

1ERE ANNEES MASTER DES SYSTEMES EMBARQUES
POUR LES APPLICATION INDUSTRIELLES

M1 SEAI

Rapport du projet

**Optimisation d'un système d'éclairage
automatique**



Réalisé Par :
Mohamed OUTZROUALTE
EL KABBOUS M'barek

Supervisé par :
Prof R. LATIF

Année Universitaire 2022 – 2023

Table des matières :

Introduction :	4
Problématique :	5
Objectif détaillée :	5
Mise en place du Projet :	6
Approche technique :	6
Composants du Système :	8
Etude algorithmique :	15
Unité de mesure :	15
Schéma électrique de la partie de mesure :	20
Implémentation en C embarqué (STM32IDE) :	21
Test et validation :	28
Process général :	28
Le signal PWM :	30
Implémentation en python :	31
Implémentation en C :	33
Implémentation en C++ :	35
Implémentation en Matlab Simulink :	37
Introduction :	37
Sous module 1 : RECEIVE DATA	37
❖ Bloc Serial Read :	38
❖ ASCII -> STRING :	38
Sous module 2: SEND REQUEST	40
❖ Bloc GPIO Read :	40
❖ Bloc Serial Write :	41
Sous module 3: COMMANDE LED	41
❖ Bloc if condition :	42
❖ Bloc PWM :	42
Conclusion :	43

Liste des figures :

Figure 1:Architecture globale	6
Figure 2:Boucle de control	7
Figure 3:Outils de développement	7
Figure 4:Raspbery 3	8
Figure 5:Raspbery description	9
Figure 6:STM32F103.....	10
Figure 7 :ST-link.....	11
Figure 8:LDR	11
Figure 9:Module LDR	12
Figure 10:Réglage du LDR	12
Figure 11:LCD avec I2C modul	13
Figure 12:LED	13
Figure 13:Resistance	14
Figure 14:Description algorithmique de l'unité de mesure	16
Figure 15:Configuration ADC.....	17
Figure 16:Cicuit RC.....	18
Figure 17:Equation différentielle du circuit RC	18
Figure 18:Descritisaion	19
Figure 19:Equation numérique du filtre	19
Figure 20:Comminication UART.....	19
Figure 21: Schéma électrique.....	20
Figure 22:Resultats de mesur.....	28
Figure 23:Description algorithmique du système.....	29
Figure 24:Signal PWM	30
Figure 25: module globale.....	37
Figure 26:Sous module 1 : RECEIVE DATA	37
Figure 27: paramètres du bloc Serial Read	38
Figure 28: block String to décimal	39
Figure 29:Sous module 2 : SEND REQUEST	40
Figure 30: block GPIO Read	40
Figure 31:paramètres du bloc Serial Write	41
Figure 32: Sous module 3 : COMMANDE LED	41
Figure 33: paramètre du bloc PWM	42

Introduction :

Dans le contexte urbain moderne, la gestion efficace de l'éclairage des rues est cruciale pour assurer la sécurité des citoyens tout en optimisant l'utilisation des ressources énergétiques. Le projet vise à développer un système de contrôle automatique de l'intensité lumineuse des lampadaires, utilisant des technologies avancées telles que le Raspberry Pi, les capteurs de lumière et la communication UART entre les cartes de développement pour créer une solution intelligente et économe en énergie.

On va essayer encore de créer et tester pratiquement le système et d'analyser son comportement sous différentes conditions.

Ce système adaptatif permet non seulement de réguler l'intensité lumineuse en fonction des conditions environnementales et des besoins des citoyens, mais aussi de réaliser des économies substantielles sur les coûts d'électricité et de maintenance. Ainsi, ce projet représente un pas significatif vers des villes plus durables et vivables.

Dans ce rapport, nous détaillerons l'architecture électrique et électronique globale du projet. Nous présenterons ensuite chaque bloc de cette architecture en mettant en avant leurs spécificités et fonctionnalités. Enfin, nous développerons des logiciels embarqués nécessaires au fonctionnement de ce système, en utilisant des langages de programmation tels que C/C++, Python et Simulink. Des tests seront effectués pour garantir l'efficacité et la fiabilité des solutions proposées.

Ce travail est rédigé suite aux travaux pratiques du module linux embarqué sous l'encadrement de Monsieur Latif Rachid.

Problématique :

De nos jours, les rues sont éclairées à l'aide de lampes à haute intensité. L'inconvénient de ces lampes à haute intensité est qu'elles consomment beaucoup d'énergie et un autre inconvénient est que l'intensité ne peut pas être modifiée en fonction des besoins.

Dans quelle mesure peut-on optimiser la consommation d'énergie d'un système d'asservissement d'éclairage automatique ?

Objectif détaillée :

L'objectif principal de ce projet est de concevoir un système d'éclairage automatique basé sur la technologie IoT, visant à optimiser la consommation d'énergie et améliorer l'efficacité de l'éclairage public. Le système régule l'intensité lumineuse en fonction de la luminosité ambiante détectée, tout en fournissant des informations en temps réel pour une gestion efficace :

- Le capteur de lumière LDR mesure l'intensité lumineuse ambiante et envoie un signal analogique au STM32F4.
- Le STM32F4 convertit ce signal en numérique et utilise un comparateur pour générer un signal numérique basé sur un seuil prédéfini.
- Le STM32F4 envoie les informations via UART au Raspberry Pi.
- Le Raspberry Pi traite ces informations pour ajuster l'intensité des LED et affiche les informations pertinentes sur un écran LCD.
- Les LED sont ajustées en temps réel pour optimiser la consommation d'énergie tout en maintenant une luminosité adéquate.

Mise en place du Projet :

La solution proposer est présentée ci-dessous :

Le système débute par l'acquisition de données à partir du capteur de lumière afin de surveiller les niveaux d'éclairage ambiants. En parallèle, il analyse les données provenant de la caméra intégrée pour identifier toute anomalie ou menace potentielle pour la sécurité. Par la suite, le système met en œuvre un contrôle automatique de l'intensité lumineuse des lampadaires en se basant à la fois sur les conditions de lumière ambiante mesurées et les résultats de l'analyse effectuée par la caméra. Enfin, des fonctionnalités de sécurité avancées sont intégrées pour détecter et répondre aux menaces telles que les accès non autorisés ou les actes de vandalisme, assurant ainsi la protection et la sûreté des installations.

Le schéma suivant décrit l'architecture globale du projet :

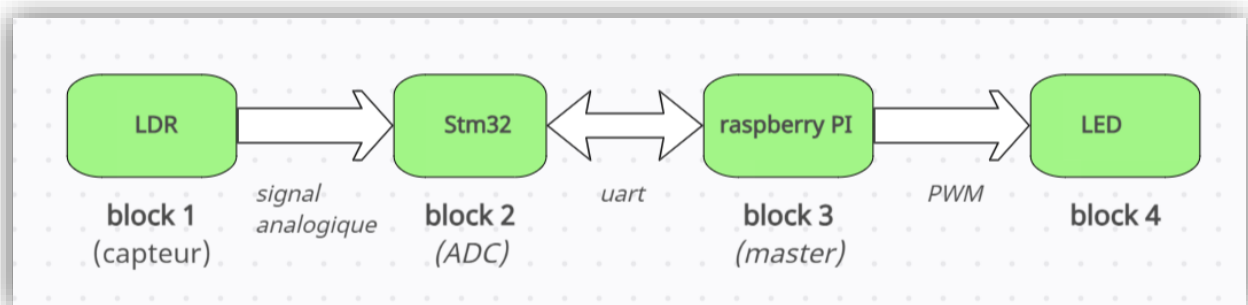


FIGURE 1:ARCHITECTURE GLOBALE

Le capteur LDR envoie un signal analogique vers stm32 qui présent l'intensité lumineux dans l'enivrement extérieur. Apres que stm32 reçoive une requête (request) du master (Raspberry) elle convertie le signal analogique du capteur LDR en un signal numérique (UART) puis l'envoi vers Raspberry pi. Enfin Raspberry pi control l'intensité lumineux l'une LED par un signal PWM.

