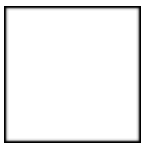


Digital Systems II
Polytechnic University of Madrid
Department of Electronic Engineering
E.T.S.I. of Telecommunications

Report of the improvements made

Course 2021/2022

Title of the developed project "CORE WATCH".

Authors (in alphabetical order): ÍÑIGO GONZÁLEZ VARAS-VALLEJO

JAVIER MARTÍNEZ RANZ

Partner code: JT-06

GENERAL INDEX

1 Introduction

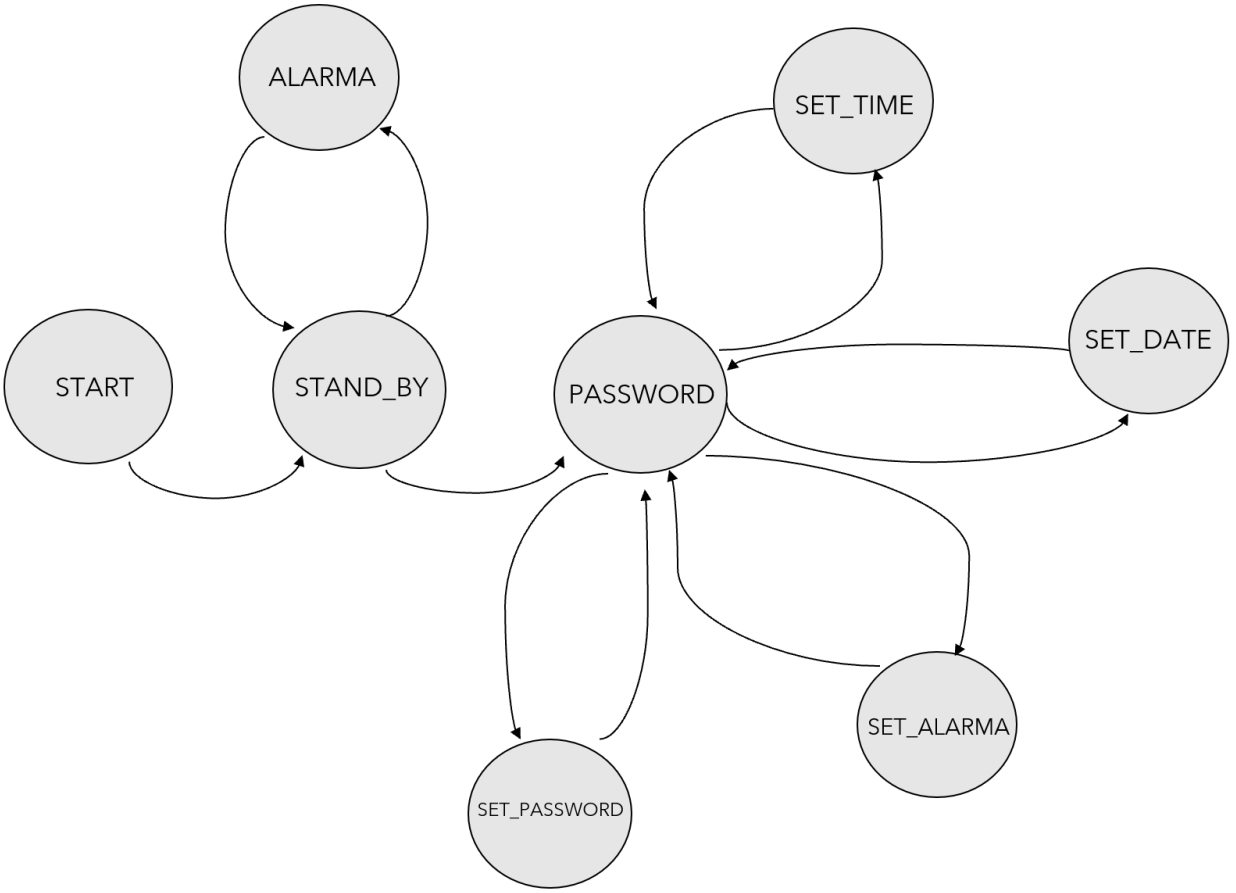
With the implementation of the improvements we wanted to provide different features to our coreWatch system. These features can be a greater robustness, security, accuracy in the display of information or simply a better user experience.

In general we have used similar approaches in all the improvements we have implemented. First, we start with a new state or states that we add to the coreWatch FSM.
Then, we add the necessary FLAGS to take care of the transition functions that link the states we started from (previous versions) and the new ones.
Finally, we code the input functions of the state transitions involved, which will check those flags. And we also program the output or update functions.
Our most important improvements are a password for the system and an alarm. In addition, we have implemented date change, and am and pm format among others.

Videos demonstrating how it works can be found at:
https://drive.google.com/drive/folders/1Wk_cARk2O4lFSYuPVbtky1X38P4DNfP7?usp=sharing

2 Extended subsystem diagram

General block diagram



3 Enhancement 1: Modification of the date (SET_DATE)

3.1 Improvement objectives

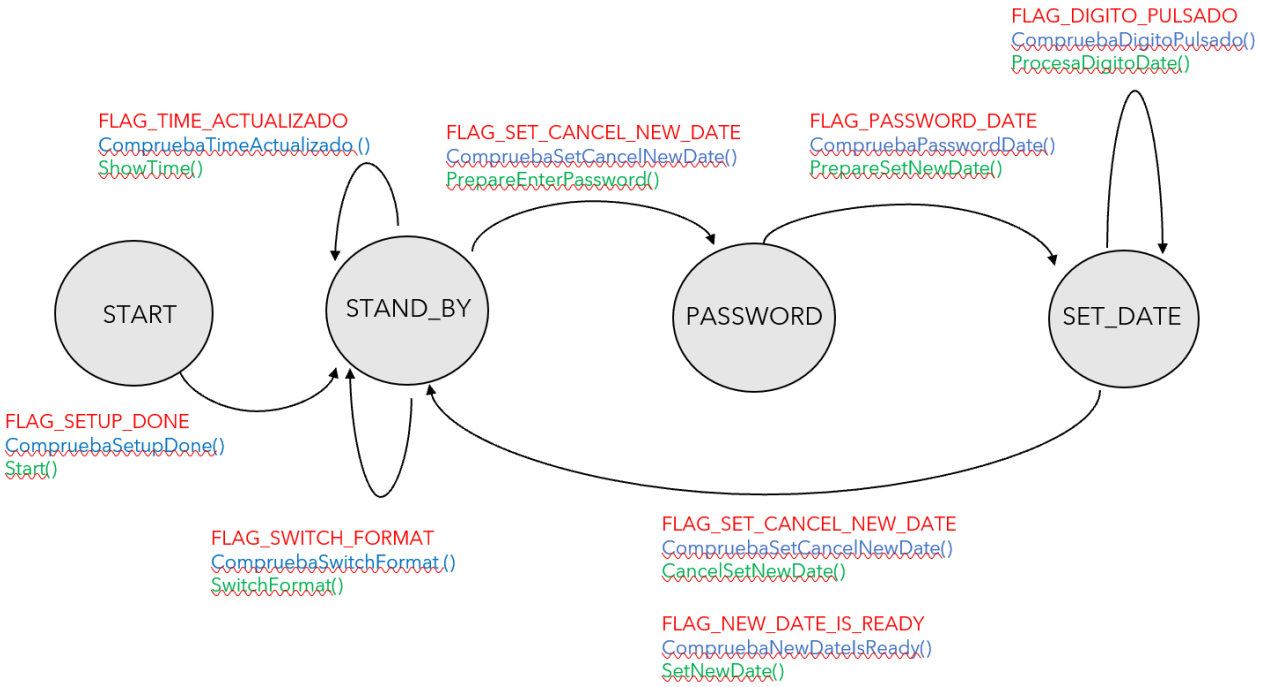
The purpose of this improvement is to allow the user to modify the CoreWatch date.

