Sistemas Digitales II
Universidad Politécnica de Madrid
Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
UNIVERSIDAD POLITÉCNICA DE MADRID

Memoria de las mejoras realizadas

Curso 2021/2022

Título del proyecto desarrollado "CORE WATCH"

Autores (orden alfabético): ÍÑIGO GONZÁLEZ VARAS-VALLEJO
JAVIER MARTÍNEZ RANZ

Código de la pareja: JT-06

ÍNDICE GENERAL

1 Introducción

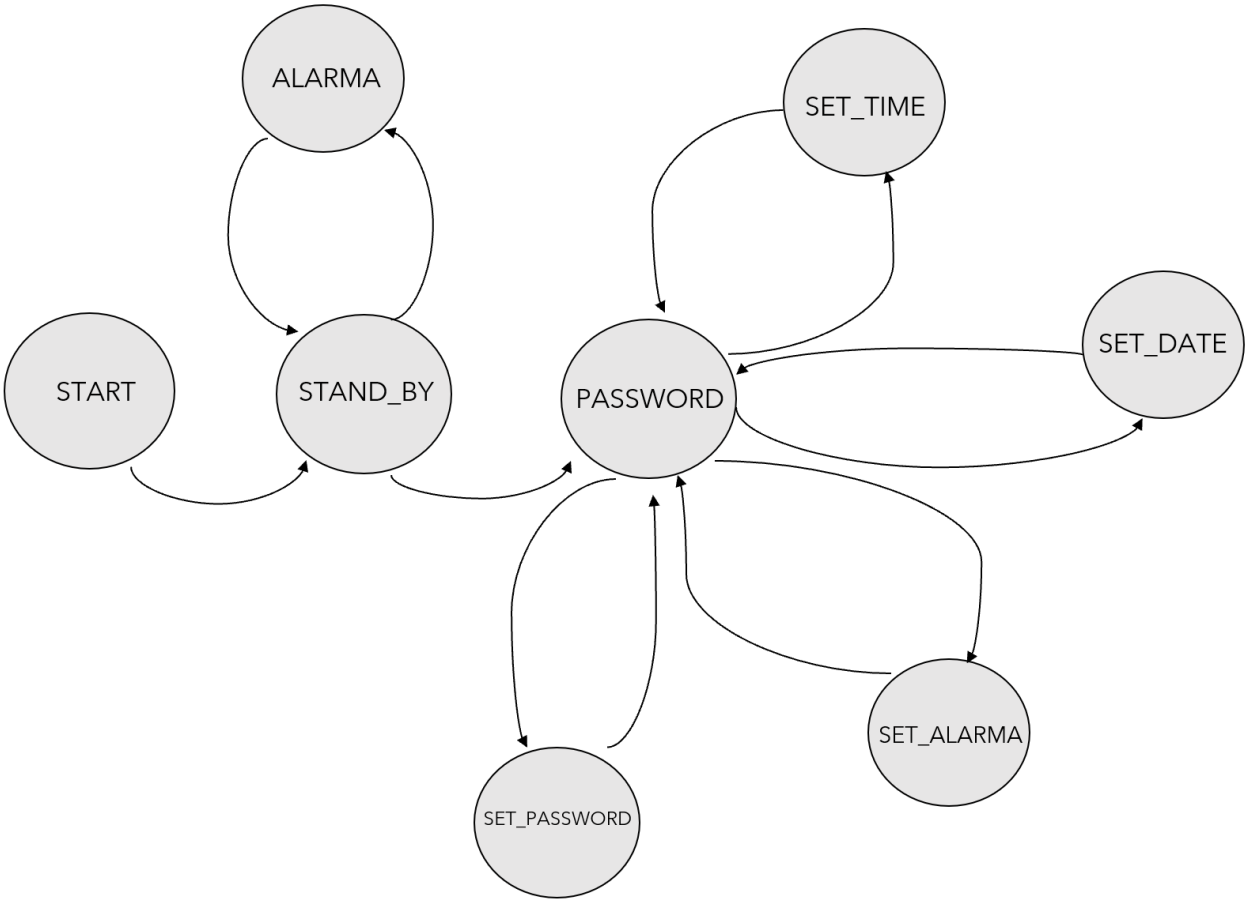
Con la implementación de las mejoras queríamos dotar de distintas características a nuestro sistema coreWatch. Estas características pueden ser una mayor robustez, seguridad, precisión en la visualización de la información o simplemente una mejor experiencia de uso.

En general hemos usado planteamientos similares en todas las mejoras que hemos implementado. En primer lugar, partimos de un nuevo o nuevos estados que añadimos a la FSM del coreWatch.
Luego, añadimos los FLAGS necesarios para atender a las funciones de transición que enlazan los estados de los que partíamos (versiones previas) y los nuevos.
Finalmente, codificamos las funciones de entrada de las transiciones de estados implicadas, que comprobarán dichos flags. Y también programamos las funciones de salida o actualización.
Nuestras mejoras más importantes son una contraseña para el sistema y una alarma. Además, hemos implementado cambio de fecha, y formato am y pm entre otras.

Los vídeos que demuestran el funcionamiento se encuentran en:
https://drive.google.com/drive/folders/1Wk_cARk2O4lFSYuPVbtky1X38P4DNfP7?usp=sharing

2 Diagrama de subsistemas ampliado

Esquema de bloques general



3 Mejora 1: Modificación de la fecha (SET_DATE)

3.1 Objetivos de la mejora

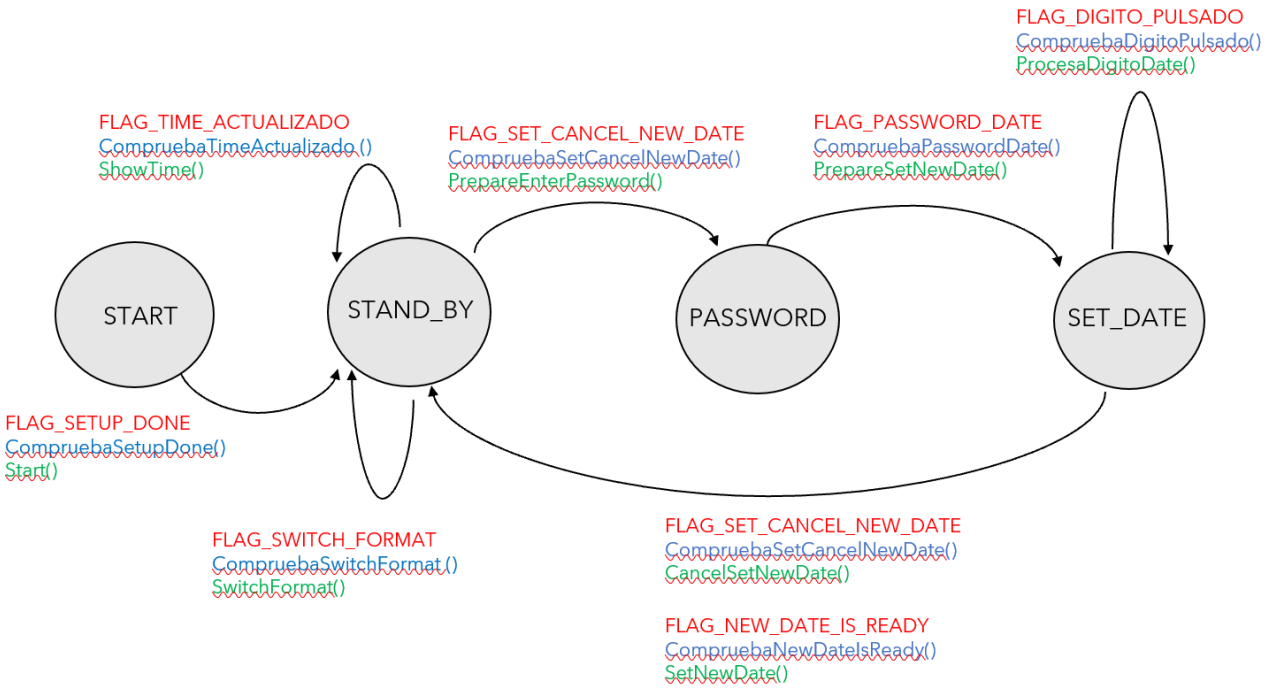
El objetivo de esta mejora es permitir al usuario modificar la fecha del CoreWatch.

3.2 Descripción del subsistema Software

Hemos utilizado el mismo planteamiento que se utilizaba con SET_TIME, al pulsar una tecla se activa un flag y pasamos al estado SET_DATE, en el cual vamos introduciendo dígitos que procesaremos con ProcesaDigitoDate(). Esta función se encarga de que los dígitos introducidos sean válidos, es decir, que no se puedan introducir meses por encima del 12, días por encima del 31 o años por debajo de 1970.

Una vez introducidos todos los dígitos, estos se pasan a unas funciones que hemos añadido en reloj.c, que son SetDia(), SetMes() y SetYear(). Estas funciones se invocan en un orden concreto, ya que es importante que la función SetDia() pueda comprobar que el día introducido es válido para el año y mes introducidos.

Por ejemplo, valida que no pueda haber un día 29 de febrero en un año no bisiesto o que no se pueda introducir el día 31 de abril.



3.2.1 Descripción de subrutinas

3.2.2 SetDia(), SetMes() y SetYear()

En reloj.c SetDia(), SetMes() y SetYear() calculan que la fecha introducida sea válida, es decir, no se permite por ejemplo el 31 de noviembre y la actualizan del TipoCalendario pasado como parámetro.

```
int SetDia(int dia, TipoCalendario *p_calendario){
	//El dia introducido no puede ser 0 ni inferior
	if(dia <= 0){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Introduzca un día válido");
		fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 1;
	}
	//Comprobamos que se han introducido 1 o 2 digitos
	int num_digitos = 0;
	int aux = dia;
	while(aux!=0){
	aux /= 10;
	num_digitos++;
	}
	if(num_digitos < 1|| num_digitos>2){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Introduzca un dia de maximo 2 digitos");
		fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 2;
	}
	int mesActual = p_calendario->MM;
	int yearActual = p_calendario->yyyy;
	int maxDiaMes = CalculaDiasMes(mesActual, yearActual);
	if(dia > maxDiaMes){
	dia = maxDiaMes;
	}
	p_calendario->dd = dia;
	return 0;

}

int SetMes(int mes, TipoCalendario *p_calendario){
	//El día introducido no puede ser 0 ni inferior
	if(mes <= 0){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Introduzca un mes valido");
		fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 1;
	}
	//Comprobamos que se han introducido 1 o 2 dígitos
	int num_digitos = 0;
```

```
        int aux = mes;
        while(aux!=0){
        aux /= 10;
        num_digitos++;
        }
        if(num_digitos < 1|| num_digitos>2){
        piLock(STD_IO_LCD_BUFFER_KEY);
                printf("Introduzca un mes de maximo 2 digitos");
                fflush(stdout);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        return 2;
        }
        if(mes > MAX_MONTH){
        mes = MAX_MONTH;
        }
        p_calendario->MM = mes;
        return 0;
}

int SetYear(int year, TipoCalendario *p_calendario){
        //El día introducido no puede ser 0 ni inferior
        if(year <= 0){
        piLock(STD_IO_LCD_BUFFER_KEY);
                printf("Introduzca un year d.C.");
                fflush(stdout);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        return 1;
        }
        if(year < MIN_YEAR){
        year = MIN_YEAR;
        }
        p_calendario->yyyy = year;
        return 0;
}
```

3.2.3 PreparaSetNewDate() CancelSetNewDate() y SetNewDate()

Por otro lado, las funciones auxiliares PreparaSetNewDate() y CancelSetNewDate() y SetNewDate() se encargan de limpiar los flags asociados a la fecha, además de limpiar los datos contenidos en las variables tempDate y digitosGuardadosDate. SetNewDate() además se encarga de guardar la fecha almacenada en tempDate en las variables del sistema invocando a SetDia() SetMes() y SetYear().

```
void PrepareSetNewDate(fsm_t* p_this){
    TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
    piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_CANCEL_NEW_DATE);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_DATE_IS_READY);
    piUnlock(SYSTEM_KEY);
    #if VERSION < 4
    piLock(STD_IO_LCD_BUFFER_KEY);
            printf("\r[SET_DATE] Introduzca la nueva fecha en formato dd/mm/yyyy\n");
            fflush(stdout);
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    #if VERSION==4
```

```
    piLock(STD_IO_LCD_BUFFER_KEY);
    lcdClear(p_sistema->lcdId);
    lcdPosition(p_sistema->lcdId, 0, 0);
    lcdPrintf(p_sistema->lcdId, "[SET_DATE]");
    lcdPosition(p_sistema->lcdId, 0, 1);
    lcdPrintf(p_sistema->lcdId, "__/__/____");
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
}

void CancelSetNewDate(fsm_t* p_this){
    TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
    p_sistema->tempDate = 0;
    p_sistema->digitosGuardadosDate = 0;
    piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_CANCEL_NEW_DATE);
    piUnlock(SYSTEM_KEY);
    #if VERSION < 4
    piLock(STD_IO_LCD_BUFFER_KEY);
            printf("\r[SET_DATE] Operacion cancelada\n");
            fflush(stdout);
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    #if VERSION==4
    piLock(STD_IO_LCD_BUFFER_KEY);
    lcdClear(p_sistema->lcdId);
    lcdPosition(p_sistema->lcdId, 0, 0);
    lcdPrintf(p_sistema->lcdId, "[SET_DATE]");
    lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "CANCELLED");
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
}
void SetNewDate(fsm_t* p_this){
    TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
    int fecha = p_sistema->tempDate;
    int dia = fecha / 1000000; //2 digitos superiores
    int mes = (fecha / 10000) % 100; //2 digitos del medio
    int year = fecha%10000; //4 ultimos digitos
    piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_DATE_IS_READY);
    piUnlock(SYSTEM_KEY);
    //piLock(RELOJ_KEY);
    SetYear(year, &p_sistema->reloj.calendario);
    SetMes(mes, &p_sistema->reloj.calendario);
    SetDia(dia, &p_sistema->reloj.calendario);
    //piUnlock(RELOJ_KEY);
    dia = p_sistema->reloj.calendario.dd;
    mes = p_sistema->reloj.calendario.MM;
    year = p_sistema->reloj.calendario.yyyy;

    int weekDay = CalculaDiaSemana(dia, mes, year);
    p_sistema->reloj.calendario.weekDay = weekDay;
    p_sistema->tempDate = 0;
    p_sistema->digitosGuardadosDate = 0;
```

```
}
```

3.2.4 ProcesaDigitoDate()

Para procesar los dígitos de la fecha, los vamos almacenando en una variable tempDate como hacíamos en ProcesaDigitoTime(). Los primeros 2 dígitos serán de día, los 2 siguientes de mes y los 4 últimos de año. Nos aseguramos siempre de que los valores introducidos sean adecuados (que no se introduzcan meses por encima de 12, días por encima de 31 o años por debajo de 1970) pero la comprobación de que la fecha introducida en sí sea válida se hará al hacer SetNewDate() con las invocaciones a SetDia(), SetMes() y SetYear().

```
void ProcesaDigitoDate(fsm_t* p_this){
    TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
    int tempDate = p_sistema->tempDate;
    int digitosGuardadosDate = p_sistema->digitosGuardadosDate;
    #if VERSION >= 2
    int ultimoDigito = g_coreWatch.digitoPulsado;
    #endif
    int aux = 0;
    piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
    piUnlock(SYSTEM_KEY);
    //procesamos el dia
    if(digitosGuardadosDate == 0){
        ultimoDigito = min(ultimoDigito, MAX_DAY/10);
        tempDate = tempDate*10 + ultimoDigito;
    }else if(digitosGuardadosDate == 1) {
        tempDate = min(tempDate*10+ultimoDigito, MAX_DAY);
    //procesamos el mes
    }else if(digitosGuardadosDate == 2) {
        ultimoDigito = min(ultimoDigito, 1);
        tempDate = tempDate*10 + ultimoDigito;
    }else if(digitosGuardadosDate == 3){
        aux = tempDate / 10;
        aux = (aux * 100) + MAX_MONTH; // aux = dd/12
        tempDate = min(tempDate*10+ultimoDigito, aux);
    //procesamos el year
    }else if(digitosGuardadosDate == 4){
        ultimoDigito = max(ultimoDigito, 1);
        tempDate = tempDate*10 + ultimoDigito;
    }else if(digitosGuardadosDate == 5){
        aux = tempDate / 10;
        aux = aux * 100 + MIN_YEAR/100; //aux = dd/mm/19xx
        tempDate = max(tempDate*10+ultimoDigito, aux);
    }else if(digitosGuardadosDate == 6){
        aux = tempDate / 100;
        aux = aux * 1000 + MIN_YEAR/10; //aux = dd/mm/197x
        tempDate = max(tempDate*10+ultimoDigito, aux);
    }else{
        aux = tempDate / 1000;
        aux = aux * 10000 + MIN_YEAR; //aux = dd/mm/1970
        tempDate = max(tempDate*10+ultimoDigito, aux);
        piLock(SYSTEM_KEY);
            g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        piUnlock(SYSTEM_KEY);
        piLock(SYSTEM_KEY);
```

```
            g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_DATE_IS_READY;
        piUnlock(SYSTEM_KEY);
    }
    digitosGuardadosDate++;
    #if VERSION<4
    piLock(STD_IO_LCD_BUFFER_KEY);
        printf("\rNueva fecha temporal: %d\n", tempDate);
        fflush(stdout);
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    #if VERSION >= 4
    piLock(STD_IO_LCD_BUFFER_KEY);
    if(digitosGuardadosDate<3){
        lcdPosition(p_sistema->lcdId, digitosGuardadosDate-1, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
    }else if(digitosGuardadosDate >= 3 && digitosGuardadosDate < 5){
        lcdPosition(p_sistema->lcdId, digitosGuardadosDate, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
    }else{
        lcdPosition(p_sistema->lcdId, digitosGuardadosDate+1, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
    }
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    p_sistema->tempDate = tempDate;
    p_sistema->digitosGuardadosDate = digitosGuardadosDate;
}
```

4 Mejora 2: Impresión de la hora y fecha con 2 dígitos

4.1 Objetivos de la mejora

El objetivo de esta mejora es que todos los números que se muestran por el display estén representados con 2 dígitos. El uso principal de esta mejora y el caso en el que será visible será cuando la hora (horas, minutos o segundos) o la fecha (día o mes) sean menores que 10.

