/// Software Design Description (SDD)

LED Control Device

Project ref: 0001

Document ref: 0001

This document covers:

- *Software specification STM Control LED*

	Abgrall, Devaux, Venier / STM_Control_LED	yyyy-mm-dd	Signature
Prepared	*Software Engineer*	2019/05/29	
Reviewed	*G. Tsiapkolis* *Software Technical Manager* *Verification and Validation Responsible* *Software Quality Responsible*		
Approved	*Technical Project Manager*		

/// Change history

REVISIONS					
Issue	**Date**	**Description**	**Prepared by**	**Checked by**	**Approved by**
A	2019/05/29	First issue	Abgrall Thomas Devaux Axel Venier Antoine	G. Tsiapkolis	
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

/// Table of contents

/// List of figures

/// List of tables

1.0 Introduction

1.1 Purpose

*<**IEEE Std 1558-2004:** In addition, identify intended audience.>*

Ce Software Design Description rpésente le projet Controle de LED, réalisé dans le cadre des cours d'Architecture ARM à l'école supérieur Ynov à Bordeaux.

Les cibles attendues de ce document sont les élèvent de la formation Aéronautique et Systèmes Embarqués de Bordeaux Ynov Campus, incluant M. Tsiapkolis en qualité de professeur d'Architecture ARM. Sont aussi concidérées comme cible les mainteneurs du projet.

1.2 Scope

*<**IEEE Std 1558-2004:** Identify the software products by name, explain what the software products will and will not do, describe the application software specified, and be consistent with the SRS or other documents.>*

Ce document contient une description du projet Controle de LED. L'architecture de base du projet et du code à été réalisé à l'aide du logiciel "STM32CubeMX". Le code est développé en C.

Le projet consiste en la réalisation d'un système permettant d'exécuter des commandes reçues en communication série et informer de son état par le même port de communication. Le système commence par allumer deux LEDs. Quatre commandes au format texte peuvent être reçues. Si la commande n'est pas reconnue, un bip est généré par l'interface des écouteurs.

Si aucune commande n'est reçue dans un délai défini, le système envoie par la liaison série l'état des LEDs, sous format texte, en boucle. Le système inclue un bouton permettant d'inverser l'état des LEDs si pressé.

L'intégralité du logiciel est développé dans le fichier main. Contenu dans le dossier ressource (Src).

1.3 Definitions, acronyms, and abbreviations

*<**IEEE Std 1558-2004:** Provide the definitions, acronyms, and abbreviations required to properly interpret this document.>*

Acronyms	Meaning
CI	Configuration Item
ECR	Engineering Change Request
IEEE	Institute of Electrical and Electronics Engineers
IC	Interface Control
ICD	Interface Control Document
IT	Information Technology
N.A	Not Applicable

Acronyms	Meaning
OS	Operating Software
SCI	Software Configuration Item
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SPMP	Software Project Management Plan
SQAP	Software Quality Assurance Plan
SRTM	Software Requirement Traceability Matrix
STP	Software Test Plan
STPr	Software Test Procedure
STR	Software Test Report
SUM	Software User Manual
SVD	Software Version Description
SVN	Subversion
SVVP	Software Verification and Validation Plan
SVVR	Software Verification and Validation Report
SW	Software
UART	Universal Asynchronous Receiver Transmitter
USART	Universal Synchronous/Asynchronous Receiver Transmitter

Table 1: Acronyms

This table defines the terms used in this plan.

Keywords	Description
Anomaly	Change of specification further to an execution error or back fitting to specification.
Baseline	A configuration used as a reference. Once established, a baseline can only be modified via modification management.
Configuration	Identification of all the components of a group of elements (I), or a macro-element (II) that stipulates the version (e.g. the configuration of a sub-system identifies all the elements of the sub-system at the instant t). A configuration can also include other configurations (e.g. project configuration identifies all the other configurations).
Configuration Management	Integral process whose aim to control products by providing a version identification and follow up mechanism for each element in the product.
Configuring (an element)	The action of submitting an element to configuration management.
Element	Software component, material component or document considered as a unit from a configuration management point of view. As soon as an element is modified, it becomes another element (e.g. RxStack.c version 7 is a different element from RxStack.c version 8).
Evolution	Specification change made by the customer.
Level of control	The attribute of an element that stipulates the configuration management rules that the element must comply with. The level of control of an element can vary according to the project class.
Modification	It results from a request for evolution or an anomaly report. The modification is taken into account by configuration management.
Product	An element that can be delivered to the customer (e.g. a plan, an executable). A product can be a macro-element (e.g. an executable).
Release	Formal action of making a product available and authorizing the use of a product that can be restored. This happens with the consent of the persons in charge of the two phases between which this occurs.

