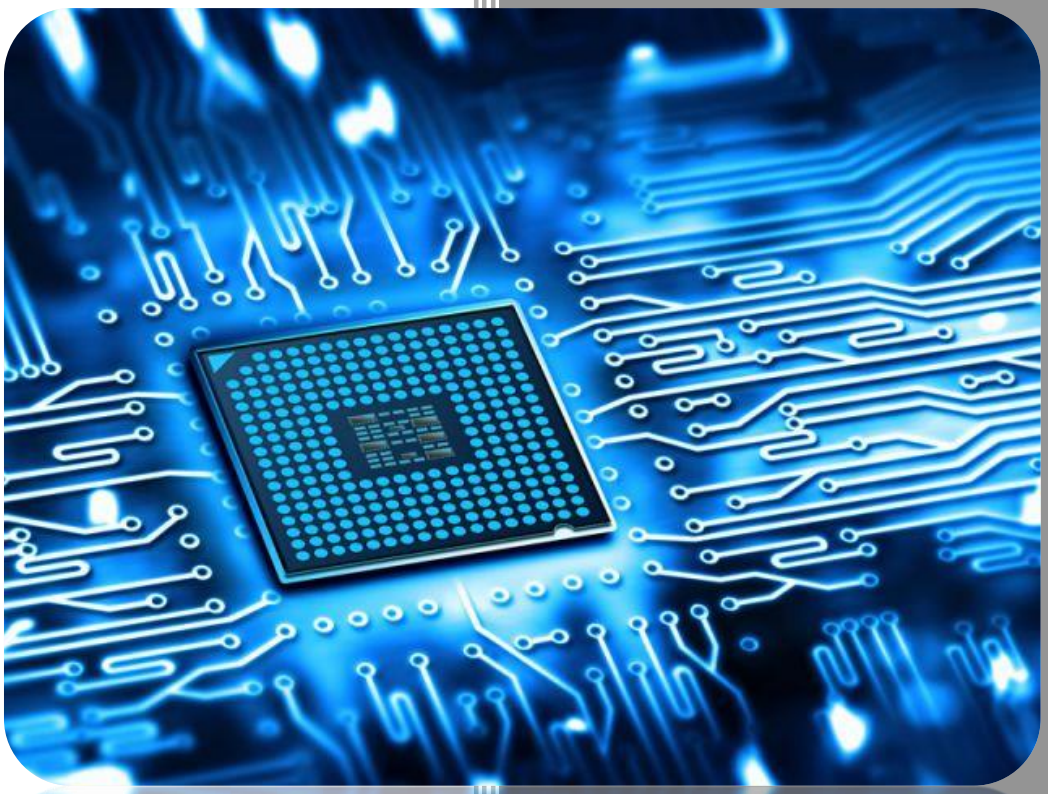
# TP3

## Circuitos digitales y microcontroladores



UNIVERSIDAD
NACIONAL
DE LA PLATA

Arreche, Cristian Carlos 01515/4
Blasco, Federico Matías 01678/4
10-7-2019

# Índice

# 1. Introducción

## 1.1. Problema

*Utilizando los periféricos SCI y TPM, se desea implementar un generador de sonidos controlado desde un teléfono inteligente mediante el protocolo Bluetooth.*

*El MCU deberá generar diferentes sonidos (con el TPM y un parlante) a partir de los comandos recibidos desde una terminal serie en un teléfono y además deberá actualizar la información en la pantalla enviando los mensajes adecuados y generando una interfaz amigable al usuario. El enlace inalámbrico se implementará con un módulo Bluetooth tipo HM-10 conectado al periférico SCI del MCU.*

## 1.2. Interpretación

Dado el microcontrolador MC9S08SH8 CPJ, un módulo bluetooth y un parlante, se deberá realizar un generador de sonidos programable, que se comunique con un teléfono mediante una terminal serie de bluetooth.

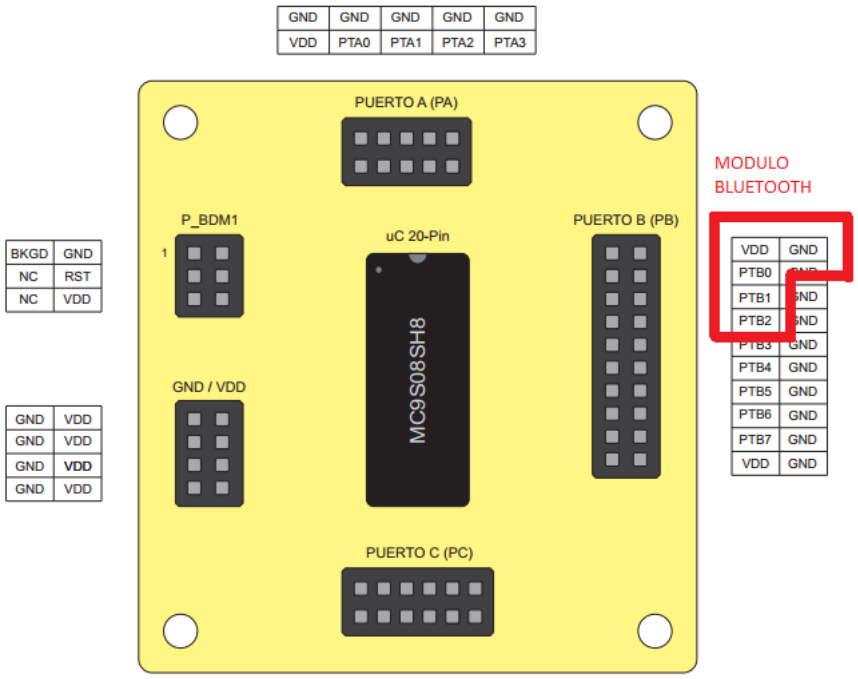
Como podemos observar, se deberá manejar la interacción con dos periféricos:

