TE2003B.500 Diseño en Chip

Evidencia: Sistema de Infoentretenimiento

Prof. Raúl Peña Ortega

Prof. Enrique González Guerrero

Integrantes del equipo:

Armando Angel Martínez Villarreal	A00830032
Yosel Eduardo Delgado Salas	A00830161
José Antonio León Navarro	A01639250
Natalia Catalina Taboada Maldonado	A01570464

Fecha de entrega: 16 de junio de 2022

Índice

Introducción	03
Justificación de Problema	03
Metodología	04
Manual de Funcionamiento	04
Marco Teórico	05
Diagrama Esquemático General del Sistema	06
Línea de Tiempo de Calendarización de Tareas	06
Código STM32 y Explicación	07
Código Python y Explicación	12
Conclusión General	15
Reflexiones Individuales	16
Video	16
Referencias	17

Introducción

La industria automotriz ha ido cambiando con el paso del tiempo. Funciones que hoy en día parecen indispensables para la vida cotidiana, tales como el sistema de encendido del motor el cual, antiguamente necesitaban un sistema mecánico más complejo para poder encender el motor del automóvil.

El uso de microcontroladores para automatizar los procesos de los automóviles tiene inició a finales de los años 70's después de que General Motors produjera el primer auto utilizando ECU (Electronic Control Unit), marcas competidoras empezaron a sacar modelos de autos con microcontroladores, tal como Cadillac que en el año 1978 introdujo el primer automóvil con microcontrolador (Motorola 6802) utilizado para su econometro (trip computer) en su modelo Seville, desde entonces el uso de microcontroladores en la industria automotriz se ha vuelto indispensable, habiendo incluso algunos modelos de autos que llegan a tener más de 50 microcontroladores.

Justificación del problema

Conforme al paso del tiempo, los avances tecnológicos crecieron de forma exponencial, cosas que en décadas anteriores parecían tan solo cosas de sueños o ideas de ciencia ficción, hoy han llegado a ser realidad.

Uno de estos avances tecnológicos de los cuales impactaron de mayor manera debido a su evolución rápida, es el uso de computadoras en todas partes. Desde un reloj que puede checar tu pulso, cuántos pasos has caminado, si te has caído y necesitas ayuda, hasta un automóvil que puede medir la temperatura del ambiente, si hay carros alrededor de sí mismo, o en el caso de nuestra problemática, puede reproducir música y desplegar la canción que se está reproduciendo junto con el nombre del autor, entre otras cosas.

Debido a que esta tendencia de crear dispositivos que trabajen entre ellos, recibiendo y mandando información de uno a otro, es el por qué y la importancia de este reto. Dado a que con ello se aprenderá de cómo establecer esta conexión y las diferentes aplicaciones que puede llegar a tener.

Metodología

Para nuestro reto se nos pidió que crearamos un sistema de infoentretenimiento utilizando un microcontrolador y una Raspberry Pi que puedan comunicarse de manera serial con el propósito de reproducir música y de simular la creación de sistemas similares a los que existen dentro de los automóviles.

Como se mencionó con anterioridad se hizo uso de un microcontrolador de tipo STM32F103C8T6 que cuenta con procesador de ARM Cortex M4, este es el dispositivo que permite la transferencia de información entre componentes, para programar el microcontrolador se hizo uso del STM32Cube IDE que utiliza C como lenguaje de programación.

Además del microcontrolador se utilizó una Raspberry Pi 3 la cual es una computadora de tamaño compacto y de bajo costo que se encargaba de recibir datos del microcontrolador y regresarlos respectivamente en el momento que se necesitaran. Para programar con la Raspberry Pi se utilizó el lenguaje de programación Python.

Por último se utilizaron varios periféricos entre ellos, un teclado matricial de 4x4 el cual era el que se utilizó para controlar el cambio de canciones, el volumen y el correr, pausar o detener una canción. Para mostrar el nombre de la canción que se estaba escuchando al igual que el nombre del artista se utilizó una LCD de 16x2 y por último se usaron bocinas para poder reproducir la música.

Manual de Instrucciones del funcionamiento del Sistema

El funcionamiento del prototipo es el siguiente, primeramente el usuario tiene que tener conectado algún tipo de periférico que tenga como propósito el de reproducir música, ya sea una bocina o unos audífonos, conectados a la Raspberry Pi, los cuales pueden ser conectados vía bluetooth o cable auxiliar.

Al estar seguro de tener conectado un periférico con las características mencionadas anteriormente, se tiene el teclado matricial que cuenta con 16 botones de los cuales se utilizaron solo la mitad para controlar el prototipo, estos siendo los siguientes:

- 2: Play
- 4: Backward
- 5: Pausa
- 6: Forward
- 8: Stop
- A: Volume Up
- B: Volume Down
- C: Mute

Dependiendo de la función que se desee llevar a cabo el usuario tiene que presionar el botón adecuado para que el prototipo la lleve a cabo.

