Lab 9 Music Player

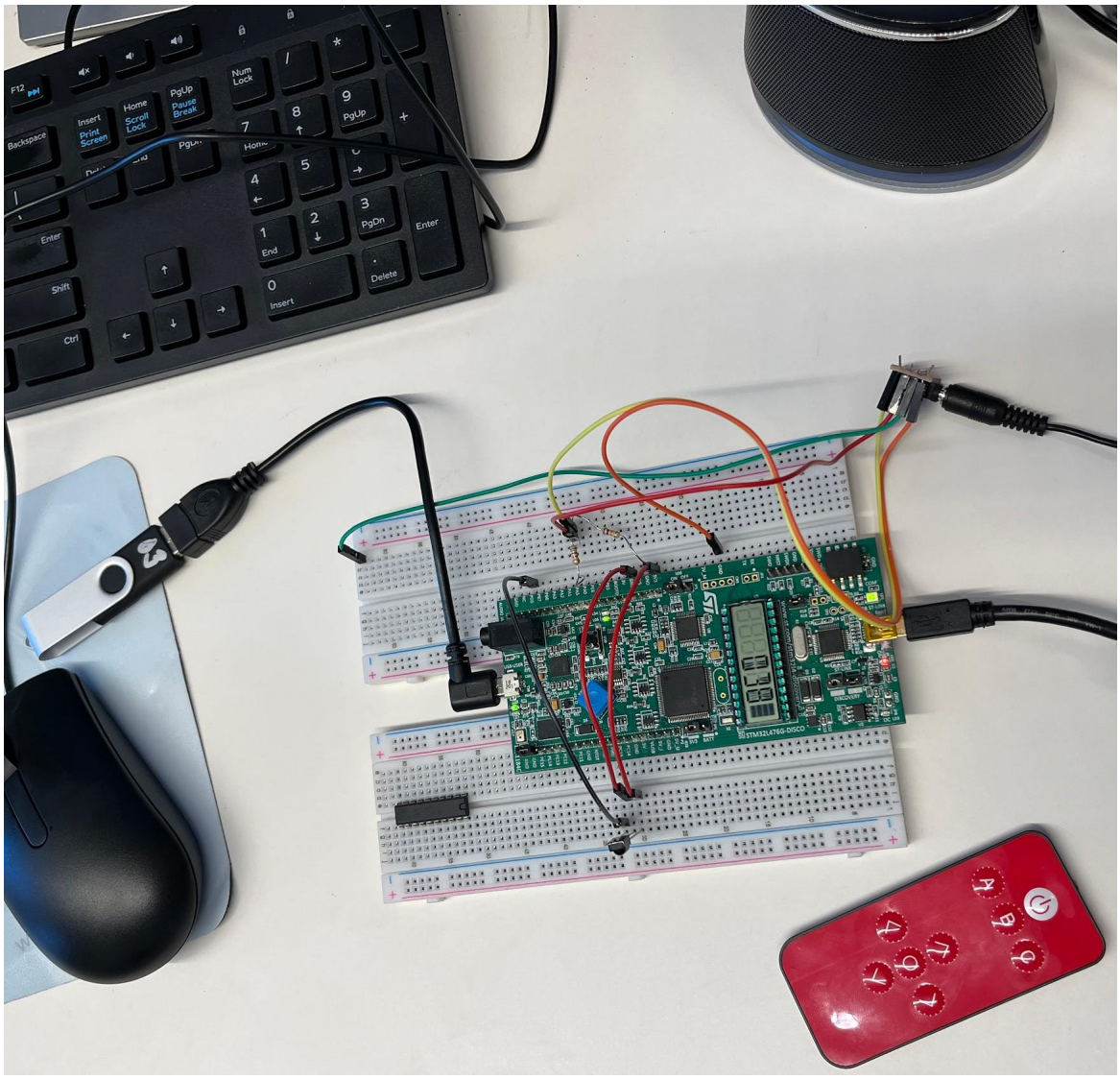
Tue 2:15PM

Henry Fang & Jack Landers

Introduction

Build a music player. It has a timer display. It has an IR remote control. It plays music from a USB storage device.

Procedure



Project1

Import project from the Lab9p1.zip. Write a TIM6 interrupt that writes audiobuffer to the DAC, back to start when reaching the end of the buffer. Play a 400 Hz sine wave on the speaker from the DAC.