3.2 Software Subsystem Description

We have used the same approach that was used with SET_TIME, pressing a key activates a flag and we go to the SET_DATE state, in which we enter digits that we will process with ProcessDigitDate(). This function makes sure that the digits entered are valid, that is, that no months over 12, days over 31 or years under 1970 can be entered.

Once all the digits have been entered, they are passed to some functions that we have added in clock.c, which are SetDay(), SetMonth() and SetYear(). These functions are called in a specific order, because it is important that the SetDay() function can check that the day entered is valid for the year and month entered.

For example, it validates that there cannot be a February 29th day in a non-leap year or that April 31st cannot be entered.



3.2.1 Description of subroutines

3.2.2 SetDay(), SetMonth() and SetYear()

In clock.c SetDay(), SetMonth() and SetYear() calculate that the date entered is valid, i.e. November 31st is not allowed for example, and update it from the CalendarType passed as a parameter.

```
int SetDay(int day, CalendarType *p_calendar){
	//The day entered cannot be 0 or lower
	if(dia <= 0){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Enter a valid day");
		fflush(stdout);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 1;
	}
	//Check that you have entered 1 or 2 digits
	int num_digits = 0;
	int aux = dia;
	while(aux!=0){
	aux /= 10;
	    num_digits++;
	}
	if(num_digits < 1|| num_digits> 2){
	piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Enter a day with a maximum of 2 digits");
		fflush(stdout);
	    piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 2;
	}
	int currentMonth = p_calendar-> MM;
	int yearActual = p_calendar-> yyyy;
	int maxDayMonth = CalculateDayMonth(currentMonth, currentYear);
	if(day > maxDayMonth){
	    day = maxDayMonth;
	}
	p_calendar-> dd = day;
	return 0;

}

int SetMonth(int month, CalendarType *p_calendar){
	//The day entered can be neither 0 nor less than 0
	if(month <= 0){
	    piLock(STD_IO_LCD_BUFFER_KEY);
		printf("Enter a valid month");
		fflush(stdout);
	    piUnlock(STD_IO_LCD_BUFFER_KEY);
	return 1;
	}
	//Check that you have entered 1 or 2 digits
	int num_digits = 0;
	int aux = month;
```

```
        while(aux!=0){
            aux /= 10;
            num_digits++;
        }
        if(num_digits < 1|| num_digits> 2){
        piLock(STD_IO_LCD_BUFFER_KEY);
               printf("Enter a month with a maximum of 2 digits");
               fflush(stdout);
            piUnlock(STD_IO_LCD_BUFFER_KEY);
        return 2;
        }
        if(month > MAX_MONTH){
        month = MAX_MONTH;
        }
        p_calendar-> MM = month;
        return 0;
}

int SetYear(int year, CalendarType *p_calendar){
        //The day entered can be neither 0 nor less than 0
        if(year <= 0){
            piLock(STD_IO_LCD_BUFFER_KEY);
               printf("Enter a year a.d.");
         fflush(stdout);
            piUnlock(STD_IO_LCD_BUFFER_KEY);
        return 1;
        }
        if(year < MIN_YEAR){
            year = MIN_YEAR;
        }
        p_calendar-> yyyy = year;
        return 0;
}
```

3.2.3 PrepareSetNewDate() CancelSetNewDate() and SetNewDate()

On the other hand, the auxiliary functions PrepareSetNewDate() and CancelSetNewDate() and SetNewDate() are in charge of clearing the flags associated to the date, as well as clearing the data contained in the variables tempDate and digitsSavedDate. SetNewDate() also saves the date stored in tempDate in the system variables by calling SetDay() SetMonth() and SetYear().

```
void PrepareSetNewDate(fsm_t* p_this){
    CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
    piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
        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] Enter the new date in dd/mm/yyyyyyyy format");
            fflush(stdout);
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    #if VERSION==4
    piLock(STD_IO_LCD_BUFFER_KEY);
    lcdClear(p_system-> lcdId);
```

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

void CancelSetNewDate(fsm_t* p_this){
    CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
    p_system-> tempDate = 0;
    p_system-> digitsSavedDate = 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] Operation canceled");
            fflush(stdout);
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    #if VERSION==4
    piLock(STD_IO_LCD_BUFFER_KEY);
    lcdClear(p_system-> lcdId);
    lcdPosition(p_system-> lcdId, 0, 0);
    lcdPrintf(p_system-> lcdId, "[SET_DATE]");
    lcdPosition(p_system-> lcdId, 0, 0);
        lcdPrintf(p_system-> lcdId, "CANCELLED");
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
}
void SetNewDate(fsm_t* p_this){
    CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
    int date = p_system-> tempDate;
    int day = date / 1000000; //2 digits upwards
    int month = (date / 10000) % 100; //2 digits of the middle digit
    int year = date%10000; //4 last digits
    piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_DATE_IS_READY);
    piUnlock(SYSTEM_KEY);
    //piLock(CLOCK_KEY);
    SetYear(year, & p_system->clock. calendar);
    SetMonth(month, & p_system->clock. calendar);
    SetDay(day, & p_system->clock. calendar);
    //piUnlock(CLOCK_KEY);
    day = p_system-> clock. calendar. dd;
    month = p_system-> clock. calendar. MM;
    year = p_system-> clock. calendar. yyyy;

    int weekDay = CalculateWeekDay(day, month, year);
    p_system-> clock. calendar. weekDay = weekDay;
    p_system-> tempDate = 0;
    p_system-> digitsSavedDate = 0;
}
```

3.2.4 ProcessDigitDate()

To process the digits of the date, we store them in a tempDate variable as we did in ProcessDigitTime(). The first 2 digits will be for the day, the next 2 for the month and the last 4 for the year. We always make sure that the values entered are appropriate (no months over 12, days over 31 or years under 1970) but the verification that the date entered is valid will be done when we do SetNewDate() with the invocations to SetDia(), SetMes() and SetYear().

