|  |  |
| --- | --- |
| **Title:** | **WINLIFT**  **SW Component < 1.0 >** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **History** | | | | |
| **Issue status**  (Index) | **Maturity/Date**  (draft/invalid/valid)  (dd-mmm-yyyy) | **Author**  Department | **Check/Release**  Department | **Description** |
| 1 | Draft  31-0ct-15 | Guillermo Ramirez  B.S. | Guillermo Ramirez  B.S. | Creation of the Software Design in repository |
| 2 | Draft  31-0ct-15 | Guillermo Ramirez  B.S. | Guillermo Ramirez  B.S. | Replacement of the SW Conceptual design diagram in section 4 and addition of component diagrams to 5 respectively. |
| 3 | Draft  31-0ct-15 | Oscar Miranda  B.S. | Oscar Miranda  B.S. | Modifications were made in chapter 5:Subtitles, Activity and sequence diagrams added. |
| 4 | Draft  31-0ct-15 | Oscar Miranda  B.S. | Oscar Miranda  B.S. | The activity diagram was modified because of some errors of the last version. |
| 5 | Draft  31-0ct-15 | Oscar Miranda  B.S. | Oscar Miranda  B.S. | Tha class diagram was added and also some functions in chapter 5 |
| 11 | Draft  2-Nov-15 | Oscar Miranda  B.S. | Oscar Miranda  B.S. | Only the return value of countPressTime was changed. |
| 12 | Draft  2-Nov-15 | Oscar Miranda  B.S. | Oscar Miranda  B.S. | References of requirements added. |
| 13 | Draft  2-Nov-15 | Guillermo Ramirez  B.S. | Guillermo Ramirez  B.S. | Functional Decomposition updated. Functions added and modification to some of the existing ones. Table of contents updated. |
| 15 | Draft  2-Nov-15 | Óscar Francisco  B.S. | Óscar Francisco  B.S. | Class diagram modified. Return values of the functions of sections 5.11 and 5.12 were changed, using naming convention. |
| 16 | Draft  2-Nov-15 | Óscar Francisco  B.S. | Óscar Francisco  B.S. | Description of the deployment, activity and class diagram modified and expanded. Class diagram was modified because of the name of the class was wrong and the elements of the e\_wState were missed. |
| 21 | Draft  12-Nov-15 | Óscar Francisco  B.S. | Guillermo Ramirez  B.S. | History chart was re-formatted. Definitions of GPIO, STM, ISR and SIUL were placed in Abbrevation part of section 2. Section 3 modified and divided into two sub-sections because the constraints were added. |
| 35 | Draft  21-Nov-15 | Óscar Francisco  B.S. | Guillermo Ramirez  B.S. | API, MAL and HAL sections added in 5-2 Functional decomposition. Some API and HAL function charts modified, and other functions added. |
| 36 | Draft  22-Nov-15 | Óscar Francisco  B.S. | Guillermo Ramirez  B.S. | 5.3 SCHEDULER section added which contains the use of the scheduler, how to set a clock tick, and to add tasks. And the functional decomposition charts of files Secheduler, Os\_Init, and Task\_Init. |
| 37 | Draft  22-Nov-15 | Óscar Francisco  B.S. | Guillermo Ramirez  B.S. | Kernel() description added in section 5.3.1.1 and section 5.6 updated, because there were some errors in the sections number. |
| 40 | Draft  23-Nov-15 | Guillermo Ramirez  B.S | Óscar Francisco  B.S. | State diagram and flow diagram of the functions added. Table of contents updated. |

**Table of Contents**

[1 Purpose 5](#_Toc436054814)

[2 Definitions and abbreviations 5](#_Toc436054815)

[3 Realization constraints and targets 5](#_Toc436054816)

[3.1 Targets 5](#_Toc436054817)

[3.2 Constraints 6](#_Toc436054818)

[4 SW Conceptual design 8](#_Toc436054819)

[5 SW Component internal breakdown 9](#_Toc436054820)

[5.1 Diagrams 9](#_Toc436054821)

[5.1.1 Use Case Diagram 9](#_Toc436054822)

[5.1.2 Class Diagram 10](#_Toc436054823)

[5.1.3 States Diagram 11](#_Toc436054824)

[5.2 Functional Decomposition 12](#_Toc436054825)

[5.2.1 SCHEDULER 12](#_Toc436054826)

[5.2.2 Tasks 13](#_Toc436054827)

[*5.2.3* *Scheduler* 14](#_Toc436054828)

[*5.3.1.1 void kernel(void)* 14](#_Toc436054829)

[*5.2.4* *Os\_init* 15](#_Toc436054830)

[*5.3.2.1 void Tick\_Flag(void)* 15](#_Toc436054831)

[*5.2.5* *Task\_init* 15](#_Toc436054832)

[*5.3.3.1* *void VALIDATE(void)* 15](#_Toc436054833)

[*5.3.3.2* *void CHECK\_MANUAL\_AUTO(void)* 16](#_Toc436054834)

[*5.3.3.3* *void ANTI\_PINCH(void)* 17](#_Toc436054835)

[*5.3.3.4* *void STATE\_MACHINE(void)* 18](#_Toc436054836)

[*5.3* *API* 18](#_Toc436054837)

[5.3.1 *WL\_AppLayer* 18](#_Toc436054838)

[*5.4.1.1* *T\_ULONG WL\_CheckValid(T\_UBYTE channel)* 18](#_Toc436054839)

[*5.4.1.2 T\_ULONG WL\_CheckAutoManual(void)* 19](#_Toc436054840)

[*5.4.1.3* *void WL\_WinMUp(void)* 20](#_Toc436054841)

[*5.4.1.4* *void WL\_WinMDw(void)* 21](#_Toc436054842)

[*5.4.1.5* *void WL\_WiAUp(void)* 23](#_Toc436054843)

[*5.4.1.6* *void WL\_WinADw(void)* 24](#_Toc436054844)

[*5.4.1.7* *void WL\_A\_Pinch(void)* 25](#_Toc436054845)

[*5.4.1.8* *void blockButtons(void)* 25](#_Toc436054846)

[5.3.2 *State\_Machine* 26](#_Toc436054847)

[*5.4.2.1 void StateMachine(void)* 26](#_Toc436054848)

[*5.4* *HAL* 26](#_Toc436054849)

[5.4.1 *Global\_Init* 26](#_Toc436054850)

[*5.6.1.1 void Global\_Init(void)* 26](#_Toc436054851)

[*5.6.1.2 void STM\_init(void)* 27](#_Toc436054852)

[*5.6.1.3 void initModesAndClock(void)* 27](#_Toc436054853)

[*5.6.1.4 void GPIO\_Init()* 27](#_Toc436054854)

[*5.6.1.5 void GPIO\_En(void)* 27](#_Toc436054855)

[*5.6.1.6 void INTC\_\_init(void)* 27](#_Toc436054856)

[5.4.2 *GPIO\_APP* 28](#_Toc436054857)

[*5.6.2.1 T\_UBYTE GPIO\_GetState(T\_UWORD ch)* 28](#_Toc436054858)

[*5.6.2.2 void GPIO\_SetState(T\_UWORD ch, T\_UBYTE value)* 28](#_Toc436054859)

# Purpose

The purpose of this project is develop a software, for an embedded system, that controls a car window movement, with anti-pitch security function. The software will be implemented in a MPC5606B Freescale development board.