4.2 Descripción del subsistema Software

Para esta implementación, hemos modificado Sho wTime() para que cuando la hora, minutos o segundos sean inferiores a 10, se añada un 0 delante en la impresión (ej: 09:08:05). Para ello hemos hecho uso de sentencias if-else y de la función sprintf(), que puede formatear del mismo modo que lo hace printf (printf("%d", hora) imprime el valor de la variable hora por pantalla) pero en vez de sacar el valor por pantalla lo guarda en una variable. Para ejecutar sprintf() sucesivas veces seguidas y que no se borre lo que ha guardado en anteriores ejecuciones, es tan sencillo como pasar como parámetro la variable donde estamos guardando todo el string. Ejemplo:

char str = "05:"
min = 3;
sprintf(str, "%s0%d:", str, min);
//valor de str tras ejectuar sprintf: str = “05:03:”

4.3 Descripción de subrutinas

4.3.1 ShowTime()

Como se ha explicado anteriormente, la modificación en ShowTime() consiste en varias sentencias if-else en que se ejecuta sprintf() con unos u otros valores. Conseguimos así almacenar en la variable — el string que queremos imprimir.

```
void ShowTime(fsm_t* p_this){
       TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
       TipoRelojShared sharedvars = GetRelojSharedVar();
       sharedvars.flags = sharedvars.flags & (~ FLAG_TIME_ACTUALIZADO);
       SetRelojSharedVar(sharedvars);
       TipoReloj reloj_sistema = p_sistema->reloj;
       int hora = reloj_sistema.hora.hh;
       int min = reloj_sistema.hora.mm;
       int seg = reloj_sistema.hora.ss;
       int dia = reloj_sistema.calendario.dd;
       int mes = reloj_sistema.calendario.MM;
       int year = reloj_sistema.calendario.yyyy;
       int ampm;
       if(reloj_sistema.hora.formato == 12){
       ampm = hora_ampm[1][hora];
       hora = hora_ampm[0][hora];
       }
       piLock(STD_IO_LCD_BUFFER_KEY);
       lcdClear(p_sistema->lcdId);
       lcdPosition(p_sistema->lcdId, 0, 0);
       //impresion hora
       char str[12];
       //usamos sprintf para anadir a la variable str la hora min y seg
```

```
        if(hora<10){
        sprintf(str, "0%d:", hora);
        }else{
        sprintf(str, "%d:", hora);
        }
        if(min<10){
        sprintf(str, "%s0%d:", str, min);
        }else{
        sprintf(str, "%s%d:", str, min);
        }
        if(seg<10){
        sprintf(str, "%s0%d", str, seg);
        }else{
        sprintf(str, "%s%d", str, seg);
        }
        if(reloj_sistema.hora.formato == 24){
        lcdPrintf(p_sistema->lcdId, "%s", str);
        }else{
        if(ampm){
               lcdPrintf(p_sistema->lcdId, "%s pm", str);
        }else{
               lcdPrintf(p_sistema->lcdId, "%s am", str);
        }
        }
    char date[12];
        sprintf(date,"%c ", diasemana[p_sistema->reloj.calendario.weekDay]);
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "%s%d/%d/%d", date, dia, mes, year);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}
```

5 Mejora 3: Formato de hora AM/PM

5.1 Objetivos de la mejora

El objetivo de esta mejora es que el usuario pueda considerar la forma de representación de hora AM-PM cuando se elige el formato 12 horas.

A su vez, también se pretende que el usuario sea capaz de cambiar el formato de visualización de la hora (12h o 24h) pulsando una tecla, en concreto, la tecla 'C'.

5.2 Descripción del subsistema Software

El objetivo que tenemos es implementar esta mejora como una mejora estética. Esto quiere decir que trabajaremos internamente con el formato 24 horas, para que cambie de hora, al pasar a SET_TIME para cambiar la hora… siempre los valores de hora, minutos y segundos del TipoHora del sistema estarán en formato 24. Sin embargo, al imprimir por pantalla o cambiar la hora haremos distinciones para cada formato, es decir, imprimir la hora en formato 12 o permitir al usuario introducir la nueva hora en formato 12.

5.2.1 ActualizaHora()

Para el desarrollo de esta funcionalidad, lo primero que hemos hecho es reestructurar el código de reloj.c relacionado con el tratamiento interno de la hora para eliminar todo aquello relacionado con el formato de 12 horas. Esto es, porque lo que buscamos es que el sistema siempre interprete de forma interna las horas en formato 24, y solo tendremos en cuenta el formato 12 horas para mostrarlo por pantalla o (si el usuario tiene seleccionado este formato) para introducir una hora en el sistema.

```
void ActualizaHora(TipoHora *p_hora){
	//actualizar segundero
	p_hora->ss = (p_hora->ss+1)%60;
	//actualizar minutero
	if(p_hora->ss == 0) {
	p_hora->mm = (p_hora->mm+1)%60;
	if(p_hora->mm == 0){
		p_hora->hh = p_hora->hh+1;
		if(p_hora->hh > MAX_HOUR_24){
		p_hora->hh = MIN_HOUR_24;
		}
	}
	}

}
```

5.2.2 ShowTime() y array hora_ampm

Una vez hecho esto, queremos que se imprima la hora en formato 12 cuando se ejecute ShowTime() si el formato elegido por el usuario es el de 12 horas. Para ello hemos añadido un array hora_ampm[24][2]. Con las horas internas del sistema (0-23) podemos acceder a las casillas del array; en la primera fila para la hora equivalente en formato 12 y en la segunda fila introducimos un 0 si es am y un 1 si es pm (ej: 23 - 11, 1 - 11 pm). Asignamos ese nuevo valor a la variable hora, y en función de ampm, imprimimos la hora con la coletilla "am" o "pm".

```
const int hora_ampm[2][24] = {
        {12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
};

void ShowTime(fsm_t* p_this){
       TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
       TipoRelojShared sharedvars = GetRelojSharedVar();
       sharedvars.flags = sharedvars.flags & (~ FLAG_TIME_ACTUALIZADO);
       SetRelojSharedVar(sharedvars);
       TipoReloj reloj_sistema = p_sistema->reloj;
       int hora = reloj_sistema.hora.hh;
       int min = reloj_sistema.hora.mm;
       int seg = reloj_sistema.hora.ss;
       int dia = reloj_sistema.calendario.dd;
       int mes = reloj_sistema.calendario.MM;
       int year = reloj_sistema.calendario.yyyy;
       int ampm;
       if(reloj_sistema.hora.formato == 12){
       ampm = hora_ampm[1][hora];
       hora = hora_ampm[0][hora];
       }
       piLock(STD_IO_LCD_BUFFER_KEY);
       lcdClear(p_sistema->lcdId);
       lcdPosition(p_sistema->lcdId, 0, 0);
       //impresion hora
       char str[12];
       //usamos sprintf para anadir a la variable str la hora min y seg
       if(hora<10){
       sprintf(str, "0%d:", hora);
       }else{
       sprintf(str, "%d:", hora);
       }
       if(min<10){
       sprintf(str, "%s0%d:", str, min);
       }else{
       sprintf(str, "%s%d:", str, min);
       }
       if(seg<10){
       sprintf(str, "%s0%d", str, seg);
       }else{
       sprintf(str, "%s%d", str, seg);
       }
       if(reloj_sistema.hora.formato == 24){
       lcdPrintf(p_sistema->lcdId, "%s", str);
       }else{
       if(ampm){
```

```
                lcdPrintf(p_sistema->lcdId, "%s pm", str);
        }else{
                lcdPrintf(p_sistema->lcdId, "%s am", str);
        }
        }
    char date[12];
        sprintf(date,"%c ", diasemana[p_sistema->reloj.calendario.weekDay]);
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "%s%d/%d/%d", date, dia, mes, year);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}
```

5.2.3 ProcesaDigitoTime() nuevo

Además de imprimir la hora en formato 12 por pantalla, queremos que si el usuario tiene seleccionado ese modo al acceder a SET_TIME, la hora se procese en formato 12. Para ello, rehacemos el flujograma de ProcesaDigitoTime(), diferenciando claramente entre formato 12 y formato 24. El formato 24 es igual que anteriormente, el formato 12 pedirá 4 dígitos al usuario para la hora (formateándolos adecuadamente al formato 12) además de un dígito adicional para seleccionar si la hora introducida es am o pm. Una vez introducidos todos los dígitos, se activará el flag que hace que se ejecute SetNewTime(). En esta función hemos realizado también modificaciones, ya que queremos tratar la hora en formato 24 internamente, por lo que los dígitos almacenados en tempTime tendrán que ser tratados. Concretamente, si la hora es pm y mayor que 12 se le sumará 12 a la hora, mientras que si son las 12 am, pondremos la hora a 0. Esta nueva hora será pasada junto con los minutos a SetHora().

```
void ProcesaDigitoTime(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        TipoReloj r = p_sistema->reloj;
        int formato = r.hora.formato;
        int tempTime = p_sistema->tempTime;
        int digitosGuardados = p_sistema->digitosGuardados;
        #if VERSION >= 2
        int ultimoDigito = g_coreWatch.digitoPulsado;
        #endif

        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        piUnlock(SYSTEM_KEY);
        if(formato == 12){
        if(digitosGuardados == 0){
                ultimoDigito = min(1, ultimoDigito);
                tempTime = tempTime*10 + ultimoDigito;
        }else if(digitosGuardados == 1){
                if(tempTime == 0){
                    ultimoDigito = max(1, ultimoDigito);
                }else{
                ultimoDigito = min(2, ultimoDigito);
                }
                tempTime = tempTime * 10 + ultimoDigito;
        }else if(digitosGuardados == 2){
                tempTime = tempTime * 10 + min(5, ultimoDigito);
        }else if (digitosGuardados == 3){
                tempTime = tempTime*10+ultimoDigito;
```

```
        }else{
                ultimoDigito = min(ultimoDigito, 1);
                tempTime = tempTime*10+ultimoDigito;
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_TIME_IS_READY;
                piUnlock(SYSTEM_KEY);
        }
        }else{
        if(digitosGuardados == 0){
                ultimoDigito = min(2, ultimoDigito);
                tempTime = tempTime*10 + ultimoDigito;
        }else if(digitosGuardados == 1){
                if(tempTime  == 2){
                    ultimoDigito = min(3, ultimoDigito);
                }
                tempTime = tempTime * 10 + ultimoDigito;
        }else if(digitosGuardados == 2){
                tempTime = tempTime * 10 + min(5, ultimoDigito);
        }else{
                tempTime = tempTime*10+ultimoDigito;
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_TIME_IS_READY;
                piUnlock(SYSTEM_KEY);
        }
        }
        digitosGuardados++;
        #if VERSION == 4
        piLock(STD_IO_LCD_BUFFER_KEY);
        if(digitosGuardados<3){
        lcdPosition(p_sistema->lcdId, digitosGuardados-1, 1);
         lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }else{
        lcdPosition(p_sistema->lcdId, digitosGuardados, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }
        if(formato == 12 && digitosGuardados == 4){
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "SELECCIONE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "AM:0 / PM:1");
        }
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif

        p_sistema->tempTime = tempTime;
        p_sistema->digitosGuardados = digitosGuardados;

}
```

5.2.4 SwitchFormat() y CompruebaSwitchFormat()

Además de todo el código empleado para poder representar y cambiar la hora en formato 12, hemos añadido otra funcionalidad al sistema: al pulsar la tecla 'C' se cambiará el formato de representación. Es decir, si estabas viendo la hora en formato 12 horas, pasas a verla en formato 24. Esto lo hemos hecho mediante la implementación de una interrupción por flag en el estado STAND_BY, que ejecuta la función SwitchFormat(), en que se cambia el valor formato de TipoCoreWatch al opuesto del actual.

```
int CompruebaSwitchFormat(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_SWITCH_FORMAT;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}
void SwitchFormat(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	int formato = 12;
	if(p_sistema->reloj.hora.formato == 12) formato = 24;
	SetFormat(formato, &p_sistema->reloj.hora);
	piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SWITCH_FORMAT);
	piUnlock(SYSTEM_KEY);
}
```

6 Mejora 4: Alarma

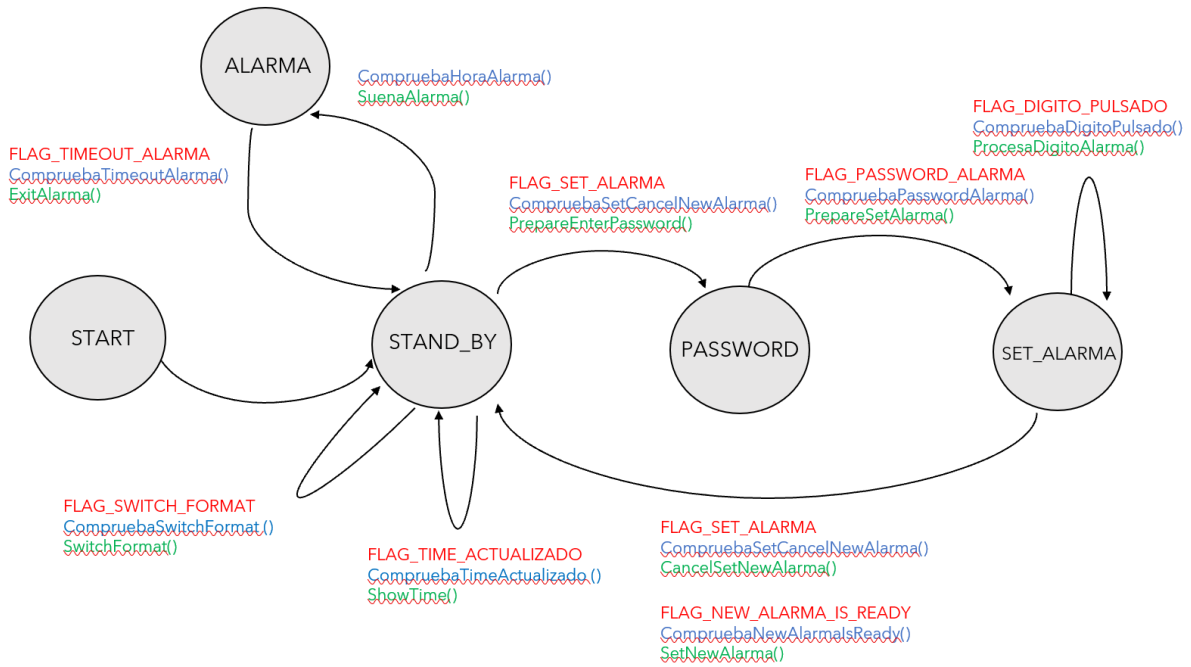
6.1 Objetivos de la mejora

El objetivo de esta mejora es permitir al usuario añadir una alarma que salte dada la hora y minutos escogidos.