```
void ProcessDigitDate(fsm_t* p_this){
    CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
    int tempDate = p_system-> tempDate;
    int digitsSavedDate = p_system-> digitsSavedDate;
    #if VERSION >= 2
    int lastDigit = g_coreWatch. digitDriven;
    #endif
    int aux = 0;
    piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
    piUnlock(SYSTEM_KEY);
    //we process the day
    if(digitosGuardadosDate == 0){
        lastDigit = min(lastDigit, MAX_DAY/10);
        tempDate = tempDate*10 + lastDigit;
    }else if(digitosGuardadosDate == 1) {
        tempDate = min(tempDate*10+lastDigit, MAX_DAY);
    //we process the month
    }else if(digitosGuardadosDate == 2) {
        lastDigit = min(lastDigit, 1);
        tempDate = tempDate*10 + lastDigit;
    }else if(digitosGuardadosDate == 3){
        aux = tempDate / 10;
        aux = (aux * 100) + MAX_MONTH; // aux = dd/12
        tempDate = min(tempDate*10+lastDigit, aux);
    //we process the year
    }else if(digitosGuardadosDate == 4){
        lastDigit = max(lastDigit, 1);
        tempDate = tempDate*10 + lastDigit;
    }else if(digitsSavedDate == 5){
        aux = tempDate / 10;
        aux = aux * 100 + MIN_YEAR/100; //aux = dd/mm/19xx
        tempDate = max(tempDate*10+lastDigit, aux);
    }else if(digitosGuardadosDate == 6){
        aux = tempDate / 100;
        aux = aux * 1000 + MIN_YEAR/10; //aux = dd/mm/197x
        tempDate = max(tempDate*10+lastDigit, aux);
    }else{
        aux = tempDate / 1000;
        aux = aux * 10000 + MIN_YEAR; //aux = dd/mm/1970
        tempDate = max(tempDate*10+lastDigit, aux);
        piLock(SYSTEM_KEY);
            g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
        piUnlock(SYSTEM_KEY);
        piLock(SYSTEM_KEY);
            g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_DATE_IS_READY;
        piUnlock(SYSTEM_KEY);
```

```
    }
    digitsSavedDate++;
    #if VERSION<4
    piLock(STD_IO_LCD_BUFFER_KEY);
        printf("New temporary date: %d", tempDate);
        fflush(stdout);
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    #if VERSION >= 4
    piLock(STD_IO_LCD_BUFFER_KEY);
    if(digitosGuardadosDate< 3){
        lcdPosition(p_system-> lcdId, digitsSavedDate-1, 1);
        lcdPutchar(p_system-> lcdId, (char)(tempDate%10 + 48));
    }else if(digitsSavedDate >= 3 && digitsSavedDate < 5){
        lcdPosition(p_system-> lcdId, digitsSavedDate, 1);
        lcdPutchar(p_system-> lcdId, (char)(tempDate%10 + 48));
    }else{
        lcdPosition(p_system-> lcdId, digitsSavedDate+1, 1);
        lcdPutchar(p_system-> lcdId, (char)(tempDate%10 + 48));
    }
    piUnlock(STD_IO_LCD_BUFFER_KEY);
    #endif
    p_system-> tempDate = tempDate;
    p_system-> digitsSavedDate = digitsSavedDate;
}
```

4 Improvement 2: Time and date printout with 2 digits

4.1 Improvement objectives

The purpose of this enhancement is that all numbers shown on the display are represented with 2 digits. The main use of this enhancement and the case in which it will be visible is when the hour (hours, minutes or seconds) or the date (day or month) is less than 10.

4.2 Software Subsystem Description

For this implementation, we have modified Sho wTime() so that when the hour, minutes or seconds are less than 10, a 0 is added in front of the printf (e.g. 09:08:05). To do this we have made use of if-else statements and the sprintf() function, which can be formatted in the same way as printf does (printf("%d", time) prints the value of the variable time to the screen) but instead of outputting the value to the screen it saves it in a variable. To execute sprintf() successive times in a row and not erase what has been saved in previous executions, it is as simple as passing as a parameter the variable where we are saving the entire string. Example:

char str = "05:"
min = 3;
sprintf(str, "%s0%d:", str, min);
//value of str after executing sprintf: str = "05:03:"

4.3 Description of subroutines

4.3.1 ShowTime()

As explained above, the modification in ShowTime() consists of several if-else statements in which sprintf() is executed with one or another value. In this way we manage to store in the variable - the string we want to print.

```
void ShowTime(fsm_t* p_this){
       CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
       sharedvars = GetRelojSharedVar();
       sharedvars. flags = sharedvars. flags & (~ FLAG_TIME_ACTUALIZED);
       SetRelojSharedVar(sharedvars);
       ClockType system_clock = p_system-> clock;
       int time = system_clock. time. hh;
       int min = system_clock. hour. mm;
       int sec = system_clock. time. ss;
       int day = system_clock. calendar. dd;
       int month = system_clock. calendar. MM;
       int year = system_clock. calendar. yyyy;
       int ampm;
       if(system_clock. time. format == 12){
           ampm = ampm_time[1][time];
           time = time_ampm_time[0][time];
       }
       piLock(STD_IO_LCD_BUFFER_KEY);
       lcdClear(p_system-> lcdId);
       lcdPosition(p_system-> lcdId, 0, 0);
       //print time
       char str[12];
       //we use sprintf to add to the variable str the min and sec time.
```

```
        if(time< 10){
            sprintf(str, "0%d:", time);
        }else{
            sprintf(str, "%d:", time);
        }
        if(min< 10){
            sprintf(str, "%s0%d:", str, min);
        }else{
            sprintf(str, "%s%d:", str, min);
        }
        if(sec< 10){
            sprintf(str, "%s0%d", str, sec);
        }else{
            sprintf(str, "%s%d", str, sec);
        }
        if(system_clock. time. format == 24){
            lcdPrintf(p_system-> lcdId, "%s", str);
        }else{
        if(ampm){
                lcdPrintf(p_system-> lcdId, "%s pm", str);
        }else{
                lcdPrintf(p_system-> lcdId, "%s am", str);
        }
        }
    char date[12];
       sprintf(date,"%c ", weekdays[p_system->clock. calendar. weekDay]);
       lcdPosition(p_system-> lcdId, 0, 1);
       lcdPrintf(p_system-> lcdId, "%s%d/%d/%d/%d", date, day, month, year);
       piUnlock(STD_IO_LCD_BUFFER_KEY);
}
```

5 Improvement 3: AM/PM time format

5.1 Improvement objectives

The purpose of this enhancement is to allow the user to consider the form of AM-PM time representation when choosing the 12-hour format.

At the same time, it is also intended that the user will be able to change the time display format (12h or 24h) by pressing a key, specifically the 'C' key.

5.2 Software Subsystem Description

The goal we have is to implement this improvement as an aesthetic improvement. This means that we will work internally with the 24 hour format, so that to change the time, when going to SET_TIME to change the time... always the values of hour, minutes and seconds of the TipoHora of the system will be in format 24. However, when printing on screen or changing the time we will make distinctions for each format, that is, print the time in format 12 or allow the user to enter the new time in format 12.

5.2.1 UpdateTime()

For the development of this functionality, the first thing we have done is to restructure the code of clock.c related to the internal treatment of the time to eliminate everything related to the 12 hours format. This is, because what we want is that the system always interprets internally the hours in 24 hours format, and we will only take into account the 12 hours format to show it on the screen or (if the user has selected this format) to enter a time in the system.