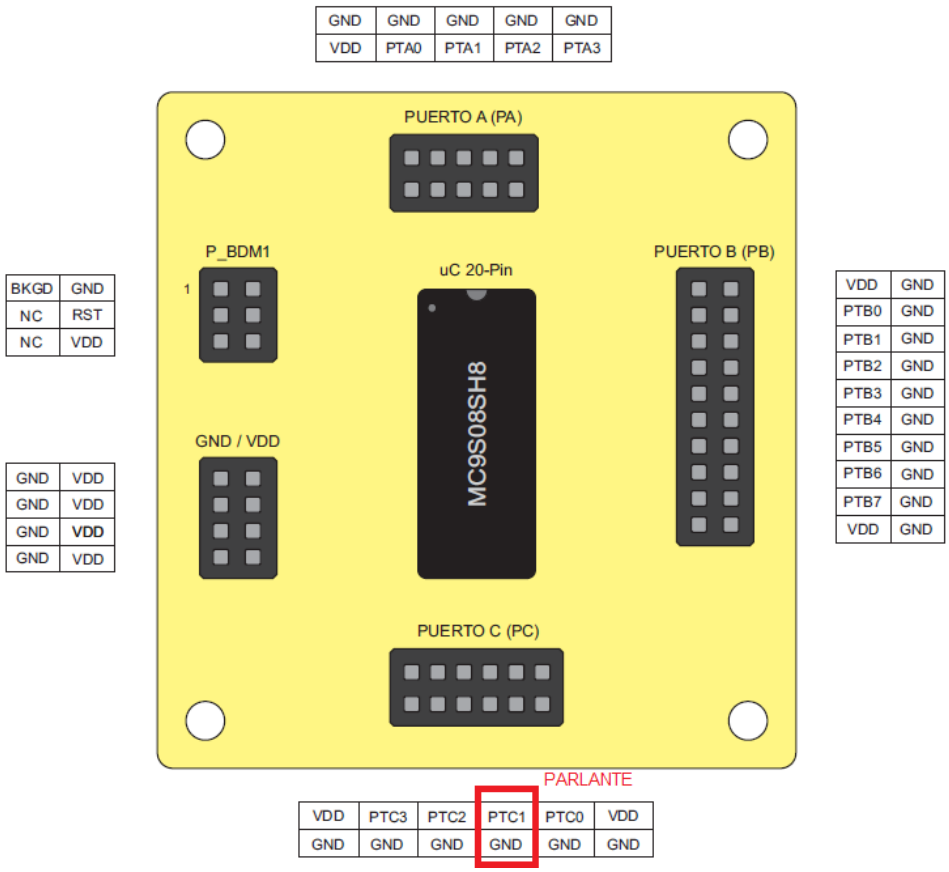
1. El TPM (temporizador): para la generación de sonido.
2. EL SCI (comunicación serie): para la comunicación con el teléfono.

Por otra parte, para la comunicación a través del puerto serie se deberá analizar el funcionamiento y configuraciones del periférico SCI provisto por el microcontrolador.

## 1.3. Resolución

El conexionado de los periféricos utilizados es de la siguiente manera:






## 2. Explicación del SCI

El módulo SCI es un periférico de comunicación serie, el cual nos servirá en la resolución de este trabajo práctico, permitiéndonos utilizar el módulo bluetooth tipo HM-10. Permite comunicación full-duplex, con formato NRZ, la transmisión y recepción son independientes, con doble buffer, puede operar en modo de bajo consumo, y permite utilizarlo en modo half-duplex si así se desea.

El mismo está compuesto por los siguientes subsistemas:

- Módulo transmisor
  - Básicamente un conversor de datos paralelo a serie.
  - Sincronizado con un reloj derivado del microcontrolador (En nuestro caso el reloj del bus).
  - Datos de 8 bits o 9 bits.
  - Permite utilizar bits de paridad de manera automática.
  - Para poder transmitir el programador debe verificar que el registro de datos SCID esté vacío (flag SCIS1_TDRE).
- Módulo receptor
  - Básicamente un conversor de datos serie a paralelo
  - Sincronizado con un reloj derivado del microcontrolador (En nuestro caso el reloj del BUS).
  - Datos presentes en el terminal de entrada del periférico son muestreados a una tasa 16 veces mayor a la seleccionada de manera de detectar el bit de comienzo y sincronizarse con los bits de datos.

    - Los datos son muestreados e introducidos al registro de desplazamiento hasta detectar el bit de parada o STOP.
    - Luego de completar la recepción se activa el flag de dato recibido SCIS1_RDRF, el cual puede disparar una interrupción si se lo desea.
    - Permite detectar distintos tipos de errores como la sobrecarga de datos, señal ruidosa, error de trama y error de paridad.
- Generador de tasa de transferencia
    - Permite elegir la velocidad a la cual se transmitirán los bits, utilizando los registros SCIBDH y SCIBDL.
- Registros de control y configuración
    - Como puede ser configurar si los datos son de 8 o 9 bits, bits de paridad, etc.
    - SCIxC2 permite configurar interrupciones, habilitar Rx y Tx, entre otras cosas
    - TDRE es otro registro útil que permite esperar hasta que el SCI esté disponible para transmitir.
    - Se puede esperar por RDRF para recibir y luego leer un dato desde SCID.

## 3. Explicación del TPM

El TPM es un módulo temporizador de 16 bits, con un módulo contador programable, interrupción por overflow y 2 canales independientes (CH0 y CH1) con generación de interrupción independiente para cada uno de ellos. A su vez, cada uno de estos canales puede ser utilizado en cualquiera de los siguientes modos de operación:

- Salida de comparación (Output compare - OC)
    - Este será el modo utilizado en esta práctica
- Entrada de captura (Input compare - IC)
- PWM alineado en flanco
- PWM alineado al centro

El microcontrolador utilizado en esta práctica posee 2 TPM, TPM1 y TPM2

## 4. Vinculación e inicialización del sistema

### 4.1. Problema

*El sistema debe iniciarse con el sonido apagado y mediante comandos AT configurar un nombre al dispositivo con el cual se realizará la conexión. Una vez establecida la conexión con el teléfono el mismo se comportará como una interfaz serie transparente.*

*En el teléfono se deberá tener una aplicación que implemente una terminal serie Bluetooth BLE. Opciones: HMBLE Terminal, BLE Terminal o Serial Bluetooth Terminal.*

### 4.2. Interpretación

Dado el microcontrolador MC9S08SH8 CPJ, un módulo bluetooth y una aplicación instalada en el celular, se necesita inicializar el microcontrolador con el sonido apagado y con todos los valores por defecto, tanto la frecuencia como el

nombre del módulo bluetooth. Luego de esta inicialización, se debe esperar a que se vincule correctamente el celular con el módulo bluetooth, para poder comenzar con el funcionamiento del sistema en cuestión.