Después de haber presionado algunos de los botones y que algunas de las canciones se estén reproduciendo, en la pantalla LCD se desplegará el nombre de la canción que se esté reproduciendo junto con el nombre del autor.

Marco Teórico

Hoy en día estamos rodeados de tecnología, es altamente difícil tratar de llevar a cabo cualquier actividad sin algún aparato electrónico ayudándonos. Puede que esto parezca una exageración, pero simplemente con el hecho de que para levantarse para ya sea ir a trabajar o a la escuela se necesita de alguna alarma ya sea que provenga de un celular propio, o actividades como bañarse o agarrar algo para comer antes de irse de la casa, seguramente se hizo uso de un aparato eléctrico, estos siendo un boiler y refrigerador respectivamente. El uso de estos dispositivos eléctricos es casi inescapable, sin embargo, en tiempos pasados esto no era así, el uso de este tipo de dispositivos en nuestra vida cotidiana fueron gradualmente incrementando gracias a la creación de un pequeño circuito integrado llamado microcontrolador.

Estos pequeños dispositivos son un circuito integrado compacto que diseñado para desempeña una función específica en un sistema embebido, estos cuenta generalmente con un procesador, memoria y puertos I/O todo dentro de un chip. Los microcontroladores están integrados dentro de un sistema con el propósito de controlar una función singular en un dispositivo. Lo hace interpretando los datos que recibe mediante los periféricos en sus puertos de I/O utilizando su procesador central. La información temporal que recibe el microcontrolador se almacena en su memoria de datos, donde el procesador accede a ella y utiliza las instrucciones almacenadas para descifrar y aplicar los datos entrantes. Después de esto utiliza sus periféricos de I/O para comunicarse y ejecutar la acción adecuada.

En 1971 Gary Boone junto con Michael Cochran diseñaron el primer microcontrolador con la intención de utilizarlo para calculadoras de la compañía Texas

Instruments. Este primer microcontrolador servirá como base para la línea de microcontroladores conocidos como TMS1000 siendo MCU de 4-bits de uso general que se anunciaría en 3 años después en 1974. Este sería un evento importante en la historia debido a que esto introdujo al público general al uso de electrónicos digitales, debido a que esta familia de MCU se utilizaría en dispositivos como alarmas, puertas automáticas, e incluso juguetes. La competencia para seguir innovando en esta área ya que poco después en 1976 Intel y Mostek introducirían la arquitectura de 8-bits que servirían para aplicaciones más demandantes tal como en el uso de la industria automotriz y el área de periféricos para PC. Para los años 80s compañías alrededor del mundo ya estaban manufacturando distintas arquitecturas de microcontroladores para todo tipo de aplicaciones, desde funciones especializadas como para el uso público. Lo cual con el paso del tiempo se integraría como una práctica común, convirtiendo a los microcontroladores en los dispositivos más ubicuos.

Como se mencionó anteriormente los microcontroladores pueden usarse por sí solos, sin embargo, las tareas y funciones que se pueden realizar con estos son limitadas por lo cual el uso de una Raspberry Pi fue necesaria para la realización del reto.

La Raspberry Pi es una serie de computadoras fabricadas por la Fundación Raspberry Pi, una organización del Reino Unido que tiene como objetivo educar a personas en el área de informática y facilitar el acceso a la educación. Esta computadora de placa única se lanzó al mercado en el 2012 y desde entonces se han lanzado varias iteraciones y variaciones. Debido al conjunto de pines GPIO que le permite controlar componentes electrónicos.