```
void UpdateTime(TimeType *p_time){
       //update second hand
       p_hour-> ss = (p_hour-> ss+1)%60;
       //update minute hand
       if(p_time-> ss == 0) {
           p_hour-> mm = (p_hour-> mm+1)%60;
       if(p_time-> mm == 0){
                p_hour-> hh = p_hour-> hh+1;
              if(p_hour-> hh > MAX_HOUR_24){
                    p_hour-> hh = MIN_HOUR_24;
              }
       }
       }

}
```

5.2.2 ShowTime() and array hora_ampm

Once this is done, we want to print the time in format 12 when ShowTime() is executed if the format chosen by the user is 12 hours. To do this we've added an array hour_ampm[24][2]. With the internal system hours (0-23) we can access the array boxes; in the first row for the equivalent time in 12 format and in the second row we enter a 0 if it is am and a 1 if it is pm (e.g. 23 - 11, 1 - 11 pm). We assign this new value to the variable time, and depending on ampm, we print the time with the string "am" or "pm".

```
const int time_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){
       CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
       sharedvars = GetRelojSharedVar();
       sharedvars. flags = sharedvars. flags & (~ FLAG_TIME_ACTUALIZED);
       SetRelojSharedVar(sharedvars);
       ClockType system_clock = p_system-> clock;
       int time = system_clock. time. hh;
       int min = system_clock. hour. mm;
       int sec = system_clock. time. ss;
       int day = system_clock. calendar. dd;
       int month = system_clock. calendar. MM;
       int year = system_clock. calendar. yyyy;
       int ampm;
       if(system_clock. time. format == 12){
           ampm = ampm_time[1][time];
           time = time_ampm_time[0][time];
       }
       piLock(STD_IO_LCD_BUFFER_KEY);
       lcdClear(p_system-> lcdId);
       lcdPosition(p_system-> lcdId, 0, 0);
       //print time
       char str[12];
       //we use sprintf to add to the variable str the min and sec time.
       if(time< 10){
           sprintf(str, "0%d:", time);
       }else{
           sprintf(str, "%d:", time);
       }
       if(min< 10){
           sprintf(str, "%s0%d:", str, min);
       }else{
           sprintf(str, "%s%d:", str, min);
       }
       if(sec< 10){
           sprintf(str, "%s0%d", str, sec);
       }else{
           sprintf(str, "%s%d", str, sec);
       }
       if(system_clock. time. format == 24){
           lcdPrintf(p_system-> lcdId, "%s", str);
       }else{
       if(ampm){
```

```
                lcdPrintf(p_system-> lcdId, "%s pm", str);
        }else{
                lcdPrintf(p_system-> lcdId, "%s am", str);
        }
        }
    char date[12];
       sprintf(date,"%c ", weekdays[p_system->clock. calendar. weekDay]);
       lcdPosition(p_system-> lcdId, 0, 1);
       lcdPrintf(p_system-> lcdId, "%s%d/%d/%d/%d", date, day, month, year);
       piUnlock(STD_IO_LCD_BUFFER_KEY);
}
```

5.2.3 ProcessDigitTime() new

In addition to printing the time in format 12 on the screen, if the user has selected this mode when accessing SET_TIME, we want the time to be processed in format 12. To do this, we redo the flowchart of ProcessDigitTime(), clearly differentiating between format 12 and format 24. Format 24 is the same as before, format 12 will ask the user for 4 digits for the time (formatting them appropriately to format 12) plus an additional digit to select whether the time entered is am or pm. Once all the digits have been entered, the flag that causes SetNewTime() to be executed will be set. In this function we have also made modifications, since we want to treat the time in 24 format internally, so the digits stored in tempTime will have to be treated. Specifically, if the time is pm and greater than 12 we will add 12 to the time, while if it is 12 am, we will set the time to 0. This new hour will be passed along with the minutes to SetTime().

```
void ProcessDigitTime(fsm_t* p_this){
       CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
       ClockType r = p_system-> clock;
       int format = r. time. format;
       int tempTime = p_system-> tempTime;
       int digitsSavedDigits = p_system-> digitsSavedDigits;
       #if VERSION >= 2
       int lastDigit = g_coreWatch. digitDriven;
       #endif

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

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

        p_system-> tempTime = tempTime;
        p_system-> digitsSaved = digitsSaved;

}
```

5.2.4 SwitchFormat() and CheckSwitchFormat()

In addition to all the code used to be able to represent and change the time in format 12, we have added another functionality to the system: pressing the 'C' key will change the representation format. That is, if you were seeing the time in 12 hour format, you will see it in 24 hour format. We have done this by implementing a flag interrupt in the STAND_BY state, which executes the SwitchFormat() function, which changes the format value of TipoCoreWatch to the opposite of the current one.