Pseudocódigo:

```
Main(){
      //Inicializar variables y frecuencia por defecto
      //Desactivar interrupciones por TPM
      MCU_init();
      Enviar_nombre_ble("AT+NAMEG8\r\n");
      //Esperar a que se realice la vinculación
}
```

## 4.3. Resolución

El primer problema por resolver es cambiar el nombre del módulo bluetooth, para esto tenemos una lista de comandos útiles para configurar el mismo:

| **Comando** | **Recibe** |
|---|---|
| AT | Chequeo si el módulo está bien conectado |
| AT+NAME | Nombre para mostrar del módulo |
| AT+PIN | Pin actual del módulo |
| AT+RESET | OK+RESET en caso de que se resetee satisfactoriamente |

Para la resolución de este trabajo utilizaremos solamente un comando AT, el AT+NAME, este nos permitirá cambiar el nombre del módulo bluetooth por el que nosotros queramos. Cabe destacar que hay muchos más comandos AT utilizables con el HM-10, pero no tiene sentido listarlos a todos cuando no van a ser utilizados. Se envía el comando hacia el HM-10 con una función que será explicada más adelante.

```
void main(void) {
    char nombre_default[]="AT+NAMEG8\r\n"; //Nombre por defecto de nuestro modulo bluetooth
    char aux;
    char comando_comparable;
    char *comando;
    MCU_init();

    SCI_enviar_cadena(nombre_default); //Se envía el nombre por defecto al módulo
```

Como frecuencia por defecto se elegirá la de 100HZ (se explicará más adelante como se determina esta frecuencia) y se desactivan las interrupciones por TPM1 canal 1, ya que el sonido debe estar apagado al comienzo del programa. Las interrupciones son desactivadas colocando el registro TPM1C1SC en 0.

```
NC = F1;
TPM1C1SC = 0;//Desactivo las interrupciones por TPM1 CH1
```

Finalmente, lo único que queda es esperar la vinculación correcta con el teléfono. Esto se realiza haciendo un polling periódico con un bucle *while*, preguntando

por el pin 2 del puerto B (PTBD_PTBD2), el cual, si está en 0, indica que la conexión todavía no fue realizada.

```
//Espero a que se haga la vinculacion con el dispositivo
while(PTBD_PTBD2==0);
```

Esta parte fue particularmente difícil de realizar, ya que, por un motivo que desconocemos, no logramos que ninguno de nuestros teléfonos se vinculara satisfactoriamente a través de la aplicación. Nos llevó dos clases tratando de realizar la vinculación, pensando que era error nuestro de programación y/o configuración, hasta que descubrimos que nuestros celulares no eran compatibles con el protocolo bluetooth del HM-10 que utilizamos. La solución fue utilizar un celular de tecnología inferior, el cual se conectó sin problemas, permitiendo proseguir con la resolución del trabajo práctico.

# 5. Comunicación y funcionamiento

## 5.1. Problema

*Una vez establecida la comunicación con el teléfono (Bluetooth apareado), el MCU deberá enviar un mensaje de bienvenida a la pantalla de la terminal (por única vez) indicando como seleccionar la frecuencia de los sonidos (ver tabla ejemplo a continuación) y la lista de comandos disponibles para controlar la generación.*

| Comando | "100" | "300" | "500" | "1k" | "2k" | "5k" | "10k" | "12k" |
|---|---|---|---|---|---|---|---|---|
| frecuencia | 100Hz | 300Hz | 500Hz | 1kHz | 2kHz | 5kHz | 10kHz | 12kHz |

*Dichos comandos son: el usuario podrá encender o apagar el sonido mediante comandos ON-OFF y en cualquier caso el usuario puede volver al estado inicial con un comando de RESET.*

## 5.2. Interpretación

Una vez establecida la comunicación entre el teléfono y el microcontrolador, se deberá enviar un mensaje de bienvenida a la terminal bluetooth para que el usuario sepa qué comandos tiene disponibles y cómo utilizarlos. El mensaje a mostrar es el siguiente:

```
Bienvenido, elija su comando:
      - Comandos para configuracion de frecuencia:
          1) 100 --> 100Hz
          2) 300 --> 300Hz
          3) 500 --> 500Hz
          4) 1k --> 1000Hz
          5) 2k --> 2000Hz
          6) 5k --> 5000Hz
          7) 10k --> 10000Hz
          8) 12k --> 12000Hz
      - Comandos de control
          A) ON --> Encender parlante
          B) OFF --> Apagar parlante
          C) RESET --> Volver a inicializar el MCU
```

Donde los comandos de configuración de frecuencia permiten elegir la nota deseada a reproducir por el parlante, mientras que los de control permiten encender, apagar y reiniciar el generador de sonidos, este último vuelve a configurar la frecuencia por defecto, apaga el sonido y muestra nuevamente el mensaje de bienvenida.

Al mismo tiempo nuestro programa debe ser capaz de recibir los comandos desde el celular, interpretarlos, y actuar en consecuencia. Para esto la cátedra quiere que implementemos un buffer de recepción, y uno de transmisión, que permita guardar datos hasta que sean necesitados, tanto por el SCI como por el microcontrolador en sí para prender el parlante.

Pseudocódigo:

```
Main(){
      Si (recibo_comando)   //La interr de Rx lo pone en 1
            Leer_comando(comando);
            Ejecutar_comando(comando);
            Recibo_comando == 0;
}
```

## 5.3. Resolución

Para resolver el problema de la transmisión de mensajes y recepción de los comandos, creamos un archivo aparte llamado Buffer.c, el cual contiene los dos buffers (Rx y Tx), así como también las funciones necesarias para su correcto funcionamiento y utilización. Cada uno de estos buffers tiene un tamaño de 150 caracteres, para que sean lo suficientemente grandes como para enviar y recibir comandos y/o mensajes.

Cada uno de los buffers tiene dos índices, uno para lectura y otro para escritura. Además, se declara una variable global llamada RX_flag, la cual es utilizada para avisar al programa principal cada vez que se recibe un comando desde el celular.

### 5.3.1. Recepción

La interrupción isrVscirx, la cual se dispara cada vez que hay un carácter a recibir, llama a una función llamada escribir_en_bufferRX, que se encarga de escribir este carácter en el buffer de escritura.

Pseudocódigo:

```
escribir_en_bufferRX(){
      si(disponible_para_leer)
            //Leer el character del SCID
      if(no_es_el_ultimo_caracter)
            //Guardar el carácter en el buffer en la posición que
            //indique el índice de escritura
      Sino //se leyó el ultimo caracter
            //Avisar que ya está disponible el comando para leer
            RX_flag=1;
}
```

Cuando esta función avisa al programa principal que ya está listo el comando para ser leído, este llama a la función *leer_comando*, que devuelve el comando en forma de vector de caracteres.

Pseudocódigo:

```
leer_comando(comando){
      mientras(haya_caracter_en_buffer)
            //Guardar siguiente carácter en la variable comando
      //Reiniciar los dos índices, de lectura y escritura
}
```

Una vez recibido el comando desde el celular, se debe determinar: primero cual de todos los comandos fue ingresado, para esto se utiliza la función determinar_comando y comparar_comando, las cuales en conjunto permiten devolver al programa principal un carácter que identifica al comando ingresado por el usuario.