Project2

Import project from the Lab9p2.zip. Write a TIM6 interrupt that alternately reads from two audiobuffers. Additionally don’t play past the end of the song. To achieve this function, we detect the int lastbuffer, if it is not zero, we do a for loop for remaining frames instead.

Project3

Copy project 2 and rename it as Lab9p3. Add in an LCD timer that counts how long since the start of the song, just minutes and seconds. We were trying to use different timers to run the timer procedure. But the timer interrupts impact the music playing a lot. The audio is glitchy. Then we implant a timer procedure into the same interrupt as the DAC. For running the timer in 1Hz, we add an 0 to 15999 counter, it is 1s when the counter reaches 15999.

Project4

Copy project 2 and rename it as Lab9p4. Add in an IR remote control. Center button is the pause. The Up and Down buttons change volume, and don't get over min and max volume. The Left button restarts the music and timer.

Results

Demoed…

Conclusion

In this lab we learned how to make a music player. In project 1, we learned how to write data from buffer to DAC. In project 2,we learned how to use dual buffers for smooth music playing. In project 3, we implant an LCD timer from lab 4. And we found out interrupts are interfering with each other, timer update is glitching audio playing. We merge the interrupts instead of moving slow steps from interrupts to the main, which is a bad idea. In project 4, we implant IR remote control from lab 6. Sadly, the implant is not really successful. IR code reading interrupt is not working fine. It can read code, but many inputs will be a bad read because DAC interrupt triggered. When DAC interrupt triggers, it makes the IR reader procedure missing a lot of frames because DAC interrupt procedure is long. We did a lot of patches to try to prevent triggers DAC procedure during IR reader procedure, or make DAC interrupt procedure lighter and shorter. We remove all the variable declarations in the interrupt to avoid allocating memory. We also moved the LCD update procedure to the main, this removes glitch from audio. Other than those, we implement button functions successfully.

Appendix

Project 1

```
void TIM6_DAC_IRQHandler(void)
{
```

```
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  HAL_DAC_IRQHandler(&hdac1);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

  /*************************************************/
     /* Put your code here to produce a 400Hz Sine Wave */
  /*************************************************/
     write_DAC1Ch2(*bufferptr, volume);
     if(bufferptr == &audiobuffer[ABUF_SIZE-1])
     {
          bufferptr=audiobuffer;
     }
     else
     {
          bufferptr++;
     }

  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

Project 2

```
void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  HAL_DAC_IRQHandler(&hdac1);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

  /*************************************************/
     /* Put your code here to play a song
      */
  /*************************************************/
     //if the block is ready the playing ptr increase
     //find which block ptr is at
     int inuseblock= (playBufferptr-&audiobuffer[0][0])/ABUF_SIZE;
     if(lastbuffer==0)//not lastbuffer yet
     {
```

```
          if(abuf_full[inuseblock])//if is full write from buf to
dac
          {
                write_DAC1Ch2(*playBufferptr, volume);
                if(playBufferptr==
&audiobuffer[NUM_ABUF-1][ABUF_SIZE-1])//at last of the this array
blocks increamenting
                {
                      playBufferptr=&audiobuffer[0][0];
                }
                else
                {
                      playBufferptr++;
                }
                if(playBufferptr== &audiobuffer[0][0]
                      ||playBufferptr== &audiobuffer[1][0])//if the
pointer reaches next buffer block
                {
                      abuf_full[inuseblock]=false;//mark last block
as empty
                }
          }
     }
     else//lastbuffer
     {
          for(int i=0; i<lastbuffer/2; i++)
          {
                if(abuf_full[inuseblock])//if is full write from buf
to dac
                {
                      write_DAC1Ch2(*playBufferptr, volume);
                      if(playBufferptr==
&audiobuffer[NUM_ABUF-1][ABUF_SIZE-1])//at last of the this array
blocks
                      {
                            playBufferptr=&audiobuffer[0][0];
                      }
                      else
                      {
                            playBufferptr++;
                      }
                      if(playBufferptr== &audiobuffer[0][0]
                            ||playBufferptr== &audiobuffer[1][0])//if
the pointer reaches next buffer block
                      {
```

```
                            abuf_full[inuseblock]=false;//mark last
block as empty
                        }
                    }
                }
                __disable_irq();
                pause=1;
                return;
            }