Diagrama Esquemático General del Sistema

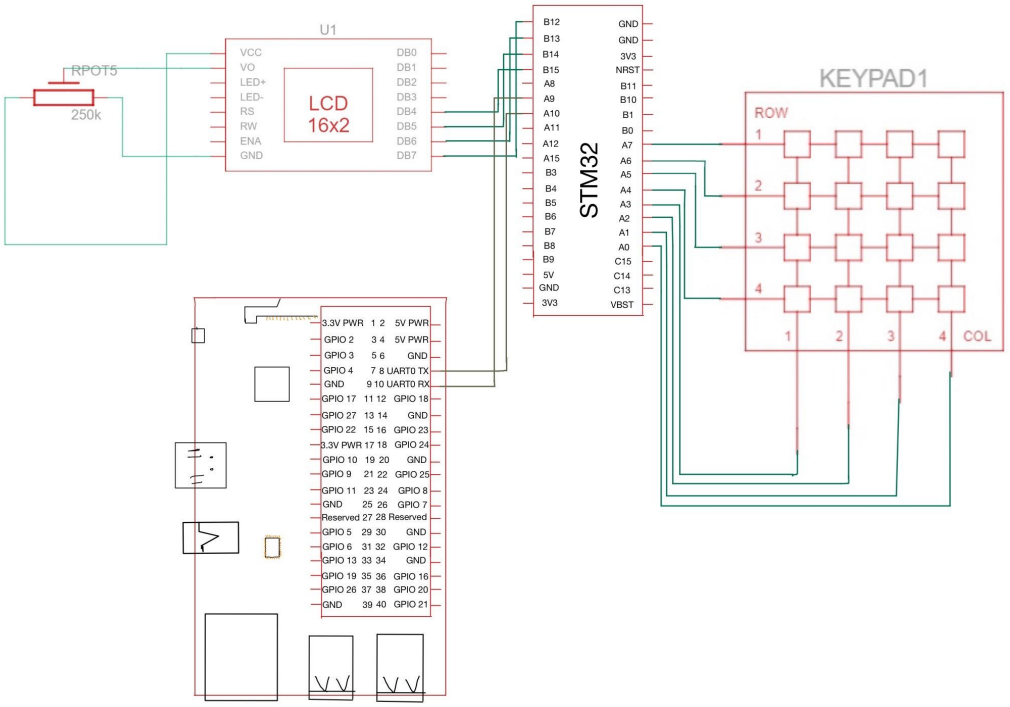

Diagrama 1. Diagrama Esquemático del sistema.

En el diagrama esquemático podemos observar las conexiones entre los distintos componentes que utilizamos para hacer este reto. Podemos ver que la STM32 es el principal componente ya que todos los demás están conectados a la BluePill. Se

conecta la LCD a los pines PB12-PB15 de la STM, el teclado matricial a los pines PA0-PA7 y la Raspberry Pi se conecta mediante los pines UART. El UART RX de la Raspberry se conecta al pin PA9 (UART TX) de la STM32 y el UART TX se conecta al pin PA10 (UART RX) de la STM.

Línea de Tiempo de Calendarización de Tareas del CMSIS-RTOS

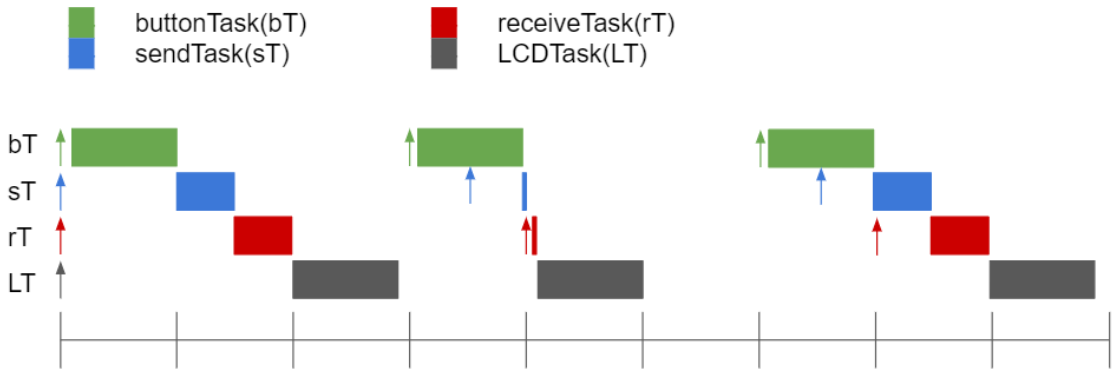

Diagrama 2. Calendarización de Tareas.

Para la realización del reto se utilizaron cuatro tareas: buttonTask, sendTask, receiveTask y LCDTask, las cuales se explicarán más adelante en la implementación de la STM32. De estas tareas, la buttonTask tiene la prioridad más alta, sendTask y receiveTask tienen prioridad media y LCDTask tiene la menor prioridad. El tiempo de ejecución de cada una de estas tareas es variable, pues dependiendo de las acciones del usuario pueden durar más o menos tiempo, por lo que en la calendarización (Diagrama 2) se representan con diferentes longitudes respetando las prioridades y los tiempos de espera de cada tarea. Puede notarse que si bien sendTask y receiveTask tienen la misma prioridad, sendTask se ejecuta antes debido a que si no se ha mandado nada antes, es poco probable que haya un dato que recibir.