```
int CheckSwitchFormat(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){
	CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
	int format = 12;
	if(p_system-> clock. time. format == 12) format = 24;
	SetFormat(format, & p_system-> clock. time);
	piLock(SYSTEM_KEY);
	    g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_SWITCH_FORMAT);
	piUnlock(SYSTEM_KEY);
}
```

6 Upgrade 4: Alarm

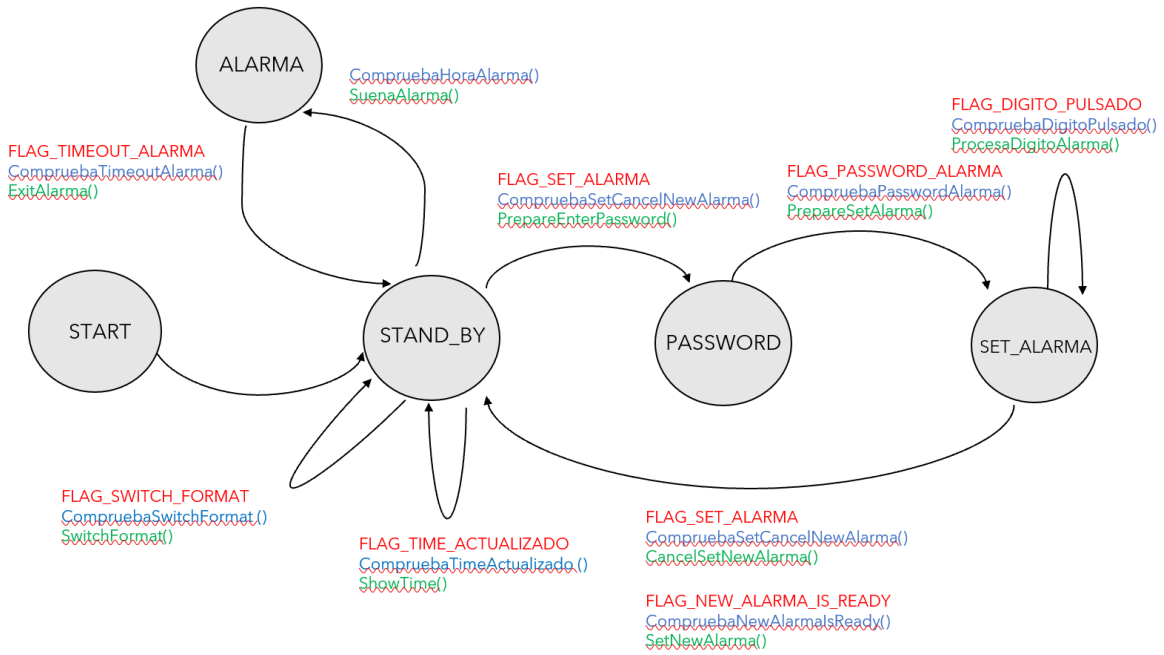
6.1 Improvement objectives

The purpose of this enhancement is to allow the user to add an alarm that will go off at the chosen hour and minutes.

6.2 Software Subsystem Description

We have implemented an alarm in our coreWatch. The way the alarm works is as follows: when the fsm is in the STAND_BY state, we check if the alarm is activated and if the current time is the alarm time with the function CheckAlarmTime(). If so, it goes to the ALARM state by executing the function SoundsAlarm(). In this function, a timer is started with a TIMEOUT_ALARM timeout that we have set to 10 seconds so that our state machine only remains in the ALARM state for that time. Once this time has elapsed, a flag is activated and it returns to the STAND_BY state. The way in which the function SuenaAlarma() warns that it has passed to the ALARM state is by printing "Bip!" messages on the screen.

Finally, SET_ALARM is identical to SET_TIME, except that it introduces an extra digit that is used to set whether the alarm is off or on.



6.2.1 CheckAlarmTime()

Checks if the current system time is the alarm time, provided the alarm is enabled. Also, the system seconds must be zero so that it only runs once and not multiple times during the minute.

```
int CheckAlarmTime(fsm_t* p_this){
	CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
	int res=0;
	piLock(CLOCK_KEY);
	if(p_system->alarm. alarmOn==1){
	int alarmtime = p_system-> alarm. time. hh;
	int minalarm = p_system-> alarm. time. hh;
	int hoursis = p_system-> clock. hour. hh;
	int minsis = p_system-> clock. time. hh;
	int segsis = p_system-> clock. time. ss;
	if(horaalarma == horasis && minsis == minalarma && segsis == 0){
		res = 1;
	}
	}
	piUnlock(CLOCK_KEY);
	return res;
}
```

6.2.2 SoundAlarm()

Function that prints "Beep!" messages on the screen to indicate that the alarm has been activated. In addition, it starts a timer with timeout TIMEOUT_ALARM that will be used to know when to exit the ALARM state.

```
 void SoundsAlarm(fsm_t* p_this){
    CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
	tmr_t* tempalarm = tmr_new(tmr_timeout_alarm);
	p_system-> alarm. timer = tempalarm;
	tmr_startms(tempalarm, TIMEOUT_ALARM);
	piLock(STD_IO_LCD_BUFFER_KEY);
	lcdClear(p_system-> lcdId);
	lcdPosition(p_system-> lcdId, 0, 0);
	lcdPrintf(p_system-> lcdId, "Beep! Beep!");
	lcdPosition(p_system-> lcdId, 0, 1);
	lcdPrintf(p_system-> lcdId, "Beep! Beep!");
	piUnlock(STD_IO_LCD_BUFFER_KEY);
}
```

6.2.3 CheckTimeoutAlarm()

Function that checks if the alarm timeout flag has been activated.

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

6.2.4 tmr_timeout_alarm()

Enables the alarm timeout flag.

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

6.2.5 ExitAlarm()

Function executed when exiting the alarm state, whose function is to clear flags and clear the LCD.

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

6.2.6 PrepareSetAlarm(), ProcessDigitAlarm(),CancelSetNewAlarm(), SetNewAlarm()

They are identical to those used for SET_TIME, with the particularity that a last digit will be entered to indicate whether the alarm is activated or not.

```
void PrepareSetAlarm(fsm_t* p_this){
	CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
	p_system-> alarm. digitsSavedAlarm = 0;
	p_system-> alarm. tempAlarm = 0;
	piLock(SYSTEM_KEY);
	    g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_ALARM);
	    g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGIT_PULSED);
	    g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_ALARMA_IS_READY);
	piUnlock(SYSTEM_KEY);
	piLock(STD_IO_LCD_BUFFER_KEY);
	lcdClear(p_system-> lcdId);
	lcdPosition(p_system-> lcdId, 0, 0);
	lcdPrintf(p_system-> lcdId, "[SET_ALARM]");
	lcdPosition(p_system-> lcdId, 0, 1);
	lcdPrintf(p_system-> lcdId, "__:__");
	delay(1000);
	piUnlock(STD_IO_LCD_BUFFER_KEY);
}

void CancelSetNewAlarm(fsm_t* p_this){
	CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
	p_system-> alarm. digitsSavedAlarm = 0;
	p_system-> alarm. tempAlarm = 0;
	piLock(SYSTEM_KEY);
	    g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_ALARM);
```

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

void ProcessDigitAlarm(fsm_t* p_this){
        CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
        ClockType r = p_system-> clock;
        int format = r. time. format;
        int tempTime = p_system-> alarm. tempAlarm;
        int digitsSavedDigits = p_system-> alarm. digitsSavedAlarm;
        int lastDigit = g_coreWatch. digitDriven;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
        piUnlock(SYSTEM_KEY);
        if(format == 12){
        if(digitosGuardados == 0){
                lastDigit = min(1, lastDigit);
               tempTime = tempTime*10 + lastDigit;
        }else if(digitosGuardados == 1){
               if(tempTime == 0){
               lastDigit = max(1, lastDigit);
               }else{
                    lastDigit = min(2, lastDigit);
               }
                tempTime = tempTime * 10 + lastDigit;
        }else if(digitosGuardados == 2){
                tempTime = tempTime * 10 + min(5, lastDigit);
        }else if (digitosGuardados == 3){
                tempTime = tempTime*10+lastDigit;
        }else if (digitosGuardados == 4){
                lastDigit = min(lastDigit, 1);
               tempTime = tempTime*10+lastDigit;
        }else {
                tempTime = tempTime*10+lastDigit;
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PUSHED);
                piUnlock(SYSTEM_KEY);
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch | FLAG_NEW_ALARMA_IS_READY;
                piUnlock(SYSTEM_KEY);
        }
        }else{
        if(digitosGuardados == 0){
               lastDigit = min(2, lastDigit);
                tempTime = tempTime*10 + lastDigit;
        }else if(digitosGuardados == 1){
```