Pseudocódigo

```
determinar_comando(){
      si(comando_coincide_con_ON){
            //Devolver el carácter de identificacion de ON
      }
      //Misma comparación para cada uno de los comandos
}
```

Esto se hace de esta manera para que sea más fácil decidir qué acciones hacer en la función *ejecutar_comando*, la cual consiste simplemente de una estructura de selección múltiple (switch), el cual activa o desactiva las interrupciones del TPM en caso de que el comando sea ON/OFF, cambia la frecuencia actual en caso de que el comando ingresado sea uno de frecuencia, y reinicia el programa en caso de que el comando ingresado sea RESET. En todos los casos, se avisa al usuario por la consola del celular si su comando fue ingresado correctamente.

Pseudocódigo

```
ejecutar_comando(){
      switch(comando){
            case ‘A’://Acciones para el comando ON
                  //Avisar al usuario en la consola que
                  //funciono el comando
                  break;
            //Asi con cada uno de los comandos
      }
}
```

### 5.3.2. **Transmisión**

La transmisión comienza cuando el programa principal llama a la función *SCI_enviar_cadena*, la cual está definida en el archivo Buffer.c, y escribe la palabra a transmitir de a un carácter en el buffer de transmisión.

Pseudocódigo:

```
SCI_enviar_cadena(cadena){
	mientras(no_sea_fin_de_cadena){
		//Enviar un carácter al buffer de transmisión
		//Pasar al siguiente carácter
	}
	//Habilitar interrupciones de transmision
}
```

Para enviar de a un carácter en el buffer, se llama a la función *SCI_escribir_char_al_buffer*, la cual simplemente escribe un carácter en la posición del buffer que indique el índice de escritura e incrementa dicho índice.

Por último, la interrupción isrVscitx llama a la función SCI_update, la cual, si hay un carácter para transmitir, lo coloca en el SCID, y si no hay, reinicia los índices de transmisión, y deshabilita la interrupción de transmisión.

Pseudocódigo:

```
SCI_update(){
	si(Hay_caracter_para_transmitir)
		//Colocar el character en el SCID
		//Incrementar el índice de lectura
	sino
		//Reinicio los indices de transmisión
		//Deshabilito las interrupciones de
		//transmision
}
```

# 6. **Parlante**

## 6.1. **Problema**

*En la salida del canal 1 del TPM1 (PTCD1) se conectará un parlante, el cual debe poder cambiar la frecuencia del tono que está haciendo sonar, asi como también ser apagado y prendido cuantas veces se desee*.

## 6.2. **Interpretación**

Para la resolución de este problema, tendremos que aprender a utilizar el módulo TPM, el cual nos servirá para enviar las señales necesarias al parlante, y que este suene a la frecuencia que desee el usuario. Al mismo tiempo, el TPM debe ser controlado por el programa principal, ya que, dependiendo de los comandos ingresados por consola, habrá que modificar los valores de ciertos registros de este.

### 6.3. Resolución

Para hacer funcionar el parlante, se utiliza el módulo TPM, configurado en modo output compare. La interrupción *isrVtpm1ch1* se habilita en modo output compare (TPM1C1SC=0x54U) cuando el comando ON es ingresado, y se deshabilita (TPM1C1SC=0) cuando el comando OFF es ingresado.

Para elegir la frecuencia deseada, se coloca un valor (será calculado más adelante) en el registro TPM1C1V, para que cuando el counter del TPM llegue a este valor, se produzca un cambio en la salida del canal 1, permitiéndonos simular una onda cuadrada. Cuanto más grande sea el valor colocado en TPM1C1V, más tiempo tardará el counter en alcanzarlo, por lo tanto, la frecuencia de la señal cuadrada será menor. Utilizando el TPM1 de la manera que lo configuramos (Prescaler=1, Modulo counter = 0, Bus rate clock) la cuenta que hay que hacer para conseguir el valor a colocar (NC) en el TPM1C1V es la siguiente:

$$NC = \mathrm{Frec}_{\mathrm{deseada}} * 400$$

Cabe destacar que hay que sumarle NC al registro TPM1C1V cada vez que se dispara la interrupción, ya que el contador del TPM no se reinicia. Para la frecuencia deseada NC, se utiliza una variable global, la cual es modificada por el programa principal, dependiendo los comandos que ingrese el usuario.

Por último, se lee el flag TPM1C1SC_CH1F y se escribe un 0 en el mismo, de esta manera se limpia el flag de interrupción, para que pueda ser disparada nuevamente en el futuro.

```
__interrupt void isrVtpm1ch1(void)
{
    TPM1C1V += NC;
    TPM1C1SC_CH1F=0;
}
```

## 7. Arquitectura

### 7.1. Problema

*Respecto a la arquitectura del programa, la misma deberá ser del tipo Background/Foreground y además deberá aplicar modularización, abstracción y el modelo productor-consumidor para el periférico SCI. Es decir, las funciones para el manejo del periférico se deberán implementar con buffers y usando las interrupciones de Transmisión y Recepción.*

### 7.2. Interpretación

La arquitectura del programa debe ser del tipo background/foreground, esta arquitectura se caracteriza por tener varias interrupciones habilitadas al mismo tiempo, las cuales habilitan un flag al ejecutarse. Este flag es analizado en una estructura condicional en el programa principal constantemente, y cuando se cumpla que un flag esté en 1, se ejecutan una serie de acciones, que responden a este flag. Utilizaremos este tipo de arquitectura para recibir los comandos desde teclado, con las interrupciones de recepción del SCI.

### 7.3. Resolución

En el generador de sonidos se deberá utilizar una arquitectura de software de tipo Background/Foreground, donde las tareas en background se ejecutan en el lazo infinito del main en forma cíclica, mientras que las tareas en foreground son aquellas que se ejecutan mediante interrupciones, utilizadas para manejar eventos asincrónicos.

En lo que refiere a las tareas en foreground, debe considerarse el proceso de generación de sonido (a través de una interrupción que maneja su frecuencia), y la comunicación con el puerto SCI serie (generándose una interrupción cada vez que el usuario ingresa un comando y cada vez que el puerto está disponible para transmitir un mensaje).

Las tareas en background se ejecutan en forma cíclica de acuerdo a la activación de un flag de recepción que indica que hay información para procesar, tal como se muestra en el pseudocódigo:

```
Main(){
      si(flag_rx){
            //Ejecutar el comando que ingreso el usuario
      }
}
```

Cabe destacar que este será el único evento que tendremos en el programa principal, ya que la transmisión de datos no tiene que ejecutar nada en el mismo, simplemente usa funciones de la librería Buffer.c.