Explicación de partes fundamentales del código en el SMT32

En la primera parte del código main, después de inicializar las funciones a utilizar y los componentes necesarios del microcontrolador, se definen los diferentes threads que se van a estar utilizando (Código 1). En este caso se utilizan cuatro, uno para cada tarea, y se establecen las prioridades como ya fue mencionado en la calendarización.

```
/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
/* Create the task, storing the handle. */
osThreadDef(sTaskHandle, senderTask, osPriorityAboveNormal, 0, 128);
sTaskHandle = osThreadCreate(osThread(sTaskHandle), NULL);

osThreadDef(rTaskHandle, receiverTask, osPriorityAboveNormal, 0, 128);
rTaskHandle = osThreadCreate(osThread(rTaskHandle), NULL);

osThreadDef(bTaskHandle, buttonTask, osPriorityHigh, 0, 128);
bTaskHandle = osThreadCreate(osThread(bTaskHandle), NULL);

osThreadDef(lTaskHandle, LCDTask, osPriorityBelowNormal, 0, 128);
lTaskHandle = osThreadCreate(osThread(lTaskHandle), NULL);
/* USER CODE END RTOS_THREADS */
```

Código 1. Creación de tareas.

Posteriormente se definen dos queues en las que se mandarán datos entre las tareas. La primera queue es utilizada para mandar información entre buttonTask y sendTask, mientras que la segunda es utilizada para mandar datos entre receiveTask y LCDTask.

```
/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
osMessageQDef(MsgQueueHandle, 1, int);
MsgQueueHandle = osMessageCreate(osMessageQ(MsgQueueHandle), NULL);

osMessageQDef(MsgQueueHandle1, 1, int);
MsgQueueHandle1 = osMessageCreate(osMessageQ(MsgQueueHandle1), NULL);
/* USER CODE END RTOS_QUEUES */
```

Código 2. Creación de queues.

La primera tarea, buttonTask, lee los valores de los pines asociados al teclado matricial conectado a la STM32 y determina si se ha presionado alguno de los botones de acción del sistema, los cuales fueron explicados en el manual de instrucciones. En caso de detectarse una acción válida, manda a través del primer queue el código Ascii de la información recibida al senderTask. De igual manera, si no se recibe una acción pero se establece el valor de la variable “b” como 0, la cual se explicará más adelante en la LCDTask, se manda un valor de 0 por el queue. Finalmente se establece un delay que le permite a las demás tareas operar y después recibe un valor a trav[es del queue que le indica que ha terminado la tarea.

```
void buttonTask(void const * argument)
{
  osStatus s_event1;
  osEvent r_event1;
  /* Infinite loop */
  int a;
  uint8_t next;

  for(;;)
  {
          a = 0;
          //Teclado matricial
          GPIOA->ODR |= GPIO_ODR_ODR4;
          GPIOA->ODR &= ~GPIO_ODR_ODR5;
          GPIOA->ODR |= GPIO_ODR_ODR6;
          GPIOA->ODR |= GPIO_ODR_ODR7;

          if (BUTTON0 == 0){
              HAL_Delay(10);
              a = 62; // >
          }
          else if (BUTTON2 == 0){
              HAL_Delay(10);
              a = 56; // 8
          }

              GPIOA->ODR |= GPIO_ODR_ODR4;
              GPIOA->ODR |= GPIO_ODR_ODR5;
              GPIOA->ODR |= GPIO_ODR_ODR6;
              GPIOA->ODR &= ~GPIO_ODR_ODR7;

              if (BUTTON0 == 0){
                  HAL_Delay(50);
                  a = 60; // <
              }
              else if (BUTTON2 == 0){
                  HAL_Delay(10);
                  a = 50; // 2
              }

              if(a!=0 || b==0){
                s_event1 = osMessagePut(MsgQueueHandle, a, 1000);
                osDelay(100);
                r_event1 = osMessageGet(MsgQueueHandle, 1000);
              }
```

```
GPIOA->ODR |= ~GPIO_ODR_ODR4;
GPIOA->ODR |= GPIO_ODR_ODR5;
GPIOA->ODR &= ~GPIO_ODR_ODR6;
GPIOA->ODR |= GPIO_ODR_ODR7;

if (BUTTON0 == 0){
    HAL_Delay(50);
    a = 61; // =
}
else if (BUTTON1 == 0){
    HAL_Delay(10);
    a = 54; // 6
}
else if (BUTTON2 == 0){
    HAL_Delay(50);
    a = 53; // 5
}
else if (BUTTON3 == 0){
    HAL_Delay(10);
    a = 52; // 4
}
```

Código 3. buttonTask.