Keywords	Description
Revision (i)	Product identifier (all the figures and/or letters) (I). Product released (II).
Software	Intellectual creation including the procedures, programs, rules and all literature relative to how a unit of data processing function.
Traceability	Ease with which a link can be established between two or several products in a development process and more particularly those possessing a predecessor to successor or master to slave relation.

Table 2: Definitions

1.4 Document organization

*<**IEEE Std 1558-2004:** Document the software methods or representations used (e.g., Data Flow Diagrams, Structure Charts, Finite state machines, Object-Oriented Diagrams, or other design techniques).*

Define the view(s) for the design entities and their attributes, and for each view, define the document format and specify where the design entities and their attributes are described within the document.>

2.0 References

*<**IEEE Std 1558-2004:** Complete list of all documents referenced within this document.>*

***Note:** If the issue is not shown, the latest issue shall apply.*

Ref.	Description	Version
[IEEE SRS]		

Table 3: External references

Ref.	Description
[SDD]	YT001001 - Software Design Description (?)

Table 4: Project documents

3.0 Software design

*<**IEEE Std 1558-2004:** Description of the design following the organization of views specified in 1.4. Regardless of the design view(s) presented, each entity attribute for each entity shall be present.>*

Pour réaliser la communication série, j'ai utilisé un USART en mode asynchrone et deux Timers. Pour la partie interface il y a deux LEDs et le User Button sur les GPIOs.
Les deux Timers sont initialisés avec un Prescaler différent, le premier offre 3 secondes de période, le second 1

seconde de période.
Une interruption est génerée par les timers à chaque période, et une interruption est généré par l'USART à chaque caractère reçu. Par polling, le bouton génère l'inversion de l'état des LEDs dans une boucle infinie avec un anti-rebond afin que l'action n'est lieux qu'une fois si on reste appuyé.

*<**IEEE Std 1016-1998:** A design is an element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced.*
Design entities result from a decomposition of the software system requirements. The objective is to divide the system into separate components that can be considered, implemented, changed, and tested with minimal effect on other entities.
Entities can exist as a system, subsystems, data stores, modules, programs, and processes; see IEEE Std 610.12-1990.
The number and type of entities required to partition a design are dependent on a number of factors, such as the complexity of the system, the design technique used, and the programming environment. Although entities are different in nature, they possess common characteristics. Each design entity will have a name, purpose, and function. There are common relationships among entities such as interfaces or shared data. The common characteristics of entities are described by design entity attributes.>

Chacunes des parties qui seront présenté sont indépendantes et peuvent être implémentées, changées, et testées séparément chacunes les une des autres.

Ces parties sont les suivantes:

- USART:

```c
/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_Data_Reception;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}
```

- *Timer2:*

```
/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 750;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 64000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
```

- *Timer3:*

```
/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 249;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 64000;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */

}
```

- GPIO:

```
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PD14 PD15 */
  GPIO_InitStruct.Pin = GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}
```

*<**IEEE Std 1016-1998:** A design entity attribute is a named characteristic or property of a design entity. It provides a statement of fact about the entity.*

Design entity attributes can be thought of as questions about design entities. The answers to those questions are the values of the attributes. All the questions can be answered, but the content of the answer will depend upon the nature of the entity. The collection of answers provides a complete description of an entity. The list of design entity attributes presented in this subclause is the minimum set required for all SDDs. The selection of these attributes is based on three criteria:

- *The attribute is necessary for all software projects;*
- *An incorrect specification of the attribute value could result in a fault in the software system to be developed;*
- *The attribute describes intrinsic design information and not information related to the design process. Examples of attributes that do not meet the second and third criteria are designer names, design status; and revision history. This important process information is maintained by other software project activities as described in IEEE Std 730-1998 and IEEE Std 828-1998.*

All attributes shall be specified for each entity. Attribute descriptions should include references and design considerations such as tradeoffs and assumptions when appropriate. In some cases, attribute descriptions may have the value none. When additional attributes are identified for a specific software project, they should be included in the design description. The. attributes and associated information items are defined as follow.>

<IEEE Std 1016-1998: Identification

The name of the entity. Two entities shall not have the same name. The names for the entities may be selected to characterize their nature. This will simplify referencing and tracking in addition to providing identification.>

<IEEE Std 1016-1998: Type

A description of the kind of entity. The type attribute shall describe the nature of the entity. It may simply name the kind of entity, such as subprogram, module, procedure, process, or data store. Alternatively, design entities may be grouped into major classes to assist in locating an entity dealing with a particular type of information. For a given design description, the-chosen entity types shall be applied consistently.>

L'entité USART a pour but de faire d'établir la communication série afin de pouvoir indiquer l'état des LEDs et de recevoir les nouveaux états des LEDs demandé pas l'utilisateur.