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

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

        if((format == 24 && digitsSaved == 4) || (format == 12 && digitsSaved == 5)){

        delay(500);
            lcdPosition(p_system-> lcdId, 0, 0);
            lcdPrintf(p_system-> lcdId, "[SET_ALARM]");
            lcdPosition(p_system-> lcdId, 0, 1);
            lcdPrintf(p_system-> lcdId, "OFF:0 / ON:1");
        }
        piUnlock(STD_IO_LCD_BUFFER_KEY);
        #endif
        p_system-> alarm. tempAlarm = tempTime;
        p_system-> alarm. digitsSavedAlarm = digitsSaved;
}

void SetNewAlarm(fsm_t* p_this){
        CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_NEW_ALARMA_IS_READY);
        piUnlock(SYSTEM_KEY);
```

```
        if (p_system-> clock. time. format == 12){
        int activated = p_system-> alarm. tempAlarm % 10;
        int ampm = (p_system-> alarm. tempAlarm / 10) % 10;
        int min = (p_system-> alarm. tempAlarm / 100) % 100;
        int time = p_system-> alarm. tempAlarm / 10000;
        if(ampm==1 && time!=12){
                hour+=12;
        }
        if(ampm==0 && time == 12){
                time = 0;
        }
            hour = hour*100 + min;
        SetTime(time, & p_system-> alarm. time);
            p_system-> alarm. alarmEnabled = enabled;
        }else{
        int time = p_system-> alarm. tempAlarm/10;
        int activated = p_system-> alarm. tempAlarm%10;
        SetTime(time, & p_system-> alarm. time);
            p_system-> alarm. alarmEnabled = enabled;
        }
        p_system-> tempTime = 0;
        p_system-> digitsSaved = 0;
}
```

7 Enhancement 5: System Password

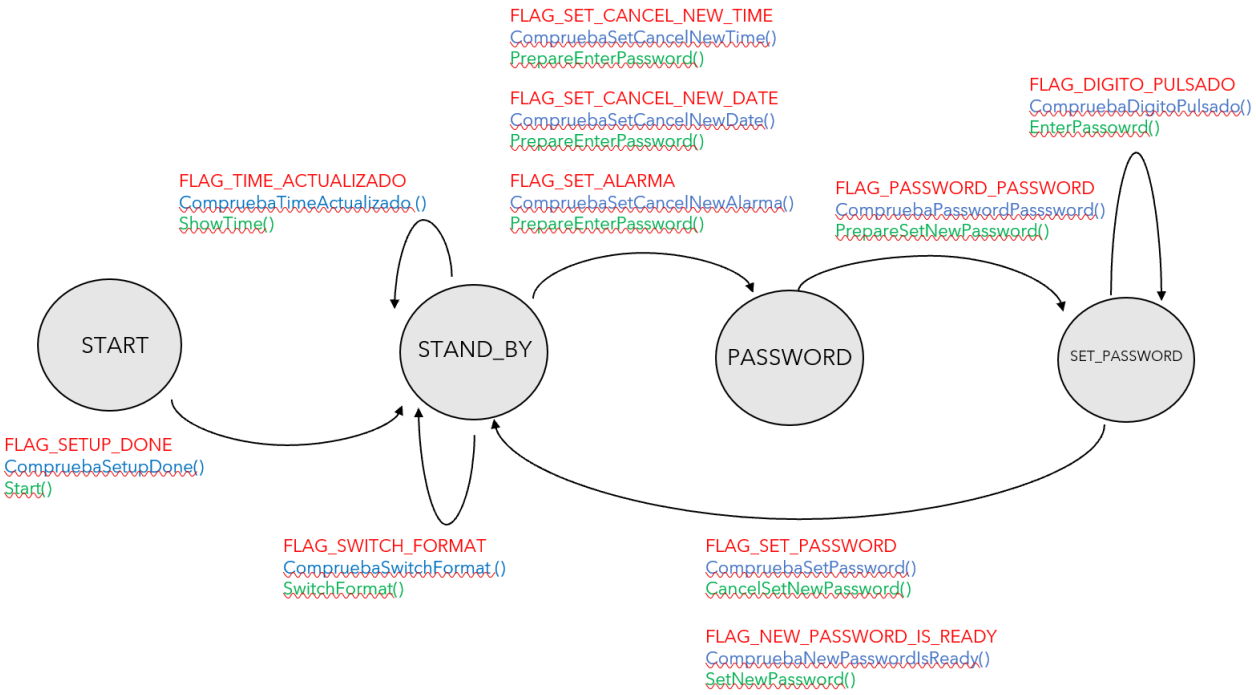
7.1 Improvement objectives

The purpose of this improvement is to require the user to enter a password ("Password" or PIN) before modifying the time, date, alarm and even the password itself.

This improvement aims to add more security to our system because if you don't enter the correct "Password" you won't be able to modify any of the modifiable variables (time, date or alarm) of CoreWatch and it will only be possible to view the time and date.

7.2 Software Subsystem Description

It consists of adding a password to the system that will be required to the user every time he wants to modify the time, date or alarm, that is, to access the SET_TIME, SET_DATE or SET_ALARM states respectively. For them, we change the state to which is passed when pressing the keys assigned to the system configuration states by the PASSWORD state in the fsm. In this state, the digits entered by the user are processed. Once entered, if they correspond to the system password, it allows access to the configuration states, and if not, the user is asked to enter the password again.



7.3 Execution and implementation of password routines

We believe that here it is interesting to gradually explain how this improvement has been implemented.

As modifications to the existing code, we have added two new states to the fsm state enumeration: PASSWORD and SET_PASSWORD.

```
in enum FSM_SYSTEM_STATES{

    START, STAND_BY, SET_TIME, SET_DATE, SET_ALARM, ALARM, ALARM, SET_PASSWORD,
PASSWORD

};
```

In addition, we have created a new struct, TipoPassword, which contains all the variables needed for the implementation. We have done it this way to keep some order in the code and have all the variables related to the password located in the same place. An instantiation of TypePassword will be stored inside TypeCoreWatch. The number variable is used to store the value of the password, the rest of the variables will be explained as they are needed.

```
typedef struct{
    int number;
    tmr_t* tmrPassword;
    int tempPassword;
    int digitsSavedPassword;
    int tempNewPassword;
    int tempNewPasswordRepeat;
}PasswordType;
```

Additionally, we have modified the state machine so that when FLAG_CANCEL_SET__TIME, FLAG_CANCEL_SET__DATE and FLAG_SET_ALARMA are activated, instead of going to the system configuration states, it goes to the PASSWORD state and executes the PrepareEnterPassword() function, in charge of dealing with zero initializations of certain variables and clearing flags in case they were activated.