La senderTask recibe el valor del queue y dependiendo de lo que se haya mandado, envía la información correspondiente a través del puerto USART a la Raspberry Pi 3 con la función USER_USART1_Transmit. Cuando se recibe el valor de una acción que implique escribir en la pantalla LCD, se establece el valor de “b” en 0 para indicar que hay cosas por escribir. En caso de que “b” ya sea 0 no se ejecuta ninguna acción y se pasa directamente a la siguiente tarea.

```
void senderTask(void const * argument)
{
    osStatus s_event;
    osEvent r_event;
      /* Infinite loop */
    for(;;){
          r_event = osMessageGet(MsgQueueHandle, 1000);
          if( r_event.status == osEventMessage ){
              if(b==1){
                  if (r_event.value.v == 50){
                      USER_USART1_Transmit("2",sizeof("2"));
                      b = 0;
                  }
                  else if (r_event.value.v == 56){
                      USER_USART1_Transmit("8",sizeof("8"));
                      b = 0;
                  }
                  else if (r_event.value.v == 53){
                      USER_USART1_Transmit("5",sizeof("5"));
                  }
                  else if (r_event.value.v == 54){
                      USER_USART1_Transmit("6",sizeof("6"));
                      b = 0;
                  }
                  else if (r_event.value.v == 52){
                      USER_USART1_Transmit("4",sizeof("4"));
                      b = 0;
                  }
                  else if (r_event.value.v == 60){
                      USER_USART1_Transmit("A",sizeof("A"));
                  }
                  else if (r_event.value.v == 61){
                      USER_USART1_Transmit("B",sizeof("B"));
                  }
                  else if (r_event.value.v == 62){
                      USER_USART1_Transmit("C",sizeof("C"));
                  }
                  else if (r_event.value.v == 0){
                  }
              }


              osDelay(100);
              s_event = osMessagePut(MsgQueueHandle, 0, 1);
```

Código 4. senderTask.

La receiverTask lee cada 90 microsegundos el valor que se haya recibido a través del puerto USART con la función USER_USART1_Receive. Si se recibe un dato válido y se indica que hay cosas pendientes por escribir, manda el valor al segundo queue que lo manda a la siguiente tarea, dando tiempo a través de un delay de que la información se reciba y se interprete en esta. Finalmente lee el valor de la queue para indicar que terminó de ejecutarse la tarea.

```
void receiverTask(void const * argument)
{
      /* USER CODE BEGIN 5 */
    osStatus s_event3;
    osEvent r_event3;

      /* Infinite loop */
      uint8_t d;

      for(;;)
      {
              HAL_Delay(90);
              d = USER_USART1_Receive();
              if(d!='{' && b==0){
                s_event3 = osMessagePut(MsgQueueHandle1, d, 1000);
                osDelay(100);
                r_event3 = osMessageGet(MsgQueueHandle1, 2000);
              }
      }
}
```

Código 5. receiverTask.

Finalmente, la LCDTask lee el valor de la segunda queue, cuyo valor es mandado desde la Raspberry. Se consideran ciertos caracteres especiales para dar indicaciones a la pantalla LCD, tales como el "}", el cual indica que se va a ingresar nueva información y debe borrarse la anterior. Si se manda un "_" indica que a partir de ahí se da información del artista, la cual debe ser escrita en la segunda fila de la pantalla. Si se manda un "." se indica que se ha terminado de mandar la información y se debe cambiar el valor de "b" nuevamente a 1 para que puedan mandarse datos a la Raspberry. Fuera de estos caracteres, se considera que los demás son parte de la información y deben ser impresos en la pantalla. Finalmente se envía un valor a través de la queue para indicar que terminó la tarea.

```
void LCDTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
osStatus s_event4;
osEvent r_event4;
uint8_t data;
int x = 0;
int y = 1;
char a;
  /* Infinite loop */
for(;;){
      r_event4 = osMessageGet(MsgQueueHandle1, 1);
      if( r_event4.status == osEventMessage ){
          data = r_event4.value.v;
          a = (char) data;
          HAL_Delay(110);
          if (data == 46){
              b = 1;
          }
          if (data != 46){
              if (data == 125){
                  LCD_Clear( );//           borra la pantalla
                  LCD_Set_Cursor( 1, 0 );
                  x = 0;
              }
              else if (data == 124){
                  if (sizeof(data))
                  LCD_Put_Char("a" );
                  x++;
                      }
              else if (data == 13){
                  y = 2;
                  x = 0;
                  LCD_Set_Cursor( y, x);
                  LCD_Put_Str("                ");
                  LCD_Set_Cursor( y, x);
              }
              else if (data == 95){
                  y = 2;
                  LCD_Set_Cursor( y, 0);
              }
              else {
                  if (x < 16){

                      if (sizeof(a))
                      LCD_Put_Char(a );
                      x++;
                  }
                  else{

                      if (sizeof(data))
                      LCD_Put_Char( data );

                      x++;
                  }
              }
          }
      s_event4 = osMessagePut(MsgQueueHandle1, 1, 1);
      }
  /* USER CODE END 5 */
}

}
/* USER CODE END 4 */
```

Código 6. LCDTask.