# Definitions and abbreviations

**Definitions**

|  |  |
| --- | --- |
| WINLIFT | Name of the project, which means Window Lifter |
| API | May refer to Application Programming Interface or Application Layer |
| HAL | Hardware Abstraction Layer |
| MAL | Middleware Application Layer |
|  |  |

**Abbreviations**

|  |  |
| --- | --- |
| GPIO  ISR  STM  SIUL | General purpose inputs and outputs  Interrupt Service Routine  System Timer Module  System Integration Unit Lite |

**References**

|  |  |  |
| --- | --- | --- |
| **N°** | **Document name** | **Reference** |
| 1 | Traceability Matrix Template Rev. 35 | 1 |
| 2 | MPC5606B Reference Manual 7.1 | 2 |
|  |  |  |
|  |  |  |

# Realization constraints and targets

## Targets

The project has several functionalities to control the window, here is an overview of the principal ones:

* It will have a function that control the opening of the window. **Req. 2.1**
* It will have a function that control the closure of the window. **Req. 2.2**
* When opening or closing the window there will be an indicator LED indicating the process in progress. **Req. 2.8**
* There will be a function that counts how much time a button have been pressed and validate the press. **Req. 3 and Req. 3.1**
* It will have an anti-pinch functionality, declared as interruption, which will stop the closure of the window and will open it. This functionality is for security purposes. **Req. 4.3**
* When anti-pinch is active, a sub-function will disable all inputs for 5 seconds. **Req. 4.6**

## Constraints

A possible constraint could be the PowerPC architecture of the hardware where the software will be implemented, which is a development board MPC5606B of Freescale. Here are some concerning specifications and a block diagram that could help when trying to export the project into another platform:

* MPC5606B MCU in a 144LQFP package.
* On-board JTAG connection via open source OSBDM circuit using the MPC9S08JM MCU
* Operating Frequency (Max): 64 MHz.
* Total DMA Channels 16.
* Internal Flash (KB): 512
* GPIOs: 149.
* EEPROM: 64 KB DataFlash®
* RAM: Up to 96 KB
* Timer: 16 bits up to 64 channels

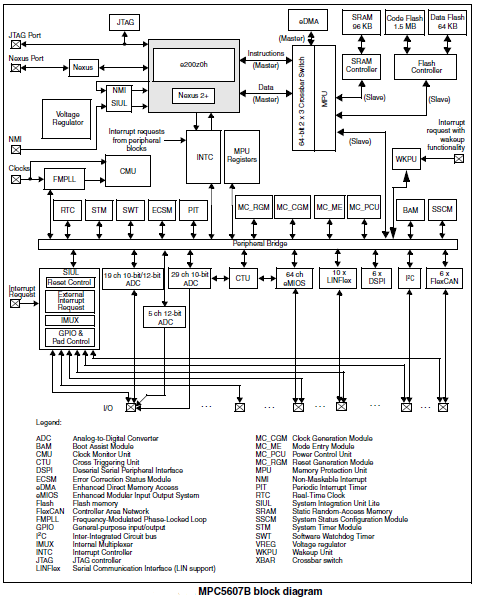
The principal modules (see block diagram) that are used in this project:

* SIUL:

-to configure GPIO and Pad Control (software control of external pins), used to declare the 10 output pins that will be connected to the LED’s that simulate the window and 3 inputs: closing/open/anti-pinch switches,

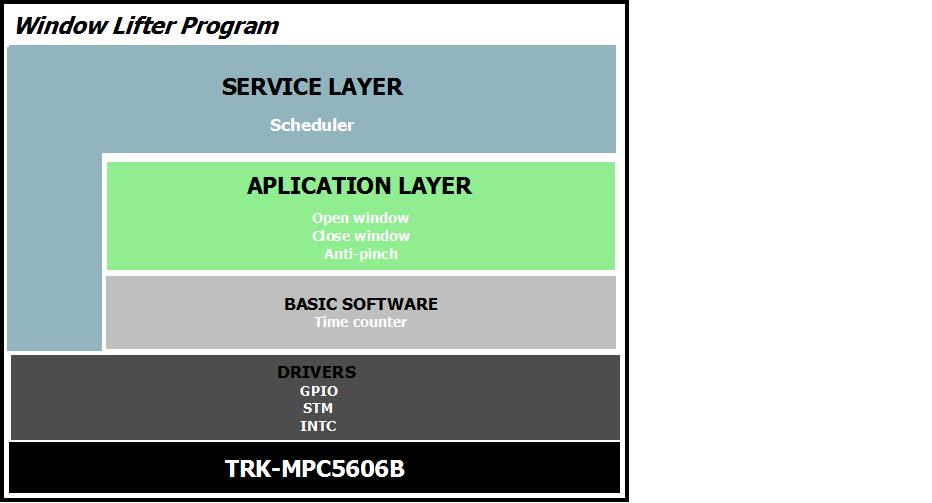
-and an ISR (used to make possible anti-pinch functionality)

* STM: used for timing purposes, that is, to configure the transitions of each led, the validation of a press button, and for deciding either to active an automatic opening/closure or a manual opening/closure. In total, the four channels of the STM are used.
* INTC: used to define the anti-pinch interrupt function.



# SW Conceptual design

The next diagram represents the inputs and outputs of the WINLIFT’s conceptual design and the general tasks that must be performed.