Approche technique :

L'approche technique consistera à utiliser un Raspberry Pi comme unité centrale de traitement pour la collecte de données, l'analyse et le contrôle. Des capteurs de lumière seront utilisés pour surveiller les niveaux d'éclairage ambiants, tandis que des caméras intégrées seront utilisées pour surveiller l'environnement urbain. Des algorithmes avancés seront développés pour analyser les données des capteurs et des caméras, et prendre des décisions de contrôle en temps réel pour ajuster l'intensité lumineuse des lampadaires.

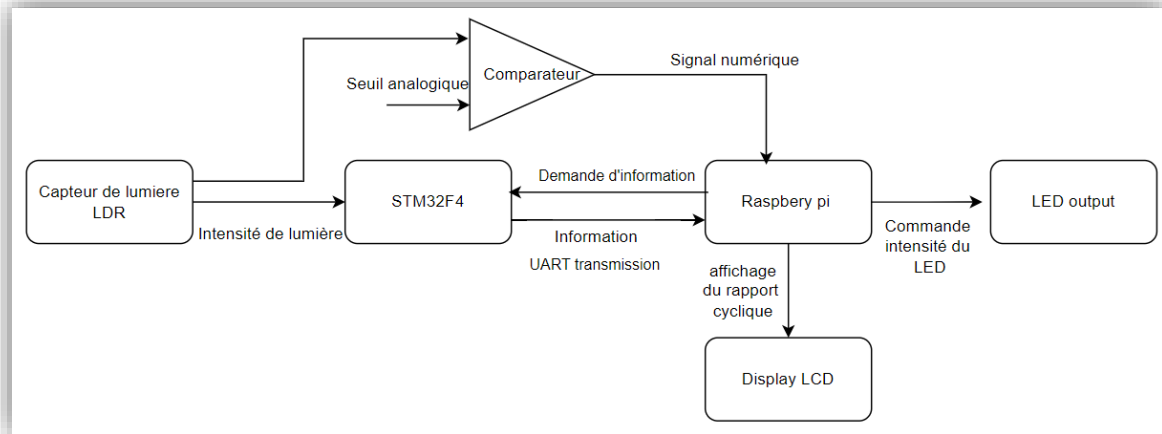


FIGURE 2:BOUCLE DE CONTROL

Le capteur de lumière LDR mesure en continu l'intensité lumineuse ambiante. Lorsqu'une interruption est détectée, le STM32F4 envoie une demande pour capturer ces mesures, les convertit et les filtre avant de les transmettre au Raspberry Pi. Le Raspberry Pi, à son tour, ajuste l'intensité des LED en fonction des données reçues et affiche les informations pertinentes sur l'écran LCD, tout en assurant un cycle de vérification toutes les secondes pour maintenir une réponse dynamique et efficace.

Les langues de développement du projet seront :

C/C++/python et Matlab Simulink.



FIGURE 3:OUTILS DE DEVELOPPEMENT

Composants du Système :

1. Raspberry Pi 3 Modèle B : Unité centrale de traitement pour l'acquisition de données, la correction, la génération de PWM, l'analyse des résultats et la mise en œuvre du protocole.



FIGURE 4: RASPBERRY 3

Le choix de la carte Raspberry Pi 3 Model B+ est motivé par sa puissance en termes de traitement des données et par son système d'exploitation sophistiqué (Linux Raspbian), qui permet une meilleure gestion des données (traitement et stockage). De plus, sa connectivité impressionnante via le réseau Ethernet est un atout majeur.

Ce qui est impressionnant, ce sont ses caractéristiques. Sur un petit bout de carte électronique, il y a un :

- Processeur ARM Cortex A-53 quatre cœurs de 64-bit cadencé à 1,4 Ghz.
- 1 Go de RAM (SDRAM).
- Un contrôleur vidéo Broadcom VideoCore IV
- Une interface de caméra (CSI).
- 4 ports USB 2.0
- Un port HDMI.
- Une interface d'affichage (DSI).
- Un connecteur Jack 3,5 mm / sortie composite.

- Une prise Ethernet 10 / 100 Mbps.
- Un emplacement de carte micro SD.
- WIFI 802.11n
- Bluetooth 4.1
- Alimentation 5V par micro USB.
- Bluetooth Low Energy (BLE).
- 40 pins GPIO

Raspberry pi représente toujours une carte de developement un grand potentiel de develloper notre projet.

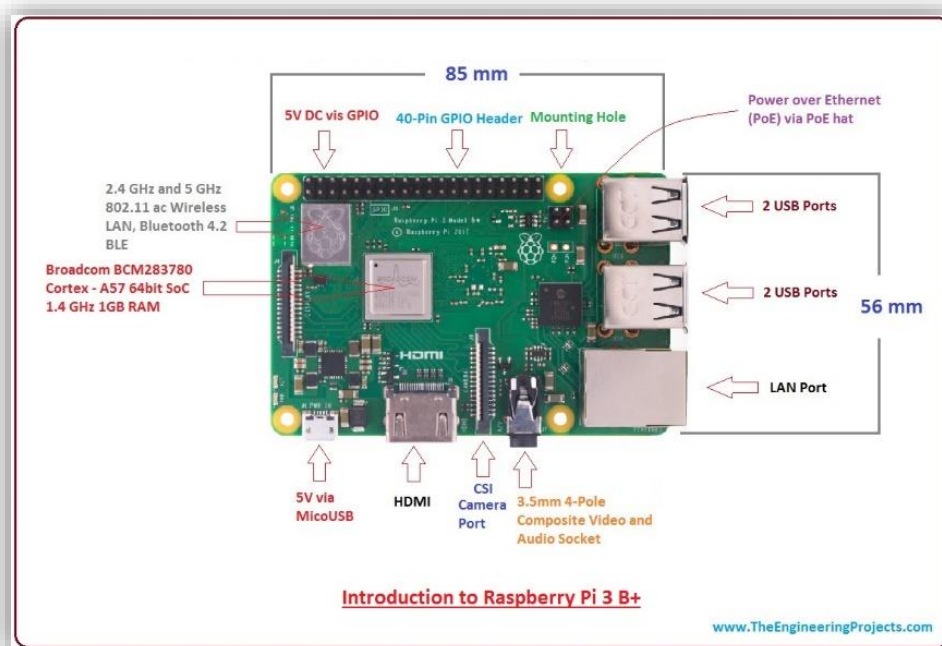


FIGURE 5:RASPBERRY DESCRIPTION

Cependant Raspberry pi ne nous permet pas la conversion analogique numérique pour interpréter les grandeurs analogiques qui représente une phase très critique dans notre application, c'est pour ça qu'on a chercher un microcontrôleur qui peut faire la mesure et le prétraitement (filtrage) et l'acquisition dans le processus de mesure.

Le microcontrôleur STM32F103 a été proposer par notre encadrant Monsieur Rachid Latif pour sa petite taille, sa consommation d'énergie optimale et ses fonctionnalité intéressante.

2. STM32F103 :

Le microcontrôleur STM32 pour l'acquisition des capturées, prétraitement (filtrage) et la transmission en temps réel.