Para el ingreso de comandos por parte del usuario, se utiliza un esquema productor/consumidor, en el que los datos no se imprimen y/o procesan todos juntos, sino que se depositan y extraen en/desde un buffer. Esto fue explicado en el punto 5.3, donde se habla de la librería Buffer.c.

## 8. Anexos

### 8.1. Main.c

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include <string.h>
#include <buffer.h>

#define N 50
#define FALSE 0
#define TRUE 1

#define F1 40000 //100Hz
#define F2 13333
#define F3 8000
#define F4 4000
#define F5 2000
#define F6 800
#define F7 400
#define F8 333
```

```
void MCU_init(void); /* Device initialization function
declaration */


extern volatile char RX_flag; //Flag de recepci□e comando
volatile short NC; //Frecuencia actual del parlante



// --------------------------------Funciones y programa
principal---------------------------------------

void menu_inicio(){

    SCI_enviar_cadena("\n\n\nBienvenido, elija su comando:\n");
    SCI_enviar_cadena(" - Comandos para configuracion de
frecuencia:\n");
    SCI_enviar_cadena("      1) 100 --> 100Hz\n");
    SCI_enviar_cadena("      2) 300 --> 300Hz\n");
    SCI_enviar_cadena("      3) 500 --> 500Hz\n");
    SCI_enviar_cadena("      4) 1k --> 1000Hz\n");
    SCI_enviar_cadena("      5) 2k --> 2000Hz\n");
    SCI_enviar_cadena("      6) 5k --> 5000Hz\n");
    SCI_enviar_cadena("      7) 10k --> 10000Hz\n");
    SCI_enviar_cadena("      8) 12k --> 12000Hz\n");
    SCI_enviar_cadena(" - Comandos de control\n");
    SCI_enviar_cadena("      A) ON --> Encender parlante\n");
    SCI_enviar_cadena("      B) OFF --> Apagar parlante\n");
    SCI_enviar_cadena("      C) RESET --> Volver a inicializar
el MCU");
}

int comparar_comando(char str1[], char str2[])
{
    int ctr=0;

    while(str1[ctr]==str2[ctr])
    {
        if(str1[ctr]=='\0'||str2[ctr]=='\0')
            break;
        ctr++;
    }
    if(str1[ctr]=='\0' && str2[ctr]=='\0')
        return 1;
    else
        return 0;
}

void ejecutar_comando(char comando){

    switch(comando){

        case '1':
            SCI_enviar_cadena("Frecuencia de 100Hz");
            NC=F1;
        break;
        case '2':
            SCI_enviar_cadena("Frecuencia de 300Hz");
            NC=F2;

        break;
```

```
            case '3':
                SCI_enviar_cadena("Frecuencia de 500Hz");
                NC=F3;

            break;
            case '4':
                SCI_enviar_cadena("Frecuencia de 1KHz");
                NC=F4;

            break;
            case '5':
                SCI_enviar_cadena("Frecuencia de 2KHz");
                NC=F5;


            break;
            case '6':
                SCI_enviar_cadena("Frecuencia de 5KHz");
                NC=F6;

            break;
            case '7':
                SCI_enviar_cadena("Frecuencia de 10KHz");
                NC=F7;

            break;
            case '8':
                SCI_enviar_cadena("Frecuencia de 12KHz");
                NC=F8;

            break;
            case 'A':
                SCI_enviar_cadena("Sonido activado");
                TPM1C1SC=0x54U;
            break;
            case 'B':
                SCI_enviar_cadena("Sonido desactivado");
                TPM1C1SC = 0;
            break;
            case 'C':
                SCI_enviar_cadena("Dispositivo reseteado");
                NC = F1;
                TPM1C1SC = 0;
                menu_inicio();
            break;
            default:
                SCI_enviar_cadena("Comando invalido");
            break;
        }

}

char determinar_comando(char command[]){
    if(comparar_comando(command,"ON\r")){
        return 'A';
    }else{

    if(comparar_comando(command,"OFF\r")){
        return 'B';
    }else{
```

```
    if(comparar_comando(command,"RESET\r")){
        return 'C';
    }else{

    if(comparar_comando(command,"100\r")){
        return '1';
    }else{

    if(comparar_comando(command,"300\r")){
            return '2';
    }else{

    if(comparar_comando(command,"500\r")){
            return '3';
    }else{

    if(comparar_comando(command,"1k\r")){
            return '4';
    }else{

    if(comparar_comando(command,"2k\r")){
            return '5';
    }else{

    if(comparar_comando(command,"5k\r")){
            return '6';
    }else{

    if(comparar_comando(command,"10k\r")){
            return '7';
    }else{

    if(comparar_comando(command,"12k\r")){
            return '8';
    }}}}}}}}}}}
}

//char comando[]="AT+PIN\r\n";  //Necesita el \r \n, es para
ver en que estado esta, deberia responder "OK\r\n" porque esta
en modo AT
void main(void) {
    char nombre_default[]="AT+NAMEG8\r\n"; //Nombre por defecto
de nuestro modulo bluetooth
    char aux;
    char comando_comparable;
    char *comando;
    MCU_init();

    SCI_enviar_cadena(nombre_default); //Se envel nombre por
defecto al m□o

    NC = F1;
    TPM1C1SC = 0;//Desactivo las interrupciones por TPM1 CH1

    //Espero a que se haga la vinculacion con el dispositivo
    while(PTBD_PTBD2==0);

    menu_inicio();

    for(;;) {
        if(RX_flag){
```

```
            comando = leer_comando();
            comando_comparable = determinar_comando(comando);
            ejecutar_comando(comando_comparable);
            RX_flag = 0;
        }
    }
}
```

## 8.2. Buffer.c

```
#include "derivative.h"
#define TX_BUFFER_SIZE 150
#define RX_BUFFER_SIZE 150

//Buffer para transmision y sus indices
unsigned short indice_lectura_TX = 0; //Indice de lectura del
buffer TX
unsigned short indice_escritura_TX = 0;//Indice de escritura
del buffer TX
char bufferTX[TX_BUFFER_SIZE]; // Buffer para transmision

//Buffer para recepcion y sus indices
unsigned short indice_lectura_RX = 0; //Indice de lectura del
buffer RX
unsigned short indice_escritura_RX = 0;//Indice de escritura
del buffer RX
char bufferRX[RX_BUFFER_SIZE]; // Buffer para recepcion


//volatile uint8 Flag_Transmisor = FALSE; // si esta en TRUE
indica que el buffer esta ocupado*/

volatile char RX_flag = 0;

void delay(unsigned short n){//Funcion que recibe por par□tro
la cantidad de miliseg de delay
  unsigned short temp;
  unsigned short cant;
  /*Aca sabemos que 1a ejecucion del for equivale a 23 ciclos
de reloj
  y que hacemos 8000 ciclos por milisegundo, eso equivale a mas
o menos
  348 ejecuciones del for por milisegundo*/
  for(cant=n; cant>0; cant--){
    for(temp=348; temp>0; temp--);
  }
}