```
void PrepareEnterPassword(fsm_t* p_this){
       CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
       p_system-> password. digitsSavedPassword = 0;
       p_system-> password. tempPassword = 0;
       piLock(SYSTEM_KEY);
           g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
           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_system-> lcdId);
           lcdPosition(p_system-> lcdId, 0, 0);
           lcdPrintf(p_system-> lcdId, "CHANGING");
           lcdPosition(p_system-> lcdId, 0, 1);
           lcdPrintf(p_system-> lcdId, "PASSWORD");
           delay(1000);
```

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

#endif
}
```

Once in the PASSWORD state, the digits entered are processed using the EnterPassword() function. The digits entered are stored in tempPassword, and the digitsSavedPassword variable is incremented with each digit saved. Once we reach 4 digits, we check if the password entered is equal to the system password. If it is, we proceed to activate the necessary flags to go to the configuration state. As all the configuration states have the same password, when FLAG_CANCEL_SET__TIME, FLAG_CANCEL_SET__DATE, FLAG_SET_ALARMA and FLAG_SET_PASSWORD were activated, we did not deactivate them in order to know at this moment which state the user wanted to access when going to the PASSWORD state. So, depending on which flags are active when the password entered is correct, we will activate FLAG_PASSWORD_TIME, FLAG_PASSWORD_DATE, FLAG_PASSWORD_ALARMA or FLAG_PASSWORD_PASSWORD_PASSWORD.

```
void EnterPassword(fsm_t* p_this){
        CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
        int tempPassword = p_system-> password. tempPassword;
        int digitsSavedPassword = p_system-> password. digitsSavedPassword;
        int lastDigit = g_coreWatch. digitDriven;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
        piUnlock(SYSTEM_KEY);
        if(digitosGuardadosPassword < 4){
            tempPassword = tempPassword*10 + lastPassword;
            digitsSavedPassword++;
        }
        if(digitosGuardadosPassword == 4){
        if(tempPassword == p_system-> password. number){
                piLock(SYSTEM_KEY);
                g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
                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_ALARM){
                    g_flagsCoreWatch = g_flagsCoreWatch | FLAG_PASSWORD_ALARMA;
               }
                piUnlock(SYSTEM_KEY);
       }else{
                digitsSavedPassword =0;
                tempPassword = 0;
                piLock(STD_IO_LCD_BUFFER_KEY);
                lcdClear(p_system-> lcdId);
                lcdPosition(p_system-> lcdId, 0, 0);
                lcdPrintf(p_system-> lcdId, "WRONG PASSWORD!");
                lcdPosition(p_system-> lcdId, 0, 1);
                lcdPrintf(p_system-> lcdId, "TRY AGAIN");
                delay(1000);
                lcdClear(p_system-> lcdId);
                lcdPosition(p_system-> lcdId, 0, 0);
                lcdPrintf(p_system-> lcdId, "ENTER");
                lcdPosition(p_system-> lcdId, 0, 1);
                lcdPrintf(p_system-> lcdId, "PASSWORD");
              piUnlock(STD_IO_LCD_BUFFER_KEY);
       }
       }
       p_system-> password. tempPassword = tempPassword;
       p_system-> password. digitsSavedPassword = digitsSavedPassword;
}
```

These flags are checked by their corresponding check functions, and depending on which one is activated, the PASSWORD state will exit from the PASSWORD state to the corresponding configuration state.

```
 int CheckPasswordTime(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 CheckPasswordDate(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 CheckPasswordAlarm(fsm_t* p_this){
```

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

int CheckPasswordPassword(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 Enhancement 6: SET_PASSWORD

8.1 Improvement objectives

We want the password added in the previous section to be modifiable by the user when pressing the 'F' key. As the keyboard is small we have had to remove the Reset functionality of this key because otherwise we had no more keys and we wanted to make this improvement.

8.2 Hardware Subsystem Description

SET_PASSWORD is the state we go to when we want to change the system password. To access, again we first go through the PASSWORD state, having to enter the current password. Once entered, the user is asked to type the same password twice. If the two passwords match, it is set as the password, otherwise the user is prompted to enter it another two times.

8.3 Description of subroutines

8.3.1 PrepareSetNewPassword() and CancelSetNewPassword()

They are in charge of handling flags and setting variables to zero, as well as some screen printing.

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

void CancelSetNewPassword(fsm_t* p_this){
       CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
       piLock(SYSTEM_KEY);
           g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_DIGIT_PULSED);
           g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_SET_PASSWORD);
           g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
       piUnlock(SYSTEM_KEY);
       p_system-> password. tempNewPassword = 0;
```

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

8.3.2 EnterNewPassword()

Before going to SET_PASSWORD, as a good configuration state, we go through PASSWORD and the user is asked to enter the current password (there is a custom print that says "ENTER CURRENT" only when the state you want to access is the SET_PASSWORD state, see the code of PreparaEnterPassword in the previous improvement). Once entered, PreparaEnterNewPassword is executed and goes to SET_PASSWORD, where the digits entered are processed by the EnterNewPassword() function and the user is asked to type the new password twice. The digits entered are processed by the EnterNewPassword() function. First 4 digits are entered, then a "REPEAT PLEASE" message is printed and then the other 4 digits are entered. The two passwords entered have been stored in the tempNewPassword and tempNewPasswordRepeat variables, and the digitsSavedPassword counter does not store exactly the total number of digits saved, otherwise it would not be possible to print the "REPEAT PLEASE" message on the screen. It increments to the first 4 digits, prints the message, increments again, and the digits of the second password go from 5 to 9. This was done because it was the most direct solution to a bug that printed the repeat message infinite times. Finally, if the passwords match, a flag is activated that will make the machine go to the STAND_BY state and execute SetNewPassword(). If they don't match, it starts entering the new passwords again.

```
void EnterNewPassword(fsm_t* p_this){
        CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
        int tempNewPassword = p_system-> password. tempNewPassword;
        int tempNewPasswordRepeat = p_system-> password. tempNewPasswordRepeat;
        int digitsSavedPassword = p_system-> password. digitsSavedPassword;
        int lastDigit = g_coreWatch. digitDriven;
        piLock(SYSTEM_KEY);
        g_flagsCoreWatch = g_flagsCoreWatch & (~ FLAG_DIGIT_PULSED);
        piUnlock(SYSTEM_KEY);
        if(digitosGuardadosPassword < 4){
            tempNewPassword = tempNewPassword*10 + lastPassword;
            digitsSavedPassword++;
        }
        if(digitosGuardadosPassword >= 5 && digitosGuardadosPassword < 9){
            tempNewPasswordRepeat = tempNewPasswordRepeat*10 + lastPassword;
            digitsSavedPassword++;
        }
        if(digitosGuardadosPassword == 9){
        if(tempNewPassword == tempNewPasswordRepeat){
                piLock(STD_IO_LCD_BUFFER_KEY);
                lcdClear(p_system-> lcdId);
                lcdPosition(p_system-> lcdId, 0, 0);
                lcdPrintf(p_system-> lcdId, "PASSWORD");
                lcdPosition(p_system-> lcdId, 0, 1);
                lcdPrintf(p_system-> 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_system-> lcdId);
                lcdPosition(p_system-> lcdId, 0, 0);
                lcdPrintf(p_system-> lcdId, "NOT MATCHING");
                lcdPosition(p_system-> lcdId, 0, 1);
                lcdPrintf(p_system-> lcdId, "TRY AGAIN");
                delay(1000);
                lcdClear(p_system-> lcdId);
                lcdPosition(p_system-> lcdId, 0, 0);
                lcdPrintf(p_system-> lcdId, "ENTER");
                lcdPosition(p_system-> lcdId, 0, 1);
                lcdPrintf(p_system-> lcdId, "NEW");
              piUnlock(STD_IO_LCD_BUFFER_KEY);
                digitsSavedPassword = 0;
                tempNewPassword = 0;
                tempNewPasswordRepeat = 0;
        }
        }