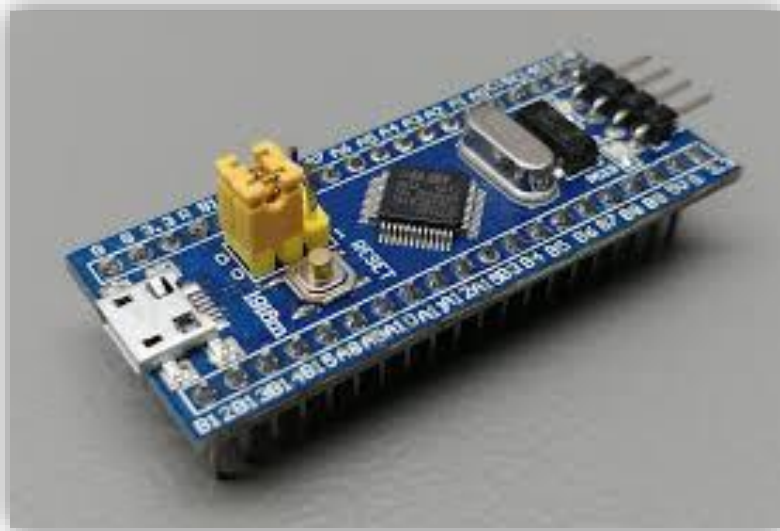


FIGURE 6:STM32F103

La configuration de la carte est réalisée par l'interface ST-link(debugger) et le logiciel de développement STM32IDE.

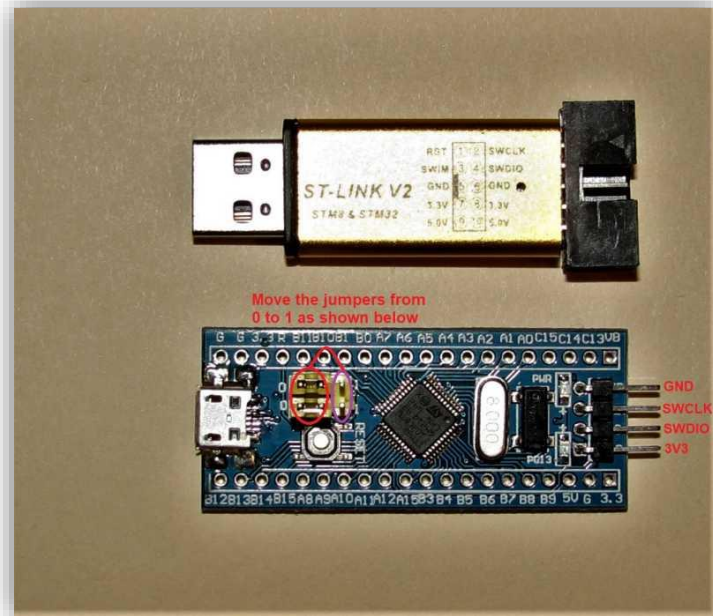


FIGURE 7 :ST-LINK

3. Capteur de Lumière (LDR) : Détecte les niveaux de lumière ambiante pour l'ajustement en temps réel de l'intensité des lampadaires.

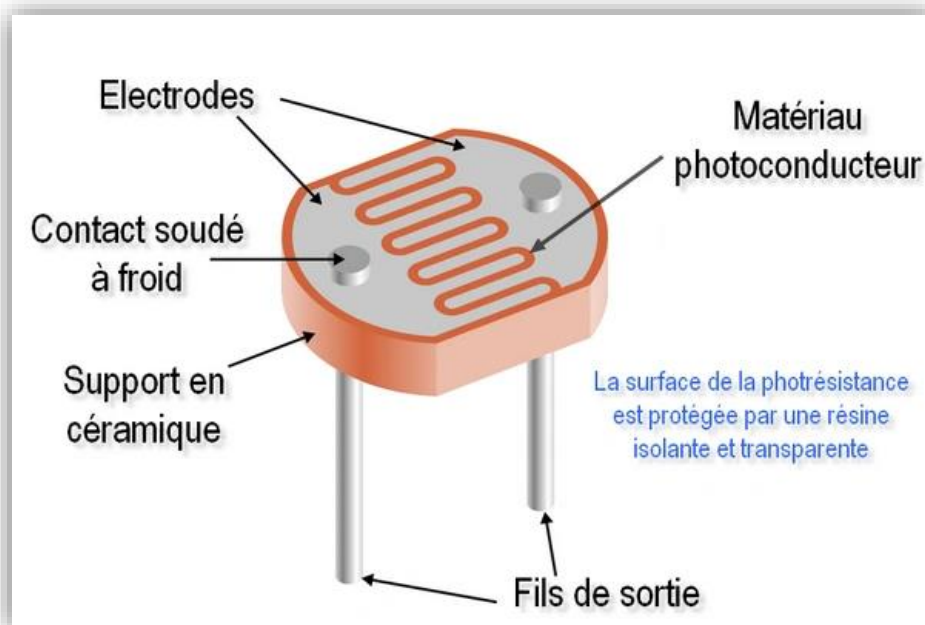


FIGURE 8:LDR

Une photorésistance est un capteur électronique dont la résistivité varie en fonction de la quantité de lumière incidente. On peut également la nommer résistance photo-dépendante (Light Dependent Resistor (LDR)) ou cellule photoconductrice.

Pour cette application on a choisi d'utiliser le module de mesure LM393.

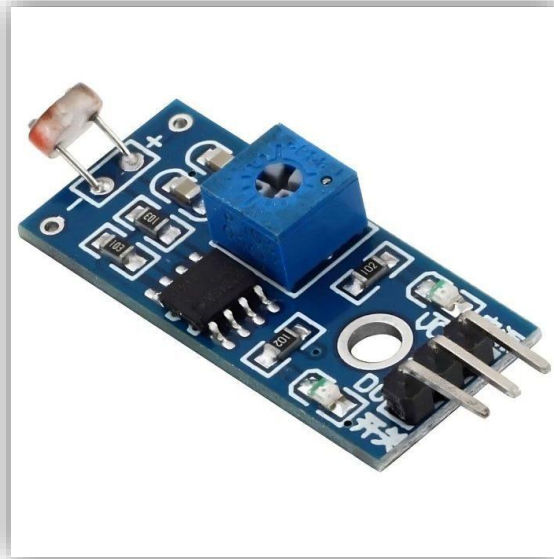


FIGURE 9:MODULE LDR

Ce module de mesure contient un comparateur analogique avec un seuil a sensibilité ajustable(manuellement), ce qui nous permettra l'application du model de la solution proposée précédemment.

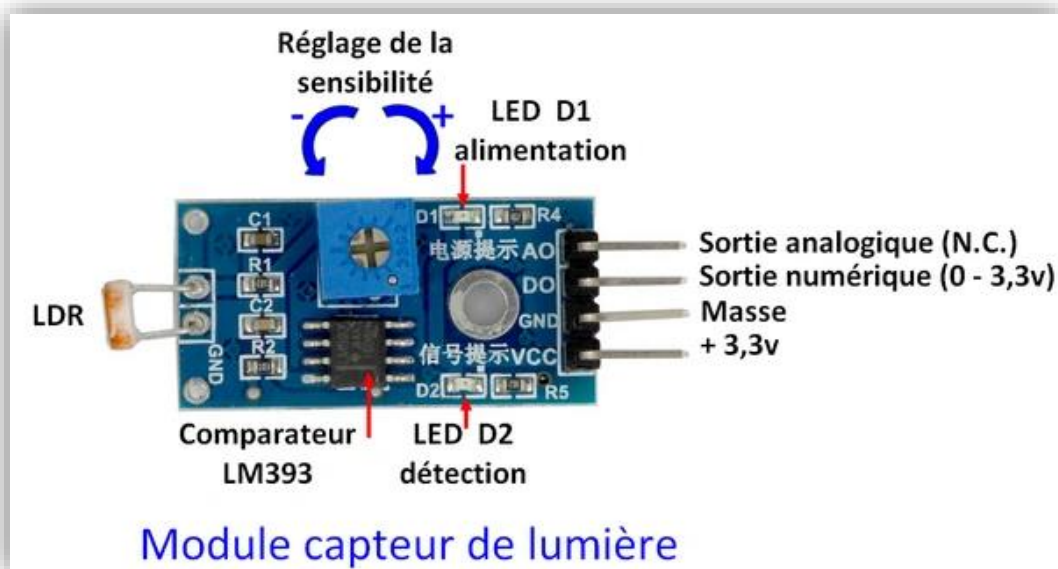


FIGURE 10:REGLAGE DU LDR

4. Afficheur LCD avec module I2C : Une unité d'affichage a module de communication I2C pour interfacer avec Raspberry pi.