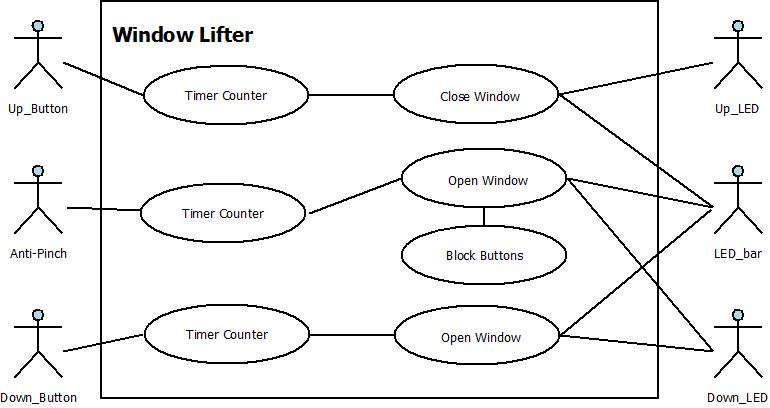


# SW Component internal breakdown

## Diagrams

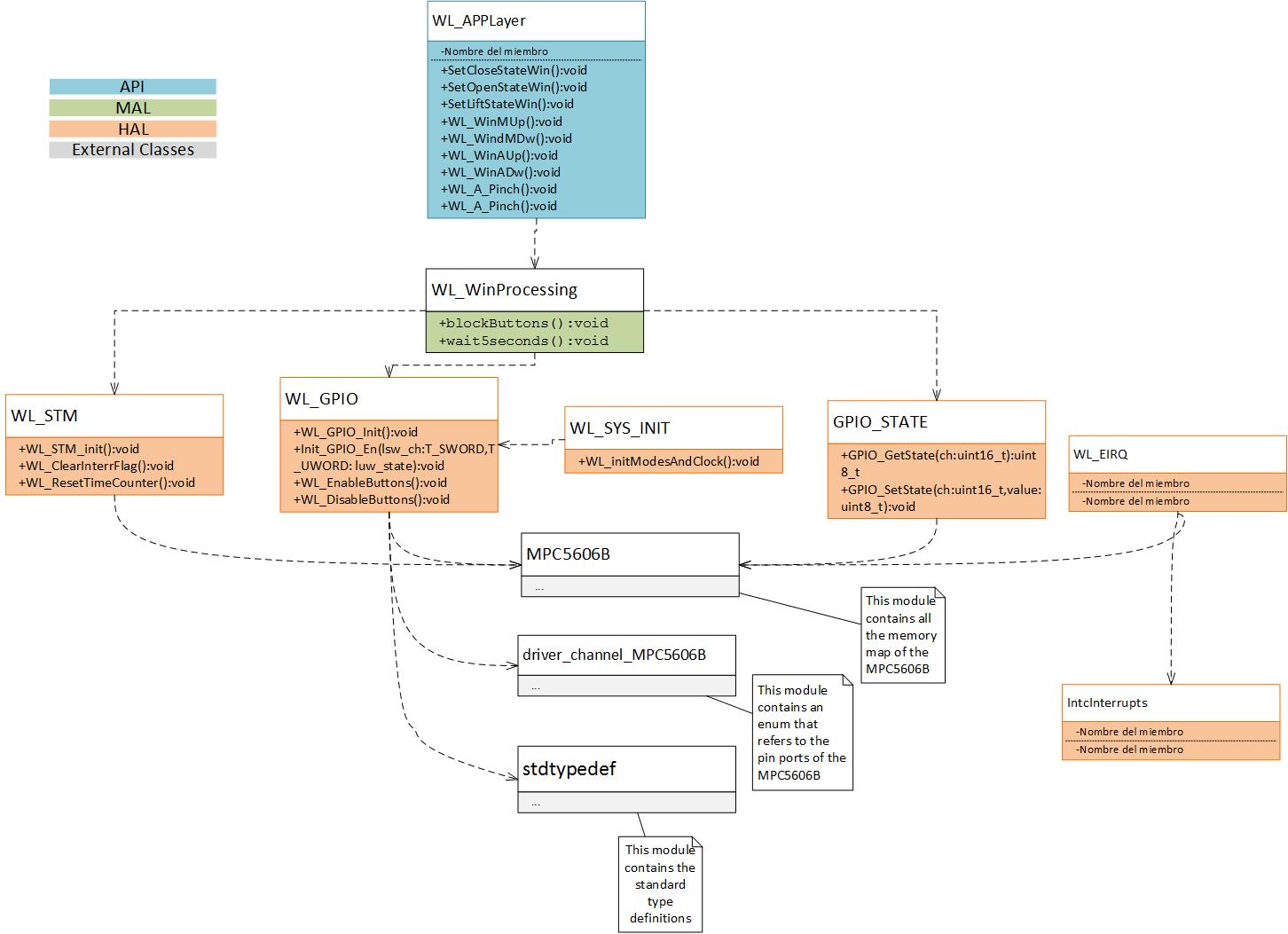
### Use Case Diagram

The following Use Case diagram describes the general interaction between the main actor and the function that will be added to the program. It describes the over-all behavior of the window lift system.



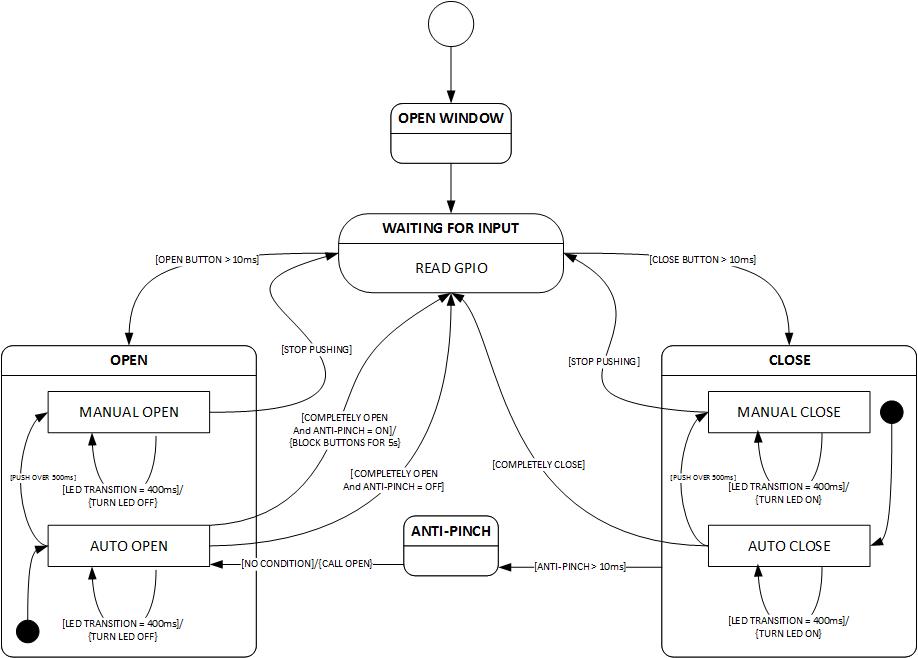
### Class Diagram

The class represents the API of WINLIFT. It has 10 methods and 1 attribute called e\_wState that is an enum and refers to the state of the window and also has three values: CLOSE, OPEN and LIFTING. For more details of the functions go to section 5.2.