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

        p_system-> password. tempNewPassword = tempNewPassword;
        p_system-> password. tempNewPasswordRepeat = tempNewPasswordRepeat;
        p_system-> password. digitsSavedPassword = digitsSavedPassword;
}
```

8.3.3 SetNewPassword()

This function updates p_system-> password.number to the new password value. In addition to disabling certain flags.

```
void SetNewPassword(fsm_t* p_this){
        CoreWatchType *p_system = (CoreWatchType*)(p_this-> user_data);
        p_system-> password. number = p_system-> password. tempNewPassword;
        p_system-> password. tempNewPassword = 0;
        p_system-> password. tempNewPasswordRepeat = 0;
        p_system-> password. digitsSavedPassword = 0;
        piLock(SYSTEM_KEY);
            g_flagsCoreWatch = g_flagsCoreWatch & (~FLAG_NEW_PASSWORD_IS_READY);
        piUnlock(SYSTEM_KEY);
}
```

9 Improvement 7: Aesthetic changes in SET_TIME, SET_DATE AND SET_ALARM

9.1 Improvement objectives

The objective of this improvement is to facilitate the usability of the corewatch as well as an important aesthetic factor. We want to print a template for the LCD in which to introduce the time (__:__) and the date (__/__/____) and modify the blank spaces or "low bars" by the numbers that the user is introducing. In this way we intend to get the user to be able to visualize in a clearer way the date and time that is being entered.

9.2 Software Subsystem Description

It consists of printing on the screen aesthetically the number of date or time that is being introduced so that the user knows what number is introduced. We do this with lcdPutChar and the counter of digitsSaved.

This code for hours

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

And this one for date

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

1.

2.

3.

10 Improvement 8: Initial Setup at current time

2. Improvement objectives

Instead of using a default date and time on system initialization or reset, we want to use the current date and time.

3. Software Subsystem Description

Now, instead of setting a default time and date when initializing the clock, we will take the current time and date using the time.h module and the struct tm. By instantiating that struct in ResetReloj(), we can access the local time and set it.

4. Description of subroutines

ResetReloj(): We instantiate the struct tm with name timeinfo and a variable time_t rawtime. We obtain in timeinfo a struct from which we can get all the necessary parameters for the initialization, and we pass them to TipoHora and TipoCalendario.

```
void ResetClock(ClockType *p_clock){
	//Obtain current date and time
	time_t rawtime;
	struct tm * timeinfo;
	time(& rawtime);
	timeinfo = localtime(& rawtime);
	int currenttime = timeinfo-> tm_hour;
	int minActual = timeinfo-> tm_min;
	int segActual = timeinfo-> tm_sec;
	int currentDay = timeinfo-> tm_mday;
	int mesActual = timeinfo-> tm_mon +1; //REFERENCED FROM 0 TO 11
	int yearActual = timeinfo-> tm_year + 1900; //the year obtained this way is referenced to 1900
	int weekDay = timeinfo-> tm_wday;
	printf("Weekday: %d", weekDay);
	CalendarCalendarType calendar = {currentDay, currentMonth, currentYear, currentWeek, currentWeekDay};
	TimeTimeType time = {currentTime, currentTime, currentMin, currentSec, DEFAULT_TIME_FORMAT};
	p_clock-> calendar = calendar;
	p_clock-> time = time;
	p_clock-> timestamp = 0; //default timestamp
	piLock (CLOCK_KEY);
	    g_clockSharedVars. flags = 0;
	piUnlock (CLOCK_KEY);
```

```
}
```

11 Improvement 9: Show the day of the week

11.1 Improvement objectives

This enhancement aims to inform the user of the day of the week by printing it on the screen along with the date. To implement this functionality we will make use of the <time.h> library.

11.2 Software Subsystem Description

Thanks to the time module we used before, we can get the day of the week as well. So, inside the CalendarType in clock.h we are going to introduce a new parameter weekDay that represents the day of the week. When updating the date this number is made modulo 7 and incremented. When changing the date it is recalculated, calculating first the seconds since the epoch 1-1-1970 and doing "time = localtime(&secondsSinceEpoch)" to calculate the struct time and get the day of the week. To print it, we declare in CoreWatch.c an array char diassemanas[7], in which we introduce the letter of the day of the week and before printing the date, with sprintf(), we add this char at the beginning of the string.

11.2.1 Description of subroutines

UpdateDate() in clock.c, makes modulo 6 to the day of the week and adds 1 to it, since the day of the week is represented from 0 to 6 and if the last one has been reached we want to start again.

```
void UpdateDate(CalendarType *p_date){
	// Irrelevant this whole code for improvement
	int weekDay = p_date-> weekDay%6;
	weekDay++;
	p_date-> weekDay = weekDay;
	p_date-> MM = currentMonth;
	p_date-> dd = currentDay;
}
```

12 Main problems encountered

In terms of the main problems encountered, we can highlight the following.

When programming the improvement that allowed to change the AM-PM format, in the first versions of the project, we found a bug. When following the script to the letter and programming UpdateDate() as proposed in this script, when using the 12 hour format, the date change did not work correctly. This bug has been solved now, since we treat all the hours in 24 format internally and we show them in 24 or 12 format depending on what the user decides when pressing the letter 'C'.

When we were setting up the functions to print by the LCD the corresponding day of the week we made use of other functions that in general terms calculate the seconds from the "Epoch" to a certain date. It turns out that to perform such operations we have to use a high bit precision since we are taking very large numbers. Therefore, the best alternative was to use 64-bit integer variables to store those seconds and not lose precision. However, time_t variables don't even have that resolution, so after a certain date, when the precision number of these variables is exceeded, it stops working correctly.

13 User manual

A: change alarm time

B: exit

C: change format (being in STAND_BY)ç

D: change date

E: change time

F: change password

14 Bibliography

The bibliography for the improvements only includes the following web, consulted to know how to use the struct included in time.h to obtain the day of the week.

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

15 ANNEX I: Final project programme code