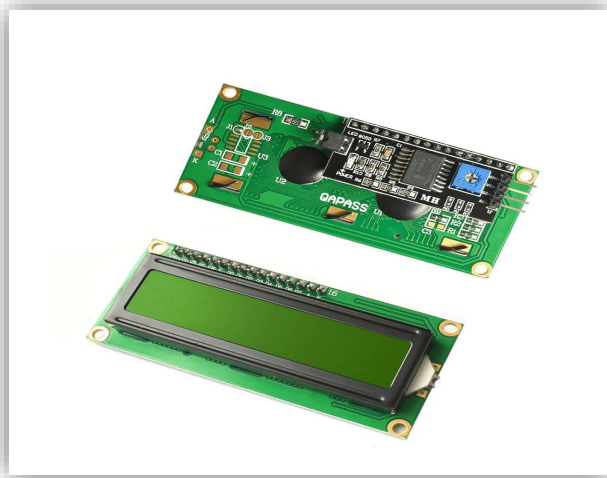


FIGURE 11:LCD AVEC I2C MODUL

5. LED : Indicateur pour ajuster la luminosité des lampadaires et simuler le comportement du système.



FIGURE 12:LED

6. Résistances : Utilisées pour limiter le courant dans les circuits des LEDs et d'autres composants selon les exigences de conception.



FIGURE 13:RESISTANCE

La conception du système :

Les résultats attendus du projet incluent la conception et la mise en œuvre réussies d'un système de contrôle automatique de l'intensité lumineuse des lampadaires, capable de réguler efficacement l'éclairage des rues en fonction des conditions environnementales et des besoins de sécurité. Le système devra démontrer des améliorations significatives en termes d'efficacité énergétique, de sécurité et de fiabilité par rapport aux méthodes de contrôle traditionnelles.

Le système d'éclairage automatique est composé de deux sous-systèmes principaux :

1. Sous-système de Contrôle Global :

- Gère l'initialisation, la détection des interruptions, la commande PWM pour les LED, et l'affichage des informations sur un écran LCD.

2. Unité de Mesure :

- Effectue l'acquisition de l'intensité lumineuse ambiante à l'aide d'un capteur LDR, convertit les signaux analogiques en numériques, applique un filtrage pour précision, et transmet les données au sous-système de contrôle via UART.

Etude algorithmique :

Unité de mesure :

On effectue dans cette partie l'acquisition de l'intensité lumineuse ambiante à l'aide d'un capteur LDR, convertit les signaux analogiques aux numériques, applique un filtrage pour précision, et transmet les données au sous-système de contrôle via UART.

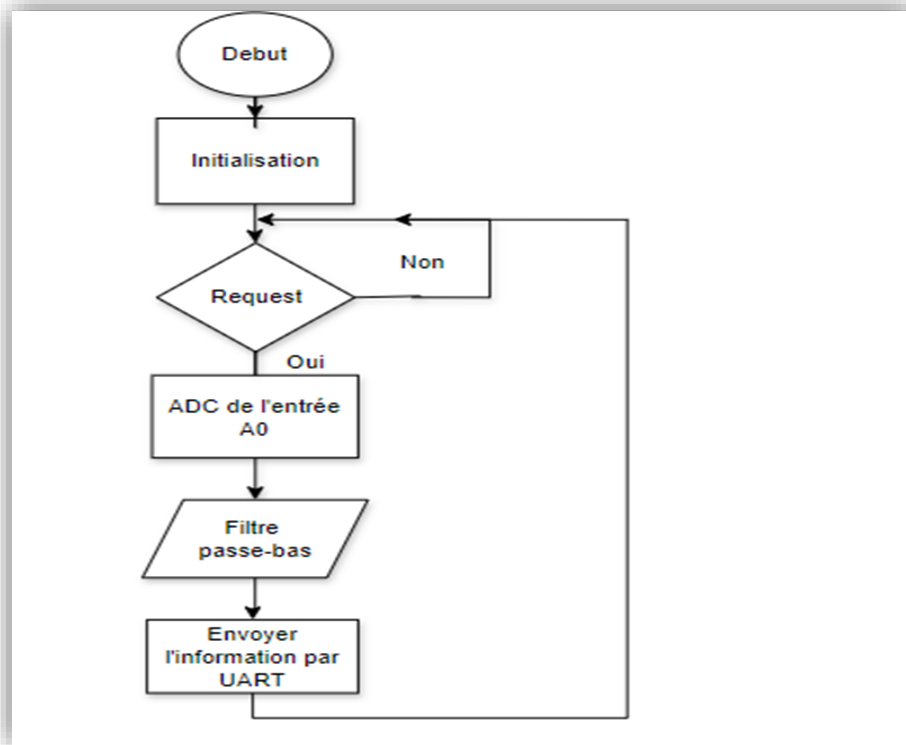


FIGURE 14: DESCRIPTION ALGORITHMIQUE DE L'UNITE DE MESURE

1. Conversion analogique numérique :

La conversion analogique-numérique (CAN) dans la carte STM32 est réalisée en utilisant le module de conversion analogique-numérique intégré au microcontrôleur STM32. Ce module CAN est une partie essentielle de nombreuses applications embarquées où il est nécessaire de convertir des signaux analogiques provenant de capteurs en données numériques que le microcontrôleur peut traiter.

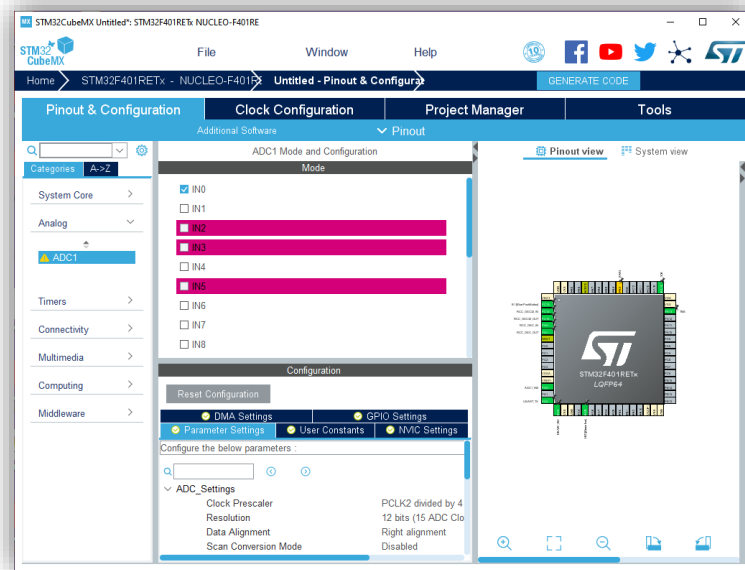


FIGURE 15: CONFIGURATION ADC

Le module CAN de la STM32 fonctionne en échantillonnant un signal analogique à des intervalles réguliers et en le convertissant en une valeur numérique correspondante. Le processus commence par la sélection de l'entrée analogique via un multiplexeur analogique. Une fois l'entrée sélectionnée, le convertisseur analogique-numérique échantillonne le signal analogique à une fréquence déterminée par un horloge d'échantillonnage. Le signal échantillonné est ensuite quantifié et codé en une valeur numérique sur un nombre déterminé de bits, la résolution est configurée à 12bits.