### State Diagram

The flow of the state machine is showed in the next image. First, the initial state of the window is OPEN WINDOW. The immediate next state will be the idle or WAITING FOR INPUT, where the readings of the switches. If the close button is pressed and validated the flow of the program will pass to CLOSE state, in this state, the automatic or manual closing of the window will be determined in the MANUAL CLOSE and AUTO CLOSE. If the button is release while the manual routine is executing the flow is passed to WAITING FOR INPUT. If the anti pinch button is pressed, the flow passes to ANTI-PINCH state and then the window will be automatic open. After that the buttons will be blocked. Once the 5 seconds has passed the flow is passed to WAITING FOR INPUT. A similiar routine will execute if the OPEN BUTTON is pressed, but for opening the window, with the difference that OPEN state can't move to the ANTI-PINCH state.



## Functional Decomposition

### SCHEDULER

#### Use of the scheduler

The scheduler controls the tasks with delays that are defined by the clock tick. To have a good performance of each task, a task must be small enough to fit within the clock tick. If a task is larger than the clock tick, it would be pause and will be dispatched in the next clock tick if other tasks are waiting. A task offset is necessary to avoid executing tasks at the same time, but this only reduces the possibility of such case, it doesn’t eliminate it.

#### Clock Tick

The clock tick is set in the timer module (STM) of the hardware that runs at 16 MHz (this value can be modified) and an ISR function is used.

To establish the clock tick use the CMP register of the STM module using the following formula:

Were Clock Tick Hex is a hexadecimal value, Clock Speed Hex is the hexadecimal magnitude of the clock speed, and Desired Clock Tick Time is the value in seconds of the Clock Time.

To keep the control of the clock tick a header, OS\_Init.h, is used. It has a variable used to mark the deadline of the tick and the interrupt function Tick\_Flag() that clears the ub\_TickFlag, the clock interrupt flag of the STM, and reset the CNT register. This function is shown in the OS\_Init.c file of the next page.

#### OS\_Init.h

**T\_UBYTE ub\_TickFlag;**

**void** **Tick\_Flag**(**void**);

#### OS\_Init.c

**void** **Tick\_Flag**(**void**)

{ /\* -------------------------------------------------------------------

\* Name : Tick\_Flag

\* Description : Check if the channel 0 of the STM as

reached 10ms and raise a flag when

reached

\* Parameters : void

\* Return : void

\* ----------------------------------------------------------------

\*/

**if** (STM.CH[0].CIR.B.CIF)

{

ub\_TickFlag = 1; /\* Clear tick clock flag \*/

STM.CH[0].CIR.B.CIF = 1; /\* Clear interrupt flag \*/

STM.CNT.R = 0; /\*Reset counter\*/

}

}

### Tasks

To add a task:

In the TASK\_Init.h add as much enums as necessary inside E\_TASK enum, the most important constant is E\_TASK\_NUM that tells the number of the total tasks of the scheduler, this one must not be modified or moved as its used in other modules of the program.

Each task is defined by the following:

A structure S\_TASK has three members:

* A handler called rp\_Tasks to call the task,
* A 32 bit rul\_Period, which is used to define the period of such task,
* A 32 bit rul\_Offset, which is used to define the task’s offset.

The tasks are declared and defined in both, the TASK\_Init.h and Task\_Init.c files:

#### TASK\_Init.h

**typedef** **struct** {

**void**(\* rp\_Tasks)(**void**); /\*Pointer that'll call every task\*/

T\_ULONG rul\_Period; /\*Period of task\*/

T\_ULONG rul\_Offset; /\*Offset of task\*/

}S\_TASK;

**typedef** **enum**{

*E\_TASK1*,

*E\_TASK2*,

…/\* NEW TASK MUST BE ADDED HERE \*/ …

*E\_TASK\_NUM*

}E\_TASK;

In the TASK\_Init.c, an array of S\_TASK variables must be used to define the tasks pointer, and to define the periods and offsets the format will be the following:

#### TASK\_Init.c

**#define** taskPeriod1 XXXX

… …

**#define** taskPeriodN XXXX

**#define** taskOffset1 XXXX

… …

**#define** taskOffsetN XXXX

**const** S\_TASK taskName[*E\_TASK\_NUM*] = {

{taskName1, taskPeriod1, taskOffset1 },

{taskName2, taskPeriod2, taskOffset2 },

{taskName3, taskPeriod3, taskOffset3 },

… … … …

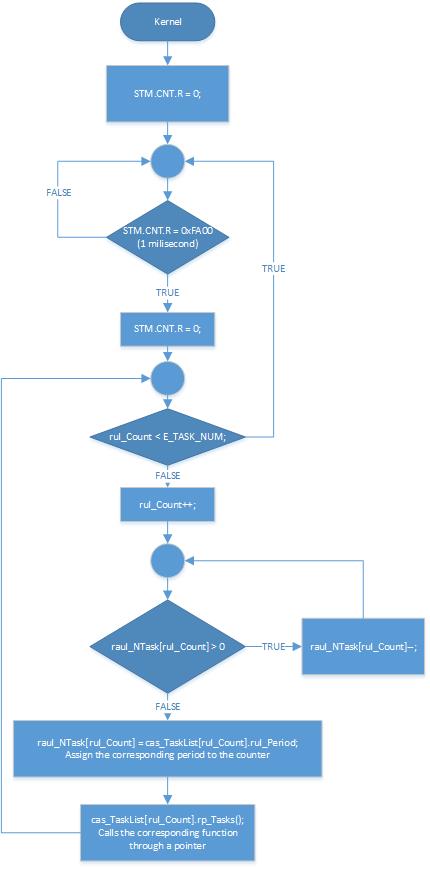
{taskNameN, taskPeriodN, taskOffsetN }

};

Note that E\_TASK\_NUM represents the number of tasks, in this case, four tasks, and that the period and the offset are numeric values.

