

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA TELEMÁTICA

**SISTEMAS ELECTRÓNICOS DIGITALES**

Practica Libre

**AUTOR:** Luis Pastor Camarillo

CURSO 2022/2023

INDICE

[INTRODUCCIÓN 3](#_Toc123639554)

[DISEÑO SOFTWARE 4](#_Toc123639555)

[DISEÑO HARDWARE 5](#_Toc123639556)

[RESULTADOS 5](#_Toc123639557)

[CONCLUSIÓN 6](#_Toc123639558)

# INTRODUCCIÓN

Para la realización de esta práctica se pide la implementación completa de un reloj digital común a partir del microprocesador LPC1768. Más en detalle, se ha de realizar el montaje de un display cuádruple que permita visualizar horas y minutos o segundos y decimas y centésimas de estos. Además de estos dos modos de visualización se pide añadir tres funcionalidades principales:

* Reloj.
* Dos alarmas.
* Dos temporizadores.

Estas funcionalidades han de ser programables, permitiendo modificar la hora del reloj, la hora a la que se van a activar las alarmas y el tiempo inicial de los temporizadores.

Además, es necesario que tanto alarmas como temporizadores puedan activarse o desactivarse mediante un switch, lo cual se visualizará utilizando una serie de leds.

Por último se pide que, al activarse las alarmas al llegar la hora del reloj a la programada en estas, o al concluir el tiempo programado en los temporizadores, se active un altavoz haciendo uso del periférico DAC del microprocesador, generando distintos tipos de señales en función de la alarma o temporizador activo.

# DISEÑO SOFTWARE

## Diagrama de estados:

Diagrama

Descripción generada automáticamente

A continuación, se va a desgranar el código generado para la implementación de esta práctica completa, el cual está formado por alrededor de quinientas líneas escritas en lenguaje C.

## Inicio del programa:

Inicialmente se encuentra la **declaración de las bibliotecas** a utilizar, siendo indispensable la específica del controlador para trabajar con él y la biblioteca ‘*math.h’* para hacer uso de la función seno utilizada con el periférico DAC. Cabe destacar que ‘*delay.h’* sólo ha sido utilizada en los manejadores de las interrupciones para evitar falsas detecciones en los pulsadores tal y como indicó el profesor, aunque estos han sido configurados además para ser activos por flanco como veremos posteriormente.

Posteriormente se declaran tanto **constantes** como **variables**, teniendo estas las funcionalidades comentadas en el propio código.

Los arrays ‘*NUMEROS*’ y ‘*HOLA*’ almacenan la sucesión de bits codificados en hexadecimal necesarios para poder visualizar los números del 0 al 9 y los caracteres de la palabra “HOLA” utilizando un display de siete segmentos de ánodo común.

Tanto ‘*tiempo*’ como ‘*alarma1*’, ‘*alarma2*’, *‘temp1’* y ‘*temp2*’ almacenan los valores que aparecen en sus respectivos comentarios. Estos valores serán con los que se trabajará toda la práctica y los que se usarán para cada funcionalidad del reloj.

El resto de las variables globales serán comentadas posteriormente a la hora de utilizarlas.

Texto

Descripción generada automáticamente

## Configuración de puertos:

Para configurar los puertos necesarios para toda la práctica se ha creado la función ‘*config\_GPIO’* que especifica las entradas de interrupción utilizadas, la definición de puertos de entrada y salida y la configuración de los switches como entradas pull-down. Además de estar explicada en los comentarios del código, se volverá de nuevo a hablar de esta configuración en el apartado de diseño hardware.

Tabla

Descripción generada automáticamente

## Configuración de SysTick y Timers:

Para la realización de la práctica se han utilizado estos tres timers.

En primer lugar, el periférico SysTick ha sido configurado para interrumpir cada décima de segundo, ya que este es el tiempo mínimo de actualización del reloj, dado que se muestran hasta las centésimas, pero siempre con el valor ‘0’. Este será utilizado para controlar la evolución del tiempo que representa el reloj, como se verá luego en su manejador.

## Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

En segundo lugar, el Timer0 ha sido configurado para interrumpir cada 5 milisegundos. Este es el tiempo de actualización que se ha utilizado en el display para la representación de cada dígito. De nuevo, veremos posteriormente como se utiliza en su manejador.

## Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

Por último, el Timer1 se ha configurado para interrumpir en función de la variable *‘f’,* la cual representa la frecuencia de la señal del DAC que ha de ser representada expresada en Hz. Cada vez que este temporizador interrumpa se reproducirá una señal en el DAC.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Configuración de interrupciones:

Una vez declaradas todas las interrupciones, la función siguiente declara todas como activas por flanco para evitar errores con los pulsadores, baja los flags y realiza la asignación de prioridades, asegurando que las únicas que puedan desalojar al resto sean *SysTick* y *Timer0*, encargadas de la representación en el display.

Por último, todas son habilitadas para que comiencen a interrumpir en función de su configuración.

## Texto Descripción generada automáticamente

## Configuración del periférico DAC:

Para el funcionamiento del convertidor analógico-digital se han utilizado dos funciones principales.

En primer lugar ‘*config\_DAC*’ establece el puerto P0.26 como salida DAC de modo que esta será la salida que conectemos posteriormente a nuestro altavoz o al osciloscopio.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para la generación de señales que reproducirá el DAC se ha escrito esta función, la cual genera una señal triangular para la alarma1, una señal cuadrada para la alarma2 y una señal senoidal para los temporizadores. Las tres ondas generadas aprovechan toda la amplitud de voltaje disponible en el DAC y tienen cien muestras cada una.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## Manejador de Timer0:

Como se ha comentado anteriormente el Timer0 interrumpe cada 5 milisegundos, lo que se utiliza para la visualización del display cuádruple. Para ello se utiliza la variable *‘m’*, la cual toma valores entre 0 y 3. Esta se utiliza en la función ‘*main*’ de manera que para cada uno de los valores se enciende uno de los cuatro displays.

Texto

Descripción generada automáticamente con confianza media

## Manejador del Timer1:

En este caso, el manejador del Timer1 interrumpe en función de las necesidades del DAC, y reproduce en este la señal necesaria en función de si ha sido activado por una alarma o un temporizador, lo cual se gestiona utilizando la variable ‘*sel\_onda’,* utilizada para elegir una de las tres ondas generadas en la función ‘*genera\_Ondas()*’.

Al final de este manejador se gestiona la evolución de la variable ‘*k*’ entre 0 y 99, lo cual permite que se reproduzcan las cien muestras de cada onda generada.