//Recepcion de datos

void escribir_en_bufferRX(void) {
    char car;
    if (SCIS1_RDRF) { //Si esta disponible para leer
        car = SCID;//Leo el primer caracter
    }
    if(car != '\n'){
        bufferRX[indice_escritura_RX] = car;
        indice_escritura_RX++;
    }
    else{
        RX_flag = 1;//Se prende cuando lee todo el string
    }
```

```
}

char* leer_comando(void){
    char lectura[RX_BUFFER_SIZE];
    short indiceAux = 0;

    while(indice_lectura_RX < indice_escritura_RX){
        lectura[indiceAux] = bufferRX[indice_lectura_RX];
        indice_lectura_RX++;
        indiceAux++;//Se podria dejar solo indice_lectura_RX,
fijate que queda mejor
    }
    lectura[indiceAux] = '\0';//Para que al leer el comando
haya un corte de control

    indice_lectura_RX = 0;//Se reinician ambos punteros
    indice_escritura_RX = 0;

    return lectura;//Devuelvo lo que lei
}


//Transmision de datos

void SCI_escribir_char_al_buffer(char dato){
    if (indice_escritura_TX < TX_BUFFER_SIZE) { // si hay lugar
en el buffer guarda un dato
        bufferTX[indice_escritura_TX] = dato;
        indice_escritura_TX++;
    }
}

void SCI_enviar_cadena(char *cadena){
    while (*cadena != '\0') {
        SCI_escribir_char_al_buffer(*cadena);//Por lo visto
toma este llamado como si fuera un prototipo, y genera el error
abajo, lo vemos desp
        cadena++;
    }
    SCIC2_TIE = 1; //Habilita las interrupciones de transmision
    delay(20);  //Delay de 400ms
}

void SCI_update(void) {
    if (indice_escritura_TX > indice_lectura_TX) {//Si el
indice de escritura es mayor, hay caracteres a transmitir
        if (SCIS1_TDRE) { //Si esta disponible el trasmisor
            SCID = bufferTX[indice_lectura_TX];
            indice_lectura_TX++;
        }
    } else { //Reinicio los indices y deshabilito la
interrupcion
        indice_lectura_TX = 0;
        indice_escritura_TX = 0;
        SCIC2_TIE = 0;
    }
}
```

## 8.3. Buffer.h

```
void delay(unsigned short);
void escribir_en_bufferRX(void);
void SCI_enviar_cadena(char*);
void SCI_escribir_char_al_buffer(char);
char* leer_comando(void);
void SCI_update(void);
```

## 8.4. MCUinit.c

```
/*
**
###################################################################
####
**     This code is generated by the Device Initialization
Tool.
**     It is overwritten during code generation.
**     USER MODIFICATION ARE PRESERVED ONLY INSIDE INTERRUPT
SERVICE ROUTINES
**     OR EXPLICITLY MARKED SECTIONS
**
**     Project   : DeviceInitialization
**     Processor : MC9S08SH8CPJ
**     Version   : Component 01.008, Driver 01.08, CPU db:
3.00.066
**     Datasheet : MC9S08SH8 Rev. 3 6/2008
**     Date/Time : 2019-07-01, 14:21, # CodeGen: 4
**     Abstract  :
**         This module contains device initialization code
**         for selected on-chip peripherals.
**     Contents  :
**         Function "MCU_init" initializes selected peripherals
**
**     Copyright : 1997 - 2010 Freescale Semiconductor, Inc.
All Rights Reserved.
**
**     http      : www.freescale.com
**     mail      : support@freescale.com
**
###################################################################
####
*/

/* MODULE MCUinit */

#include <mc9s08sh8.h>                 /* I/O map for
MC9S08SH8CPJ */
#include "MCUinit.h"

/* Standard ANSI C types */
#ifndef int8_t
typedef signed char int8_t;
#endif
#ifndef int16_t
typedef signed int int16_t;
#endif
#ifndef int32_t
typedef signed long int int32_t;
#endif
```

```
#ifndef uint8_t
typedef unsigned char uint8_t;
#endif
#ifndef uint16_t
typedef unsigned int uint16_t;
#endif
#ifndef uint32_t
typedef unsigned long int uint32_t;
#endif

/* User declarations and definitions */
extern volatile short NC;
/*   Code, declarations and definitions here will be preserved
during code generation */
/* End of user declarations and definitions */


/*
**
=================================================================
====
**     Method      :  MCU_init (component MC9S08SH8_20)
**
**     Description :
**         Device initialization code for selected peripherals.
**
=================================================================
====
*/
void MCU_init(void)
{
  /* ### MC9S08SH8_20 "Cpu" init code ... */
  /*  PE initialization code after reset */
  /* Common initialization of the write once registers */
  /* SOPT1: COPT=0,STOPE=0,IICPS=0,BKGDPE=1,RSTPE=0 */
  SOPT1 = 0x02U;
  /* SOPT2: COPCLKS=0,COPW=0,ACIC=0,T1CH1PS=1,T1CH0PS=0 */
  SOPT2 = 0x02U;
  /* SPMSC1:
LVWF=0,LVWACK=0,LVWIE=0,LVDRE=1,LVDSE=1,LVDE=1,BGBE=0 */
  SPMSC1 = 0x1CU;
  /* SPMSC2: LVDV=0,LVWV=0,PPDF=0,PPDACK=0,PPDC=0 */
  SPMSC2 = 0x00U;
  /*  System clock initialization */
  /*lint -save  -e923 Disable MISRA rule (11.3) checking. */
  if (*(unsigned char*far)0xFFAFU != 0xFFU) { /* Test if the
device trim value is stored on the specified address */
    ICSTRM = *(unsigned char*far)0xFFAFU; /* Initialize ICSTRM
register from a non volatile memory */
    ICSSC = *(unsigned char*far)0xFFAEU; /* Initialize ICSSC
register from a non volatile memory */
  }
  /*lint -restore Enable MISRA rule (11.3) checking. */
  /* ICSC1: CLKS=0,RDIV=0,IREFS=1,IRCLKEN=0,IREFSTEN=0 */
  ICSC1 = 0x04U;                       /* Initialization of the
ICS control register 1 */
  /* ICSC2:
BDIV=1,RANGE=0,HGO=0,LP=0,EREFS=0,ERCLKEN=0,EREFSTEN=0 */
  ICSC2 = 0x40U;                       /* Initialization of the
ICS control register 2 */
```