- Configuration et Utilisation

Pour utiliser le module CAN sur une carte STM32, plusieurs étapes de configuration sont nécessaires. Cela comprend la configuration des broches d'entrée analogiques, l'initialisation du module CAN, et la configuration des paramètres tels que la résolution de la conversion, la fréquence d'échantillonnage, et le mode de fonctionnement (simple, continu, ou scan).

- Configuration des Broches:

Les broches de la STM32 qui seront utilisées comme entrées analogiques doivent être configurées en mode analogique. Cela désactive les fonctions numériques et les résistances pull-up/pull-down sur ces broches.

- Initialisation du Module CAN:

Le module CAN est initialisé en configurant les registres appropriés. Cela inclut la sélection de la fréquence de l'horloge d'échantillonnage, la définition de la résolution de la conversion, et la configuration des canaux d'entrée.

- Lancement de la Conversion:

La conversion peut être lancée en mode simple (une seule conversion par déclenchement), en mode continu (conversions continues), ou en mode scan (conversion séquentielle de plusieurs

canaux). Une fois la conversion terminée, la valeur numérique résultante peut être lue à partir du registre de données de l'ADC.

2. Filtrage numérique :

La conception d'un filtre adéquat passe bas est très important dans ce genre d'application, il consiste à éliminer le bruit entrainer dans la récupération du signal et l'isolation de la bande passante, pour un le prototype de notre application on a choisi 150Hz comme une fréquence de coupure du filtre avec le comportement numérique d'un filtre passif RC.

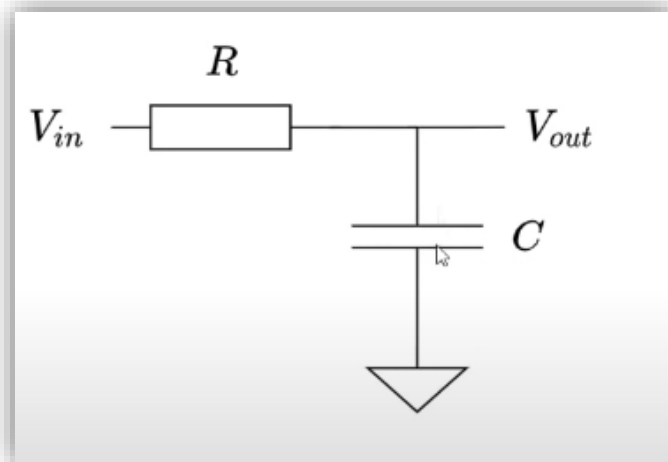


FIGURE 16:CICUIT RC

L'étude du circuit nous donne l'équation différentielle suivantes :

$$V_{out} + RC \frac{dV_{out}}{dt} = V_{in}$$

FIGURE 17:EQUATION DIFFERENTIELLE DU CIRCUIT RC

Discretisation du filtre en utilisant la méthode 'backward d'Euler':

$$\frac{dV}{dt} \approx \frac{V[n] - V[n - 1]}{T}$$

FIGURE 18:DESCRITISAION

On obtiendra par la suite l'équation suivante :

$$V_{out}[n] + RC \frac{V_{out}[n] - V_{out}[n - 1]}{T} = V_{in}[n]$$

FIGURE 19:EQUATION NUMERIQUE DU FILTRE

3. Transmission UART

Dans ce projet, la communication UART est utilisée pour établir une liaison série entre un Raspberry Pi et un microcontrôleur STM32. Le processus commence lorsque le Raspberry Pi envoie une demande sous forme de caractère ASCII 'a' (équivalent à 97 en décimal) à l'STM32 via l'UART. En réponse à cette demande, l'STM32 transmet des données du rapport cyclique, représentant un nombre entre 0 et 100, qui sera utilisé pour commander la LED connectée au Raspberry.

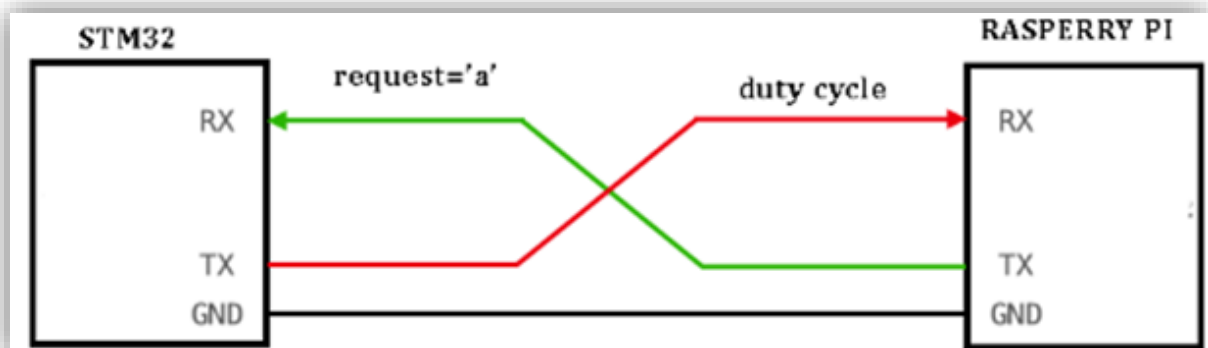


FIGURE 20:COMMUNICATION UART

Schéma électrique de la partie de mesure :

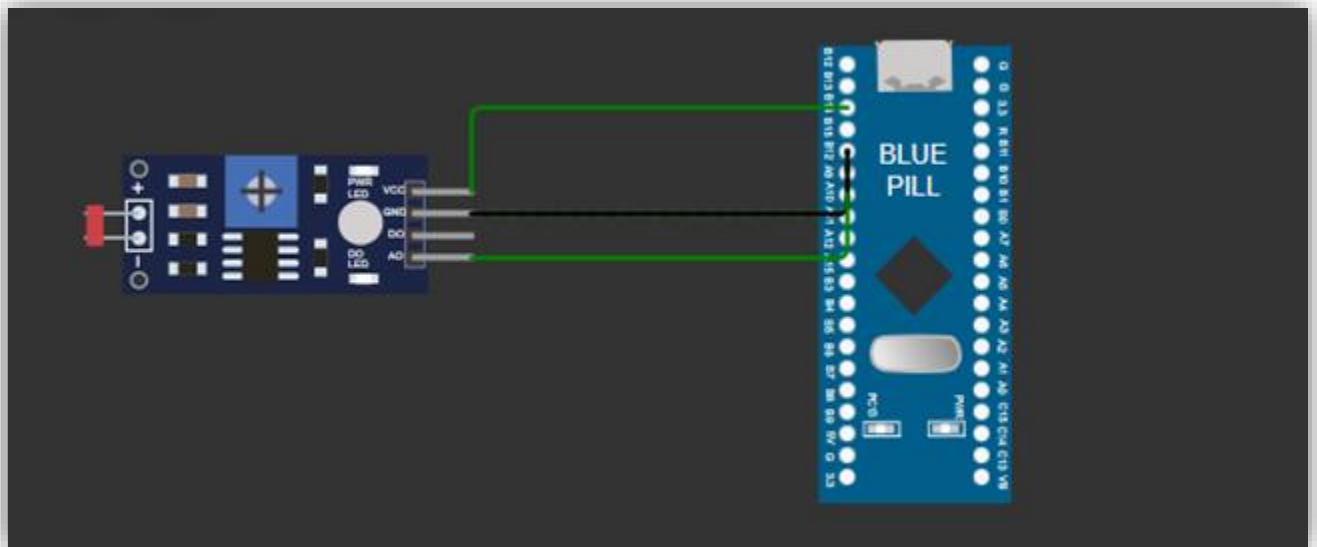


FIGURE 21: SCHEMA ELECTRIQUE

Implémentation en C embarqué (STM32IDE) :

