|  |  |
| --- | --- |
| **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. |

**Table of Contents**

[1 Purpose 4](#_Toc435095907)

[2 Definitions and abbreviations 4](#_Toc435095908)

[3 Realization constraints and targets 4](#_Toc435095909)

[3.1 Targets 4](#_Toc435095910)

[3.2 Constraints 5](#_Toc435095911)

[4 SW Conceptual design 7](#_Toc435095912)

[5 SW Component internal breakdown 8](#_Toc435095913)

[5.1 Diagrams 8](#_Toc435095914)

[5.1.1 Use Case Diagram 8](#_Toc435095915)

[5.1.2 Deployment Diagram 9](#_Toc435095916)

[5.1.3 Component Diagram 10](#_Toc435095917)

[5.1.4 Activity Diagram 11](#_Toc435095918)

[5.1.5 Class Diagram 12](#_Toc435095919)

[5.1.6 Sequence Diagram 13](#_Toc435095920)

[5.2 Functional Decomposition 13](#_Toc435095921)

[*5.3* *void WL\_WinMUp(void)* 13](#_Toc435095922)

[*5.4* *void WL\_WinMDw(void)* 14](#_Toc435095923)

[5.5 *void WL\_WinAUp(void)* 14](#_Toc435095924)

[*5.6* *void WL\_WinADw(void)* 14](#_Toc435095925)

[5.7 *void WL\_A\_Pinch (void)* 14](#_Toc435095926)

[5.8 *void WL\_CheckValid()* 15](#_Toc435095927)

[*5.9* *void blockButtons()* 15](#_Toc435095928)

[*5.10* *void wait5seconds()* 15](#_Toc435095929)

[*5.11* *T\_ULONG WL\_CheckAutoManualUp(void)* 15](#_Toc435095930)

[*5.12* *T\_ULONG WL\_CheckAutoManualDw(void)* 16](#_Toc435095931)

# 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

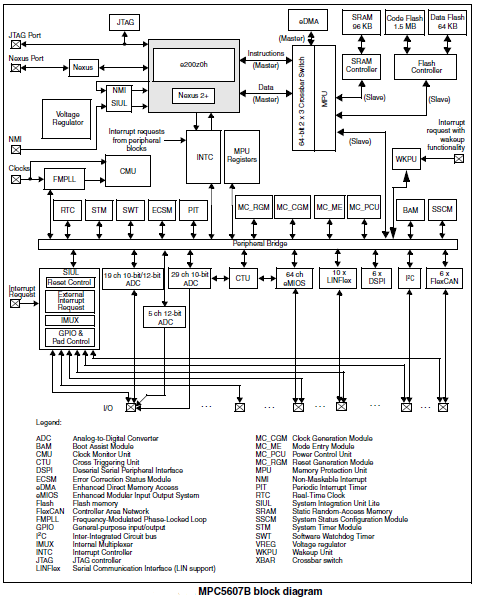
The principal modules (see block diagram) that are used in this project:

* 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,

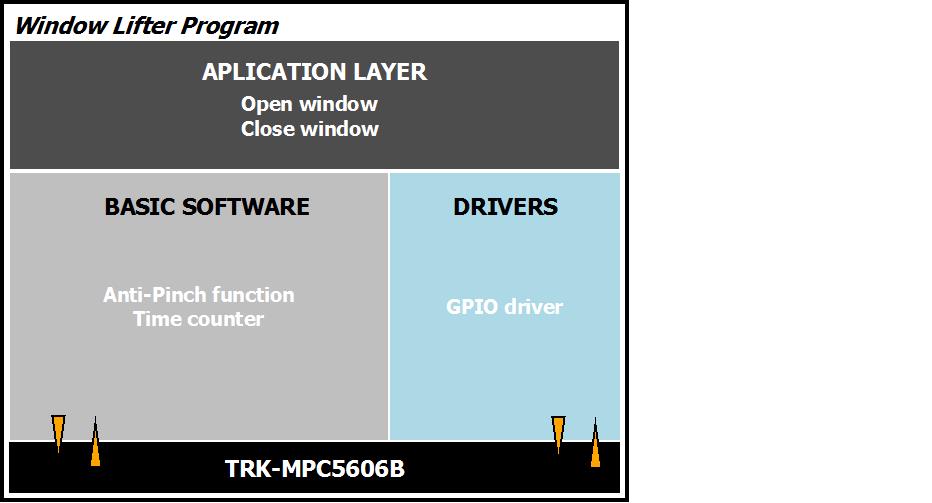
-and an ISR (used to make possible anti-pinch functionality)

* 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.