### Scheduler

### 5.3.1.1 void kernel(void)



|  |  |
| --- | --- |
| **Description** | The scheduler is contained inside a function called kernel. This function starts by doing an indexing of the task, assigning the offset. This process occurs only once in each execution of the code.  Then an infinite loop starts, the tick of the scheduler is defined by a flag called ub\_TickFlag, which was configured to raise every millisecond. Once it was acquired it is restarted to zero.  Now, inside the if there is a for cycle which who through all the task accessing to their offset with and if, whenever the offset is over zero it will decrease, for every task. When one of the offset, of whichever task, reach zero, the else of the if will be active and will stablish the period of the task and will call it using the pointer previously defined. |
| **Parameters** | void |
| **Return Value** | *void* |
| **Precondition** | ------------ |
| **Post condition** | ------------------ |
| **Error Conditions** | *Not defined* |
| **Requirement** | -------- |

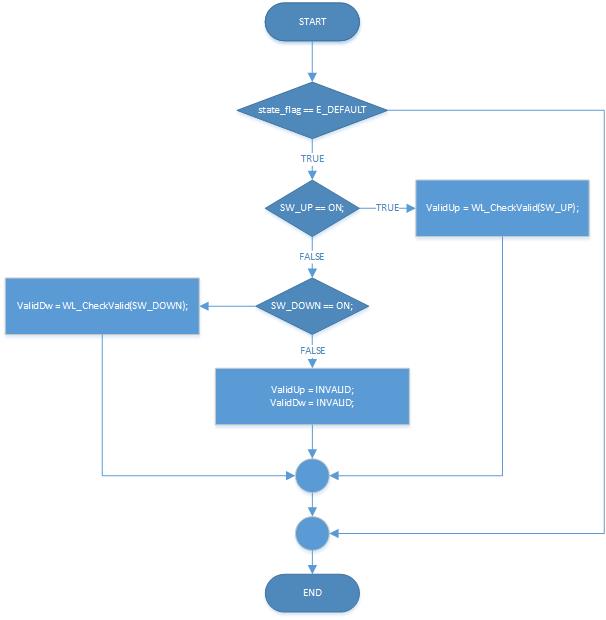
### Os\_init

### 5.3.2.1 void Tick\_Flag(void)

|  |  |
| --- | --- |
| **Description** | Check if the channel 0 of the STM as reached 1ms  and set a flag called ub\_TickFlag when reached. |
| **Parameters** | void |
| **Return Value** | *void* |
| **Precondition** | The counter value of the STM is equal to the compare value of channel 0 and counter interrupt flag has been set. |
| **Post condition** | ------------------ |
| **Error Conditions** | *Not defined* |
| **Requirement** | -------- |

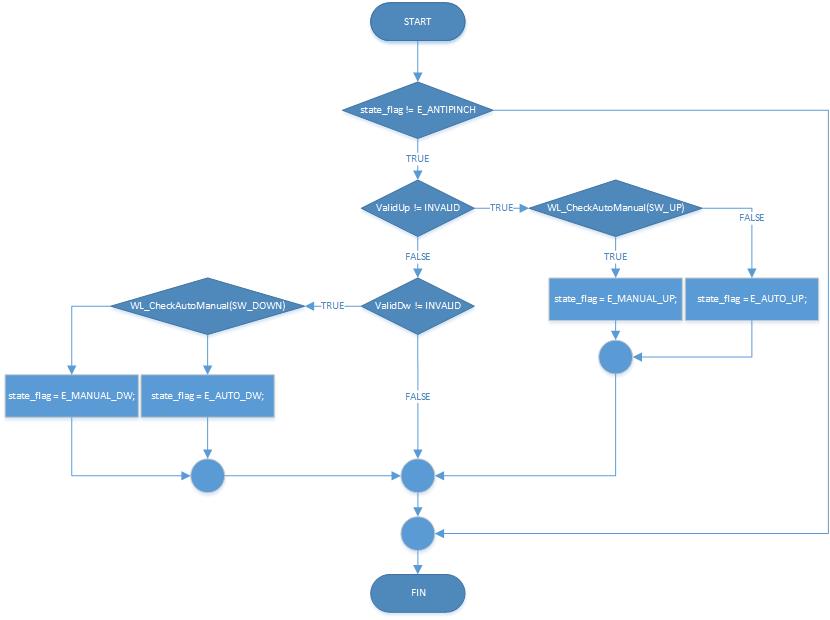
### Task\_init

### 5.3.3.1 void VALIDATE(void)



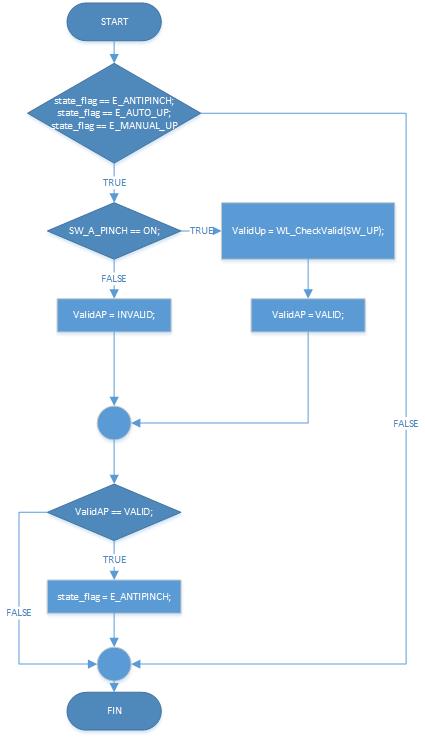
|  |  |
| --- | --- |
| **Description** | Read the state of the UP and DOWN switches, if  one of them is on it will call the WL\_CheckValid  function to verify if its valid. This will happen  as long as the state\_flag is E\_DEFAULT, this to  prevent that the function is called once it had been  validated and the window is in movement. |
| **Parameters** | No parameters |
| **Return Value** | *No return value* |
| **Precondition** | Either is the first task the scheduler calls or task STATE\_MACHINE() has been called by the scheduler. And UP or DOWN are on. |
| **Post condition** | Task ANTI\_PINCH is the next task that the scheduler will select |
| **Error Conditions** | *Not defined* |
| **Requirement** | 3.1 |