L'entité TIMER2 permet le rafraichissement des données dans la liasion série.

Le TIMER3 est l'entité qui permet de déclencher l'affichage.

Et l'entité GPIO est celle qui permettra de modifier l'état des LEDs.

*<**IEEE Std 1016-1998:** **Purpose***

A description of why the entity exists. The purpose attribute shall provide the rationale for the creation of the entity. Therefore, it shall designate the specific functional and performance requirements for which this entity was created; see IEEE Std 830-1998. The purpose attribute shall also describe special requirements that must be met by the entity that are not included in the software requirements specification.>

Materiel utilisé : Une carte STM32

Logiciels utilisés : CubeMX, Keil et Herculess. Le programme a pour objectif de recevoir des commandes via une liaison UART avec un terminal:
Si la commande recue par le programme est :

- *LED 1 ON : la led verte doit d'allumer*
- *LED 2 ON: la led orange doit d'allumer*
- *LED 1 OFF: la led verte doit s'eteindre*
- *LED 2 OFF : la led orange doit s'eteindre*

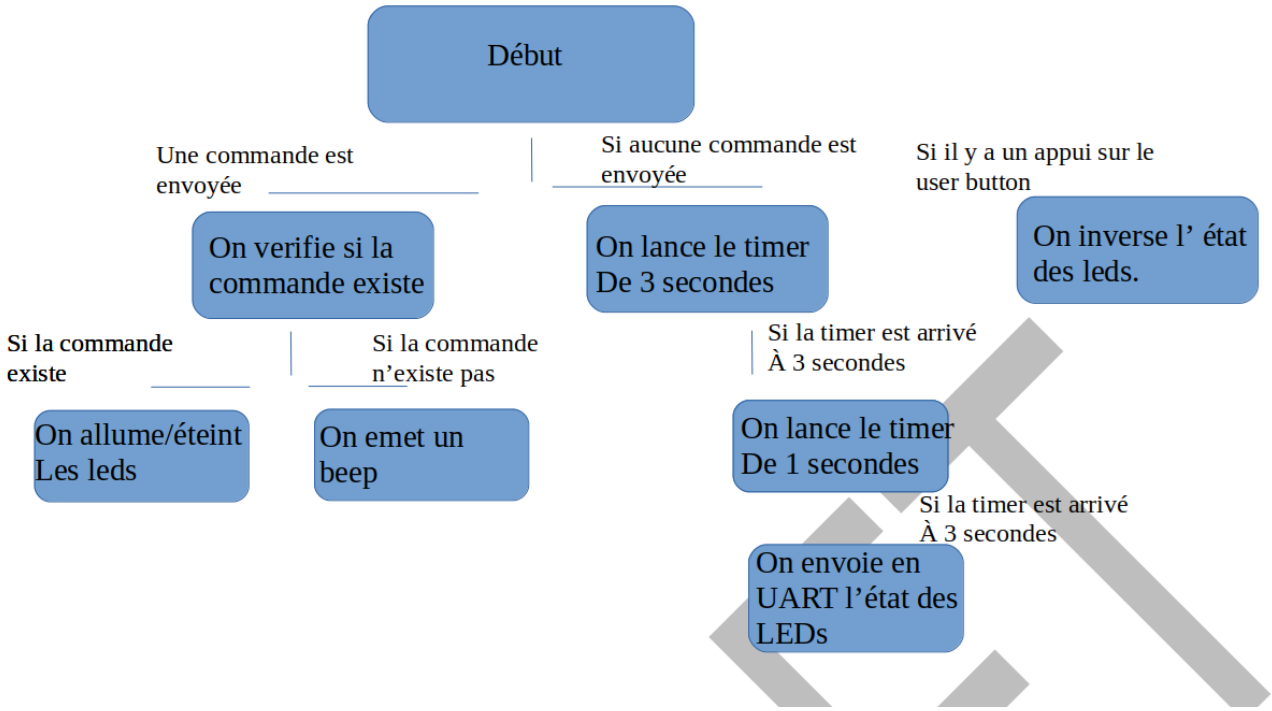
Si une autre de ces commandes est envoyée alors un beep est emit pour signifier à l'utilisateur que la commande n'est pas comprise par le programme.