  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

Project 3

Vars

```
extern volatile int pause;
extern ApplicationTypeDef Appli_state;
extern int seccount;
extern int mincount;

int16_t audiobuffer[NUM_ABUF][ABUF_SIZE];  /* File read buffer for
audio */
volatile bool abuf_full[NUM_ABUF];
bool file_open = false;
int fill_buf = 0;
int lastbuffer = 0;
volatile uint8_t volume = DFLT_VOLUME;          // Volume range is 0
to 255
uint8_t noisehigh[8] = {0xff, 0xdd, 0xbb, 0x99, 0x77, 0x55, 0x33,
0x11};

char *rfilename = "songfile.raw";


FIL MyFile;                   /* File object */

int16_t *writeBufferptr;
int16_t *playBufferptr=&audiobuffer[0][0];
int playing_buf = 0;
```

TIM6 IRQ

```
void TIM6_DAC_IRQHandler(void)
{
```

```
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  HAL_DAC_IRQHandler(&hdac1);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

  /*****************************************************/
     /* Put your code here to play a song
      */
  /*****************************************************/
     //if the block is ready the playing ptr increase
     //find which block ptr is at
     if(period==16000)//timer process
     {
          if(pause==0&&file_open==true){
               seccount ++;
               seccount = seccount%60;

               if(seccount==59){
                    mincount++;
               }

               mincount = mincount%60;

               period=0;
          }
     }
     else
     {
          period++;
     }

     int inuseblock= (playBufferptr-&audiobuffer[0][0])/ABUF_SIZE;
     if(lastbuffer==0)//not lastbuffer yet
     {
          if(abuf_full[inuseblock])//if is full write from buf to
dac
          {
               write_DAC1Ch2(*playBufferptr, volume);
               if(playBufferptr==
&audiobuffer[NUM_ABUF-1][ABUF_SIZE-1])//at last of the this array
blocks increamenting
               {
                    playBufferptr=&audiobuffer[0][0];
```

```
                }
                else
                {
                      playBufferptr++;
                }
                if(playBufferptr== &audiobuffer[0][0]
                      ||playBufferptr== &audiobuffer[1][0])//if the
pointer reaches next buffer block
                {
                      abuf_full[inuseblock]=false;//mark last block
as empty
                }
           }
      }
      else//lastbuffer
      {
           for(int i=0; i<lastbuffer/2; i++)
           {
                if(abuf_full[inuseblock])//if is full write from buf
to dac
                {
                      write_DAC1Ch2(*playBufferptr, volume);
                      if(playBufferptr==
&audiobuffer[NUM_ABUF-1][ABUF_SIZE-1])//at last of the this array
blocks
                      {
                            playBufferptr=&audiobuffer[0][0];
                      }
                      else
                      {
                            playBufferptr++;
                      }
                      if(playBufferptr== &audiobuffer[0][0]
                            ||playBufferptr== &audiobuffer[1][0])//if
the pointer reaches next buffer block
                      {
                            abuf_full[inuseblock]=false;//mark last
block as empty
                      }
                }

           }
           __disable_irq();
           pause=1;
           return;
```

```
      }


  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

MAIN WHILE TRUE

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

    /* USER CODE BEGIN 3 */
          uint8_t temp = seccount+48;//acii to the char numbers
          //BSP_LCD_GLASS_DisplayChar(&temp, POINT_OFF,
DOUBLEPOINT_ON, 5);
          temp = seccount%10+48;
          BSP_LCD_GLASS_DisplayChar(&temp, POINT_OFF,
DOUBLEPOINT_OFF, 3);//one of the secc
          temp = seccount/10+48;
          BSP_LCD_GLASS_DisplayChar(&temp, POINT_OFF,
DOUBLEPOINT_OFF, 2);//ten of the sec
          temp = mincount%10+48;
          BSP_LCD_GLASS_DisplayChar(&temp, POINT_OFF,
DOUBLEPOINT_ON, 1);//ONE OF the min
          temp = mincount/10+48;
          BSP_LCD_GLASS_DisplayChar(&temp, POINT_OFF,
DOUBLEPOINT_OFF, 0);//then of the min
    /* Mass Storage Application State Machine */
    switch(Appli_state) {
      case APPLICATION_READY:
        if (!file_open) FS_FileOpen();//if file not open, open the
file
                if (file_open && (lastbuffer == 0)) {//if file is
open
                        if (!abuf_full[fill_buf]) {//if filling
buf is not full
                                FS_FileRead((uint8_t *)
audiobuffer[fill_buf]);//write from usb to filling buf
                                abuf_full[fill_buf++] = true;//set
this buffer be full
                                fill_buf = fill_buf %
NUM_ABUF;//increment to next buffer
                        }
                }
                if ((file_open == true) && (lastbuffer != 0)) {
```

```
                    FS_FileClose();
      Appli_state = APPLICATION_IDLE;
                  }
    break;

  case APPLICATION_IDLE:
    default:
    break;
  }




     }
```