Le code présenté ci-dessous est l'implémentation de l'algorithme de mesure expliquée ci-dessus :

```
/* Includes -----  
*/  
#include "main.h"  
#include <stdio.h>  
#include "string.h"  
#include "RCFilter.h"  
  
/* Private includes -----  
*/  
/* USER CODE BEGIN Includes */  
  
/* USER CODE END Includes */  
  
/* Private typedef -----  
*/  
/* USER CODE BEGIN PTD */  
  
/* USER CODE END PTD */  
  
/* Private define -----  
*/  
/* USER CODE BEGIN PD */  
  
char* user_data="1000\r\n";  
char data[2];  
uint16_t lux;  
//uint8_t filtred[2];  
uint8_t x;  
//uint8_t lux_Start=0xff;  
//uint16_t lux_Int;  
//uint16_t lux_Dec;  
uint8_t CheckSum;  
uint8_t Ask;  
uint8_t req;  
  
RCFilter lplux[3];  
  
/* USER CODE END PD */  
  
/* Private macro -----  
*/
```

```

/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----
*/
ADC_HandleTypeDef hadc1;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----
*/
void SystemClock_Config(void);
//static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----
*/
/* USER CODE BEGIN 0 */
void Send_Data()
{
    /* transmit start of frame byte
    */
    //    sprintf(data, "%d", lux_Start); /* convert int to string*/
    //    HAL_UART_Transmit(&huart2, (uint8_t*)data, (uint16_t)(strlen(data)),
    HAL_MAX_DELAY);
    /*
    * transmit integer byte
    */
    sprintf(data, "%d\n\r", (uint8_t)x); /* convert int to string*/
    HAL_UART_Transmit(&huart2, (uint8_t*)data, (uint16_t)(strlen(data)),
    HAL_MAX_DELAY);
    /*
    * transmit decimal byte
    */
    sprintf(data, "%d\n\r", (uint8_t)filtred[1]); /* convert int to string*/
    HAL_UART_Transmit(&huart2, (uint8_t*)data, (uint16_t)(strlen(data)),
    HAL_MAX_DELAY);
    */
}

```

```

}
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration----- */

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* Configure the system clock */
    SystemClock_Config();

    /* Initialize all configured peripherals */
    MX_ADC1_Init();
    MX_USART2_UART_Init();

    //FIRFilter_Init(&lplux);
    RCFilter_Init(&lplux , 150.0f , 0.001f);

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {

        while(req!=97)
        {
            HAL_UART_Receive(&huart2, &req,1, HAL_MAX_DELAY);
        }
        req=48; /* 0 reset */
//        Ask=48; /* 0 reset */

        /* send data after getting request */
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, 20);
        lux=HAL_ADC_GetValue(&hadc1);
        //FIRFilter_Update(&lplux,lux);
        x =(int)RCFilter_Update(&lplux,lux)/27;
    }
}

```

```

    /*filtred[0]=x/100;
    filtred[1]=x%100;*/

    Send_Data();

    /* wait for Ask code */
//    while(Ask!=49)
//    {
//        HAL_UART_Receive(&huart2, &Ask,1, HAL_MAX_DELAY);
//        if(Ask==48)
//        {
//            HAL_ADC_Start(&hadc1);
//            HAL_ADC_PollForConversion(&hadc1, 20);
//            lux=HAL_ADC_GetValue(&hadc1);
//            Send_Data();
//        }
//    }
/* END of while loop 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{

}

/**
 * @brief ADC1 Initialization Function
 * @param None
 * @retval None
 */
static void MX_ADC1_Init(void)
{

    /* USER CODE BEGIN ADC1_Init 0 */

    /* USER CODE END ADC1_Init 0 */

    ADC_ChannelConfTypeDef sConfig = {0};

    /* USER CODE BEGIN ADC1_Init 1 */

    /* USER CODE END ADC1_Init 1 */
    /** Configure the global features of the ADC (Clock, Resolution, Data

```



```

Alignment and number of conversion)
*/
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.ScanConvMode = DISABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.NbrOfConversion = 1;
hadc1.Init.DMAContinuousRequests = DISABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
    Error_Handler();
}
/** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
*/
sConfig.Channel = ADC_CHANNEL_0;
sConfig.Rank = 1;
sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/* USER CODE BEGIN ADC1_Init 2 */

/* USER CODE END ADC1_Init 2 */

}

static void MX_USART2_UART_Init(void)
{
    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_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    //huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
}

```

```

    }
}

void Error_Handler(void)
{
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line
    number,
    tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
    */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/***** (C) COPYRIGHT STMicroelectronics *****END OF
FILE*****/

```

Le code aussi fait appel à la bibliothèque header `#include "RCFilter.h"` qui contient les prototypes de l'implémentation des fonctions du filtre numérique, voici son implémentation :

```

#include "RCFilter.h"

void RCFilter_Init(RCFilter *filt, float cutoffFreqHz, float sampleTimeS)
{
    /* Compute equivalent 'RC' constant from cut-off frequency */
    float RC = 1.0f / (6.28318530718f * cutoffFreqHz);

    /* Pre-compute filter coefficients for first-order low-pass filter */
    filt->coeff[0] = sampleTimeS / (sampleTimeS + RC);
    filt->coeff[1] = RC / (sampleTimeS + RC);

    /* Clear output buffer */
    filt->out[0] = 0.0f;
}

```

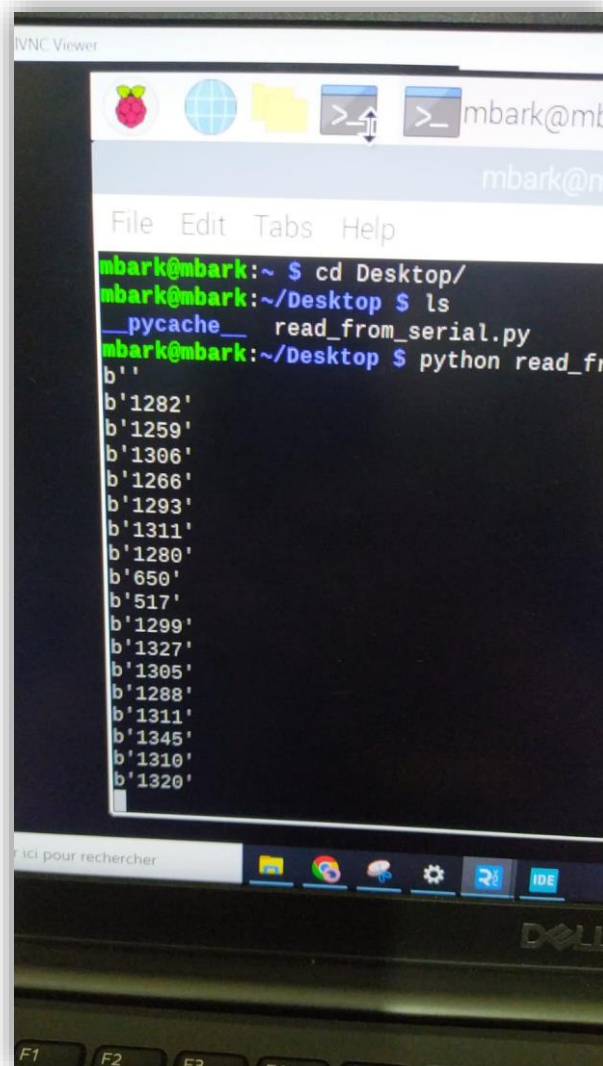
```
filt->out[1] = 0.0f;
}
float RCFilter_Update(RCFilter *filt, float inp) {

    /* Shift output samples */
    filt->out[1] = filt->out[0];

    /* Compute new output sample */
    filt->out[0] = filt->coeff[0] * inp + filt->coeff[1] * filt->out[1];

    /* Return filtered sample */
    return (filt->out[0]);
}
```