Interfaz de usuario gráfica, Texto, Aplicación

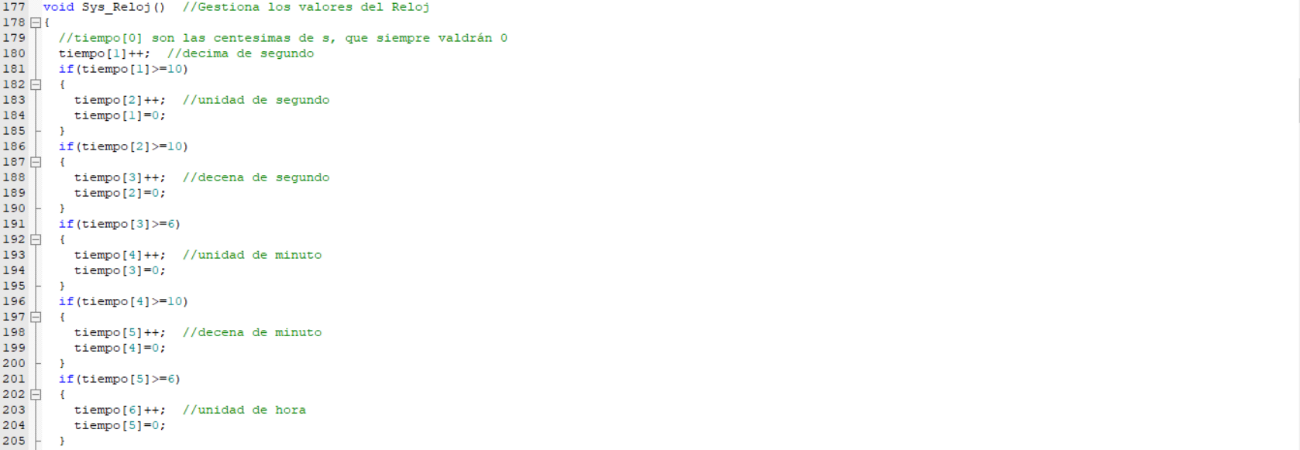
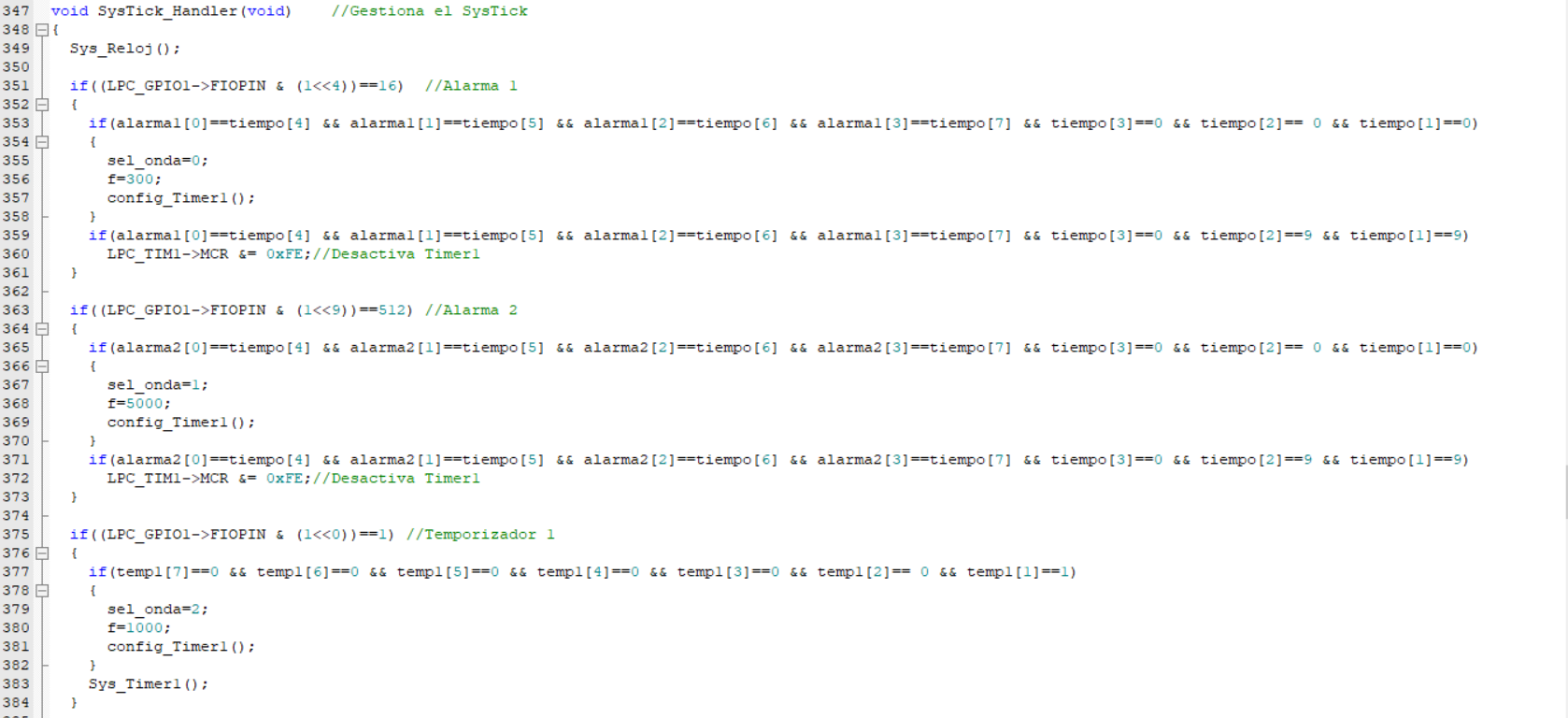
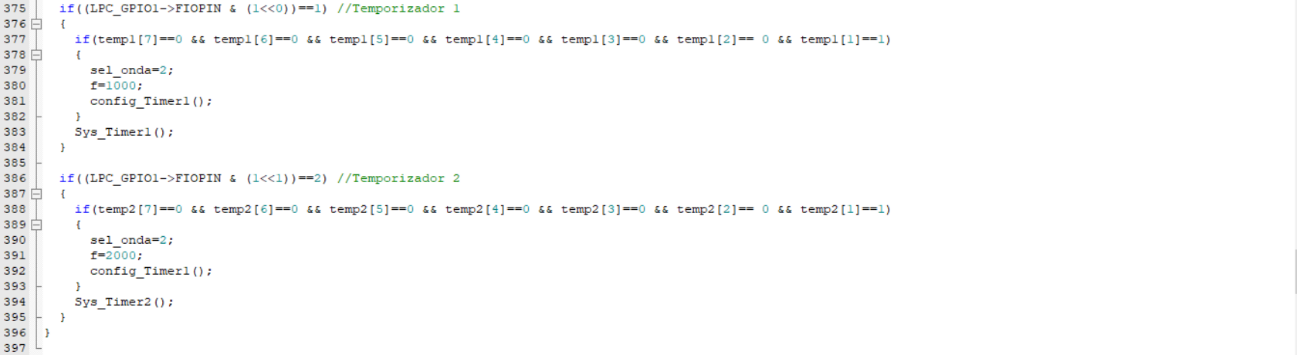
Descripción generada automáticamente