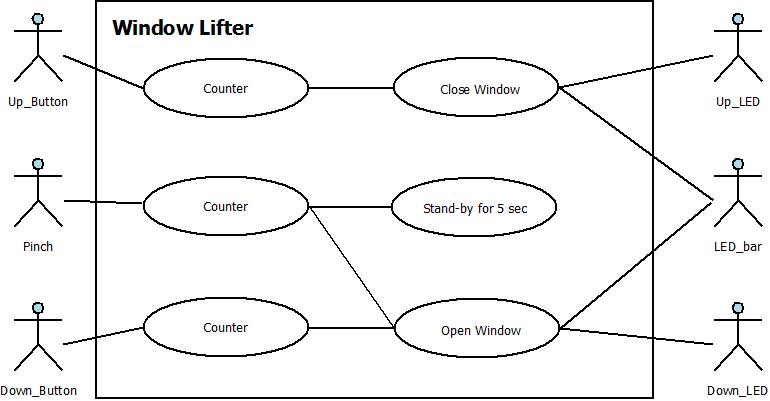


# SW Component internal breakdown

## Diagrams

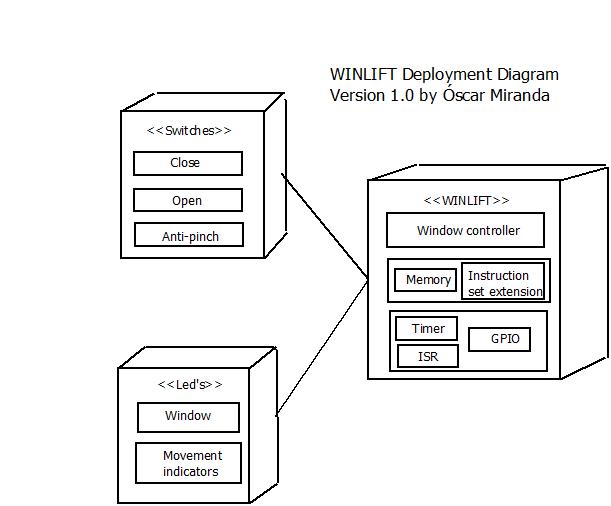
### Use Case Diagram

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.