Explicación de partes fundamentales del código en la Raspberry Pi

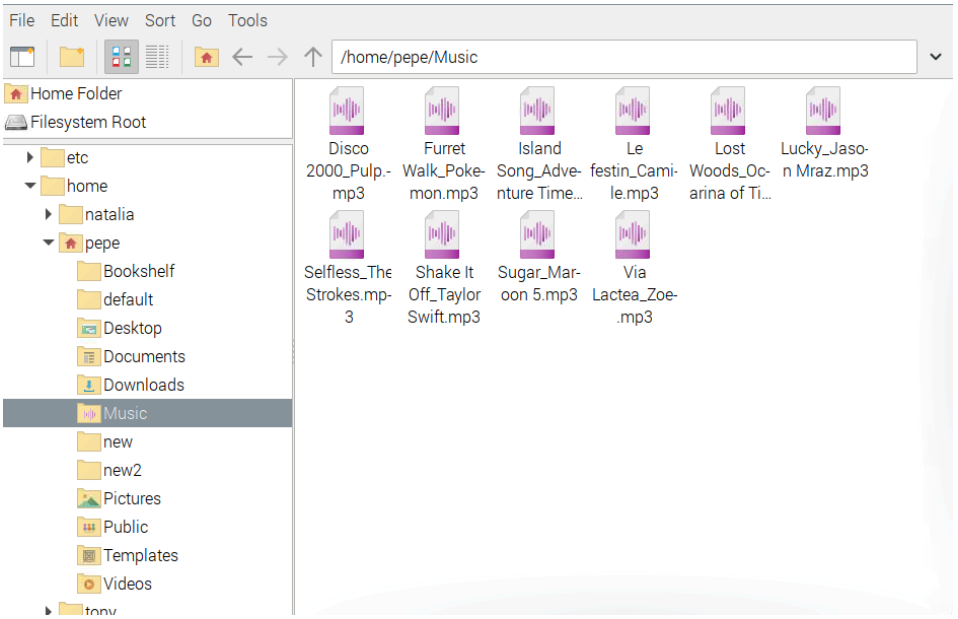

Imagen 1. Carpeta de música.

Se utilizó la carpeta Music que se encontraba en la Raspberry para guardar las canciones que se iban a reproducir y poder acceder a ellas desde el programa de Python. Los nombres de los archivos siguen el formato [Nombre_Artista.mp3] para facilitar la escritura de este en la LCD.

Creación de la Playlist

```
import serial
import time
import os
import vlc
```

```
12 instance = vlc.Instance('--aout=alsa')
13
14 dir = '/home/pepe/Music/'
15
16 with os.scandir(dir) as ficheros:
17     ficheros = [fichero.name for fichero in ficheros if fichero.is_file()]
18
19 player = vlc.Instance()
20
21 media_player = vlc.MediaListPlayer()
22
23 media_list = vlc.MediaList()
24
25 for i in range (len(ficheros)):
26     media = player.media_new(dir + ficheros[i])
27     media_list.add_media(media)
28
29 media_player.set_media_list(media_list)
```

Código 6. Inicialización de objetos para el reproductor.

Se guardó la dirección del directorio música que contiene los archivos a reproducir y con la librería os se realizó un escaneo del mismo para agregar los nombres de cada archivo a una lista. Con esta lista, se utilizó la librería vlc para crear una instancia de MediaList Player, la cual permite manipular una lista de reproducción a través de los archivos que incluyas en ella, en este caso se crea un objeto Media para cada canción usando sus nombres y la dirección en la que se encuentran y se añaden a la lista. Finalmente, esta lista se asigna a el MediaPlayer que permitirá hacer las acciones del usuario en la playlist.

Comunicación Serial

```
6  ser = serial.Serial ("/dev/ttyS0", 9600, timeout=1000,
7      bytesize=serial.EIGHTBITS,
8      parity=serial.PARITY_NONE,
9      stopbits=serial.STOPBITS_ONE)     #Open port with baud rate
10
```

Código 7. Abrir puerto serial.

Se definió el puerto de comunicación serial de la Raspberry Pi utilizando la dirección /dev/ttyS0 correspondiente a los pines Rx y Tx presentes en ella. Después, se configuró el baudrate a 9600 de forma que tuviese el mismo que la STM32 para lograr comunicar efectivamente los datos de una a otra. El resto de parámetros se definieron de igual manera de la misma manera que en la STM32, no se paridad, se configura un solo bit de stop y el tamaño en bits de los datos es de 8.

```
while True:
                    #transmit data serially
    dataRed = ser.read().decode("Ascii")
    if (dataRed == "2") :
        media_player.play()
        ser.write("}".encode("Ascii"))
        time.sleep(t)
        if (index == -1):
            index =0
        dataT = ficheros[index]
        size= len(dataT)
        for i in range (size-3) :
            ser.write(dataT[i].encode("Ascii"))
            time.sleep(t)
```

Código 8. Leer UART y función de Play.