6.2 Descripción del subsistema Software

Hemos implementado una alarma en nuestro coreWatch. El funcionamiento de la alarma es el siguiente: cuando la fsm está en el estado STAND_BY, se comprueba si la alarma está activada y si la hora actual es la hora de la alarma con la función CompruebaHoraAlarma(). En caso afirmativo, se pasa al estado ALARMA ejecutando la función SuenaAlarma(). En esta función se arranca un temporizador con un timeout de TIMEOUT_ALARMA que hemos puesto a 10 segundos que sirve para que nuestra máquina de estados solamente permanezca ese tiempo en el estado ALARMA. Una vez transcurrido ese tiempo, se activa un flag y se vuelve al estado STAND_BY. La forma en la que la función SuenaAlarma() avisa de que se ha pasado al estado de ALARMA es imprimiendo por pantalla mensajes de "Bip!".

Finalmente, SET_ALARMA es idéntico a SET_TIME, salvo por la particularidad de que introduce un dígito más que se utiliza para establecer si la alarma está apagada o encendida.



6.2.1 CompruebaHoraAlarma()

Comprueba si la hora actual del sistema es la hora de la alarma, siempre que la alarma esté activada. Además, los segundos del sistema deben ser cero para que solamente se ejecute una vez y no varias veces durante el minuto.

```
int CompruebaHoraAlarma(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	int res=0;
	piLock(RELOJ_KEY);
	if(p_sistema->alarma.alarmaActivada==1){
	int horaalarma = p_sistema->alarma.hora.hh;
	int minalarma = p_sistema->alarma.hora.hh;
	int horasis = p_sistema->reloj.hora.hh;
	int minsis = p_sistema->reloj.hora.hh;
	int segsis = p_sistema->reloj.hora.ss;
	if(horaalarma == horasis && minsis == minalarma && segsis == 0){
		res = 1;
	}
	}
	piUnlock(RELOJ_KEY);
	return res;
}
```

6.2.2 SuenaAlarma()

Función que imprime por pantalla mensajes de “Bip!”, para indicar que la alarma ha sido activada. Además, inicia un timer con timeout TIMEOUT_ALARMA que servirá para saber posteriormente cuándo salir del estado ALARMA.

```
 void SuenaAlarma(fsm_t* p_this){
    TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	tmr_t* tempalarma = tmr_new(tmr_timeout_alarma);
	p_sistema->alarma.timer = tempalarma;
	tmr_startms(tempalarma, TIMEOUT_ALARMA);
	piLock(STD_IO_LCD_BUFFER_KEY);
	lcdClear(p_sistema->lcdId);
	lcdPosition(p_sistema->lcdId, 0, 0);
	lcdPrintf(p_sistema->lcdId, "Bip! Bip!");
	lcdPosition(p_sistema->lcdId, 0, 1);
	lcdPrintf(p_sistema->lcdId, "Bip! Bip!");
	piUnlock(STD_IO_LCD_BUFFER_KEY);
}
```

6.2.3 CompruebaTimeoutAlarma()

Función que comprueba si se ha activado el flag de timeout de alarma.

```
int CompruebaTimeoutAlarma(fsm_t* this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_TIMEOUT_ALARMA;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}
```

6.2.4 tmr_timeout_alarma()

Activa el flag de timeout de alarma.

```
void tmr_timeout_alarma(union sigval value) {
	piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch | FLAG_TIMEOUT_ALARMA;
	piUnlock(SYSTEM_KEY);
}
```

6.2.5 ExitAlarma()

Función ejecutada al salir del estado alarma, cuya funcionalidad es limpiar flags y limpiar el lcd.

```
void ExitAlarma(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	piLock(STD_IO_LCD_BUFFER_KEY);
	lcdClear(p_sistema->lcdId);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_TIMEOUT_ALARMA);
	piUnlock(SYSTEM_KEY);
}
```

6.2.6 PrepareSetAlarma(), ProcesaDigitoAlarma(),CancelSetNewAlarma(), SetNewAlarma()

Son idénticas a las empleadas para SET_TIME, con la particularidad de que se va a introducir un último dígito que servirá para indicar si la alarma está activada o no.

```
void PrepareSetAlarma(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	p_sistema->alarma.digitosGuardadosAlarma = 0;
	p_sistema->alarma.tempAlarma = 0;
	piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_ALARMA);
	g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGITO_PULSADO);
	g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_ALARMA_IS_READY);
	piUnlock(SYSTEM_KEY);
	piLock(STD_IO_LCD_BUFFER_KEY);
	lcdClear(p_sistema->lcdId);
	lcdPosition(p_sistema->lcdId, 0, 0);
	lcdPrintf(p_sistema->lcdId, "[SET_ALARMA]");
	lcdPosition(p_sistema->lcdId, 0, 1);
	lcdPrintf(p_sistema->lcdId, "__:__");
	delay(1000);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
}

void CancelSetNewAlarma(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	p_sistema->alarma.digitosGuardadosAlarma = 0;
	p_sistema->alarma.tempAlarma = 0;
	piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_ALARMA);
```

```
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_ALARMA_IS_READY);
        piUnlock(SYSTEM_KEY);
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_ALARMA]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CANCELLED");
        delay(1000);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}

void ProcesaDigitoAlarma(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        TipoReloj r = p_sistema->reloj;
        int formato = r.hora.formato;
        int tempTime = p_sistema->alarma.tempAlarma;
        int digitosGuardados = p_sistema->alarma.digitosGuardadosAlarma;
        int ultimoDigito = g_coreWatch.digitoPulsado;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        piUnlock(SYSTEM_KEY);
        if(formato == 12){
        if(digitosGuardados == 0){
                ultimoDigito = min(1, ultimoDigito);
                tempTime = tempTime*10 + ultimoDigito;
        }else if(digitosGuardados == 1){
                if(tempTime == 0){
                ultimoDigito = max(1, ultimoDigito);
                }else{
                ultimoDigito = min(2, ultimoDigito);
                }
                tempTime = tempTime * 10 + ultimoDigito;
        }else if(digitosGuardados == 2){
                tempTime = tempTime * 10 + min(5, ultimoDigito);
        }else if (digitosGuardados == 3){
                tempTime = tempTime*10+ultimoDigito;
        }else if (digitosGuardados == 4){
                ultimoDigito = min(ultimoDigito, 1);
                tempTime = tempTime*10+ultimoDigito;
        }else {
                tempTime = tempTime*10+ultimoDigito;
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_ALARMA_IS_READY;
                piUnlock(SYSTEM_KEY);
        }
        }else{
        if(digitosGuardados == 0){
                ultimoDigito = min(2, ultimoDigito);
                tempTime = tempTime*10 + ultimoDigito;
        }else if(digitosGuardados == 1){
```

```
                if(tempTime  == 2){
                ultimoDigito = min(3, ultimoDigito);
                }
                tempTime = tempTime * 10 + ultimoDigito;
        }else if(digitosGuardados == 2){
                tempTime = tempTime * 10 + min(5, ultimoDigito);
        }else if(digitosGuardados == 3){
                tempTime = tempTime * 10 + ultimoDigito;
        }else{
                tempTime = tempTime*10+ultimoDigito;
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_ALARMA_IS_READY;
                piUnlock(SYSTEM_KEY);

        }
        }
        digitosGuardados++;
        #if VERSION == 4
        piLock(STD_IO_LCD_BUFFER_KEY);
        if(digitosGuardados<3){
        lcdPosition(p_sistema->lcdId, digitosGuardados-1, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }else{
        lcdPosition(p_sistema->lcdId, digitosGuardados, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }
        if(formato == 12 && digitosGuardados == 4){
        delay(500);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "SELECCIONE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "AM:0 / PM:1");
        }

        if((formato == 24 && digitosGuardados == 4) || (formato == 12 && digitosGuardados
== 5)){

        delay(500);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_ALARMA]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "OFF:0 / ON:1");
        }
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        p_sistema->alarma.tempAlarma = tempTime;
        p_sistema->alarma.digitosGuardadosAlarma = digitosGuardados;
}

void SetNewAlarma(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_ALARMA_IS_READY);
```

```
        piUnlock(SYSTEM_KEY);
        if (p_sistema->reloj.hora.formato == 12){
        int activada = p_sistema->alarma.tempAlarma % 10;
        int ampm = (p_sistema->alarma.tempAlarma / 10) % 10;
        int min = (p_sistema->alarma.tempAlarma / 100) % 100;
        int hora = p_sistema->alarma.tempAlarma / 10000;
        if(ampm==1 && hora!=12){
                hora+=12;
        }
        if(ampm==0 && hora == 12){
                hora = 0;
        }
        hora = hora*100 + min;
        SetHora(hora, &p_sistema->alarma.hora);
        p_sistema->alarma.alarmaActivada = activada;
        }else{
        int hora = p_sistema->alarma.tempAlarma/10;
        int activada = p_sistema->alarma.tempAlarma%10;
        SetHora(hora, &p_sistema->alarma.hora);
        p_sistema->alarma.alarmaActivada = activada;
        }
        p_sistema->tempTime = 0;
        p_sistema->digitosGuardados = 0;
}
```

7 Mejora 5: Contraseña del sistema

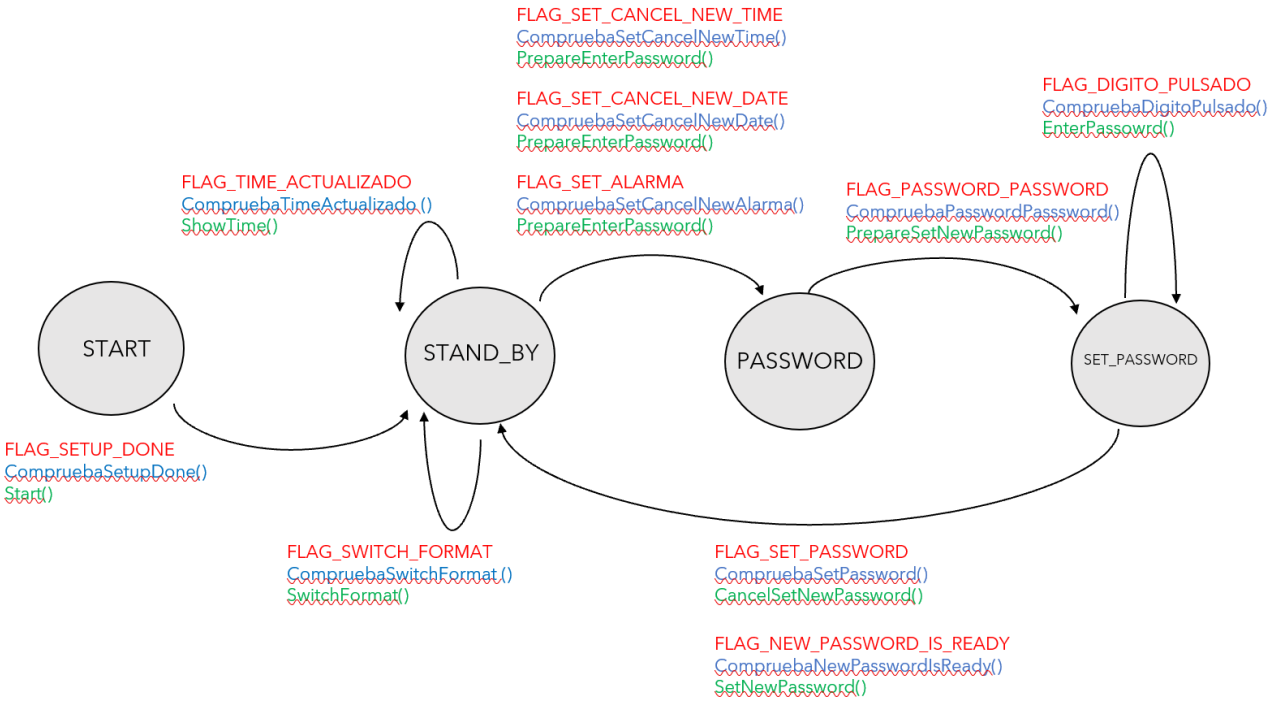
7.1 Objetivos de la mejora

El objetivo de esta mejora es requerir al usuario una contraseña (“Password” o PIN) antes de modificar la hora, la fecha, la alarma e incluso la propia contraseña.

Esta mejora busca añadirle una mayor seguridad a nuestro sistema ya que de no introducir el “Password” correcto no se permitirá modificar ninguna de las variables modificables (hora, fecha o alarma) del CoreWatch y sólo será posible visualizar la hora y fecha.

7.2 Descripción del subsistema Software

Consiste en la adición de una contraseña al sistema que será requerida al usuario cada vez que quiera realizar modificaciones de hora, fecha o alarma, es decir, acceder a los estados SET_TIME, SET_DATE o SET_ALARMA respectivamente. Para ellos, cambiamos el estado al que se pasa al pulsar las teclas asignadas a los estados de configuración del sistema por el estado PASSWORD en la fsm. En este estado, se procesan los dígitos que introduce el usuario. Una vez introducidos, si se corresponden con la contraseña del sistema deja acceder a los estados de configuración, y sino se pide al usuario que introduzca de nuevo la contraseña.



7.3 Ejecución e implementación de las rutinas de contraseña

Creemos que aqui es interesante ir explicando de forma gradual cómo se ha ido implementando esta mejora.