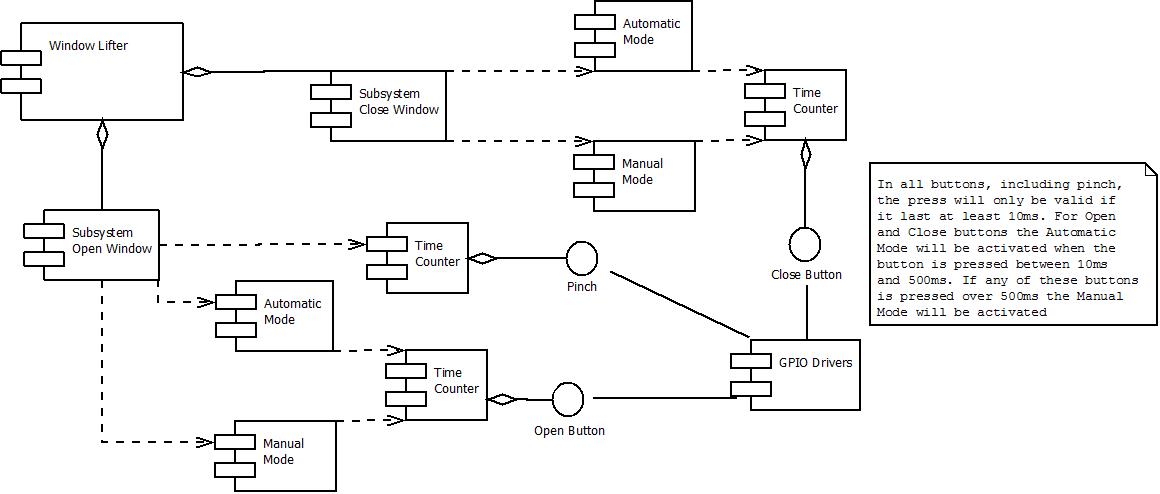
### Deployment Diagram

The following deployment diagram represents the inputs that are divided in three: Close, Open and anti-pinch; the outputs that are the LED simulation of the window, and the indicator LEDS, that display whether the window is lifting or lowering; the processing module that corresponds to the API, the HAL and the MCAL, in these modules the timings, the inputs and ouputs of the microcontroller and the interrupts are defined.



### Component Diagram

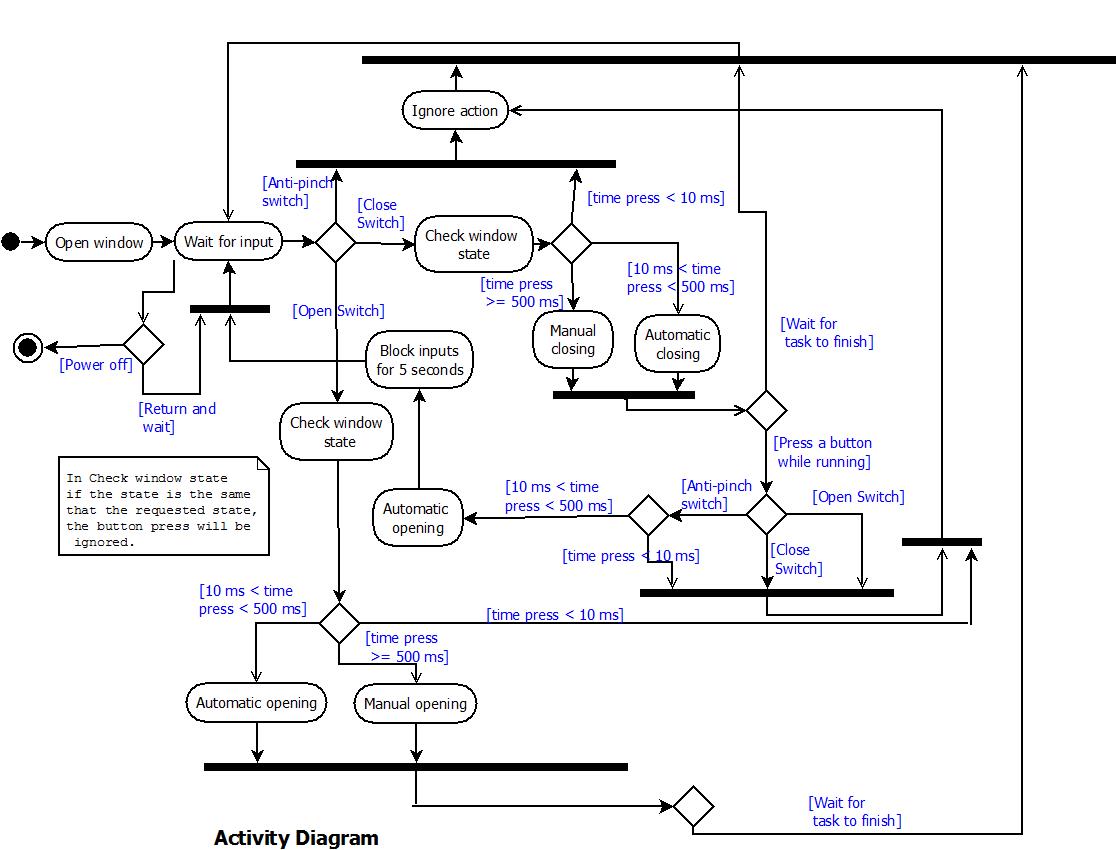
The following Component Diagram describes the structure and relations between the sub-systems comprehended in the Window Lifter system.