Posteriormente, con ayuda de la función read se reciben datos provenientes del STM32, el cual se guarda en la variable que determina las acciones del reproductor tales como play, pause, forward, entre otras. En caso de ser necesario, se utiliza la función write para mandar el nombre del archivo guardado previamente a la STM32 caracter por caracter. Esto último se realiza de esta manera debido a que es necesario dejar pasar un tiempo entre cada caracter enviado para que la STM32

pueda leerlo correctamente, lo cual se implementó utilizando la función sleep de la librería time.

Manipulación de la Playlist

```
                    elif (dataRed == "4"):
                        media_player.previous()
                        if (index < 0 ):
                            index = 0
                            ser.write("}".encode("Ascii"))
                            time.sleep(t)
                            dataT = ficheros[index]
                            size = len(dataT)
                            for i in range (size-3) :
                                ser.write(dataT[i].encode("Ascii"))
                                time.sleep(t)
elif (dataRed == "5"):
    media_player.pause()

elif (dataRed == "6"):
    ser.write("}".encode("Ascii"))
    time.sleep(t)

    if (index < len(ficheros)-1):
        media_player.next()
        index += 1
    else:
        media_player.stop()
        media_player.play()
        index = 0
    dataT = ficheros[index]
    size= len(dataT)
    for i in range (size-3) :
        ser.write(dataT[i].encode("Ascii"))
        time.sleep(t)
```

```
elif (dataRed == "8"):
    media_player.stop()
    ser.write("}".encode("Ascii"))
    time.sleep(t)
    index = -1
    dataT = "Stopped."
    size= len(dataT)
    for i in range (size) :
        ser.write(dataT[i].encode("Ascii"))
        time.sleep(t)
```

Código 9. Funciones de Previous, Pause, Forward y Stop.

Aquí se configuró que hace cada una de las teclas del teclado matricial para poder manipular la playlist. Se utiliza el 5 para poner pausa a las canciones, el 6 para pasar a la siguiente canción, si la canción no es la última pone la siguiente, si es la última, vuelve a empezar desde el principio. El 8 se utiliza para detener las canciones y te regresa al principio, además despliega “Stopped” en el LCD. Y la tecla 4 reproduce la canción anterior, si es la primera canción, vuelve a empezar.

```
    elif (dataRed == "A"):
        player = media_player.get_media_player()
        if (vol <= 90):
            vol = vol + 10
        else :
            vol = 100
        player.audio_set_volume(vol)
        time.sleep(.01)

    elif (dataRed == "B"):
        player = media_player.get_media_player()
        if (vol >= 10):
            vol = vol - 10
        else :
            vol = 0

    elif (dataRed == "C"):
        player = media_player.get_media_player()
        mute = player.audio_get_mute()
        player.audio_set_mute(not mute)
```

Código 10. Funciones de control de volumen.

También se configuraron las teclas A, B y C para modificar el volumen. La tecla A sube el volumen 10 decibeles, la B disminuye el volumen 10 decibeles y la letra C pone y quita mute a las canciones.

Conclusión

Durante el transcurso de 10 semanas reforzamos conocimientos previos que conseguimos en el bloque pasado tal como el uso de sistemas embebidos, al igual que su funcionamiento. Sin embargo, también adquirimos nuevas habilidades y conocimientos, entre ellos aprendimos acerca de las arquitecturas ARM y LEG, al igual que cómo emplearlas.

Conocimos acerca de los microcontroladores, sus usos, el cómo programarlos utilizando el lenguaje de programación C. Con esto llegamos a tener un mejor entendimiento de cómo es que los dispositivos que utilizamos día con día funcionan.

Por último aprendimos acerca de un sistema operativo nuevo, Linux, y de cómo se puede programar en este utilizando la terminal gracias al lenguaje Bash.

Todo esto en conjunto hizo de este proyecto un tanto retador al momento de tener que juntarlo todo y lograr que los datos que quisiéramos enviar desde el microcontrolador a la Raspberry Pi si se recibieran de la manera correcta. Sin embargo, fue un reto bastante interesante.