Project 4

Vars

```
/* USER CODE BEGIN PV */
extern volatile int pause;
extern ApplicationTypeDef Appli_state;
extern int seccount;
extern int mincount;

int16_t audiobuffer[NUM_ABUF][ABUF_SIZE];  /* File read buffer for
audio */
volatile bool abuf_full[NUM_ABUF];
bool file_open = false;
int fill_buf = 0;
int lastbuffer = 0;
volatile uint8_t volume = DFLT_VOLUME;          // Volume range is 0
to 255
uint8_t noisehigh[8] = {0xff, 0xdd, 0xbb, 0x99, 0x77, 0x55, 0x33,
0x11};

char *rfilename = "songfile.raw";


FIL MyFile;                   /* File object */

int16_t *writeBufferptr;
int16_t *playBufferptr=&audiobuffer[0][0];
```

```
int playing_buf = 0;
//ir components
volatile int irdat[SAMPLE_COUNT];
volatile int lock = 0;
volatile int *head = irdat;
volatile int IRCode[32];
unsigned int ExtractCode;
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"};
uint32_t output;
/* USER CODE END PV */
```

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_USB_HOST_Init();
```

```
  MX_FATFS_Init();
  MX_DAC1_Init();
  MX_TIM6_Init();
  MX_RTC_Init();
  MX_LCD_Init();
  MX_TIM7_Init();
  /* USER CODE BEGIN 2 */
     BSP_LCD_GLASS_Init();

     /* initialize data structures */

     for (int i=0; i<NUM_ABUF; i++) {
          abuf_full[i] = false;
     }
     /* start devices */

  HAL_DAC_Start(&hdac1, DAC1_CHANNEL_2);
     HAL_TIM_Base_Start_IT(&htim6);             /* Start Timer 6 at
16KHz to run DAC */
     HAL_TIM_Base_Start_IT(&htim7); //TImer 7 at 10KHz

     NVIC_SetPriority(TIM7_IRQn,(uint32_t)1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    MX_USB_HOST_Process();

    /* USER CODE BEGIN 3 */

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


          switch(IRcodeHolder)
          {
               case IR_POWER:
```

```
            break;
        case IR_A:

            break;
        case IR_B:

            break;
        case IR_C:

            break;
        case IR_UP:
            if((int)volume+VOLUME_INCREMENT>=255)break;
            volume+=VOLUME_INCREMENT;
            break;
        case IR_DOWN:
            if((int)volume-VOLUME_INCREMENT<=0)break;
            volume-=VOLUME_INCREMENT;
            break;
        case IR_LEFT:
            FS_FileClose();

            seccount=0;
            mincount=0;
            FS_FileOpen();
            break;
        case IR_RIGHT:

            break;
        case IR_CIRCLE:
            pause++;
            pause%=2;
            if(pause) NVIC_DisableIRQ(TIM6_DAC_IRQn);
            else NVIC_EnableIRQ(TIM6_DAC_IRQn);
            HAL_GPIO_TogglePin(LD_R_GPIO_Port,LD_R_Pin);

            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;
           }

           uint8_t temp1 = seccount+48;//acii to the char numbers
           //BSP_LCD_GLASS_DisplayChar(&temp, POINT_OFF,
DOUBLEPOINT_ON, 5);
           temp1 = seccount%10+48;
           BSP_LCD_GLASS_DisplayChar((uint8_t*)&temp1, POINT_OFF,
DOUBLEPOINT_OFF, 3);//one of the secc
           temp1 = seccount/10+48;
           BSP_LCD_GLASS_DisplayChar((uint8_t*)&temp1, POINT_OFF,
DOUBLEPOINT_OFF, 2);//ten of the sec
           temp1 = mincount%10+48;
           BSP_LCD_GLASS_DisplayChar((uint8_t*)&temp1, POINT_OFF,
DOUBLEPOINT_ON, 1);//ONE OF the min
           temp1 = mincount/10+48;
           BSP_LCD_GLASS_DisplayChar((uint8_t*)&temp1, POINT_OFF,
DOUBLEPOINT_OFF, 0);//then of the min