### 5.3.3.2 void CHECK\_MANUAL\_AUTO(void)



|  |  |
| --- | --- |
| **Description** | Once the button had been validated, read the state  to verify that is still on, if one of them is on  it will call the WL\_CheckAutoManual To change the  state\_flag to the corresponding MANUAL or AUTO routine.  This will happen as long as the state\_flag is different  from E\_ANTIPINCH, this to prevent that the function is  called during the anti-pinch routine. |
| **Parameters** | No parameters |
| **Return Value** | *No return value* |
| **Precondition** | Task VALIDATE() has been called by the scheduler. And a button is on. |
| **Post condition** | Task ANTI\_PINCH is the next task that the scheduler will select |
| **Error Conditions** | *Not defined* |
| **Requirement** | 3.3 |

### 5.3.3.3 void ANTI\_PINCH(void)



|  |  |
| --- | --- |
| **Description** | This function can only be called if the window is  moving up (closing), or if the anti-pinch has already  been activated. It will read the state of the SW\_A\_PINCH  if its on, it check if its a valid press. Once it has been  validated it will change the state\_flag to E\_ANTIPINCH |
| **Parameters** | No parameters |
| **Return Value** | *No return value* |
| **Precondition** | Task CHECK\_MANUAL\_AUTO() has been called by the scheduler. And the window is lifting. |
| **Post condition** | Task STATE\_MACHINE() is the next task the scheduler will select |
| **Error Conditions** | *Not defined* |
| **Requirement** | 3.3, 4.3, 4.4, 4.5 and 4.6 |

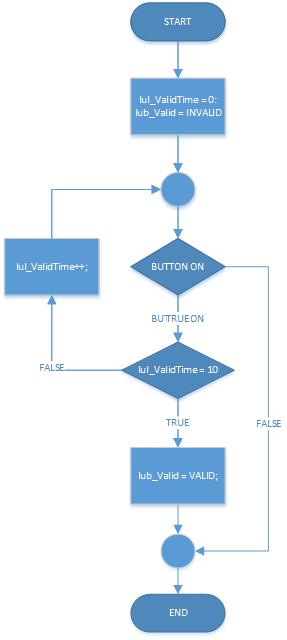
### 5.3.3.4 void STATE\_MACHINE(void)

|  |  |
| --- | --- |
| **Description** | This function calls the state machine that controls  the window movement, depending on the flags defined  by the previous functions. It will always be called. See function 5.4.2.1 StateMachine() |
| **Parameters** | No parameters |
| **Return Value** | *No return value* |
| **Precondition** | Task ANTI\_PINCH() has been called by the scheduler. |
| **Post condition** | Task VALIDATE() is the next task the scheduler will select |
| **Error Conditions** | *Not defined* |
| **Requirement** | ------- |

## ***API***

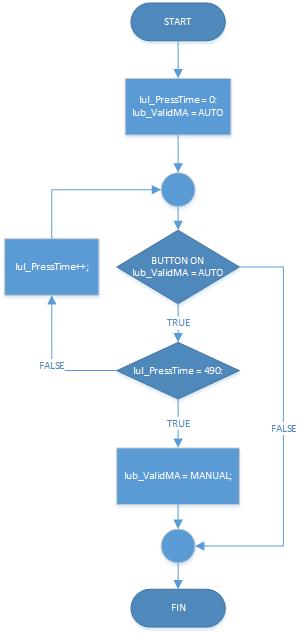
### ***WL\_AppLayer***

### 5.4.1.1 T\_ULONG WL\_CheckValid(T\_UBYTE channel)



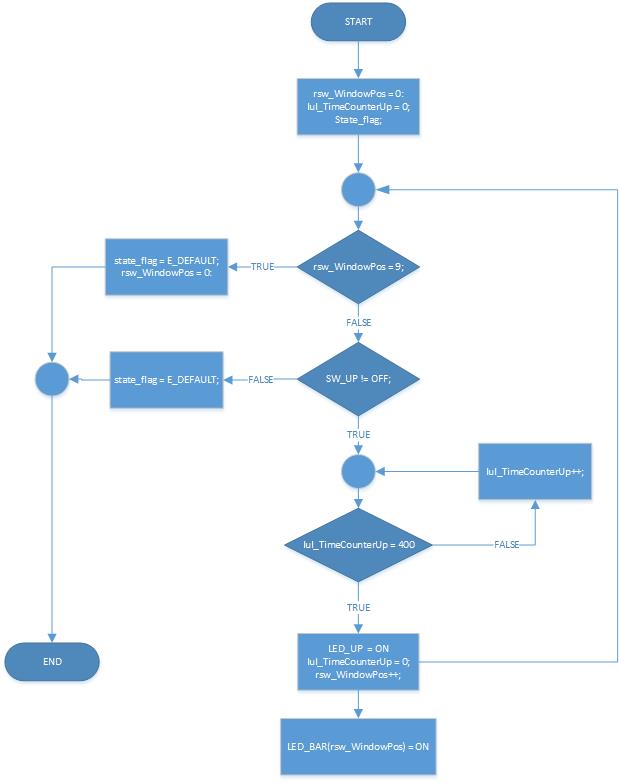
|  |  |
| --- | --- |
| **Description** | As long as a button is press, check if the button  press is valid (to avoid debounce), the button  verifies is the one specified in channel. If it  have not reached yet increase the lul\_ValidTime  by one, every time that the function is called,  and returns a invalid value in the lub\_Valid  variable. When the counter reach 10, lub\_Valid  return a valid value |
| **Parameters** | T\_UBYTE channel  The GPIO register number of the push buttons, up(close), down(open) and anti-pinch |
| **Return Value** | *T\_ULONG lub\_Valid*  *This variable determines whether a press button is valid (greater than 10 ms) or invalid (lower than 10 ms)* |
| **Precondition** | up(close), down(open) or anti-pinch button must have been pressed |
| **Post condition** | ------------------ |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 3.1 |