```
  while(ICSSC_IREFST == 0U) {          /* Wait until the source
of reference clock is internal clock */
  }
  /* GNGC:
GNGPS7=0,GNGPS6=0,GNGPS5=0,GNGPS4=0,GNGPS3=0,GNGPS2=0,GNGPS1=0,
GNGEN=0 */
  GNGC = 0x00U;
  /* Common initialization of the CPU registers */
  /* PTCPE: PTCPE1=0 */
  PTCPE &= (unsigned char)~(unsigned char)0x02U;
  /* PTASE: PTASE4=0,PTASE3=0,PTASE2=0,PTASE1=0,PTASE0=0 */
  PTASE &= (unsigned char)~(unsigned char)0x1FU;
  /* PTBSE:
PTBSE7=0,PTBSE6=0,PTBSE5=0,PTBSE4=0,PTBSE3=0,PTBSE2=0,PTBSE1=0,
PTBSE0=0 */
  PTBSE = 0x00U;
  /* PTCSE: PTCSE3=0,PTCSE2=0,PTCSE1=0,PTCSE0=0 */
  PTCSE &= (unsigned char)~(unsigned char)0x0FU;
  /* PTADS: PTADS4=0,PTADS3=0,PTADS2=0,PTADS1=0,PTADS0=0 */
  PTADS = 0x00U;
  /* PTBDS:
PTBDS7=0,PTBDS6=0,PTBDS5=0,PTBDS4=0,PTBDS3=0,PTBDS2=0,PTBDS1=0,
PTBDS0=0 */
  PTBDS = 0x00U;
  /* PTCDS: PTCDS3=0,PTCDS2=0,PTCDS1=0,PTCDS0=0 */
  PTCDS = 0x00U;
  /* ### Init_SCI init code */
  /* SCIC2: TIE=0,TCIE=0,RIE=0,ILIE=0,TE=0,RE=0,RWU=0,SBK=0 */
  SCIC2 = 0x00U;                       /* Disable the SCI
module */
  (void)(SCIS1 == 0U);                 /* Dummy read of the
SCIS1 register to clear flags */
  (void)(SCID == 0U);                  /* Dummy read of the
SCID register to clear flags */
  /* SCIS2:
LBKDIF=1,RXEDGIF=1,RXINV=0,RWUID=0,BRK13=0,LBKDE=0,RAF=0 */
  SCIS2 = 0xC0U;
  /* SCIBDH:
LBKDIE=0,RXEDGIE=0,SBR12=0,SBR11=0,SBR10=0,SBR9=0,SBR8=0 */
  SCIBDH = 0x00U;
  /* SCIBDL:
SBR7=0,SBR6=0,SBR5=1,SBR4=1,SBR3=0,SBR2=1,SBR1=0,SBR0=0 */
  SCIBDL = 0x34U;
  /* SCIC1: LOOPS=0,SCISWAI=0,RSRC=0,M=0,WAKE=0,ILT=0,PE=0,PT=0
*/
  SCIC1 = 0x00U;
  /* SCIC3:
R8=0,T8=0,TXDIR=0,TXINV=0,ORIE=0,NEIE=0,FEIE=0,PEIE=0 */
  SCIC3 = 0x00U;
  /* SCIC2: TIE=0,TCIE=0,RIE=1,ILIE=0,TE=1,RE=1,RWU=0,SBK=0 */
  SCIC2 = 0x2CU;
  /* ### Init_TPM init code */
  (void)(TPM1C1SC == 0U);              /* Channel 0 int. flag
clearing (first part) */
  /* TPM1C1SC: CH1F=0,CH1IE=1,MS1B=0,MS1A=1,ELS1B=0,ELS1A=1 */
  TPM1C1SC = 0x54U;                    /* Int. flag clearing
(2nd part) and channel 0 contr. register setting */
  TPM1C1V = 0x00U;                     /* Compare 0 value
setting */
  /* TPM1SC:
TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=0,PS2=0,PS1=0,PS0=0 */
```

```
  TPM1SC = 0x00U;                      /* Stop and reset
counter */
  TPM1MOD = 0x00U;                     /* Period value setting
*/
  (void)(TPM1SC == 0U);                /* Overflow int. flag
clearing (first part) */
  /* TPM1SC:
TOF=0,TOIE=0,CPWMS=0,CLKSB=0,CLKSA=1,PS2=0,PS1=0,PS0=0 */
  TPM1SC = 0x08U;                      /* Int. flag clearing
(2nd part) and timer control register setting */
  /* ### */
  /*lint -save  -e950 Disable MISRA rule (1.1) checking. */
  asm CLI;                             /* Enable interrupts */
  /*lint -restore Enable MISRA rule (1.1) checking. */

} /*MCU_init*/


/*lint -save  -e765 Disable MISRA rule (8.10) checking. */
/*
**
============================================================
====
**     Interrupt handler : isrVscitx
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/
__interrupt void isrVscitx(void)
{
  SCI_update();
}
/* end of isrVscitx */


/*
**
============================================================
====
**     Interrupt handler : isrVscirx
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/
__interrupt void isrVscirx(void)
{
  /* Write your interrupt code here ... */
    escribir_en_bufferRX();
}
/* end of isrVscirx */
```

```
/*
**
============================================================
====
**      Interrupt handler : isrVscierr
**
**      Description :
**          User interrupt service routine.
**      Parameters  : None
**      Returns     : Nothing
**
============================================================
====
*/
__interrupt void isrVscierr(void)
{
  /* Write your interrupt code here ... */

}
/* end of isrVscierr */


/*
**
============================================================
====
**      Interrupt handler : isrVtpm1ovf
**
**      Description :
**          User interrupt service routine.
**      Parameters  : None
**      Returns     : Nothing
**
============================================================
====
*/
__interrupt void isrVtpm1ovf(void)
{
  /* Write your interrupt code here ... */

}
/* end of isrVtpm1ovf */


/*
**
============================================================
====
**      Interrupt handler : isrVtpm1ch1
**
**      Description :
**          User interrupt service routine.
**      Parameters  : None
**      Returns     : Nothing
**
============================================================
====
*/
__interrupt void isrVtpm1ch1(void)
{
```

```
    TPM1C1V += NC;
    TPM1C1SC_CH1F=0;
}


/*lint -restore Enable MISRA rule (8.10) checking. */

/*lint -save  -e950 Disable MISRA rule (1.1) checking. */
/* Initialization of the CPU registers in FLASH */
/* NVPROT:
FPS7=1,FPS6=1,FPS5=1,FPS4=1,FPS3=1,FPS2=1,FPS1=1,FPDIS=1 */
static const unsigned char NVPROT_INIT @0x0000FFBDU = 0xFFU;
/* NVOPT: KEYEN=0,FNORED=1,SEC01=1,SEC00=0 */
static const unsigned char NVOPT_INIT @0x0000FFBFU = 0x7EU;
/*lint -restore Enable MISRA rule (1.1) checking. */



extern near void _Startup(void);