Como modificaciones al código ya existente, hemos añadido dos nuevos estados al enumerado de estados de la fsm: PASSWORD y SET_PASSWORD.

```
enum FSM_ESTADOS_SISTEMA{

      START, STAND_BY, SET_TIME, SET_DATE, SET_ALARMA, ALARMA, SET_PASSWORD,
PASSWORD

};
```

Además, hemos creado una nueva struct, TipoPassword, que contiene todas las variables necesarias para la implementación. Lo hemos hecho de esta manera para mantener cierto orden en el código y tener localizadas todas las variables relacionadas con la contraseña en el mismo lugar. Una instanciación de TipoPassword será almacenada dentro de TipoCoreWatch. La variable number sirve para almacenar el valor de la contraseña, el resto de variables las iremos explicando conforme vayan siendo necesarias.

```
typedef struct{
    int  number;
    tmr_t* tmrPassword;
    int  tempPassword;
    int  digitosGuardadosPassword;
    int tempNewPassword;
    int tempNewPasswordRepeat;
}TipoPassword;
```

Adicionalmente, hemos modificado la máquina de estados para que al activarse FLAG_CANCEL_SET__TIME, FLAG_CANCEL_SET__DATE y FLAG_SET_ALARMA en lugar de pasar a los estados de configuración del sistema se pase al estado PASSWORD y se ejecute la función PrepareEnterPassword(), encargada de lidiar con inicializaciones a cero de ciertas variables y de borrar flags por si se encontraban activados.

```
void PrepareEnterPassword(fsm_t* p_this){
       TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
       p_sistema->password.digitosGuardadosPassword = 0;
       p_sistema->password.tempPassword = 0;
       piLock(SYSTEM_KEY);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_DATE);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_TIME);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_ALARMA);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_PASSWORD);
       piUnlock(SYSTEM_KEY);
#if VERSION==4
       if(g_flagsCoreWatch & FLAG_SET_PASSWORD){
       piLock(STD_IO_LCD_BUFFER_KEY);
       lcdClear(p_sistema->lcdId);
       lcdPosition(p_sistema->lcdId, 0, 0);
       lcdPrintf(p_sistema->lcdId, "CHANGING");
       lcdPosition(p_sistema->lcdId, 0, 1);
       lcdPrintf(p_sistema->lcdId, "PASSWORD");
```

```
        delay(1000);
        lcdClear(p_sistema->lcdId);
         lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "INTRODUCE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CURRENT");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        }else{
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "ENTER");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "PASSWORD");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        }

#endif
}
```

Una vez en el estado PASSWORD, se van procesando los dígitos introducidos mediante la función EnterPassword(). Los dígitos introducidos los almacenamos en tempPassword, y la variable digitosGuardadosPassword se incrementa con cada dígito guardado. Una vez llegamos a 4 dígitos, se comprueba si la contraseña introducida es igual a la contraseña del sistema. En caso afirmativo, procedemos a activar los flags necesarios para pasar al estado de configuración. Al tener todos los estados de configuración la misma contraseña, cuando se activaron FLAG_CANCEL_SET__TIME, FLAG_CANCEL_SET__DATE, FLAG_SET_ALARMA y FLAG_SET_PASSWORD no los desactivamos para en este momento saber a qué estado quería acceder el usuario cuando pasamos al estado PASSWORD. Así, en función de qué flags estén activos cuando se ha corroborado que la contraseña introducida es correcta, activaremos FLAG_PASSWORD_TIME, FLAG_PASSWORD_DATE, FLAG_PASSWORD_ALARMA o FLAG_PASSWORD_PASSWORD.

```
void EnterPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        int tempPassword = p_sistema->password.tempPassword;
        int digitosGuardadosPassword = p_sistema->password.digitosGuardadosPassword;
        int ultimoDigito = g_coreWatch.digitoPulsado;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        piUnlock(SYSTEM_KEY);
        if(digitosGuardadosPassword < 4){
        tempPassword = tempPassword*10 + ultimoDigito;
        digitosGuardadosPassword++;
        }
        if(digitosGuardadosPassword == 4){
        if(tempPassword == p_sistema->password.number){
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                if(g_flagsCoreWatch & FLAG_SET_PASSWORD){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_PASSWORD;
```

```
                }
                if(g_flagsCoreWatch & FLAG_SET_CANCEL_NEW_TIME){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_TIME;
                }
                if(g_flagsCoreWatch & FLAG_SET_CANCEL_NEW_DATE){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_DATE;
                }
                if(g_flagsCoreWatch & FLAG_SET_ALARMA){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_ALARMA;
                }
                piUnlock(SYSTEM_KEY);
        }else{
                digitosGuardadosPassword =0;
                tempPassword = 0;
                piLock(STD_IO_LCD_BUFFER_KEY);
                lcdClear(p_sistema->lcdId);
                lcdPosition(p_sistema->lcdId, 0, 0);
                lcdPrintf(p_sistema->lcdId, "WRONG PASSWORD!");
                lcdPosition(p_sistema->lcdId, 0, 1);
                lcdPrintf(p_sistema->lcdId, "TRY AGAIN");
                delay(1000);
                lcdClear(p_sistema->lcdId);
                lcdPosition(p_sistema->lcdId, 0, 0);
                lcdPrintf(p_sistema->lcdId, "ENTER");
                lcdPosition(p_sistema->lcdId, 0, 1);
                lcdPrintf(p_sistema->lcdId, "PASSWORD");
                piUnlock(STD_IO_LCD_BUFFER_KEY);
        }
        }
        p_sistema->password.tempPassword = tempPassword;
        p_sistema->password.digitosGuardadosPassword = digitosGuardadosPassword;
}
```

Estos flags son comprobados por sus correspondientes funciones de comprobación, y en función de cuál se active se saldrá del estado PASSWORD al estado de configuración correspondiente.

```
 int CompruebaPasswordTime(fsm_t* this){
        int res;
        piLock(SYSTEM_KEY);
        res = g_flagsCoreWatch & FLAG_PASSWORD_TIME;
        piUnlock(SYSTEM_KEY);
        if(res!= 0)return 1;
        return 0;
}

int CompruebaPasswordDate(fsm_t* this){
        int res;
        piLock(SYSTEM_KEY);
        res = g_flagsCoreWatch & FLAG_PASSWORD_DATE;
        piUnlock(SYSTEM_KEY);
        if(res!= 0)return 1;
        return 0;
}
```

```
int CompruebaPasswordAlarma(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_PASSWORD_ALARMA;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}

int CompruebaPasswordPassword(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_PASSWORD_PASSWORD;
	piUnlock(SYSTEM_KEY);
	if(res!=0)return 1;
	return 0;
}
```

8 Mejora 6: SET_PASSWORD

8.1 Objetivos de la mejora

Queremos que la contraseña añadida en el apartado anterior sea modificable por el usuario al pulsar la tecla 'F'. Como el teclado es pequeño hemos tenido que quitar la funcionalidad de Reset de dicha tecla ya que sino no teníamos más teclas y queríamos realizar esta mejora.

8.2 Descripción del subsistema Hardware

SET_PASSWORD es el estado al que pasamos cuando queremos modificar la contraseña del sistema. Para acceder, de nuevo pasamos primero por el estado PASSWORD, teniendo que introducir la contraseña actual. Una vez introducida, se le pide al usuario que teclee dos veces la misma contraseña. Si coinciden las dos contraseñas, se establece como la contraseña, sino se le pide al usuario que la introduzca otras dos veces.

8.3 Descripción de subrutinas

8.3.1 PrepareSetNewPassword() y CancelSetNewPassword()

Se encargan de hacer manejo de flags y de poner variables a cero, así como alguna impresión por pantalla.

```
void PrepareSetNewPassword(fsm_t* p_this){
       TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
       piLock(SYSTEM_KEY);
    g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_PASSWORD);
       g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
       piUnlock(SYSTEM_KEY);
       p_sistema->password.tempNewPassword = 0;
       p_sistema->password.tempNewPasswordRepeat = 0;
       p_sistema->password.digitosGuardadosPassword = 0;
       #if VERSION==4
       piLock(STD_IO_LCD_BUFFER_KEY);
       lcdClear(p_sistema->lcdId);
       lcdPosition(p_sistema->lcdId, 0, 0);
       lcdPrintf(p_sistema->lcdId, "INTRODUCE");
       lcdPosition(p_sistema->lcdId, 0, 1);
       lcdPrintf(p_sistema->lcdId, "NEW");
       piUnlock(STD_IO_LCD_BUFFER_KEY);
       #endif
}

void CancelSetNewPassword(fsm_t* p_this){
       TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
       piLock(SYSTEM_KEY);
       g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGITO_PULSADO);
       g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_PASSWORD);
       g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
       piUnlock(SYSTEM_KEY);
```

```
        p_sistema->password.tempNewPassword = 0;
        p_sistema->password.tempNewPasswordRepeat = 0;
        p_sistema->password.digitosGuardadosPassword = 0;
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "SET_PASSWORD");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CANCELLED");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}
```

8.3.2 EnterNewPassword()

Antes de pasar a SET_PASSWORD, como buen estado de configuración que es, pasamos por PASSWORD y se pide al usuario que introduzca la contraseña actual (hay una impresión personalizada que pone "INTRODUZCA ACTUAL" únicamente cuando el estado al que se quiere acceder es el estado SET_PASSWORD, ver el código de PreparaEnterPassword en la mejora anterior). Una vez introducida, se ejecuta PreparaEnterNewPassword y se pasa a SET_PASSWORD, donde se procesan los dígitos introducidos mediante la función EnterNewPassword() se pide al usuario que teclee dos veces la nueva contraseña. Los dígitos introducidos se procesan mediante la función EnterNewPassword(). Primero se introducen 4 dígitos, a continuación se imprime un mensaje de "REPEAT PLEASE" y posteriormente se introducen los otros 4 dígitos. Las dos contraseñas introducidas se han almacenado en las variables tempNewPassword y tempNewPasswordRepeat, y el contador digitosGuardadosPassword no almacena exactamente el número total de dígitos guardados, ya que sino no sería posible imprimir por pantalla el mensaje de "REPEAT PLEASE". Se incrementa hasta los 4 primeros dígitos, imprime el mensaje, se vuelve a incrementar, y los dígitos de la segunda contraseña van de 5 a 9. Esto se ha hecho así ya que era la solución más directa a un bug que imprimía el mensaje de repetición infinitas veces. Finalmente, si las contraseñas coinciden, se activa un flag que hará pasar a la máquina al estado STAND_BY y ejecutar SetNewPassword(). Si no coinciden, se empiezan a introducir las nuevas contraseñas de nuevo.

```
void EnterNewPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        int tempNewPassword = p_sistema->password.tempNewPassword;
        int tempNewPasswordRepeat = p_sistema->password.tempNewPasswordRepeat;
        int digitosGuardadosPassword = p_sistema->password.digitosGuardadosPassword;
        int ultimoDigito = g_coreWatch.digitoPulsado;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        piUnlock(SYSTEM_KEY);
        if(digitosGuardadosPassword < 4){
        tempNewPassword = tempNewPassword*10 + ultimoDigito;
        digitosGuardadosPassword++;
        }
        if(digitosGuardadosPassword >= 5 && digitosGuardadosPassword < 9){
        tempNewPasswordRepeat = tempNewPasswordRepeat*10 + ultimoDigito;
        digitosGuardadosPassword++;
        }
        if(digitosGuardadosPassword == 9){
        if(tempNewPassword == tempNewPasswordRepeat){
                piLock(STD_IO_LCD_BUFFER_KEY);
                lcdClear(p_sistema->lcdId);
                lcdPosition(p_sistema->lcdId, 0, 0);
                lcdPrintf(p_sistema->lcdId, "PASSWORD");
```

```
                lcdPosition(p_sistema->lcdId, 0, 1);
                lcdPrintf(p_sistema->lcdId, "CHANGED");
                delay(1000);
                piUnlock(STD_IO_LCD_BUFFER_KEY);
                //FLAGS
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_PASSWORD_IS_READY;
                piUnlock(SYSTEM_KEY);
        }else{
                piLock(STD_IO_LCD_BUFFER_KEY);
                lcdClear(p_sistema->lcdId);
                lcdPosition(p_sistema->lcdId, 0, 0);
                lcdPrintf(p_sistema->lcdId, "NOT MATCHING");
                lcdPosition(p_sistema->lcdId, 0, 1);
                lcdPrintf(p_sistema->lcdId, "TRY AGAIN");
                delay(1000);
                lcdClear(p_sistema->lcdId);
                lcdPosition(p_sistema->lcdId, 0, 0);
                lcdPrintf(p_sistema->lcdId, "INTRODUCE");
                lcdPosition(p_sistema->lcdId, 0, 1);
                lcdPrintf(p_sistema->lcdId, "NEW");
                piUnlock(STD_IO_LCD_BUFFER_KEY);
                digitosGuardadosPassword = 0;
                tempNewPassword = 0;
                tempNewPasswordRepeat = 0;
        }
        }

        if(digitosGuardadosPassword == 4){
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "REPEAT");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "PLEASE");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        digitosGuardadosPassword++;
        }

        p_sistema->password.tempNewPassword = tempNewPassword;
        p_sistema->password.tempNewPasswordRepeat = tempNewPasswordRepeat;
        p_sistema->password.digitosGuardadosPassword = digitosGuardadosPassword;
}
```

8.3.3 SetNewPassword()

En esta función se actualiza p_sistema->password.number al nuevo valor de contraseña. Además de desactivar ciertos flags.

```
void SetNewPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->password.number = p_sistema->password.tempNewPassword;
        p_sistema->password.tempNewPassword = 0;
        p_sistema->password.tempNewPasswordRepeat = 0;
        p_sistema->password.digitosGuardadosPassword = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
```

```
        piUnlock(SYSTEM_KEY);
}
```

9 Mejora 7: Cambios estéticos en SET_TIME, SET_DATE Y SET_ALARMA

9.1 Objetivos de la mejora

El objetivo de esta mejora es facilitar la usabilidad del corewatch así como un importante factor estético. Queremos imprimir por el LCD un template o plantilla en que introducir la hora (__:__) y la fecha (__/__/____) y se vayan modificando los espacios en blanco o "barras bajas" por los números que el usuario vaya introduciendo. De esta manera pretendemos conseguir que el usuario sea capaz de visualizar de una forma más clara la fecha y la hora que está introduciendo.

9.2 Descripción del subsistema Software

Consiste en ir imprimiendo por pantalla estéticamente el número de fecha u hora que se va introduciendo para que el usuario sepa qué numero introduce. Esto lo hacemos con lcdPutChar y el contador de digitosGuardados.

Este código para horas

```
if(digitosGuardados<3){
        lcdPosition(p_sistema->lcdId, digitosGuardados-1, 1);
         lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }else{
        lcdPosition(p_sistema->lcdId, digitosGuardados, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }
```

Y este para fecha

```
if(digitosGuardadosDate<3){
        lcdPosition(p_sistema->lcdId, digitosGuardadosDate-1, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
        }else if(digitosGuardadosDate >= 3 && digitosGuardadosDate < 5){
        lcdPosition(p_sistema->lcdId, digitosGuardadosDate, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
        }else{
        lcdPosition(p_sistema->lcdId, digitosGuardadosDate+1, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
        }
```

10 Mejora 8: Configuración Inicial a la hora actual

Objetivos de la mejora

Queremos que en vez de utilizar una fecha y hora por defecto al inicializar el sistema o al hacer reset, se tome la fecha y hora actuales.

Descripción del subsistema Software

Ahora, en vez de establecer una hora y fecha por defecto al inicializar el reloj, tomaremos la hora y fecha actual haciendo uso del módulo time.h y de la struct tm. Al realizar una instanciación de esa struct en ResetReloj(), podemos acceder a la hora local y establecerla.

Descripción de subrutinas

ResetReloj(): Instanciamos la struct tm con nombre timeinfo y una variable time_t rawtime. Obtenemos en timeinfo una struct de la que podemos sacar todos los parámetros necesarios para la inicialización, y se los pasamos a TipoHora y TipoCalendario.

```
void ResetReloj(TipoReloj *p_reloj){
	//Obtenemos fecha y hora actual
	time_t rawtime;
	struct tm * timeinfo;
	time(&rawtime);
	timeinfo = localtime(&rawtime);
	int horaActual = timeinfo->tm_hour;
	int minActual = timeinfo->tm_min;
	int segActual = timeinfo->tm_sec;
	int diaActual = timeinfo->tm_mday;
	int mesActual = timeinfo->tm_mon +1; //REFERENCIADO DE 0 A 11
	int yearActual = timeinfo->tm_year + 1900; //el year obtenido así es referenciado
a 1900
	int weekDay = timeinfo->tm_wday;
	printf("Weekday: %d", weekDay);
	TipoCalendario calendario = {diaActual, mesActual, yearActual, weekDay};
	TipoHora hora = {horaActual, minActual, segActual, DEFAULT_TIME_FORMAT};
	p_reloj->calendario = calendario;
	p_reloj->hora = hora;
	p_reloj->timestamp = 0; //default timestamp
	piLock (RELOJ_KEY);
	g_relojSharedVars.flags = 0;
	piUnlock (RELOJ_KEY);
}
```

11 Mejora 9: Mostrar el día de la semana

11.1 Objetivos de la mejora

Esta mejora tiene por objetivo informar al usuario del día de la semana imprimiéndolo por pantalla junto con la fecha. Para implementar esta funcionalidad haremos uso de la librería <time.h>

11.2 Descripción del subsistema Software

Gracias al módulo time que hemos usado antes, podemos obtener el día de la semana también. Así, dentro de TipoCalendario en reloj.h vamos a introducir un nuevo parámetro weekDay que represente el día de la semana. Al actualizar la fecha este número se hace módulo 7 y se incrementa. Al cambiar la fecha se recalcula, calculando primero los segundos desde el epoch 1-1-1970 y haciendo "time = localtime(&segundosDesdeEpoch)" para que calcule la struct time y poder obtener el día de la semana. Para imprimirlo, declaramos en CoreWatch.c un array char diassemanas[7], en que introducimos la letra del día de la semana y antes de imprimir la fecha, con sprintf(), añadimos este char al principio de la cadena.