Si au bout de 3 secondes aucune commande n'est envoyée alors le programme envoie sur la liaison serie toutes les secondes l'état des leds.

Si l'utilisateur appui sur le user button de la carte alors l'état des leds s'inversent.

*<**IEEE Std 1016-1998:** **Function***

A statement of what the entity does. The function attribute shall state the transformation applied by the entity to inputs to produce the desired output. In the case of a data entity, this attribute shall state the type of information stored or transmitted by the entity.>



*<**IEEE Std 1016-1998:** **Subordinates***

The identification of all entities composing this entity. The subordinates attribute shall identify the composed of relationship for an entity. This information is used to trace requirements to design entities and to identify parent/child structural relationships through a software system decomposition.>

On vérifie si la commande existe : La verification de la commande se fait de la callback de reception de l'UART à l'aide de swit case le programme decide quelle instruction il execute en function de la commande qui lui ai parvenu via liaison Série.

```
/* if sur les parties constantes des commande valides */
if(Data_Reception[0] == 'L' && Data_Reception[1] == 'E' && Data_Reception[2] == 'D' && Data_Reception[3] == ' ' &&
    Data_Reception[5] == ' ' && Data_Reception[6] == 'O')
{
    /* vérification sur les parie variable (LED et état) */
    switch(Data_Reception[4]){
        case '1':
            LED = GPIO_PIN_14;
            break;
        case '2':
            LED = GPIO_PIN_15;
            break;
        default:
            return PARSE_STATUS_ERR;
    }
    switch(Data_Reception[7]){
        case 'N':
            etat = GPIO_PIN_SET;
            break;
        case 'F':
            etat = GPIO_PIN_RESET;
            break;
        default :
            return PARSE_STATUS_ERR;
    }
```

On allume/éteint les leds: Pour allumer et éteindre les Leds il à été choisit d'utiliser une structure pour la lisibilité du code.

```
typedef struct{
    GPIO_PinState LED14;
    GPIO_PinState LED15;
}LEDs_Stat;
```

On emet un beep: L'emission du beep se fait via la sortie série de la STM32 un message BIP est affiché sur le terminale.

```
void emitBip(){
    HAL_UART_Transmit(&huart2, (uint8_t*) "BIP\r\n", 6, 10);
}
```

Timer de 3 secondes: le timer de 3 secondes est déclencher par la function interruption

```
// Comme une commande a été détéctée, on arrête d'affiche l'état des LEDs
HAL_TIM_Base_Stop_IT(&htim3);
```

Timer de 1 secondes: le timer de 1 seconde est déclencher par la function intérruption

```
// On redémare alors le timer 2 au cas ou il a été stoppé
HAL_TIM_Base_Start_IT(&htim2);
```

Envoie de l'état des Leds: l'etat des LEDs est envoyée par liaison série au terminale

Inverse l'etat des Leds: l'inversion des led est réaliser par la function HAL_GPIO_TooglePin()

<IEEE Std 1016-1998: Dependencies

A description of the relationships of this entity with other entities. The dependencies attribute shall identify the uses or requires the presence of relationship for an entity. These relationships are often graphically depicted by structure charts, data flow diagrams, and transaction diagrams.

This attribute shall describe the nature of each interaction including such characteristics as timing and conditions for interaction. The interactions may involve the initiation, order of execution, data sharing, creation, duplicating, usage, storage, or destruction of entities.>

On vérifie si la commande existe : Cette fonction depend de la fonciton HAL_UART_Receive_IT(&huart2,Buff_Data_Reception,1), Si cette function recoit quelle que chose alors la partie du programme qui se charge de verifier si la commande existe est apellé.
On allume/éteint les leds: Cette function n'est dépendante d'aucune des autres.
On emet un beep: L'emission du beep ne se fait que si l'utilisateur envoie une mauvaise commande dans le cas contraire la function d'emission d'un beep n'est pas appelé.
Timer de 3 secondes: le timer de 3 secondes ne se déclenche si aucune commande n'est envoyé au programme.
Timer de 1 secondes: le timer de 1 seconde ne se déclenche qu'une fois que le timer de 3 secondes s'est déclenché.
Envoie de l'état des Leds: Cette function n'est appelé qu'une fois que le timer de 1 seconde est lance.
Inverse l'etat des Leds: l'inversion des led est réaliser Lorsqu'il y a un appui sur le buton utilisateur qui est effectué.