Test et validation :



```
VNC Viewer
mbark@mbark:~$ cd Desktop/
mbark@mbark:~/Desktop$ ls
__pycache__  read_from_serial.py
mbark@mbark:~/Desktop$ python read_fr
b''
b'1282'
b'1259'
b'1306'
b'1266'
b'1293'
b'1311'
b'1280'
b'650'
b'517'
b'1299'
b'1327'
b'1305'
b'1288'
b'1311'
b'1345'
b'1310'
b'1320'
```

FIGURE 22:RESULTATS DE MESUR

Process général :

Système d'éclairage automatique utilise une combinaison de capteurs, de microcontrôleurs, et de plateformes de traitement pour offrir une solution efficace, intelligente et économe en énergie pour la gestion de l'éclairage public.

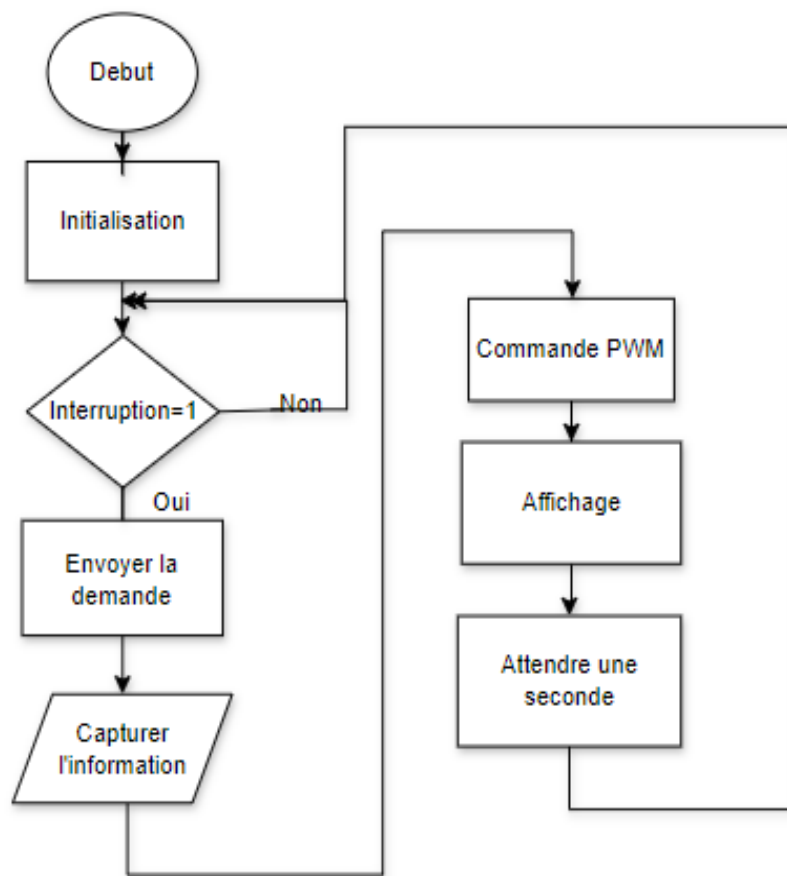


FIGURE 23:DESCRIPTION ALGORITHMIQUE DU SYSTEME

Le système surveille une interruption pour déterminer si de nouvelles données doivent être capturées.

L'interruption est déclenchée lorsque le comparateur LM393 si le comparateur (Interruption = 1), le système procède à l'étape suivante.

On décrit le système présenté par les étapes suivants :

- Début et Initialisation :

Le système commence par l'initialisation de tous les composants, incluant le capteur de lumière LDR, le microcontrôleur STM32F4, le Raspberry Pi, l'écran LCD, et le comparateur LM393.

- Interruption :

Le système surveille une interruption pour déterminer si de nouvelles données doivent être capturées.

Si une interruption est détectée (Interruption = 1), le système procède à l'étape suivante.

- Envoi de la Demande et Capture de l'Information:

Le Raspberry pi envoie une demande de capture l'information sur l'intensité lumineuse.

- Commande PWM et Affichage :

Le Raspberry Pi utilise les données reçues pour ajuster l'intensité lumineuse des LED à l'aide de la modulation de largeur d'impulsion (PWM).

L'état et les informations de l'éclairage sont affichés sur un écran LCD.

- Attente et Boucle:

Le système attend une seconde avant de recommencer le processus en vérifiant à nouveau l'état de l'interruption.

Le signal PWM :

La modulation de largeur d'impulsion (PWM) est une technique largement utilisée pour contrôler la puissance délivrée à des dispositifs tels que les moteurs, les LED, les servomoteurs, etc. Elle implique la modulation de la largeur des impulsions d'un signal périodique pour ajuster la quantité de puissance transmise à un dispositif.

On a utilisé PWM pour le control de luminosité de la LED.