### ***5.4.1.2 T\_ULONG WL\_CheckAutoManual(void)***



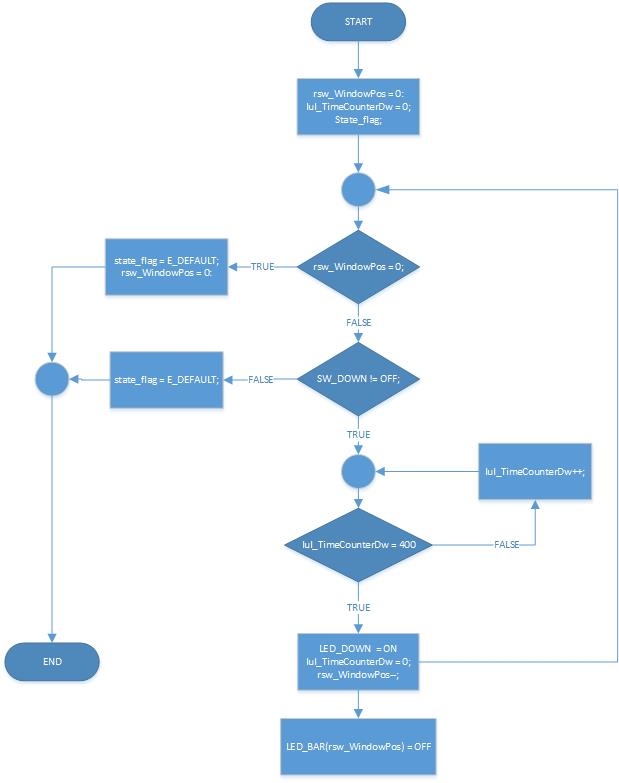
|  |  |
| --- | --- |
| **Description** | As long as a button is press, and it has been  validated, check if the routine to execute is  manual or automatic, this will be for the  movement specified in channel (Up or Down).  If the counter (lul\_PressTime) do not reached 490  counts (which will be 500 when the 10 in WL\_CheckValid()  are counted) is AUTO. But if the counter (lul\_PressTime)  reached 490 counts it will be manual. |
| **Parameters** | T\_UBYTE channel  The GPIO register number of the push buttons, up(close), down(open) and anti-pinch |
| **Return Value** | *T\_ULONG lub\_ValidMA*  *Determines either to execute a manual action or an automatic action* |
| **Precondition** | Up (Close) or Down (Open) button pressed and the button press was validated by WL\_CheckValid() |
| **Post condition** | *Prepares the window to move manually or automatically using the variable lub\_ValidMA* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 3.3 |

### ***5.4.1.3 void WL\_WinMUp(void)***



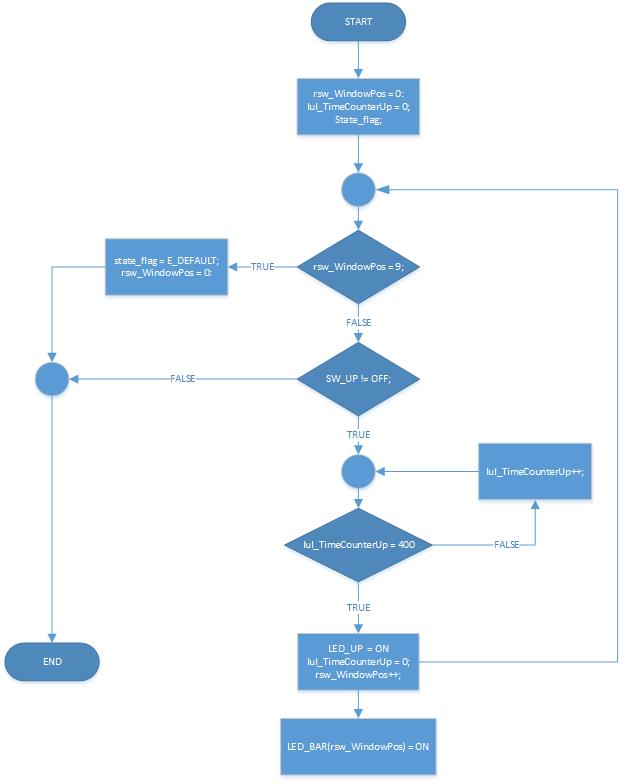
|  |  |
| --- | --- |
| **Description** | Close the window manually by making a "sweep"  through the LED\_BAR, as long as rsw\_WindowPos  is bigger than NUMLEDS, which means that the window  is not completely close yet. As long as the SW\_UP  button is pressed the sweep will turn on a LED  every 400 counts of lul\_TimeCounterUp. Once it reaches  400 the counter is reset and rsw\_WindowPos increase by  one, to turn on a different LED the next time.  While the button is pressed, state\_flag, will be E\_MANUAL\_UP,  but when is completely close, or the button is not pressed any  more, state\_flag will be E\_DEFAULT. |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Only can be called when the when up button is pressed during 500 ms |
| **Post condition** | *Leds’ transition down-to-up executes* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 2.6, 3.3 |

### ***5.4.1.4 void WL\_WinMDw(void)***



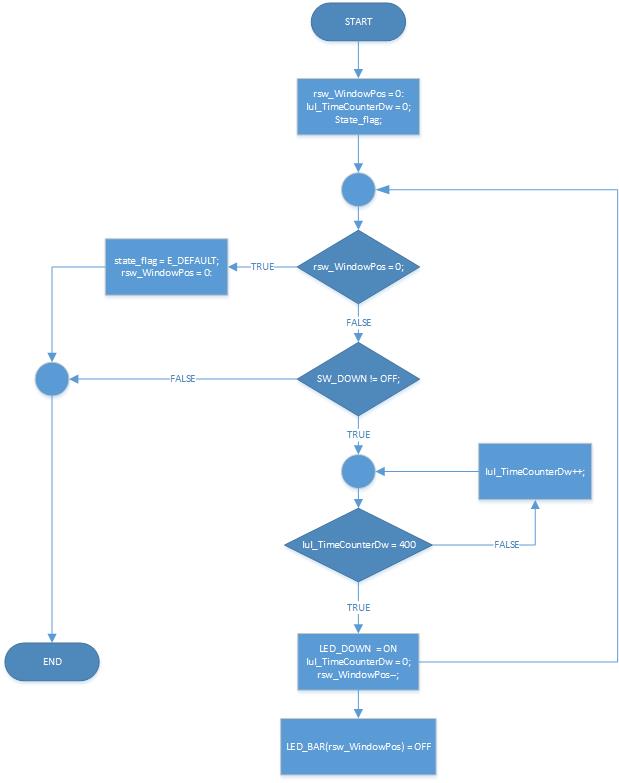
|  |  |
| --- | --- |
| **Description** | Open the window manually by making a "sweep"  through the LED\_BAR, as long as rsw\_WindowPos  is minor than NUMLEDS, which means that the window  is not completely open yet. As long as the SW\_DOWN  button is pressed the sweep will turn off a LED  every 400 counts of lul\_TimeCounterDw. Once it reaches  400 the counter is reset and rsw\_WindowPos decrease by  one, to turn off a different LED the next time.  While the button is pressed, state\_flag, will be E\_MANUAL\_DW,  but when is completely open, or the button is not pressed any  more, state\_flag will be E\_DEFAULT. |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Only can be called when the down(open) button is pressed more than 500 ms |
| **Post condition** | *Leds’ transition up-to-down executes* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 2.6, 3.3 |