### Activity Diagram

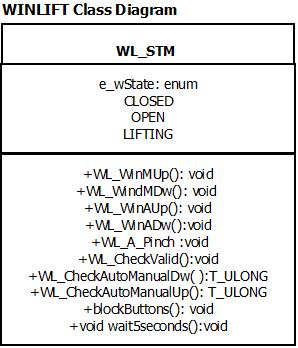
The following flowchart defines all the default and other possible states of the window and requests of the processor, the operations like open or close window, the anti-pinch functionality, the validation of a button and the end of the program flow that is when the system is reset, or turned off.

Roughly, the process of the program begins with a default state, that is open, then the system have to wait for an input. If a button is pressed, first of all, the current state of the window must be checked to avoid errors, for example, only the anti pinch functionality have to be activated if the state of the window is Lifting. Depending on the button press time, the program will evaluate whether to select a manual closing, an automatic closing, or goes to wait. If the anti pinch button is pressed, all the inputs must be disable for 5 seconds and then goes wait. To exit of wait state, there are two options, either to turn off the system or enter an input.

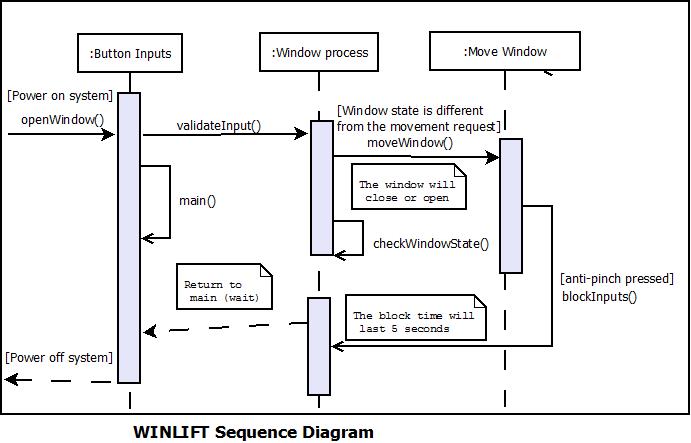


### Class Diagram

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.



### Sequence Diagram

This diagram represents a general flow of the WINLIFT software. Note that the function moveWindow() could be a openWindow() or a closeWindow() function, and while the system is running, the most of the time, it will wait for an input, until is turn off. The initial or default state is window open.

## Functional Decomposition

## ***API***

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

### 5.3.1.1 void WL\_CheckValid(void)

|  |  |
| --- | --- |
| **Description** | As long as a button is press, check if the button  press is valid (to avoid debounce), the button  verified 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 |
| **Return Value** | *T\_ULONG* |
| **Precondition** | up(close), down(open) or anti-pinch button must have been pressed |
| **Post condition** | ------------------ |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 3.1 |

### ***5.3.1.2 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. |
| **Return Value** | *T\_ULONG lub\_ValidMA* |
| **Precondition** | Up (Close) or Down (Open) button pressed |
| **Post condition** | *Prepares the window to move manually or automatically using the variable lub\_ValidMA* |
| **Error Conditions** | *Not defined* |
| **Requirement** | Req 3.3 |

### ***5.3.1.3 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 |

### ***5.3.1.4 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 |

### ***5.3.1.5 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 |

### ***5.3.1.6 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 |

### ***5.3.1.7 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 4.3, 4.4, 4.6 |

### ***5.3.1.8 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***

### ***5.3.2.1 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. |
| **Parameters** | No parameters |
| **Return Value** | *There is no return value* |
| **Precondition** | ---- |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | 3.3 |

## ***MAL***

## ***HAL***

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

### ***5.5.1.1 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** | ----- |

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

|  |  |
| --- | --- |
| **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** | ------ |

### ***5.5.1.3 void initModesAndClock()***

|  |  |
| --- | --- |
| **Description** | Enables the run mode and thsignal 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** | ------ |

### ***5.5.1.4 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 |

### ***5.5.1.5 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** | ------ |

### ***5.5.1.6 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***

### ***5.5.2.1 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 |
| **Return Value** | *T\_UBYTE result*  *Pin reading (on/off)* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | ------ |

### ***5.5.2.1 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  T\_UBYTE value  Turn off/on the specified ch |
| **Return Value** | *void* |
| **Precondition** | Global\_Init(called) |
| **Post condition** | *----* |
| **Error Conditions** | *Not defined* |
| **Requirement** | ------ |