11.2.1 Descripción de subrutinas

ActualizaFecha() en reloj.c, hace módulo 6 al día de la semana y le suma 1, ya que el día de la semana va representado de 0 a 6 y si se ha llegado al último queremos empezar de nuevo.

```
void ActualizaFecha(TipoCalendario *p_fecha){
	// Irrelevante todo este código para la mejora
	int weekDay = p_fecha->weekDay%6;
	weekDay++;
	p_fecha->weekDay = weekDay;
	p_fecha->MM = mesActual;
	p_fecha->dd = diaActual;
}
```

12 Principales problemas encontrados

En cuanto a los principales problemas encontrados podemos destacar los siguientes.

A la hora de programar la mejora que permitía cambiar el formato AM-PM, en las primeras versiones del proyecto, nos encontramos con un bug. Al seguir el guión al pie de la letra y programar ActualizaFecha() como se proponía en este, al utilizar el formato 12 horas no funcionaba correctamente el cambio de fecha. Este bug se ha resuelto ahora, ya que tratamos todas las horas en formato 24 de forma interna y las mostramos en formato 24 o 12 dependiendo de lo que decida el usuario al pulsar la letra 'C'.

Cuando estábamos configurando las funciones para imprimir por el LCD el día de la semana correspondiente hacíamos uso de otras funciones que en términos generales calculan los segundos desde el "Epoch" hasta una fecha determinada. Resulta que para realizar dichas operaciones hay que utilizar una gran precisión de bits ya que estamos tomando números muy grandes. Por ello, la mejor alternativa fue usar variables de enteros de 64 bits para almacenar dichos segundos y no perder precisión. Sin embargo, las variables de tipo time_t no tienen siquiera esa resolución, por lo que a partir de cierta fecha, cuando se excede el número de precisión de estas variables, deja de funcionar correctamente.

13 Manual de usuario

A: cambiar hora de alarma

B: exit

C: cambiar formato (estando en STAND_BY)ç

D: cambiar fecha

E: cambiar hora

F: cambiar contraseña

14 Bibliografía

La bibliografía para las mejoras únicamente incluye la siguiente web, consultada para saber cómo se utiliza la struct incluida en time.h para obtener el día de la semana.