### ***5.4.1.5 void WL\_WiAUp(void)***



|  |  |
| --- | --- |
| **Description** | Close the window automatically by making a "sweep"  through the LED\_BAR, as long as rsw\_WindowPos  is bigger than NUMLEDS, which means that the window  is not completely close yet. It will turn on a LED  every 400 counts of lul\_TimeCounterUp. Once it reaches  400 the counter is reset and rsw\_WindowPos increase by  one, to turn on a different LED the next time.  While the window is not close yet, state\_flag, will be E\_MANUAL\_UP,  but when is completely close, state\_flag will be E\_DEFAULT. |
| **Return Value** | *There is no return value* |
| **Precondition** | Only can be called when the when up(close) button is pressed between 10ms (valid press) and 400 ms |
| **Post condition** | *Leds’ transition down-to-up executes* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 2.6, 3.3 |

### ***5.4.1.6 void WL\_WinADw(void)***

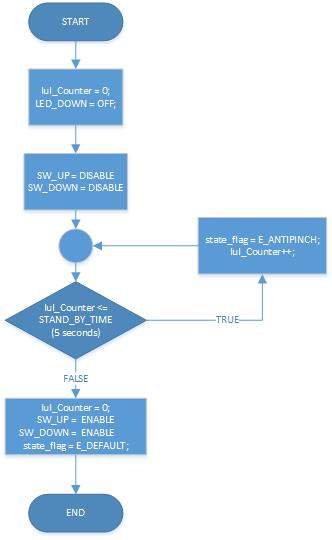


|  |  |
| --- | --- |
| **Description** | Open the window automatically by making a "sweep"  through the LED\_BAR, as long as rsw\_WindowPos  is minor than NUMLEDS, which means that the window  is not completely open yet. It will turn on a LED  every 400 counts of lul\_TimeCounterDw. Once it reaches  400 the counter is reset and rsw\_WindowPos decrease by  one, to turn off a different LED the next time.  While the window is not open yet, state\_flag, will be E\_MANUAL\_DW,  but when is completely open, state\_flag will be E\_DEFAULT. |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Only can be called when the down button is pressed between 10 ms (valid press) and 400 ms |
| **Post condition** | *Leds’ transition up-to-down executes* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 2.6, 3.3 |

### ***5.4.1.7 void WL\_A\_Pinch(void)***

|  |  |
| --- | --- |
| **Description** | This function can only be called when the window is  going up, it ensures that the LED\_UP is turned off.  Then open the window automatically using WL\_WinADw, when  the window is completely open calls the blockButtons  function. |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Up (close) button pressed and WL\_WinAUp() or WL\_WinMUp() were executed |
| **Post condition** | *Block all inputs during 5 seconds using blockButtons()* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 3.1, 4.3, 4.4, 4.5 4.6 |

### ***5.4.1.8 void blockButtons(void)***



|  |  |
| --- | --- |
| **Description** | Disable the inputs buttons and creates and wait 5s  before re-enable them. Every time the function is called  lul\_Counter will increase by one, if it has not reached  5s (or 5000 counts) state\_flag will be E\_ANTIPINCH. When  it reached the 5s the buttons re-enable and state\_flag changes  to E\_DEFAULT. |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | rsw\_WindowPos is equal to zero when WL\_A\_Pinch() was executed |
| **Post condition** | *Enable all the inputs* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 4.6 |

### ***State\_Machine***

### ***5.4.2.1 void StateMachine(void)***

|  |  |
| --- | --- |
| **Description** | It calls a different state, or window movement case,  depending on the value of state\_flag, which is  established by the tasks in the Scheduler. There are 6 states: E\_MANUAL\_UP, E\_AUTO\_UP, E\_MANUAL\_DW, E\_AUTO\_DW, E\_ANTIPINCH and E\_DEFAULT |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | ---- |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | 3.3, 4.3,4.4,4.5,4.6 |

E\_MANUAL\_UP refers to a manual closing of the window.

E\_AUTO\_UP refers to an automatic closing of the window

E\_MANUAL\_DW refers to a manual opening of the window

E\_AUTO\_DW refers to automatic opening of the window

E\_ANTIPINCH refers to the anti\_pinch functionality of the window

E\_DEFAULT refers to the default state of the window, where no indicator leds are on.

## ***HAL***

### ***Global\_Init***

### ***5.6.1.1 void Global\_Init(void)***

|  |  |
| --- | --- |
| **Description** | Initializes the modes and clock of the system, the STM module, the GPIO ports, and the software interrupts of INTC |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Main(has started) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | ----- |

### ***5.6.1.2 void STM\_init(void)***

|  |  |
| --- | --- |
| **Description** | Enables the STM module and configures the prescaler, the channel 0, and the compare registers |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | ------ |

### ***5.6.1.3 void initModesAndClock(void)***

|  |  |
| --- | --- |
| **Description** | Enables the run mode and the signal clock speed to 64 MHz |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | ------ |

### ***5.6.1.4 void GPIO\_Init()***

|  |  |
| --- | --- |
| **Description** | Enables the pins –using GPIO\_En() function- that will be used for the led bar, the led up and down indicators as outputs and the up(close), down(open), and anti-pinch buttons as inputs. |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | 1.2, 2.1, 2.7 and 4.2 |

### ***5.6.1.5 void GPIO\_En(void)***

|  |  |
| --- | --- |
| **Description** | Enables a pin of the GPIO that will be used |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | ------ |

### ***5.6.1.6 void INTC\_\_init(void)***

|  |  |
| --- | --- |
| **Description** | Enables software interrupts |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | ------ |

### ***GPIO\_APP***

### ***5.6.2.1 T\_UBYTE GPIO\_GetState(T\_UWORD ch)***

|  |  |
| --- | --- |
| **Description** | Get the state of a pin, the pin is defined in ch, that is, it reads if the pin is on or off. |
| **Parameters** | T\_UWORD ch  Number of the GPIO register/channel |
| **Return Value** | *T\_UBYTE result*  *Pin reading (on/off) of specified channel* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | 1.2, 4.2 |

### ***5.6.2.2 void GPIO\_SetState(T\_UWORD ch, T\_UBYTE value)***

|  |  |
| --- | --- |
| **Description** | Set the value received to the pin specified in ch, using value to turn it on/off. |
| **Parameters** | T\_UWORD ch  Number of the GPIO register/channel  T\_UBYTE value  Turn off/on the specified channel |
| **Return Value** | *void* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | 2.7 |