## Manejador del SysTick:

En cuanto al manejador del SysTick, esta es una de las funciones más compleja del programa ya que actualiza los valores de la hora que representa el reloj llamando a la función ‘*Sys\_Reloj()*’ que veremos después, pero también comprueba el estado de alarmas y temporizadores si es que estos están activos. De ser así, en el caso de las alarmas comprueba que la hora coincida con la programada en estas. En el caso de los temporizadores comprueba que quede tan solo una décima de segundo para finalizar, por lo que al llamar a posteriori a ‘*Sys\_timer1’* o ‘*Sys\_timer2’* el temporizador llega a cero.

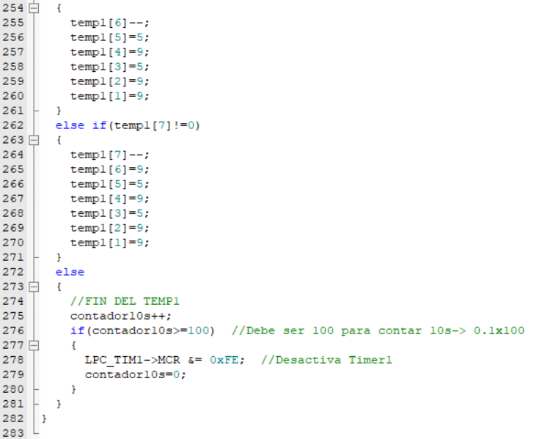
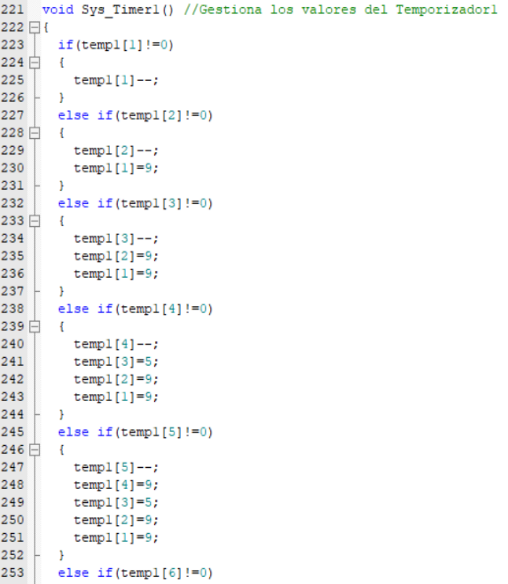
Cuando una de las condiciones se cumple se selecciona la onda mediante la variable ‘*sel\_onda’,* se establece la frecuencia con la que va a trabajar del DAC con la variable *‘f’* y se llama a la función ‘config\_Timer1’ que actualiza el tiempo con el que este timer interrumpirá, de manera que se reproduzca la salida deseada en el DAC y habilita a este para que comience a interrumpir en ese preciso instante.

Por último, pasados diez segundos, el deja de interrumpir, parando la salida del DAC y dejando de sonar el altavoz. En el caso de las alarmas, dado que se programan horas y minutos, el DAC se interrumpirá a los 10 segundos de dicha hora. Por otro lado, en los temporizadores programamos minutos y segundos, por lo que mas tarde veremos como se ha solventado este requerimiento.



En el caso de la función ‘*Sys\_Reloj*’, esta se encarga de modificar los valores de ‘*tiempo*’ cada vez que SysTick interrumpe sumando una décima de segundo. En función de la medida de tiempo que representen se aumentan los valores de este array. Por último, cuando el reloj llega a 24 horas, o lo sobrepasa, se reinicia desde las 00:00

De una manera similar, pero restando, funcionan ‘*Sys\_timer1’* y ‘*Sys\_timer2’*. Estas dos funciones son idénticas, con la salvedad de que la primera opera con el array de variables *‘temp1’* y la segunda con *‘temp2’*; una para cada temporizador.