https://linux.die.net/man/3/ctime

15 ANEXO I: Código del programa del proyecto final

```
/*
 * reloj.h
 *
 *  Created on: 13 feb. 2022
 *      Author: pi
 */

#ifndef RELOJ_H_
#define RELOJ_H_
// INCLUDES
#include "systemConfig.h"
#include "util.h"
// DEFINES Y ENUMS
enum FSM_ESTADOS_RELOJ{
	WAIT_TIC
};
// FLAGS FSM
#define FLAG_ACTUALIZA_RELOJ 0x01 //01
#define FLAG_TIME_ACTUALIZADO 0x02 //10
// DECLARACION ESTRUCTURAS
typedef  struct {int dd;int MM;int  yyyy; int weekDay;} TipoCalendario;
typedef  struct {int hh;int mm;int ss;int  formato;} TipoHora;
typedef  struct {int  timestamp;TipoHora  hora;TipoCalendario  calendario;tmr_t* tmrTic;} TipoReloj;
typedef  struct {int  flags;} TipoRelojShared;
// DECLARACION VARIABLES
//asociadas a TipoCalendario
#define MIN_DAY 1
#define MAX_DAY 31
#define MIN_MONTH 1
#define MAX_MONTH 12
#define MIN_YEAR 1970
//asociadas a TipoReloj
#define MIN_HOUR_12 1
#define MIN_HOUR_24 0
#define MAX_HOUR_12 12
#define MAX_HOUR_24 23
#define MAX_MIN 59
//array de tabla de transiciones de fsm
extern  fsm_trans_t  g_fsmTransReloj[];
//array de numero de dias de cada mes para años bisiestos y no bisiestos
extern const int DIAS_MESES[2][MAX_MONTH];
// DEFINICION VARIABLES
#define DEFAULT_DAY 28
#define DEFAULT_MONTH 2
#define DEFAULT_YEAR 2020
#define DEFAULT_HOUR 23
#define DEFAULT_MIN 59
#define DEFAULT_SEC 58
#define DEFAULT_TIME_FORMAT 12
#define PRECISION_RELOJ_MS 1000
// FUNCIONES DE INICIALIZACION DE LAS VARIABLES
```

```
int ConfiguraInicializaReloj (TipoReloj *p_reloj);
void ResetReloj(TipoReloj *p_reloj);
// FUNCIONES PROPIAS
void ActualizaFecha(TipoCalendario *p_fecha);
void ActualizaHora(TipoHora *p_hora);
int CalculaDiasMes(int month, int year);
int EsBisiesto(int year);
TipoRelojShared GetRelojSharedVar();
int SetFormato(int nuevoFormato, TipoHora *p_hora);
int SetHora(int nuevaHora, TipoHora *p_hora);
int SetFormat(int formato, TipoHora *p_hora);
void SetRelojSharedVar(TipoRelojShared value);
int SetDia(int dia, TipoCalendario *p_fecha);
int SetMes(int mes, TipoCalendario *p_fecha);
int SetYear(int year, TipoCalendario *p_fecha);
int CalculaDiaSemana(int dia, int mes, int year);
int64_t CalculaSegundosEpoch(int dia, int mes, int year);
int CalculaBisiestosDesdeEpochHasta(int year);
// FUNCIONES DE ENTRADA DE LA MAQUINA DE ESTADOS
int CompruebaTic(fsm_t *p_this);
// FUNCIONES DE SALIDA DE LA MAQUINA DE ESTADOS
void ActualizaReloj(fsm_t *p_this);
// SUBRUTINAS DE ATENCION A LAS INTERRUPCIONES
void tmr_actualiza_reloj_isr(union sigval value);
// FUNCIONES LIGADAS A THREADS ADICIONALES
#endif /* RELOJ_H_ */
```

```
/*
 * reloj.c
 *
 *  Created on: 13 feb. 2022
 *      Author: pi
 */

#include "reloj.h"
#include <time.h>

// {EstadoIni, FuncCompruebaCondicion, EstadoSig, FuncAccionesSiTransicion}
fsm_trans_t fsmTransReloj[] = {{WAIT_TIC, CompruebaTic, WAIT_TIC, ActualizaReloj},{-1, NULL, -1, NULL}};
static int DIAS_MESES[2][MAX_MONTH] = {{31,28,31,30,31,30,31,31,30,31,30,31},{31,29,31,30,31,30,31,31,30,31,30,31}};
static TipoRelojShared g_relojSharedVars;

void ResetReloj(TipoReloj *p_reloj){
	//Obtenemos fecha y hora actual
	time_t rawtime;
	struct tm * timeinfo;
	time(&rawtime);
	timeinfo = localtime(&rawtime);
	int horaActual = timeinfo->tm_hour;
	int minActual = timeinfo->tm_min;
	int segActual = timeinfo->tm_sec;
	int diaActual = timeinfo->tm_mday;
	int mesActual = timeinfo->tm_mon +1; //REFERENCIADO DE 0 A 11
a 1900	int yearActual = timeinfo->tm_year + 1900; //el year obtenido así es referenciado
	int weekDay = timeinfo->tm_wday;
	printf("Weekday: %d", weekDay);
	TipoCalendario calendario = {diaActual, mesActual, yearActual, weekDay};
	TipoHora hora = {horaActual, minActual, segActual, DEFAULT_TIME_FORMAT};
	p_reloj->calendario = calendario;
	p_reloj->hora = hora;
	p_reloj->timestamp = 0; //default timestamp
	piLock (RELOJ_KEY);
	g_relojSharedVars.flags = 0;
	piUnlock (RELOJ_KEY);
}

int ConfiguraInicializaReloj(TipoReloj *p_reloj){
	ResetReloj(p_reloj);
	tmr_t* temp1 = tmr_new(tmr_actualiza_reloj_isr);
	p_reloj->tmrTic = temp1;
	tmr_startms_periodic(temp1, PRECISION_RELOJ_MS);
	return 0;
}

int CompruebaTic(fsm_t *p_this){
	int result;
	piLock(RELOJ_KEY);
	result = (g_relojSharedVars.flags & FLAG_ACTUALIZA_RELOJ);
	piUnlock(RELOJ_KEY);
	return result;
}
```

```
void ActualizaReloj(fsm_t *p_this){
	TipoReloj *p_reloj = (TipoReloj*)(p_this->user_data);
	p_reloj->timestamp += 1;
	ActualizaHora(&p_reloj->hora);
	//si la hora es 00:00:00, hay que actualizar la fecha
	if(p_reloj->hora.hh == 0 && p_reloj->hora.mm == 0 && p_reloj->hora.ss == 0){
	ActualizaFecha(&p_reloj->calendario);
	}
	//00 01 10 11
	//ENTRAS CON flags = 01
	//SALES CON flags = 10
	piLock(RELOJ_KEY);
	g_relojSharedVars.flags = FLAG_TIME_ACTUALIZADO;
	piUnlock(RELOJ_KEY);


#if VERSION == 1
	int hora = p_reloj->hora.hh;
	int min = p_reloj->hora.mm;
	int seg = p_reloj->hora.ss;
	int dia = p_reloj->calendario.dd;
	int mes = p_reloj->calendario.MM;
	int year = p_reloj->calendario.yyyy;
	piLock(STD_IO_LCD_BUFFER_KEY);
	printf("%d:%d:%d", hora, min, seg);
	printf(" del %d/%d/%d\n", dia, mes, year);
	fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
#endif
}

void  tmr_actualiza_reloj_isr (union  sigval  value){
	piLock(RELOJ_KEY);
	g_relojSharedVars.flags = FLAG_ACTUALIZA_RELOJ;
	piUnlock(RELOJ_KEY);
}

void ActualizaFecha(TipoCalendario *p_fecha){
	int diasMes = CalculaDiasMes(p_fecha->MM, p_fecha->yyyy);
	int diaActual = p_fecha->dd;
	int mesActual = p_fecha->MM;
	if(diaActual >= diasMes){
	diaActual = 1;
	if(mesActual >= 12){
		mesActual = 1;
		p_fecha->yyyy++;
	}else{
		mesActual++;
	}
	}else{
	diaActual++;
	}
	int weekDay = p_fecha->weekDay%6;
	weekDay++;
	p_fecha->weekDay = weekDay;
```

```
	p_fecha->MM = mesActual;
	p_fecha->dd = diaActual;
}

void ActualizaHora(TipoHora *p_hora){
	//actualizar segundero
	p_hora->ss = (p_hora->ss+1)%60;
	//actualizar minutero
	if(p_hora->ss == 0) {
	p_hora->mm = (p_hora->mm+1)%60;
	if(p_hora->mm == 0){
		p_hora->hh = p_hora->hh+1;
		if(p_hora->hh > MAX_HOUR_24){
		p_hora->hh = MIN_HOUR_24;
		}
	}
	}

}

int CalculaDiasMes(int month, int year){
	int bisiesto = EsBisiesto(year);
	return DIAS_MESES[bisiesto][month-1];
}

int EsBisiesto(int year){
	int bisiesto = 0;
	if(((year%4 == 0) && (year%100!=0)) || (year%400 == 0)){
	bisiesto = 1;
	}
	return bisiesto;
}

int SetHora(int horaInt ,TipoHora *p_hora){
	if(horaInt < 0){ return 1;}
	int num_digitos = 0;
	int aux = horaInt;
	while(aux!=0){
	aux /= 10;
	num_digitos++;
	}
	if(num_digitos<1 || num_digitos>4){
	return 2;
	}
	//obtenemos los valores de hora y minutos introducidos
	int hora = horaInt/100;
	int minutos = horaInt%100;
	//si los mins exceden el maximo, se asigna el valor max_min
	if(minutos>MAX_MIN) minutos = MAX_MIN;
	if(hora>MAX_HOUR_24) hora = MAX_HOUR_24;
	//Asignación de valores
	p_hora->hh = hora;
	p_hora->mm = minutos;
	p_hora->ss = 0;
	return 0;
```

```
}

int SetDia(int dia, TipoCalendario *p_calendario){
	//El dia introducido no puede ser 0 ni inferior
	if(dia <= 0){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Introduzca un día válido");
		fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 1;
	}
	//Comprobamos que se han introducido 1 o 2 digitos
	int num_digitos = 0;
	int aux = dia;
	while(aux!=0){
	aux /= 10;
	num_digitos++;
	}
	if(num_digitos < 1|| num_digitos>2){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Introduzca un dia de maximo 2 digitos");
		fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 2;
	}
	int mesActual = p_calendario->MM;
	int yearActual = p_calendario->yyyy;
	int maxDiaMes = CalculaDiasMes(mesActual, yearActual);
	if(dia > maxDiaMes){
	dia = maxDiaMes;
	}
	p_calendario->dd = dia;
	return 0;

}

int SetMes(int mes, TipoCalendario *p_calendario){
	//El día introducido no puede ser 0 ni inferior
	if(mes <= 0){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Introduzca un mes valido");
		fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 1;
	}
	//Comprobamos que se han introducido 1 o 2 dígitos
	int num_digitos = 0;
	int aux = mes;
	while(aux!=0){
	aux /= 10;
	num_digitos++;
	}
	if(num_digitos < 1|| num_digitos>2){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Introduzca un mes de maximo 2 digitos");
```

```
                fflush(stdout);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        return 2;
        }
        if(mes > MAX_MONTH){
        mes = MAX_MONTH;
        }
        p_calendario->MM = mes;
        return 0;
}

int SetYear(int year, TipoCalendario *p_calendario){
        //El día introducido no puede ser 0 ni inferior
        if(year <= 0){
        piLock(STD_IO_LCD_BUFFER_KEY);
                printf("Introduzca un year d.C.");
                fflush(stdout);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        return 1;
        }
        if(year < MIN_YEAR){
        year = MIN_YEAR;
        }
        p_calendario->yyyy = year;
        return 0;
}

int SetFormat(int formato, TipoHora *p_hora){
        if(formato != 12 && formato != 24){
        formato = 24;
        }
        p_hora->formato = formato;
        return 0;
}

int CalculaDiaSemana(int dia, int mes, int year ){
        // int64_t res = CalculaSegundosEpoch(dia, mes, year);
        int res = CalculaSegundosEpoch(dia, mes, year);
        time_t segs = (time_t)(res);
        struct tm * timeinfo;
        timeinfo = localtime(&segs);
        printf("%d", res);
        printf("%s", asctime(timeinfo));
        int weekDay = timeinfo->tm_wday;
        return weekDay;
}

int64_t CalculaSegundosEpoch(int dia, int mes, int year){
        int esbisiesto = EsBisiesto(year);
        int64_t bisiestos = CalculaBisiestosDesdeEpochHasta(year);
        int64_t yearToDays = bisiestos*366 + (year-1970-bisiestos)*365;
        int64_t monthToDays =0;
        int i;
        for(i=0; i<mes-1; i++){
        monthToDays += DIAS_MESES[esbisiesto][i];
```

```
        }
        int64_t totalDiasEnSeg = 0;
        totalDiasEnSeg = ((dia-1)+monthToDays+yearToDays)*24*3600;
        return totalDiasEnSeg;
}

int CalculaBisiestosDesdeEpochHasta(int year){
        int i;
        int cont;
        for(i = 1970; i<year; i++){
        cont += EsBisiesto(i);
        }
        return cont;
}

TipoRelojShared GetRelojSharedVar(){
        piLock(RELOJ_KEY);
        TipoRelojShared copia = g_relojSharedVars;
        piUnlock(RELOJ_KEY);
        return copia;
}

void SetRelojSharedVar(TipoRelojShared value){
        piLock(RELOJ_KEY);
        g_relojSharedVars = value;
        piUnlock(RELOJ_KEY);
}
```

```
#ifndef COREWATCH_H_
#define COREWATCH_H_

// INCLUDES
// Propios:
#include "systemConfig.h"
y entorno, setup de periféricos y otros.
#include "reloj.h"
#include "teclado_TL04.h"
// DEFINES Y ENUMS

enum FSM_ESTADOS_SISTEMA{

      START, STAND_BY, SET_TIME, SET_DATE, SET_ALARMA, ALARMA, SET_PASSWORD,
PASSWORD

};
enum FSM_DETECCION_COMANDOS{
    WAIT_COMMAND
};

// FLAGS FSM DEL SISTEMA CORE WATCH
#define FLAG_SETUP_DONE 0x01 //0000 0001
#define FLAG_TIME_ACTUALIZADO 0x02 //0000 0010
#define FLAG_RESET 0x04 //0000 0100
#define FLAG_SET_CANCEL_NEW_TIME 0x08 //0000 1000
#define FLAG_NEW_TIME_IS_READY 0x10 //0001 0000
#define FLAG_DIGITO_PULSADO 0x20 //0010 0000
#define FLAG_SET_CANCEL_NEW_DATE 0x40 //0100 0000
#define FLAG_NEW_DATE_IS_READY 0x80 //1000 0000
#define FLAG_SWITCH_FORMAT 0x100 //0001 0000 0000
#define FLAG_PASSWORD_TIME 0x200 //0010 0000 0000
#define FLAG_PASSWORD_DATE 0x400
#define FLAG_SET_ALARMA 0x800
/*
 * coreWatch.h
 *
 */

#define FLAG_NEW_ALARMA_IS_READY 0x1000
#define FLAG_TIMEOUT_ALARMA 0x2000
#define FLAG_TIMEOUT_PASSWORD 0x4000
#define FLAG_PASSWORD_ALARMA 0x8000
#define FLAG_SET_PASSWORD 0x10000
#define FLAG_PASSWORD_PASSWORD 0x20000
#define FLAG_NEW_PASSWORD_IS_READY 0x40000

// DECLARACIÓN ESTRUCTURAS
typedef  struct {
    TipoHora hora;
    tmr_t* timer;
    int  alarmaActivada;
    int tempAlarma;
    int digitosGuardadosAlarma;
}TipoAlarma;

typedef struct{
```

```
    int  number;
    tmr_t* tmrPassword;
    int  tempPassword;
    int  digitosGuardadosPassword;
    int tempNewPassword;
    int tempNewPasswordRepeat;
}TipoPassword;

typedef  struct {
    TipoReloj  reloj;
    TipoTeclado  teclado;
    int  lcdId;
    int  tempTime;
    int  digitosGuardados;
    int  tempDate;
    int  digitosGuardadosDate;
    int  digitoPulsado;
    TipoPassword password;
    TipoAlarma alarma;
} TipoCoreWatch;

// DECLARACIÓN VARIABLES
#define TECLA_ALARMA 'A'
//#define TECLA_RESET 'F'
#define TECLA_PASSWORD 'F'
#define TECLA_SET_CANCEL_TIME 'E'
#define TECLA_SET_CANCEL_DATE 'D'
#define TECLA_EXIT 'B'
#define TECLA_SWITCH_FORMAT 'C'
#define TIMEOUT_ALARMA 5000 //5 segundos
#define TIMEOUT_PASSWORD 8000 //8 segundos

// DEFINICIÓN VARIABLES
extern fsm_trans_t g_fsmTransCoreWatch[];

//-------------------------------------------------------
// FUNCIONES DE INICIALIZACION DE LAS VARIABLES
//-------------------------------------------------------

//-------------------------------------------------------
// FUNCIONES PROPIAS
//-------------------------------------------------------
void DelayUntil(unsigned int next);
int ConfiguraInicializaSistema(TipoCoreWatch *p_sistema);
int EsNumero(char value);
int min(int a, int b);
int max(int a, int b);
//-------------------------------------------------------
// FUNCIONES DE ENTRADA O DE TRANSICION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------
int CompruebaDigitoPulsado(fsm_t* p_this);
int CompruebaNewTimeIsReady(fsm_t* p_this);
int CompruebaNewDateIsReady(fsm_t* p_this);
int CompruebaReset(fsm_t* p_this);
int CompruebaSetCancelNewTime(fsm_t* p_this);
```

```
int CompruebaSetCancelNewDate(fsm_t* p_this);
int CompruebaSwitchFormat(fsm_t* p_this);
int CompruebaSetupDone(fsm_t* p_this);
int CompruebaTeclaPulsada(fsm_t* p_this);
int CompruebaTimeActualizado(fsm_t* p_this);
int CompruebaPasswordTime(fsm_t* p_this);
int CompruebaPasswordDate(fsm_t* p_this);
int CompruebaPasswordAlarma(fsm_t* p_this);
int CompruebaHoraAlarma(fsm_t* p_this);
int CompruebaTimeoutAlarma(fsm_t* p_this);
int CompruebaSetCancelNewAlarma(fsm_t* p_this);
int CompruebaNewAlarmaIsReady(fsm_t* p_this);
int CompruebaTimeoutPassword(fsm_t* p_this);
int CompruebaSetPassword(fsm_t* p_this);
int CompruebaPasswordPassword(fsm_t* p_this);
int CompruebaNewPasswordIsReady(fsm_t* p_this);
//-------------------------------------------------------
// FUNCIONES DE SALIDA O DE ACCION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------
void CancelSetNewTime(fsm_t* p_this);
void PrepareSetNewTime(fsm_t* p_this);
void CancelSetNewDate(fsm_t* p_this);
void PrepareSetNewDate(fsm_t* p_this);
void ProcesaDigitoTime(fsm_t* p_this);
void ProcesaDigitoDate(fsm_t* p_this);
void ProcesaTeclaPulsada(fsm_t* p_this);
void Reset(fsm_t* p_this);
void SetNewTime(fsm_t* p_this);
void SetNewDate(fsm_t* p_this);
void SwitchFormat(fsm_t* p_this);
void ShowTime(fsm_t* p_this);
void Start(fsm_t* p_this);
void EnterPassword(fsm_t* p_this);
void PrepareEnterPassword(fsm_t* p_this);
void CancelEnterPassword(fsm_t* p_this);
void PrepareSetAlarma(fsm_t* p_this);
void SuenaAlarma(fsm_t* p_this);
void ExitAlarma(fsm_t* p_this);
void CancelSetNewAlarma(fsm_t* p_this);
void SetNewAlarma(fsm_t* p_this);
void ProcesaDigitoAlarma(fsm_t* p_this);
void PrepareSetNewPassword(fsm_t* p_this);
void ProcesaDigitoNewPassword(fsm_t* p_this);
void EnterNewPassword(fsm_t* p_this);
void SetNewPassword(fsm_t* p_this);
void CancelSetNewPassword(fsm_t* p_this);
//-------------------------------------------------------
// SUBRUTINAS DE ATENCION A LAS INTERRUPCIONES
//-------------------------------------------------------
void tmr_timeout_alarma(union sigval value);
void tmr_timeout_password(union sigval value);
//-------------------------------------------------------
// FUNCIONES LIGADAS A THREADS ADICIONALES
//-------------------------------------------------------
PI_THREAD(ThreadExploraTecladoPC);
```

```
#endif /* EAGENDA_H */
```

```
#include "coreWatch.h"
#include <stdio.h>
#include <string.h>
//-------------------------------------------------------
// VARIABLES GLOBALES
//-------------------------------------------------------
static TipoCoreWatch g_coreWatch;
static int defaultPassword = 1234;
static int defaultHoraAlarma = 0;
static int defaultAlarmaActivada = 0;
static int g_flagsCoreWatch;
fsm_trans_t g_fsmTransCoreWatch[] = {
            //START
            {START, CompruebaSetupDone, STAND_BY, Start},
            //STAND_BY
            //{STAND_BY, CompruebaReset, STAND_BY, Reset},
            {STAND_BY,CompruebaSetCancelNewTime, PASSWORD, PrepareEnterPassword},
            {STAND_BY,CompruebaSetCancelNewDate, PASSWORD, PrepareEnterPassword},
            {STAND_BY,CompruebaSetCancelNewAlarma, PASSWORD, PrepareEnterPassword},
            {STAND_BY,CompruebaSetPassword, PASSWORD, PrepareEnterPassword},
            {STAND_BY,CompruebaHoraAlarma, ALARMA, SuenaAlarma},
            {STAND_BY,CompruebaSwitchFormat, STAND_BY, SwitchFormat},
            {STAND_BY, CompruebaTimeActualizado, STAND_BY, ShowTime},
            //PASSWORD
            {PASSWORD, CompruebaDigitoPulsado, PASSWORD, EnterPassword},
            {PASSWORD, CompruebaPasswordTime, SET_TIME, PrepareSetNewTime},
            {PASSWORD, CompruebaPasswordDate, SET_DATE, PrepareSetNewDate},
            {PASSWORD, CompruebaPasswordAlarma, SET_ALARMA, PrepareSetAlarma},
            {PASSWORD, CompruebaPasswordPassword, SET_PASSWORD, PrepareSetNewPassword},
            {PASSWORD, CompruebaTimeoutPassword, STAND_BY, CancelEnterPassword},
            //SET_PASSWORD
            {SET_PASSWORD, CompruebaSetPassword, STAND_BY, CancelSetNewPassword},
            {SET_PASSWORD, CompruebaNewPasswordIsReady, STAND_BY, SetNewPassword},
            {SET_PASSWORD, CompruebaDigitoPulsado, SET_PASSWORD, EnterNewPassword},
            //SET_TIME
            {SET_TIME, CompruebaSetCancelNewTime, STAND_BY, CancelSetNewTime},
            {SET_TIME, CompruebaNewTimeIsReady, STAND_BY, SetNewTime},
            {SET_TIME, CompruebaDigitoPulsado, SET_TIME, ProcesaDigitoTime},
            //SET_DATE
            {SET_DATE, CompruebaNewDateIsReady, STAND_BY, SetNewDate},
            {SET_DATE, CompruebaSetCancelNewDate, STAND_BY, CancelSetNewDate},
            {SET_DATE, CompruebaDigitoPulsado, SET_DATE, ProcesaDigitoDate},
            //SET_ALARMA
            {SET_ALARMA, CompruebaSetCancelNewAlarma, STAND_BY, CancelSetNewAlarma},
            {SET_ALARMA, CompruebaNewAlarmaIsReady, STAND_BY, SetNewAlarma},
            {SET_ALARMA, CompruebaDigitoPulsado, SET_ALARMA, ProcesaDigitoAlarma},
            //ALARMA
            {ALARMA, CompruebaTimeoutAlarma, STAND_BY, ExitAlarma},
            {-1, NULL, -1, NULL}};
fsm_trans_t g_fsmTransDeteccionComandos[] = {