Reflexiones individuales

Armando: Durante este bloque aprendimos acerca de la estructura y operación de microcontroladores a través de lenguaje ensamblador, así como funciones del mismo, aplicado en microprocesadores ARM, tal como el Cortex-M empleado en la STM32. Esto nos permitió comprender a mayor profundidad la manipulación a nivel bit necesaria para

configurar este sistema y poder emplearlo en una aplicación de la vida real. De igual forma, empleamos la blue pill para poner en práctica el uso de sistemas en tiempo real, manejo de tareas y multithreading, los cuales son conceptos de suma importancia para desarrollarnos en nuestra carrera. Por otro lado, aprendimos acerca de sistemas embebidos y el uso del sistema operativo Linux como una opción práctica y sin costo para la implementación de aplicaciones que requieran un sistema operativo para el control de periféricos, tales como unidades de procesamiento central en aplicaciones específicas. Realizar este reto nos da herramientas para realizar proyectos en la vida real y nos da una buena base para seguir aprendiendo y experimentando con esta clase de temas interesantes que veremos a lo largo de la carrera.

Yosel: Durante este bloque pude aprender muchísimo acerca del funcionamiento y la presencia de los sistemas embebidos en mi vida cotidiana. Este reto en particular me mostró cómo crear algo que llevo utilizando la mayor parte de mi vida, un sistema reproductor de audio que responde a interacciones de un usuario a partir de un microcontrolador. Aprender a configurar cada sección del STM32 para que cumpliera con ciertas funciones o se habilitarán los modos de operación me permitió ver el grado de complejidad que tienen las herramientas que uso a diario que operan con sistemas embebidos. Me siento mucho más preparado para continuar trabajando con este tipo de temas y espero poder aprender más sobre esto y el uso de periféricos y herramientas de computación para mejorar los productos que realice a lo largo de mi carrera.

José Antonio: Este bloque fue bastante interesante y muy retador. Se nos impartieron distintos temas en las clases y tuvimos que reunir todos los conocimientos adquiridos desde el día 1 y utilizarlos para poder realizar este reto. Al final del día fue muy gratificante ver cómo todo nos fue útil para entender lo que se nos pedía y poder realizarlo de la manera correcta, el poder aplicar los conocimientos en algo práctico y tangible. Siento que los temas vistos en este bloque son muy importantes y relevantes para el futuro de la carrera de mi profesión y tengo interés en aprender más al respecto.

Natalia: Este bloque que llevamos por 10 semanas fue bastante interesante, me gusto que fue algo de continuación del bloque pasado, por que con ello puede entender algunas cosas que anteriormente no lo había comprendido en el momento. Fue muy interesante el reto que se nos pidió desarrollar, aunque hay todavía varios temas que no llegué a entender del todo sí me gustaría poder llegar a hacer mi propio proyecto aparte y utilizar los conocimientos que se me impartieron este semestre.

Video:
https://youtu.be/5K5LPer3tvA

https://drive.google.com/file/d/157F94K-0c7QHzbYy3taGMDigypSL6Dfp/view?usp=sharing

Referencias

1974: General-purpose microcontroller family is announced. 1974: General-Purpose Microcontroller Family is Announced | The Silicon Engine | Computer History Museum. (n.d.). Retrieved June 16, 2022, from https://www.computerhistory.org/siliconengine/general-purpose-microcontroller-family-is-announced/

Computer Chips Inside Cars. Vintage Computer Chip Collectibles, Memorabilia & Jewelry. (n.d.). Retrieved June 16, 2022, from https://www.chipsetc.com/computer-chips-inside-the-car.html

Computer Chips Inside Cars. Vintage Computer Chip Collectibles, Memorabilia & Jewelry. (n.d.). Retrieved June 16, 2022, from https://www.chipsetc.com/computer-chips-inside-the-car.html#:~:text=1978%20%2D%20Cadillac%20introduces%20a%20microprocessor,bit%20chips%20manufactured%20by%20Toshiba.

Lutkevich, B. (2019, November 7). *What is a microcontroller and how does it work?* IoT Agenda. Retrieved June 16, 2022, from https://www.techtarget.com/iotagenda/definition/microcontroller

Nice, K. (2001, April 11). *How car computers work*. HowStuffWorks. Retrieved June 16, 2022, from https://auto.howstuffworks.com/under-the-hood/trends-innovations/car-computer.htm#:~:text=Each%20year%2C%20cars%20seem%20to,your%20car%20easier%20to%20service.

Nice, K. (2001, April 11). *How car computers work*. HowStuffWorks. Retrieved June 16, 2022, from https://auto.howstuffworks.com/under-the-hood/trends-innovations/car-computer1.htm

Tomlinson, Z. (2018, December 11). *What are microcontrollers and why should you care?* Interesting Engineering. Retrieved June 16, 2022, from https://interestingengineering.com/what-are-microcontrollers-and-why-should-you-care

What is a Raspberry Pi and how does it work? Pi Day. (n.d.). Retrieved June 16, 2022, from https://www.piday.org/whats-a-raspberry-pi-and-how-does-it-work/

What is a Raspberry Pi? Opensource.com. (n.d.). Retrieved June 16, 2022, from https://opensource.com/resources/raspberry-pi