Como se había adelantado al explicar el manejador del SysTick, es aquí donde además del código que actualiza los valores del temporizador, se cuentan los 10 segundos en los que el DAC está habilitado.

Una vez todos los valores del temporizador llegan a 0, se entra en el ‘*else*’ final, dado que el SysTick interrumpe cada décima de segundo, cuando la variable ‘*contador10s*’ llega a 100 significa que han pasado los 10 segundos estipulados. Una vez esto se produce se desactiva Timer1 y con ello el DAC. Además, se pone dicha variable a 0 de manera que pueda volver a ser utilizada por este u otro temporizador.

## Manejador de EINT0:

Este manejador se ejecuta cuando se pulsa el pulsador ISP, asignado al puerto P2.10. Cuando esto sucede, la variable ‘*entradas*’ aumenta su valor de uno en uno hasta llegar a 4. Al pulsarse de nuevo, este vuelve a 0. Esto permite dentro de la función ‘*main*’ moverse por los diferentes modos de funcionamiento siendo:

* Visualización del reloj para entradas == 0.
* Visualización de alarma1 para entradas == 1.
* Visualización de alarma2 para entradas == 2.
* Visualización de temporizador1 para entradas == 3.
* Visualización de temporizador2 para entradas == 4.

Imagen que contiene Texto

Descripción generada automáticamente

## Manejador de EINT1:

Este segundo manejador de interrupción externa se activa con el pulsador KEY1 asignado al puerto P2.11. De una manera muy similar al anterior, modifica la variable ‘*prog*’ entre los valores 0 y 4 cada vez que se activa el pulsador. Como veremos en la función ‘*main*’, cuando dicha variable vale 0, el reloj funciona de manera normal, pero para los valores 1, 2, 3 y 4 se visualiza solo uno de los displays, cuyo valor podemos modificar utilizando los dos botones que veremos a continuación.

Texto

Descripción generada automáticamente

## Manejadores de EINT2 y EINT3:

Captura de pantalla de un celular

Descripción generada automáticamenteEstos dos últimas manejadores asignados a los puertos P2.12 y P2.13 se activan utilizando KEY2 y un pulsador externo respectivamente. Ambas modifican la variable ‘*incr*’, una aumentándola y otra disminuyéndola, pero siempre entre los valores 0 y 9. Al usarse dentro del modo programación modifican el valor de tiempo con el que se este trabajando en ese momento.

## Función main:

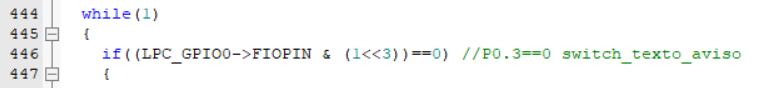
Texto

Descripción generada automáticamente

Al arrancar el programa se configuran los puertos, las interrupciones, el DAC, se generan sus ondas y se configuran los timers, a excepción de *‘Timer0’*, ya que este solo se habilitara cuando salte una alarma o temporizador.

Una vez toda la configuración previa ha sido realizada podemos entrar en el bucle while(), donde el programa quedará indefinidamente.

Este comienza con la comprobación del puerto P0.3. Si este vale 0 se visualizará en el display el reloj.