        {WAIT_COMMAND,CompruebaTeclaPulsada ,WAIT_COMMAND, ProcesaTeclaPulsada},
        {-1, NULL, -1, NULL}};
//am = 0 pm = 1
const int hora_ampm[2][24] = {
```

```
        {12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
};
const char diasemana[7] = {'D', 'L', 'M', 'X', 'J', 'V', 'S'};

int rows = 2;
int cols = 12;
int bits = 8;
int rs = GPIO_LCD_RS;
int strb = GPIO_LCD_EN;
int d0 = GPIO_LCD_D0;
int d1 = GPIO_LCD_D1;
int d2 = GPIO_LCD_D2;
int d3 = GPIO_LCD_D3;
int d4 = GPIO_LCD_D4;
int d5 = GPIO_LCD_D5;
int d6 = GPIO_LCD_D6;
int d7 = GPIO_LCD_D7;



static int arrayFilas[4] = {GPIO_KEYBOARD_ROW_1, GPIO_KEYBOARD_ROW_2,
GPIO_KEYBOARD_ROW_3, GPIO_KEYBOARD_ROW_4};

static int arrayColumnas[4] = {GPIO_KEYBOARD_COL_1, GPIO_KEYBOARD_COL_2,
GPIO_KEYBOARD_COL_3, GPIO_KEYBOARD_COL_4};
//-------------------------------------------------------
// FUNCIONES PROPIAS
//-------------------------------------------------------
// Wait until next_activation (absolute time)
// Necesita de la funci?n "delay" de WiringPi.
void DelayUntil(unsigned int next) {
       unsigned int now = millis();
       if (next > now) {
       delay(next - now);
       }
}

int min(int a, int b){
       return (a > b) ? b : a;
}

int max(int a, int b){
       return (a > b) ? a : b;
}

int ConfiguraInicializaSistema(TipoCoreWatch *p_sistema){
       piLock(SYSTEM_KEY);
       g_flagsCoreWatch = 0;
       piUnlock(SYSTEM_KEY);
       p_sistema->tempTime = 0;
       p_sistema->digitosGuardados = 0;
       p_sistema->tempDate = 0;
       p_sistema->digitosGuardadosDate = 0;
       p_sistema->password.tempPassword = 0;
       p_sistema->password.digitosGuardadosPassword = 0;
```

```
        p_sistema->password.number = defaultPassword;
        p_sistema->alarma.alarmaActivada = defaultAlarmaActivada;
        SetHora(defaultHoraAlarma, &p_sistema->alarma.hora);
        int configReloj = ConfiguraInicializaReloj(&p_sistema->reloj);
        if(configReloj!=0){return 1;}
        memcpy(p_sistema->teclado.filas, arrayFilas, sizeof(arrayFilas));
        memcpy(p_sistema->teclado.columnas, arrayColumnas, sizeof(arrayColumnas));
        int setupgpio = wiringPiSetupGpio();
        if(setupgpio != 0) return 1;
        ConfiguraInicializaTeclado(&p_sistema->teclado);
        #if VERSION == 4
        p_sistema->lcdId = lcdInit(rows, cols, bits, rs, strb, d0, d1, d2, d3, d4, d5, d6,
d7);
        if(p_sistema->lcdId == -1) return 1;
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdDisplay(p_sistema->lcdId, 1);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0,0);
        lcdPrintf(p_sistema->lcdId, "    Hii!" );
        lcdPosition(p_sistema->lcdId, 0,1);
        lcdPrintf(p_sistema->lcdId, "  /(^o^)|" );
        delay(500);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0,0);
        lcdPrintf(p_sistema->lcdId, "    Hii!" );
        lcdPosition(p_sistema->lcdId, 0,1);
        lcdPrintf(p_sistema->lcdId, "  |(^O^)/" );
        delay(500);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0,0);
        lcdPrintf(p_sistema->lcdId, "    Hii!" );
        lcdPosition(p_sistema->lcdId, 0,1);
        lcdPrintf(p_sistema->lcdId, "  /(^o^)|" );
        delay(500);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0,0);
        lcdPrintf(p_sistema->lcdId, "    Hii!" );
        lcdPosition(p_sistema->lcdId, 0,1);
        lcdPrintf(p_sistema->lcdId, "  |(^O^)/" );
        delay(500);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = FLAG_SETUP_DONE;
        piUnlock(SYSTEM_KEY);
        return 0;
}

//FUNCIONES COMPROBACION

int EsNumero(char value){
        int valorAscii = (int)value;
        if(valorAscii>=48 && valorAscii<=57){
        return 1;
        }
```

```
	return 0;
}

int CompruebaDigitoPulsado(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_DIGITO_PULSADO;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaNewTimeIsReady(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_NEW_TIME_IS_READY;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaReset(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_RESET;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaSetCancelNewTime(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_SET_CANCEL_NEW_TIME;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaSetupDone(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_SETUP_DONE;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaTimeActualizado(fsm_t* p_this){
	TipoRelojShared sharedvars = GetRelojSharedVar();
	int shv = sharedvars.flags;
	int res = shv & FLAG_TIME_ACTUALIZADO;
	if(res!=0){return 1;}
	return 0;
}
```

```
int CompruebaSetCancelNewDate(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_SET_CANCEL_NEW_DATE;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaNewDateIsReady(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_NEW_DATE_IS_READY;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaSwitchFormat(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_SWITCH_FORMAT;
	piUnlock(SYSTEM_KEY);
	if(res!=0){return 1;}
	return 0;
}

int CompruebaPasswordTime(fsm_t* this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_PASSWORD_TIME;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}

int CompruebaPasswordDate(fsm_t* this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_PASSWORD_DATE;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}

int CompruebaPasswordAlarma(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_PASSWORD_ALARMA;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}
```

```
int CompruebaTeclaPulsada(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	TipoTecladoShared ts = GetTecladoSharedVar();
	res = ts.flags & FLAG_TECLA_PULSADA;
	piUnlock(SYSTEM_KEY);
	return res;
}

int CompruebaHoraAlarma(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	int res=0;
	piLock(RELOJ_KEY);
	if(p_sistema->alarma.alarmaActivada==1){
	int horaalarma = p_sistema->alarma.hora.hh;
	int minalarma = p_sistema->alarma.hora.hh;
	int horasis = p_sistema->reloj.hora.hh;
	int minsis = p_sistema->reloj.hora.hh;
	int segsis = p_sistema->reloj.hora.ss;
	if(horaalarma == horasis && minsis == minalarma && segsis == 0){
		res = 1;
	}
	}
	piUnlock(RELOJ_KEY);
	return res;
}

int CompruebaTimeoutAlarma(fsm_t* this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_TIMEOUT_ALARMA;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}

int CompruebaSetCancelNewAlarma(fsm_t* this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_SET_ALARMA;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}

int CompruebaNewAlarmaIsReady(fsm_t* this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_NEW_ALARMA_IS_READY;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}

int CompruebaTimeoutPassword(fsm_t* this){
```

```
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_TIMEOUT_PASSWORD;
	piUnlock(SYSTEM_KEY);
	if(res!= 0)return 1;
	return 0;
}

int CompruebaSetPassword(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_SET_PASSWORD;
	piUnlock(SYSTEM_KEY);
	if(res!=0)return 1;
	return 0;
}

int CompruebaPasswordPassword(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_PASSWORD_PASSWORD;
	piUnlock(SYSTEM_KEY);
	if(res!=0)return 1;
	return 0;
}

int CompruebaNewPasswordIsReady(fsm_t* p_this){
	int res;
	piLock(SYSTEM_KEY);
	res = g_flagsCoreWatch & FLAG_NEW_PASSWORD_IS_READY;
	piUnlock(SYSTEM_KEY);
	if(res!=0)return 1;
	return 0;
}

//FUNCION PROCESAR TECLAS

void ProcesaTeclaPulsada(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	char teclaPulsada;
	piLock(SYSTEM_KEY);
	TipoTecladoShared ts = GetTecladoSharedVar();
	//limpiar el flag
	ts.flags = ts.flags & (~FLAG_TECLA_PULSADA);
	SetTecladoSharedVar(ts);
	teclaPulsada = ts.teclaDetectada;
	piUnlock(SYSTEM_KEY);

//  if(teclaPulsada == (int)TECLA_RESET){
//     //activa flag reset
//     piLock(SYSTEM_KEY);
//     g_flagsCoreWatch = g_flagsCoreWatch | FLAG_RESET;
//      piUnlock(SYSTEM_KEY);
	if(teclaPulsada == (int)TECLA_PASSWORD){
	//activa flag set password
```

```
piLock(SYSTEM_KEY);
g_flagsCoreWatch = g_flagsCoreWatch | FLAG_SET_PASSWORD;
piUnlock(SYSTEM_KEY);
}else if(teclaPulsada== (int)TECLA_SET_CANCEL_TIME){
//activa flag set cancel time
piLock(SYSTEM_KEY);
g_flagsCoreWatch = g_flagsCoreWatch | FLAG_SET_CANCEL_NEW_TIME;
piUnlock(SYSTEM_KEY);
}else if(teclaPulsada== (int)TECLA_SET_CANCEL_DATE){
//activa flag set cancel date
piLock(SYSTEM_KEY);
g_flagsCoreWatch = g_flagsCoreWatch | FLAG_SET_CANCEL_NEW_DATE;
piUnlock(SYSTEM_KEY);
}else if(teclaPulsada== (int)TECLA_SWITCH_FORMAT){
//activa flag set cancel date
piLock(SYSTEM_KEY);
   g_flagsCoreWatch = g_flagsCoreWatch | FLAG_SWITCH_FORMAT;
piUnlock(SYSTEM_KEY);
}else if(teclaPulsada== (int)TECLA_ALARMA){
//activa flag set cancel date
piLock(SYSTEM_KEY);
g_flagsCoreWatch = g_flagsCoreWatch | FLAG_SET_ALARMA;
piUnlock(SYSTEM_KEY);
}else if(EsNumero(teclaPulsada)){
//activa flag digito pulsado y guarda el digito
piLock(SYSTEM_KEY);
g_coreWatch.digitoPulsado = (int)(teclaPulsada) - 48;
  g_flagsCoreWatch = g_flagsCoreWatch | FLAG_DIGITO_PULSADO;
piUnlock(SYSTEM_KEY);
}else if(teclaPulsada == TECLA_EXIT){
//sale del sistema
#if VERSION<=3
piLock(STD_IO_LCD_BUFFER_KEY);
printf("\rSaliendo del sistema\n");
piUnlock(STD_IO_LCD_BUFFER_KEY);
#endif
#if VERSION==4
piLock(STD_IO_LCD_BUFFER_KEY);
lcdClear(p_sistema->lcdId);
lcdPosition(p_sistema->lcdId, 0,0);
lcdPrintf(p_sistema->lcdId, "    Bye!" );
lcdPosition(p_sistema->lcdId, 0,1);
lcdPrintf(p_sistema->lcdId, "  /(^o^)|" );
delay(500);
lcdClear(p_sistema->lcdId);
lcdPosition(p_sistema->lcdId, 0,0);
lcdPrintf(p_sistema->lcdId, "    Bye!" );
lcdPosition(p_sistema->lcdId, 0,1);
lcdPrintf(p_sistema->lcdId, "  |(^O^)/" );
delay(500);
lcdClear(p_sistema->lcdId);
lcdPosition(p_sistema->lcdId, 0,0);
lcdPrintf(p_sistema->lcdId, "    Bye!" );
lcdPosition(p_sistema->lcdId, 0,1);
lcdPrintf(p_sistema->lcdId, "  /(^o^)|" );
```

```
        delay(500);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0,0);
        lcdPrintf(p_sistema->lcdId, "    Bye!" );
        lcdPosition(p_sistema->lcdId, 0,1);
        lcdPrintf(p_sistema->lcdId, "  |(^O^)/" );
        delay(500);
        lcdDisplay(p_sistema->lcdId, 0);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif

        exit(0);

        }else if((teclaPulsada != '\n') && (teclaPulsada != '\r') && (teclaPulsada != 0xA)){
        #if VERSION<=3
        piLock(STD_IO_LCD_BUFFER_KEY);
        printf("\rTecla desconocida\n");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        }

}

//FUNCIONES PASSWORD

void PrepareEnterPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->password.digitosGuardadosPassword = 0;
        p_sistema->password.tempPassword = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_DATE);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_TIME);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_ALARMA);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_PASSWORD_PASSWORD);
        piUnlock(SYSTEM_KEY);
#if VERSION==4
        if(g_flagsCoreWatch & FLAG_SET_PASSWORD){
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "CHANGING");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "PASSWORD");
        delay(1000);
        lcdClear(p_sistema->lcdId);
         lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "INTRODUCE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CURRENT");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        }else{
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
```

```
        lcdPrintf(p_sistema->lcdId, "ENTER");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "PASSWORD");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        }

#endif
//  piLock(SYSTEM_KEY);
//  tmr_t* tmrpassword = tmr_new(tmr_timeout_password);
//  p_sistema->password.tmrPassword= tmrpassword;
//  tmr_startms(p_sistema->password.tmrPassword, TIMEOUT_PASSWORD);
//  piUnlock(SYSTEM_KEY);
}

void EnterPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        int tempPassword = p_sistema->password.tempPassword;
        int digitosGuardadosPassword = p_sistema->password.digitosGuardadosPassword;
        int ultimoDigito = g_coreWatch.digitoPulsado;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        piUnlock(SYSTEM_KEY);
        if(digitosGuardadosPassword < 4){
        tempPassword = tempPassword*10 + ultimoDigito;
        digitosGuardadosPassword++;
        }
        if(digitosGuardadosPassword == 4){
        if(tempPassword == p_sistema->password.number){
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                if(g_flagsCoreWatch & FLAG_SET_PASSWORD){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_PASSWORD;
                }
                if(g_flagsCoreWatch & FLAG_SET_CANCEL_NEW_TIME){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_TIME;
                }
                if(g_flagsCoreWatch & FLAG_SET_CANCEL_NEW_DATE){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_DATE;
                }
                if(g_flagsCoreWatch & FLAG_SET_ALARMA){
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_ALARMA;
                }
                piUnlock(SYSTEM_KEY);
        }else{
                digitosGuardadosPassword =0;
                tempPassword = 0;
                piLock(STD_IO_LCD_BUFFER_KEY);
                lcdClear(p_sistema->lcdId);
                lcdPosition(p_sistema->lcdId, 0, 0);
                lcdPrintf(p_sistema->lcdId, "WRONG PASSWORD!");
                lcdPosition(p_sistema->lcdId, 0, 1);
                lcdPrintf(p_sistema->lcdId, "TRY AGAIN");
                delay(1000);
```

```
            lcdClear(p_sistema->lcdId);
            lcdPosition(p_sistema->lcdId, 0, 0);
            lcdPrintf(p_sistema->lcdId, "ENTER");
            lcdPosition(p_sistema->lcdId, 0, 1);
            lcdPrintf(p_sistema->lcdId, "PASSWORD");
            piUnlock(STD_IO_LCD_BUFFER_KEY);
        }
        }
        p_sistema->password.tempPassword = tempPassword;
        p_sistema->password.digitosGuardadosPassword = digitosGuardadosPassword;
}

void tmr_timeout_password(union sigval value) {
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch | FLAG_TIMEOUT_PASSWORD;
        piUnlock(SYSTEM_KEY);
}

void CancelEnterPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->password.tempPassword = 0;
        p_sistema->password.digitosGuardadosPassword = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & ~FLAG_TIMEOUT_PASSWORD;
        g_flagsCoreWatch = g_flagsCoreWatch & ~FLAG_SET_CANCEL_NEW_DATE;
        g_flagsCoreWatch = g_flagsCoreWatch & ~FLAG_SET_CANCEL_NEW_TIME;
        g_flagsCoreWatch = g_flagsCoreWatch & ~FLAG_SET_ALARMA;
        piUnlock(SYSTEM_KEY);
}

void PrepareSetNewPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(SYSTEM_KEY);
    g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_PASSWORD);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
        piUnlock(SYSTEM_KEY);
        p_sistema->password.tempNewPassword = 0;
        p_sistema->password.tempNewPasswordRepeat = 0;
        p_sistema->password.digitosGuardadosPassword = 0;
        #if VERSION==4
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "INTRODUCE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "NEW");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
}

void EnterNewPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        int tempNewPassword = p_sistema->password.tempNewPassword;
        int tempNewPasswordRepeat = p_sistema->password.tempNewPasswordRepeat;
```

```
int digitosGuardadosPassword = p_sistema->password.digitosGuardadosPassword;
int ultimoDigito = g_coreWatch.digitoPulsado;
piLock(SYSTEM_KEY);
g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
piUnlock(SYSTEM_KEY);
if(digitosGuardadosPassword < 4){
tempNewPassword = tempNewPassword*10 + ultimoDigito;
digitosGuardadosPassword++;
}
if(digitosGuardadosPassword >= 5 && digitosGuardadosPassword < 9){
tempNewPasswordRepeat = tempNewPasswordRepeat*10 + ultimoDigito;
digitosGuardadosPassword++;
}
if(digitosGuardadosPassword == 9){
if(tempNewPassword == tempNewPasswordRepeat){
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
         lcdPrintf(p_sistema->lcdId, "PASSWORD");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CHANGED");
        delay(1000);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        //FLAGS
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_PASSWORD_IS_READY;
        piUnlock(SYSTEM_KEY);
}else{
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "NOT MATCHING");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "TRY AGAIN");
        delay(1000);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "INTRODUCE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "NEW");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        digitosGuardadosPassword = 0;
        tempNewPassword = 0;
        tempNewPasswordRepeat = 0;
}
}

if(digitosGuardadosPassword == 4){
piLock(STD_IO_LCD_BUFFER_KEY);
lcdClear(p_sistema->lcdId);
lcdPosition(p_sistema->lcdId, 0, 0);
lcdPrintf(p_sistema->lcdId, "REPEAT");
lcdPosition(p_sistema->lcdId, 0, 1);
lcdPrintf(p_sistema->lcdId, "PLEASE");
piUnlock(STD_IO_LCD_BUFFER_KEY);
```

```
        digitosGuardadosPassword++;
        }

        p_sistema->password.tempNewPassword = tempNewPassword;
        p_sistema->password.tempNewPasswordRepeat = tempNewPasswordRepeat;
        p_sistema->password.digitosGuardadosPassword = digitosGuardadosPassword;
}

void CancelSetNewPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_PASSWORD);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
        piUnlock(SYSTEM_KEY);
        p_sistema->password.tempNewPassword = 0;
        p_sistema->password.tempNewPasswordRepeat = 0;
        p_sistema->password.digitosGuardadosPassword = 0;
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "SET_PASSWORD");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CANCELLED");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}

void SetNewPassword(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->password.number = p_sistema->password.tempNewPassword;
        p_sistema->password.tempNewPassword = 0;
        p_sistema->password.tempNewPasswordRepeat = 0;
        p_sistema->password.digitosGuardadosPassword = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
        piUnlock(SYSTEM_KEY);
}

//FUNCIONES STANDBY

void Start(fsm_t* p_this){
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SETUP_DONE);
        piUnlock(SYSTEM_KEY);
}

void ShowTime(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        TipoRelojShared sharedvars = GetRelojSharedVar();
        sharedvars.flags = sharedvars.flags & (~ FLAG_TIME_ACTUALIZADO);
        SetRelojSharedVar(sharedvars);