/* Interrupt vector table */
#ifndef UNASSIGNED_ISR
  #define UNASSIGNED_ISR ((void(*near const)(void)) 0xFFFF) /*
unassigned interrupt service routine */
#endif

/*lint -save  -e923 Disable MISRA rule (11.3) checking. */
/*lint -save  -e950 Disable MISRA rule (1.1) checking. */
static void (* near const _vect[])(void) @0xFFC0 = { /*
Interrupt vector table */
/*lint -restore Enable MISRA rule (1.1) checking. */
         UNASSIGNED_ISR,               /* Int.no. 31
VReserved31 (at FFC0)           Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 30 Vacmp (at
FFC2)                 Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 29
VReserved29 (at FFC4)           Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 28
VReserved28 (at FFC6)           Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 27
VReserved27 (at FFC8)           Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 26 Vmtim (at
FFCA)                 Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 25 Vrtc (at
FFCC)                  Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 24 Viic (at
FFCE)                  Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 23 Vadc (at
FFD0)                  Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 22
VReserved22 (at FFD2)           Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 21 Vportb (at
FFD4)                Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 20 Vporta (at
FFD6)                Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 19
VReserved19 (at FFD8)           Unassigned */
         isrVscitx,                    /* Int.no. 18 Vscitx (at
FFDA)                Used */
         isrVscirx,                    /* Int.no. 17 Vscirx (at
FFDC)                Used */
```

```
         isrVscierr,                   /* Int.no. 16 Vscierr
(at FFDE)               Used */
         UNASSIGNED_ISR,               /* Int.no. 15 Vspi (at
FFE0)                   Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 14 Vtpm2ovf
(at FFE2)               Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 13 Vtpm2ch1
(at FFE4)               Unassigned */
         UNASSIGNED_ISR,               /* Int.no. 12 Vtpm2ch0
(at FFE6)               Unassigned */
         isrVtpm1ovf,                  /* Int.no. 11 Vtpm1ovf
(at FFE8)               Used */
         UNASSIGNED_ISR,               /* Int.no. 10
VReserved10 (at FFEA)           Unassigned */
         UNASSIGNED_ISR,               /* Int.no.  9 VReserved9
(at FFEC)             Unassigned */
         UNASSIGNED_ISR,               /* Int.no.  8 VReserved8
(at FFEE)             Unassigned */
         UNASSIGNED_ISR,               /* Int.no.  7 VReserved7
(at FFF0)             Unassigned */
         isrVtpm1ch1,                  /* Int.no.  6 Vtpm1ch1
(at FFF2)               Used */
         UNASSIGNED_ISR,               /* Int.no.  5 Vtpm1ch0
(at FFF4)               Unassigned */
         UNASSIGNED_ISR,               /* Int.no.  4 VReserved4
(at FFF6)             Unassigned */
         UNASSIGNED_ISR,               /* Int.no.  3 Vlvd (at
FFF8)                   Unassigned */
         UNASSIGNED_ISR,               /* Int.no.  2 Virq (at
FFFA)                   Unassigned */
         UNASSIGNED_ISR,               /* Int.no.  1 Vswi (at
FFFC)                   Unassigned */
         _Startup                      /* Int.no.  0 Vreset (at
FFFE)                Reset vector */
};
/*lint -restore Enable MISRA rule (11.3) checking. */



/* END MCUinit */

/*
**
###################################################################
####
**
**     This file was created by Processor Expert 5.00 [04.48]
**     for the Freescale HCS08 series of microcontrollers.
**
**
###################################################################
####
*/
```

## 8.5. MCUinit.h

```
/*
**
###################################################################
####
**     This code is generated by the Device Initialization
Tool.
**     It is overwritten during code generation.
**     USER MODIFICATION ARE PRESERVED ONLY INSIDE EXPLICITLY
MARKED SECTIONS.
**
**     Project   : DeviceInitialization
**     Processor : MC9S08SH8CPJ
**     Version   : Component 01.008, Driver 01.08, CPU db:
3.00.066
**     Datasheet : MC9S08SH8 Rev. 3 6/2008
**     Date/Time : 2019-06-24, 16:05, # CodeGen: 1
**     Abstract  :
**         This module contains device initialization code
**         for selected on-chip peripherals.
**     Contents  :
**         Function "MCU_init" initializes selected peripherals
**
**     Copyright : 1997 - 2010 Freescale Semiconductor, Inc.
All Rights Reserved.
**
**     http      : www.freescale.com
**     mail      : support@freescale.com
**
###################################################################
####
*/

#ifndef __DeviceInitialization_H
#define __DeviceInitialization_H 1

/* Include shared modules, which are used for whole project */


/* User declarations and definitions */
/*   Code, declarations and definitions here will be preserved
during code generation */
/* End of user declarations and definitions */

#ifdef __cplusplus
extern "C" {
#endif
extern void MCU_init(void);
#ifdef __cplusplus
}
#endif
/*
**
===================================================================
====
**     Method      :  MCU_init (component MC9S08SH8_20)
**
**     Description :
**         Device initialization code for selected peripherals.
```

```
**
============================================================
====
*/


/*lint -save  -e765 Disable MISRA rule (8.10) checking. */
__interrupt void isrVscitx(void);
/*
**
============================================================
====
**     Interrupt handler : isrVscitx
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/


__interrupt void isrVscirx(void);
/*
**
============================================================
====
**     Interrupt handler : isrVscirx
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/


__interrupt void isrVscierr(void);
/*
**
============================================================
====
**     Interrupt handler : isrVscierr
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/
```

```
__interrupt void isrVtpm1ovf(void);
/*
**
============================================================
====
**      Interrupt handler : isrVtpm1ovf
**
**      Description :
**          User interrupt service routine.
**      Parameters  : None
**      Returns     : Nothing
**
============================================================
====
*/



__interrupt void isrVtpm1ch1(void);
/*
**
============================================================
====
**      Interrupt handler : isrVtpm1ch1
**
**      Description :
**          User interrupt service routine.
**      Parameters  : None
**      Returns     : Nothing
**
============================================================
====
*/




/*lint -restore Enable MISRA rule (8.10) checking. */

/* END DeviceInitialization */

#endif
/*
**
############################################################
####
**
**      This file was created by Processor Expert 5.00 [04.48]
**      for the Freescale HCS08 series of microcontrollers.
**
**
############################################################
####
*/
```