           /* Mass Storage Application State Machine */
    switch(Appli_state) {
      case APPLICATION_READY:
        if (!file_open) FS_FileOpen();//if file not open, open the
file
                  if (file_open && (lastbuffer == 0)) {//if file is
open
                            if (!abuf_full[fill_buf]) {//if filling
buf is not full
                                      FS_FileRead((uint8_t *)
audiobuffer[fill_buf]);//write from usb to filling buf
                                      abuf_full[fill_buf++] = true;//set
this buffer be full
                                      fill_buf = fill_buf %
NUM_ABUF;//increment to next buffer
                            }
                       }
```

```
                    if ((file_open == true) && (lastbuffer != 0)) {
                         FS_FileClose();
          Appli_state = APPLICATION_IDLE;
                    }
        break;

      case APPLICATION_IDLE:
        default:
        break;
      }
          }
  /* USER CODE END 3 */
}
```

Vars

```
/* USER CODE BEGIN PV */
extern int irdat[];
extern int lock;
extern volatile int *head;
extern volatile int *tail;
extern volatile int flag;
int inuseblock;
int i;
// IR Remote Variables
int irval;
int startCount;
extern volatile int pause;
int remainbytes;
int irperiod=0;
int lastb=NUM_ABUF-1;
int lasti=ABUF_SIZE-1;
/* USER CODE END PV */
```

TIM6 IRQ

```
void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  HAL_DAC_IRQHandler(&hdac1);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

  /*************************************************/
     /* Put your code here to play a song
      */
```

```
  /*************************************************/
     //if the block is ready the playing ptr increase
     //find which block ptr is at
     NVIC_DisableIRQ(TIM7_IRQn);

     if(period==15999)//timer process
     {
          if(pause==0&&file_open==true){
               seccount ++;
               seccount = seccount%60;

               if(seccount==59){
                    mincount++;
               }

               mincount = mincount%60;

               period=0;
          }
     }
     else
     {
          period++;
     }

     inuseblock= (playBufferptr-&audiobuffer[0][0])/ABUF_SIZE;
     if(lastbuffer==0)//not last buffer yet
     {
          if(abuf_full[inuseblock])//if is full write from buf to
dac
          {
               write_DAC1Ch2(*playBufferptr, volume);
               if(playBufferptr== &audiobuffer[lastb][lasti])//at
last of the this array blocks incrementing
               {
                    playBufferptr=&audiobuffer[0][0];
               }
               else
               {
                    playBufferptr++;
               }
               if(playBufferptr==
&audiobuffer[0][0]||playBufferptr== &audiobuffer[1][0])//if the
pointer reaches next buffer block
               {
```

```
                        abuf_full[inuseblock]=false;//mark last block
as empty
                }
           }
      }
      else//lastbuffer
      {
           for(i=0; i<lastbuffer/2; i++)
           {
                if(abuf_full[inuseblock])//if is full write from buf
to dac
                {
                     write_DAC1Ch2(*playBufferptr, volume);
                     if(playBufferptr==
&audiobuffer[NUM_ABUF-1][ABUF_SIZE-1])//at last of the this array
blocks
                     {
                          playBufferptr=&audiobuffer[0][0];
                     }
                     else
                     {
                          playBufferptr++;
                     }
                     if(playBufferptr==
&audiobuffer[0][0]||playBufferptr== &audiobuffer[1][0])//if the
pointer reaches next buffer block
                     {
                          abuf_full[inuseblock]=false;//mark last
block as empty
                     }
                }

           }
           __disable_irq();//end the song playing
           pause=1;
           return;
      }
      NVIC_EnableIRQ(TIM7_IRQn);

  /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

TIM7 IRQ

```
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 */
     //NVIC_DisableIRQ(TIM6_DAC_IRQn);
     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])//increment of pointer
     {
          head=irdat;
          flag = 1;//set full
     }
     else
     {
          head++;
     }
     //NVIC_EnableIRQ(TIM6_DAC_IRQn);
  /* USER CODE END TIM7_IRQn 1 */
}
```