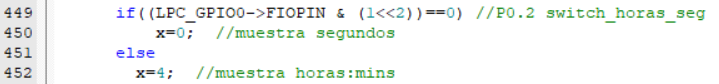


En caso contrario se visualizará por pantalla la cadena *‘HOLA’.*

Texto

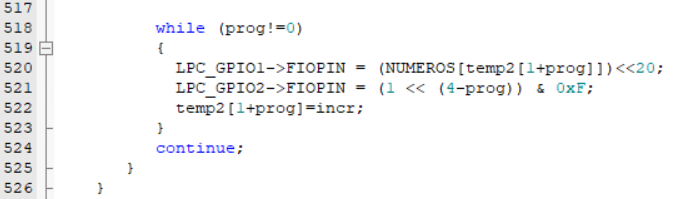
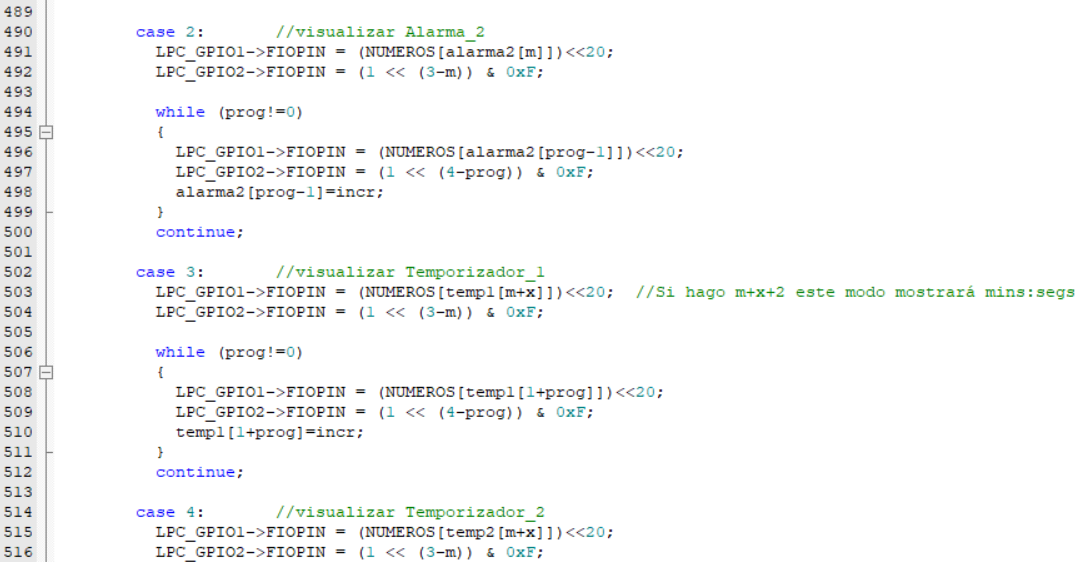
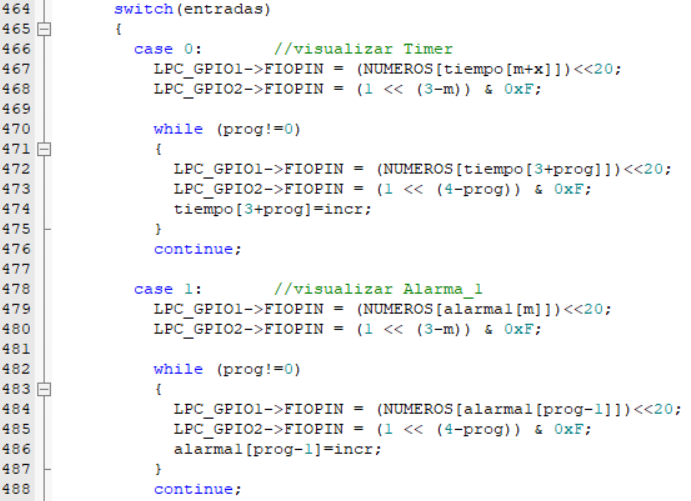
Descripción generada automáticamente

A su vez, para la visualización del reloj con P0.3==0, se comprueba el puerto P0.2, asignado al switch que permite elegir entre la visualización de segundos y la de horas y minutos en función de la variable ‘*x*’.



Finalmente entramos en el switch-case que se encarga de la visualización en el display:

Dentro suyo, podemos ver los diferentes modos en función de la variable ‘*entradas*’. A su vez, dentro de cada una de ellas podemos ver la visualización de su array en función de las variables ‘*m*’, que indica el display que se enciende y ‘*x*’ que selecciona si se visualizan horas y minutos o segundos y sus décimas. A su vez, dentro de cada uno para valores de ‘*prog*’ diferentes a 0 se visualiza solo el valor que esta siendo modificado en cada instante haciendo uso de la variable ‘incr’ que podemos modificar utilizando los pulsadores de P2.12 y P2.13.