<IEEE Std 1016-1998: Interface

A description of how other entities interact with this entity. The interface attribute shall describe the methods of interaction and the rules governing those interactions. The methods of interaction include the mechanisms for invoking or interrupting the entity, for communicating through parameters, common data areas or messages, and for direct access to internal data. The rules governing the interaction include the communications protocol, data format, acceptable values, and the meaning of each value.

This attribute shall provide a description of the input ranges, the meaning of inputs and outputs, the type and format of each input or output, and output error codes. For information systems, it should include inputs, screen formats, and a complete description of the interactive language.>

Nous interagissons avec le programme via le terminal, nous avons 3 possibilités.
La première entrer une commande, si celle-ci est prédéfinie (LED 1 ON, LED 2 ON, LED 1 OFF, LED 2 OFF) on aura l'action correspondante à cette commande sinon on aura un signal sonore.
La deuxième est d'attendre, dans ce cas là, si le timer arrive à 3 secondes, on envoie sur le terminal l'état des leds.
Et la troisième option est d'appuyer sur le bouton "user" qui inversera l'état des leds.
Si l'on écrit une commande avec une lettre ayant une casse qui ne correspond pas à celle de la commande prédéfinie, un signal audio nous prévient de l'erreur faite. En somme la commande doit être entièrement écrite en lettres majuscules.
Après avoir effectué une commande valide, le programme va decomposer la commande envoyée. Dans un premier temps on regarde si le caratère d'index 4 et "1" ou "2" ce qui déterminera la led don't on veut changer l'état. Puis on analysera le caractère d'index "7" pour savoir si c'est un "N" ou un "F" ce qui nous dira si on veut éteindre ou allumer la led précédemment définie.
Si la commande n'est pas valide c'est la function "emitBip" qui s'executera.
Nous avons deux timer, htim2 qui, lorsqu'il sera arrivé à trois seconds, lancera le timer htim3 qui, toutes les seconds affichera l'état des leds en appellant la function afficherEtatLEDs. L'état des leds est envoyé en UART.

<IEEE Std 1016-1998: Resources

*<**IEEE Std 1016-1998:** **Processing***

A description of the rules used by the entity to achieve its function. The processing attribute shall describe the algorithm used by the entity to perform a specific task and shall include contingencies. This description is a refinement of the function attribute. It is the most detailed level of refinement for this entity.

This description should include timing, sequencing of events or processes, prerequisites for process initiation, priority of events, processing level, actual process steps, path conditions, and loop back or loop termination criteria. The handling of contingencies should describe the action to be taken in the case of overflow conditions or in the case of a validation check failure.>

Nous vérifions en continu l'état du bouton « **user** », si celui-ci un **TogglePin** permettra d'inverser l'état des leds. Nous avons également ajoutés un délais qui permettra de servir « **d'anti rebond** » .

```
while (1)
{
  /* Gestion du bouton qui toggle les deux leds*/
  if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET){
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14);
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_15);

    /* Mise à jour des variables qui tracks l'état des LEDs*/
    LEDs_Stat.LED14 = (GPIO_PinState) !LEDs_Stat.LED14;
    LEDs_Stat.LED15 = (GPIO_PinState) !LEDs_Stat.LED15;

    /*anti-ebonds du bouton*/
    do{
      HAL_Delay(200);
    }while(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_PIN_SET);
  }
  HAL_Delay(100);
```

*<**IEEE Std 1016-1998:** **Data***

A description of data elements internal to the entity. The data attribute shall describe the method of representation, initial values, use, semantics, format, and acceptable values of internal data.

The description of data may be in the form of a data dictionary that describes the content, structure, and use of all data elements. Data information shall describe everything pertaining to the use of data or internal data structures by this entity. It shall include data specifications such as formats, number of elements, and initial values. It shall also include the structures to be used for representing data such as file structures. arrays, stacks, queues, and memory partitions.

The meaning and use of data elements shall be specified. This description includes such things as static versus dynamic, whether it is to be shared by transactions, used as a control parameter, or used as a value, loop iteration count, pointer, or link field. In addition, data information shall include a description of data validation needed for the process.>