        TipoReloj reloj_sistema = p_sistema->reloj;
        int hora = reloj_sistema.hora.hh;
        int min = reloj_sistema.hora.mm;
        int seg = reloj_sistema.hora.ss;
```

```
        int dia = reloj_sistema.calendario.dd;
        int mes = reloj_sistema.calendario.MM;
        int year = reloj_sistema.calendario.yyyy;
        int ampm;
        if(reloj_sistema.hora.formato == 12){
        ampm = hora_ampm[1][hora];
        hora = hora_ampm[0][hora];
        }
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        //impresion hora
        char str[12];
        //usamos sprintf para anadir a la variable str la hora min y seg
        if(hora<10){
        sprintf(str, "0%d:", hora);
        }else{
        sprintf(str, "%d:", hora);
        }
        if(min<10){
        sprintf(str, "%s0%d:", str, min);
        }else{
        sprintf(str, "%s%d:", str, min);
        }
        if(seg<10){
        sprintf(str, "%s0%d", str, seg);
        }else{
        sprintf(str, "%s%d", str, seg);
        }
        if(reloj_sistema.hora.formato == 24){
        lcdPrintf(p_sistema->lcdId, "%s", str);
        }else{
        if(ampm){
                lcdPrintf(p_sistema->lcdId, "%s pm", str);
        }else{
                lcdPrintf(p_sistema->lcdId, "%s am", str);
        }
        }
    char date[12];
        sprintf(date,"%c ", diasemana[p_sistema->reloj.calendario.weekDay]);
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "%s%d/%d/%d", date, dia, mes, year);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}

//FUNCION RESET

void Reset(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        ResetReloj(&(p_sistema->reloj));
        piLock(SYSTEM_KEY);
                //ponemos a 0 flag_reset
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_RESET);
        piUnlock(SYSTEM_KEY);
        #if VERSION < 4
```

```
        piLock(STD_IO_LCD_BUFFER_KEY);
        printf("\r[RESET] Fecha y hora reiniciadas\n");
        fflush(stdout);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        #if VERSION==4
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "  [RESET]");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
}

//FUNCIONES TIME

void PrepareSetNewTime(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        piUnlock(SYSTEM_KEY);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_CANCEL_NEW_TIME);
        piUnlock(SYSTEM_KEY);
        #if VERSION==4
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_TIME]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "__:__");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
}

void CancelSetNewTime(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->tempTime = 0;
        p_sistema->digitosGuardados = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_CANCEL_NEW_TIME);
        piUnlock(SYSTEM_KEY);
        #if VERSION==4
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_TIME]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CANCELLED");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
}

void ProcesaDigitoTime(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
```

```
TipoReloj r = p_sistema->reloj;
int formato = r.hora.formato;
int tempTime = p_sistema->tempTime;
int digitosGuardados = p_sistema->digitosGuardados;
#if VERSION >= 2
int ultimoDigito = g_coreWatch.digitoPulsado;
#endif

piLock(SYSTEM_KEY);
g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
piUnlock(SYSTEM_KEY);
if(formato == 12){
if(digitosGuardados == 0){
       ultimoDigito = min(1, ultimoDigito);
       tempTime = tempTime*10 + ultimoDigito;
}else if(digitosGuardados == 1){
       if(tempTime == 0){
           ultimoDigito = max(1, ultimoDigito);
       }else{
       ultimoDigito = min(2, ultimoDigito);
       }
       tempTime = tempTime * 10 + ultimoDigito;
}else if(digitosGuardados == 2){
       tempTime = tempTime * 10 + min(5, ultimoDigito);
}else if (digitosGuardados == 3){
       tempTime = tempTime*10+ultimoDigito;
}else{
       ultimoDigito = min(ultimoDigito, 1);
       tempTime = tempTime*10+ultimoDigito;
       piLock(SYSTEM_KEY);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
       piUnlock(SYSTEM_KEY);
       piLock(SYSTEM_KEY);
       g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_TIME_IS_READY;
       piUnlock(SYSTEM_KEY);
}
}else{
if(digitosGuardados == 0){
       ultimoDigito = min(2, ultimoDigito);
       tempTime = tempTime*10 + ultimoDigito;
}else if(digitosGuardados == 1){
       if(tempTime  == 2){
           ultimoDigito = min(3, ultimoDigito);
       }
       tempTime = tempTime * 10 + ultimoDigito;
}else if(digitosGuardados == 2){
       tempTime = tempTime * 10 + min(5, ultimoDigito);
}else{
       tempTime = tempTime*10+ultimoDigito;
       piLock(SYSTEM_KEY);
       g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
       piUnlock(SYSTEM_KEY);
       piLock(SYSTEM_KEY);
       g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_TIME_IS_READY;
       piUnlock(SYSTEM_KEY);
```

```
        }
        }
        digitosGuardados++;
        #if VERSION == 4
        piLock(STD_IO_LCD_BUFFER_KEY);
        if(digitosGuardados<3){
        lcdPosition(p_sistema->lcdId, digitosGuardados-1, 1);
         lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }else{
        lcdPosition(p_sistema->lcdId, digitosGuardados, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }
        if(formato == 12 && digitosGuardados == 4){
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "SELECCIONE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "AM:0 / PM:1");
        }
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif

        p_sistema->tempTime = tempTime;
        p_sistema->digitosGuardados = digitosGuardados;

}

void SetNewTime(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_TIME_IS_READY);
        piUnlock(SYSTEM_KEY);
        if (p_sistema->reloj.hora.formato == 12){
        int ampm = p_sistema->tempTime % 10;
        int min = (p_sistema->tempTime / 10) % 100;
        int hora = p_sistema->tempTime / 1000;
        if(ampm==1 && hora!=12){
                hora+=12;
        }
        if(ampm==0 && hora == 12){
                hora = 0;
        }
        hora = hora*100 + min;
        SetHora(hora, &p_sistema->reloj.hora);
        }else{
        SetHora(p_sistema->tempTime, &p_sistema->reloj.hora);
        }
        p_sistema->tempTime = 0;
        p_sistema->digitosGuardados = 0;
}

//FUNCIONES DATE

void PrepareSetNewDate(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(SYSTEM_KEY);
```

```
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_CANCEL_NEW_DATE);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_DATE_IS_READY);
        piUnlock(SYSTEM_KEY);
        #if VERSION < 4
        piLock(STD_IO_LCD_BUFFER_KEY);
                printf("\r[SET_DATE] Introduzca la nueva fecha en formato dd/mm/yyyy\n");
                fflush(stdout);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        #if VERSION==4
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_DATE]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "__/__/____");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
}

void CancelSetNewDate(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->tempDate = 0;
        p_sistema->digitosGuardadosDate = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SET_CANCEL_NEW_DATE);
        piUnlock(SYSTEM_KEY);
        #if VERSION < 4
        piLock(STD_IO_LCD_BUFFER_KEY);
                printf("\r[SET_DATE] Operacion cancelada\n");
                fflush(stdout);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        #if VERSION==4
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_DATE]");
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "CANCELLED");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
}

void ProcesaDigitoDate(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        int tempDate = p_sistema->tempDate;
        int digitosGuardadosDate = p_sistema->digitosGuardadosDate;
        #if VERSION >= 2
        int ultimoDigito = g_coreWatch.digitoPulsado;
        #endif
        int aux = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
```

```
piUnlock(SYSTEM_KEY);
//procesamos el dia
if(digitosGuardadosDate == 0){
ultimoDigito = min(ultimoDigito, MAX_DAY/10);
tempDate = tempDate*10 + ultimoDigito;
}else if(digitosGuardadosDate == 1) {
tempDate = min(tempDate*10+ultimoDigito, MAX_DAY);
//procesamos el mes
}else if(digitosGuardadosDate == 2) {
ultimoDigito = min(ultimoDigito, 1);
tempDate = tempDate*10 + ultimoDigito;
}else if(digitosGuardadosDate == 3){
aux = tempDate / 10;
aux = (aux * 100) + MAX_MONTH; // aux = dd/12
tempDate = min(tempDate*10+ultimoDigito, aux);
//procesamos el year
}else if(digitosGuardadosDate == 4){
ultimoDigito = max(ultimoDigito, 1);
tempDate = tempDate*10 + ultimoDigito;
}else if(digitosGuardadosDate == 5){
aux = tempDate / 10;
aux = aux * 100 + MIN_YEAR/100; //aux = dd/mm/19xx
tempDate = max(tempDate*10+ultimoDigito, aux);
}else if(digitosGuardadosDate == 6){
aux = tempDate / 100;
aux = aux * 1000 + MIN_YEAR/10; //aux = dd/mm/197x
tempDate = max(tempDate*10+ultimoDigito, aux);
}else{
aux = tempDate / 1000;
aux = aux * 10000 + MIN_YEAR; //aux = dd/mm/1970
tempDate = max(tempDate*10+ultimoDigito, aux);
piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
piUnlock(SYSTEM_KEY);
piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_DATE_IS_READY;
piUnlock(SYSTEM_KEY);
}
digitosGuardadosDate++;
#if VERSION<4
piLock(STD_IO_LCD_BUFFER_KEY);
printf("\rNueva fecha temporal: %d\n", tempDate);
fflush(stdout);
piUnlock(STD_IO_LCD_BUFFER_KEY);
#endif
#if VERSION >= 4
piLock(STD_IO_LCD_BUFFER_KEY);
if(digitosGuardadosDate<3){
lcdPosition(p_sistema->lcdId, digitosGuardadosDate-1, 1);
lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
}else if(digitosGuardadosDate >= 3 && digitosGuardadosDate < 5){
lcdPosition(p_sistema->lcdId, digitosGuardadosDate, 1);
lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
}else{
lcdPosition(p_sistema->lcdId, digitosGuardadosDate+1, 1);
```

```
        lcdPutchar(p_sistema->lcdId, (char)(tempDate%10 + 48));
        }
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        p_sistema->tempDate = tempDate;
        p_sistema->digitosGuardadosDate = digitosGuardadosDate;
}

void SetNewDate(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        int fecha = p_sistema->tempDate;
        int dia = fecha / 1000000; //2 digitos superiores
        int mes = (fecha / 10000) % 100; //2 digitos del medio
        int year = fecha%10000; //4 ultimos digitos
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_DATE_IS_READY);
        piUnlock(SYSTEM_KEY);
        //piLock(RELOJ_KEY);
        SetYear(year, &p_sistema->reloj.calendario);
        SetMes(mes, &p_sistema->reloj.calendario);
        SetDia(dia, &p_sistema->reloj.calendario);
        //piUnlock(RELOJ_KEY);
        dia = p_sistema->reloj.calendario.dd;
        mes = p_sistema->reloj.calendario.MM;
        year = p_sistema->reloj.calendario.yyyy;

        int weekDay = CalculaDiaSemana(dia, mes, year);
        p_sistema->reloj.calendario.weekDay = weekDay;
        p_sistema->tempDate = 0;
        p_sistema->digitosGuardadosDate = 0;
}

//FUNCIONES FORMATO

void SwitchFormat(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        int formato = 12;
        if(p_sistema->reloj.hora.formato == 12) formato = 24;
        SetFormat(formato, &p_sistema->reloj.hora);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SWITCH_FORMAT);
        piUnlock(SYSTEM_KEY);
}

//FUNCIONES ALARMA

void SuenaAlarma(fsm_t* p_this){
    TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        tmr_t* tempalarma = tmr_new(tmr_timeout_alarma);
        p_sistema->alarma.timer = tempalarma;
        tmr_startms(tempalarma, TIMEOUT_ALARMA);
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "Bip! Bip!");
```

```
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "Bip! Bip!");
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}

void ExitAlarma(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_TIMEOUT_ALARMA);
        piUnlock(SYSTEM_KEY);
}

void tmr_timeout_alarma(union sigval value) {
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch | FLAG_TIMEOUT_ALARMA;
        piUnlock(SYSTEM_KEY);
}

void PrepareSetAlarma(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->alarma.digitosGuardadosAlarma = 0;
        p_sistema->alarma.tempAlarma = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_ALARMA);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_ALARMA_IS_READY);
        piUnlock(SYSTEM_KEY);
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_ALARMA]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "__:__");
        delay(1000);
        piUnlock(STD_IO_LCD_BUFFER_KEY);
}

void CancelSetNewAlarma(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        p_sistema->alarma.digitosGuardadosAlarma = 0;
        p_sistema->alarma.tempAlarma = 0;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_ALARMA);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGITO_PULSADO);
        g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_ALARMA_IS_READY);
        piUnlock(SYSTEM_KEY);
        piLock(STD_IO_LCD_BUFFER_KEY);
        lcdClear(p_sistema->lcdId);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_ALARMA]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "CANCELLED");
```

```
	delay(1000);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
}

void ProcesaDigitoAlarma(fsm_t* p_this){
	TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
	TipoReloj r = p_sistema->reloj;
	int formato = r.hora.formato;
	int tempTime = p_sistema->alarma.tempAlarma;
	int digitosGuardados = p_sistema->alarma.digitosGuardadosAlarma;
	int ultimoDigito = g_coreWatch.digitoPulsado;
	piLock(SYSTEM_KEY);
	g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
	piUnlock(SYSTEM_KEY);
	if(formato == 12){
	if(digitosGuardados == 0){
		ultimoDigito = min(1, ultimoDigito);
		tempTime = tempTime*10 + ultimoDigito;
	}else if(digitosGuardados == 1){
		if(tempTime == 0){
		ultimoDigito = max(1, ultimoDigito);
		}else{
		ultimoDigito = min(2, ultimoDigito);
		}
		tempTime = tempTime * 10 + ultimoDigito;
	}else if(digitosGuardados == 2){
		tempTime = tempTime * 10 + min(5, ultimoDigito);
	}else if (digitosGuardados == 3){
		tempTime = tempTime*10+ultimoDigito;
	}else if (digitosGuardados == 4){
		ultimoDigito = min(ultimoDigito, 1);
		tempTime = tempTime*10+ultimoDigito;
	}else {
		tempTime = tempTime*10+ultimoDigito;
		piLock(SYSTEM_KEY);
		g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
		piUnlock(SYSTEM_KEY);
		piLock(SYSTEM_KEY);
		g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_ALARMA_IS_READY;
		piUnlock(SYSTEM_KEY);
	}
	}else{
	if(digitosGuardados == 0){
		ultimoDigito = min(2, ultimoDigito);
		tempTime = tempTime*10 + ultimoDigito;
	}else if(digitosGuardados == 1){
		if(tempTime  == 2){
		ultimoDigito = min(3, ultimoDigito);
		}
		tempTime = tempTime * 10 + ultimoDigito;
	}else if(digitosGuardados == 2){
		tempTime = tempTime * 10 + min(5, ultimoDigito);
	}else if(digitosGuardados == 3){
		tempTime = tempTime * 10 + ultimoDigito;
	}else{
```

```
                tempTime = tempTime*10+ultimoDigito;
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGITO_PULSADO);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_ALARMA_IS_READY;
                piUnlock(SYSTEM_KEY);

        }
        }
        digitosGuardados++;
        #if VERSION == 4
        piLock(STD_IO_LCD_BUFFER_KEY);
        if(digitosGuardados<3){
        lcdPosition(p_sistema->lcdId, digitosGuardados-1, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }else{
        lcdPosition(p_sistema->lcdId, digitosGuardados, 1);
        lcdPutchar(p_sistema->lcdId, (char)(tempTime%10 + 48));
        }
        if(formato == 12 && digitosGuardados == 4){
        delay(500);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "SELECCIONE");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "AM:0 / PM:1");
        }

        if((formato == 24 && digitosGuardados == 4) || (formato == 12 && digitosGuardados
== 5)){

        delay(500);
        lcdPosition(p_sistema->lcdId, 0, 0);
        lcdPrintf(p_sistema->lcdId, "[SET_ALARMA]");
        lcdPosition(p_sistema->lcdId, 0, 1);
        lcdPrintf(p_sistema->lcdId, "OFF:0 / ON:1");
        }
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        p_sistema->alarma.tempAlarma = tempTime;
        p_sistema->alarma.digitosGuardadosAlarma = digitosGuardados;
}

void SetNewAlarma(fsm_t* p_this){
        TipoCoreWatch *p_sistema = (TipoCoreWatch*)(p_this->user_data);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_ALARMA_IS_READY);
        piUnlock(SYSTEM_KEY);
        if (p_sistema->reloj.hora.formato == 12){
        int activada = p_sistema->alarma.tempAlarma % 10;
        int ampm = (p_sistema->alarma.tempAlarma / 10) % 10;
        int min = (p_sistema->alarma.tempAlarma / 100) % 100;
        int hora = p_sistema->alarma.tempAlarma / 10000;
        if(ampm==1 && hora!=12){
                hora+=12;
        }
```

```
        if(ampm==0 && hora == 12){
                hora = 0;
        }
        hora = hora*100 + min;
        SetHora(hora, &p_sistema->alarma.hora);
        p_sistema->alarma.alarmaActivada = activada;
        }else{
        int hora = p_sistema->alarma.tempAlarma/10;
        int activada = p_sistema->alarma.tempAlarma%10;
        SetHora(hora, &p_sistema->alarma.hora);
        p_sistema->alarma.alarmaActivada = activada;
        }
        p_sistema->tempTime = 0;
        p_sistema->digitosGuardados = 0;
}

//-------------------------------------------------------
// MAIN
//-------------------------------------------------------
int main() {
        unsigned int next;

#if VERSION == 4
        TipoCoreWatch corePrueba;
        int sol = ConfiguraInicializaSistema(&corePrueba);
        TipoReloj r = corePrueba.reloj;
        if(sol != 0){printf("error");exit(0);}

        fsm_t* fsmReloj = fsm_new(WAIT_TIC, g_fsmTransReloj, &(corePrueba.reloj));

        fsm_t* fsmCoreWatch = fsm_new(START, g_fsmTransCoreWatch, &(corePrueba));

        fsm_t*  fsmDeteccionComandos  =  fsm_new(WAIT_COMMAND,  g_fsmTransDeteccionComandos,
&(corePrueba));

        fsm_t*  fsmTeclado  =  fsm_new(TECLADO_ESPERA_COLUMNA,  g_fsmTransExcitacionColumnas,
&(corePrueba.teclado));

        next = millis();
        while(1){
        fsm_fire(fsmReloj);
        fsm_fire(fsmDeteccionComandos);
        fsm_fire(fsmTeclado);
        fsm_fire(fsmCoreWatch);
        next+=CLK_MS;
        DelayUntil(next);
        }
        //destruir
        tmr_destroy(r.tmrTic);
        fsm_destroy(fsmReloj);
        fsm_destroy(fsmCoreWatch);

#endif

}
```