# DISEÑO HARDWARE

|  |  |
| --- | --- |
| **Esquema de Conexión** | |
| **Señal** | **Pin GPIO** |
| Switch\_cambio\_visualización | P2.10 ISP |
| Switch\_horas/segundos | P0.2 |
| Switch\_alarma1\_On/Off | P1.4 |
| Switch\_alarma2\_On/Off |  |
| Switch Temp1 | P1.0 |
| Pulsador\_programación | P2.11 KEY1 |
| Pulsador\_incrementar | P2.12 KEY2 |
| Pulsador\_decrementar |  |
| Generación\_señales\_de\_aviso | P0.26 |
| Led\_visualiza\_reloj |  |
| Led\_visualiza\_alarma1 |  |
| Led\_visualiza\_alarma2 |  |
| Led\_visualiza\_temporizador1 |  |
| Led\_visualiza\_temporizador2 |  |
| 7\_segmentos\_a | P1.20 |
| 7\_segmentos\_b | P1.21 |
| 7\_segmentos\_c | P1.22 |
| 7\_segmentos\_d | P1.23 |
| 7\_segmentos\_e | P1.24 |
| 7\_segmentos\_f | P1.25 |
| 7\_segmentos\_g | P1.26 |
| 7\_segmentos\_dp | 1.19 |
| Cátodo\_o\_Ánodo\_Display\_0 | P2.0 |
| Cátodo\_o\_Ánodo\_Display\_1 | P2.1 |
| Cátodo\_o\_Ánodo\_Display\_2 | P2.2 |
| Cátodo\_o\_Ánodo\_Display\_3 | P2.3 |
| Switch\_texto\_aviso | P0.3 |

Para la realización de este proyecto se propone el cumplimiento de los siguientes objetivos:

1. **Estudio del estado del arte de las arquitecturas orientadas a eventos.** En esta fase se deberán estudiar los avances más recientes de este tipo de arquitecturas software con el fin de comprender cual ha sido el desarrollo que ha seguido este tipo de tecnología y hacia donde avanza la vanguardia referente a este ámbito, para posteriormente aplicarlo en el resto de objetivos.
2. **Implementación de una arquitectura del modelo “productor/consumidor” aplicada a un flujo de trabajo real.** Durante esta etapa se deberá realizar una implementación de un modelo productor/consumidor y evaluar su funcionamiento bajo diversas condiciones.
3. **Implementación de una arquitectura del modelo “flujo de eventos” a un flujo de trabajo real.** En esta etapa se desarrollará otra aplicación, esta vez utilizando el modelo flujo de eventos. De la misma manera que en el objetivo anterior, esta será sometida a diferentes condiciones con el fin de poder evaluar su comportamiento posteriormente.
4. **Análisis de resultados y generación de informes o evidencias científicas.** Para finalizar, se deberá recoger los datos obtenidos en las etapas anteriores con el fin de comparar los dos modelos especificados. Esto permitirá extraer conclusiones que quedarán plasmadas en los pertinentes informes finales.

# RESULTADOS

En este diagrama de Gantt se puede observar la estimación de la planificación del proyecto, con las semanas que se emplearán para cada fase.

Interfaz de usuario gráfica, Aplicación, Tabla, Escala de tiempo, Excel

Descripción generada automáticamente

Ilustración 1: Diagrama de Gantt

# CONCLUSIÓN

Una vez concluida la práctica cabe destacar que, si bien es una implementación más grande que las realizadas habitualmente, no es excesivamente difícil si se sigue un buen método de trabajo. En mi caso, comencé diseñando el diagrama de estados y haciendo un diseño de la manera más simplificada posible para poder comprobar el funcionamiento correcto del display. A continuación, la adición y prueba de pequeñas funciones ha permitido depurar de manera rápida el funcionamiento del código escrito, realizando la práctica en menos tiempo del que inicialmente había contemplado.

Creo además, que esta es una buena conclusión para la asignatura, ya que resume las prácticas realizadas durante el resto del curso, ayudándonos a repasar lo ya aprendido, a reforzar conocimientos y a poner en práctica lo anterior de manera conjunta en una implementación mayor llevada a una utilidad más común y que utilizamos día a día, como es un reloj digital.

Debido a la carga de trabajo del resto de asignaturas y el limitado tiempo para la realización de la práctica libre no ha sido posible, pero en mi caso me hubiera gustado implementar otro display cuádruple más, lo que hubiera permitido visualizar el reloj completo, desde las decenas de horas hasta las centésimas de segundo. Pese a no haberlo realizado finalmente, todo el código creado sería funcional con esta ampliación, necesitando realizar muy breves cambios en este mas allá de la configuración de puertos necesarios para otro display y obviamente, el montaje en la placa.

Aún así he quedado bastante satisfecho con el trabajo que he realizado durante la asignatura y lo he disfrutado más de lo que en principio había imaginado, especialmente la realización de esta última práctica.