return 0;

counter=0;
while(*temp==1)
{
      counter++;
      if(temp == &irdat[SAMPLE_COUNT-1])//increament of pointer
            {
                  temp=irdat;
            }
            else
            {
                  temp++;
            }
}
if(counter<=40||counter>=50)return 0;

//start to extract data
while(*temp==0)//skip first low pulse
{
      if(temp == &irdat[SAMPLE_COUNT-1])//increament of pointer
      {
            temp=irdat;
      }
      else
      {
            temp++;
      }
}

for(i=0;i<32;i++)//check 32bits
```

```
	{
		counter=0;
		while(*temp==1)
		{
			counter++;
			if(temp == &irdat[SAMPLE_COUNT-1])//increament of
pointer
			{
				temp=irdat;
			}
			else
			{
				temp++;
			}
		}

		if(counter>=5 && counter<=7)
		{
			IRCode[i]=0;//only 5ms is 0
		}
		else//If bug happens check this else
		{
			IRCode[i]=1;//beyond 7ms is 1
		}

		//skip Low pulse region
		while(*temp==0)
		{
			if(temp == &irdat[SAMPLE_COUNT-1])//increament of
pointer
			{
				temp=irdat;
			}
			else
			{
				temp++;
			}
		}
	}
	//convert to unsigned int 32
	uint32_t output = 0x00000000;
	for(i=0;i<31;i++)
	{
		output= (uint32_t)IRCode[i]|output;
		output= output<<1;
```

```
      }
      output= (uint32_t)IRCode[i]|output;
      return output;
}
```

TIM7 IRQHANDLER

```
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 */
      if(lock) return;//return if locked

      *head = HAL_GPIO_ReadPin(IR_IN_GPIO_Port, IR_IN_Pin);//get the
value from input

      if(head == &irdat[SAMPLE_COUNT-1])//increament of pointer
      {
            head=irdat;
            flag = 1;//set full
      }
      else
      {
            head++;
      }



  /* USER CODE END TIM7_IRQn 1 */
}
```