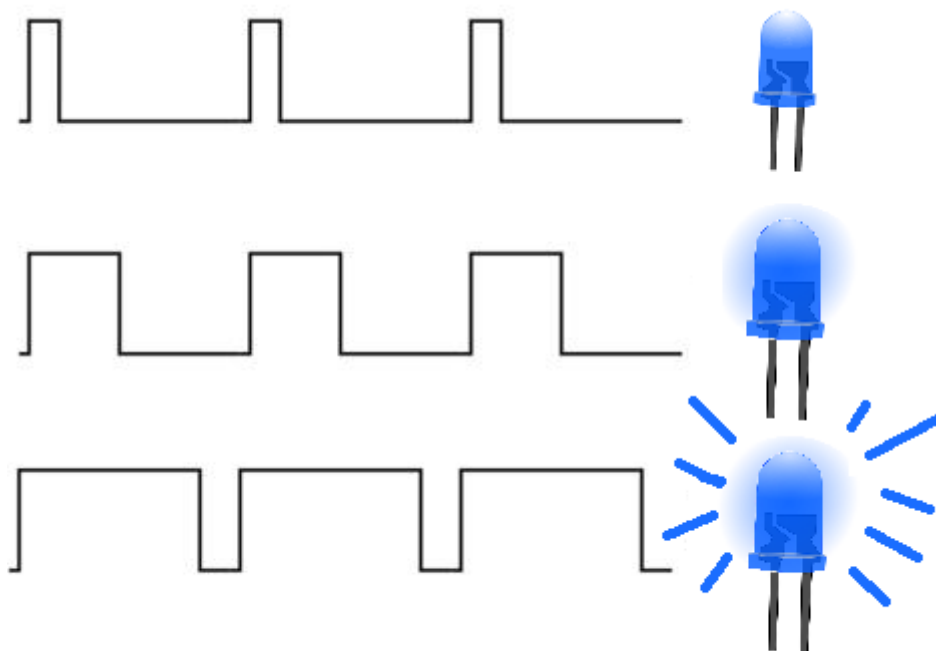


FIGURE 24: SIGNAL PWM

Implémentation en python :

```
import RPi.GPIO as GPIO
import serial
import time
from smbus2 import SMBus
import lcd_i2c

# Constants
INP_PIN = 4
SERIAL_PORT = "/dev/ttyACM0"
LCD_ADDR = 0x27
PWM_PIN = 18
ALPHA = 0.5

# Initialize global variables
filtered_duty_cycle = 0.0

# Initialize GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(INP_PIN, GPIO.IN)
GPIO.setup(PWM_PIN, GPIO.OUT)

# Initialize PWM
pwm = GPIO.PWM(PWM_PIN, 1000) # 1000 Hz frequency
pwm.start(0) # Start with 0% duty cycle

# Initialize serial communication
ser = serial.Serial(SERIAL_PORT, 115200, timeout=1)

# Initialize I2C for LCD
bus = SMBus(1)
lcd = lcd_i2c.lcd(bus, LCD_ADDR)

def filter(input_duty_cycle):
    global filtered_duty_cycle
    filtered_duty_cycle = ALPHA * input_duty_cycle + (1 - ALPHA) * filtered_duty_cycle

# Main loop
try:
    while True:
        lcd.lcd_display_string("Bright", 1)

        if GPIO.input(INP_PIN) == GPIO.HIGH:
            # Send request to STM32
            ser.write(b'a')

            # Capture dt from measurement unit
            dt_str = ser.read(2).decode('utf-8')
            if dt_str:
                dt = int(dt_str)

            # Apply simple filter (low-pass)
            filter(dt)
```

```

        # Set PWM duty cycle based on filtered value
        pwm.ChangeDutyCycle(filtered_duty_cycle)

        # Display behavior
        lcd lcd_display_string("Duty cycle is:", 1)
        lcd lcd_display_string(f"{filtered_duty_cycle:.1f}%", 2)

        time.sleep(0.1)
except KeyboardInterrupt:
    pass
finally:
    # Clean up
    ser.close()
    pwm.stop()
    GPIO.cleanup()

```

On utilise des bibliothèques adaptées à un environnement Raspberry Pi. Il commence par importer les modules nécessaires: RPi.GPIO pour la gestion des GPIO, serial pour la communication série, time pour les fonctions de temporisation, et smbus2 et lcd_i2c pour la communication I2C et le contrôle de l'affichage LCD. Les constantes pour les broches GPIO, le port série, l'adresse LCD, la broche PWM et la valeur alpha du filtre sont définies.

L'initialisation configure les broches GPIO pour l'entrée et la sortie PWM, initialise le PWM avec une fréquence de 1000 Hz, et démarre la communication série et l'affichage LCD. La fonction de filtrage applique un filtre passe-bas simple pour lisser les valeurs du cycle de travail en entrée. Dans la boucle principale, le programme vérifie en continu l'état de la broche d'entrée. Lorsque la broche est en état HAUT, il envoie une demande au STM32 via la communication série, lit la réponse, applique le filtre aux données reçues, met à jour le cycle de travail PWM et affiche le cycle de travail sur l'écran LCD. Le code utilise un bloc try-except pour gérer les interruptions clavier et assure que le port série est fermé, le PWM est arrêté et les broches GPIO sont nettoyées en cas d'arrêt du programme.

Implémentation en C :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wiringPi.h>
#include <stdint.h>
#include <wiringSerial.h>
#include <wiringPiI2C.h>
#include <i2c1602.h>

#define INP_PIN 4
#define SERIAL_PORT "/dev/ttyACM0"
#define LCD_ADDR 0x27
I2C16x2 lcd;
// GPIO pin for PWM
#define PWM_PIN 18

// FIR filter constants
#define ALPHA 0.5

// Function prototypes
void filter(double input_duty_cycle);

// Global variables
int filtered_duty_cycle = 0.0;
int serial_fd;

// Simple second order FIR filter
void filter(double input_duty_cycle) {
    filtered_duty_cycle = ALPHA * input_duty_cycle + (1 - ALPHA) * filtered_duty_cycle;
}

int main() {
    // Initialize wiringPi and GPIO pins and the display and the communication

    /******
    *****
    wiringPiSetup();
    fd=serialOpen("/dev/ttyACM0", 115200);
    lcd_init(LCD_ADDR);
    ClrLcd();

    if (wiringPiSetup () == -1)
        exit (1) ;
    pinMode(INP_PIN, INPUT);
    pinMode (PWM_PIN, PWM_OUTPUT);

    // Initialize PWM
    softPwmCreate(PWM_PIN, 0, 100);

    // Initialize serial communication
    if ((serial_fd = serialOpen(SERIAL_PORT, 115200)) < 0) {
        fprintf(stderr, "Unable to open serial device\n");
        return 1;
    }
}
```

```

}

//*****
// Main loop
while (1) {
    lcdLoc(LINE1+1);
    typeString("Bright");
    // Wait for GPIO pin interrupt
    if (digitalRead(GPIO_PIN) == HIGH) {
        // Send request to STM32
        serialPuchar(serial_fd, 'a');

        // Capture dt from measurement unit
        char dt_str[2];
        int i = 0;
        while (serialDataAvail(serial_fd) > 0 && i <= 1) {
            dt_str[i++] = serialGetchar(serial_fd);
        }
        dt_str[i] = '\0';
        int dt = atoi(dt_str);

        //Simple filter application(Low passe)
        filter(dt);

        // Set PWM duty cycle based on filtered value
        softPwmWrite(PWM_PIN, (int)filtered_duty_cycle);

        //Display behavior
        lcdLoc(LINE1+1);
        typeString("Duty cycle is:");
        lcdLoc(LINE2+0);
        typeString(filtered_duty_cycle);
        lcdLoc(LINE2+10);
        typeString("%");
        usleep(100000);
    }
}

// Clean up
serialClose(serial_fd);
return 0;
}

```

Le programme commence par inclure des bibliothèques standard ainsi que les bibliothèques wiringPi pour la gestion des GPIO, la communication série et I2C. Les constantes pour la broche d'entrée, le port série, l'adresse LCD, la broche PWM et le filtre FIR sont définies. L'initialisation configure la bibliothèque wiringPi, la communication série, les broches GPIO, et l'affichage LCD. La fonction de filtrage applique un filtre passe-bas simple pour lisser les valeurs du cycle de travail en entrée. Dans la boucle principale, le programme vérifie en continu l'état d'une broche d'entrée. Lorsque la broche est en état HAUT, il envoie une demande à un

microcontrôleur STM32 pour obtenir une valeur d'intensité lumineuse (dt), applique le filtre à cette donnée, met à jour le cycle de travail PWM et affiche le cycle de travail sur l'écran LCD. Enfin, le programme ferme la communication série et se termine proprement.

Implémentation en C++ :

```
#include <iostream>
#include <wiringPi.h>
#include <softPwm.h>
#include <wiringSerial.h>
#include <lcd.h>

#define INP_PIN 4
#define SERIAL_PORT "/dev/ttyACM0"
#define LCD_ADDR 0x27
#define GPIO_PIN INP_PIN
#define LINE1 0x80
#define LINE2 0xC0

// FIR filter constants
const double ALPHA = 0.5;

// Function prototypes
void filter(double input_duty_cycle);

// Global variables
double filtered_duty_cycle = 0.0;
int serial_fd;

// Simple second order FIR filter
void filter(double input_duty_cycle) {
    filtered_duty_cycle = ALPHA * input_duty_cycle + (1 - ALPHA) * filtered_duty_cycle;
}

int main() {
    // Initialize wiringPi and GPIO pins and the display and the communication
    wiringPiSetup();
    serial_fd = serialOpen(SERIAL_PORT, 115200);
    int lcd_fd = lcdInit(2, 16, 4, LCD_ADDR, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    if (lcd_fd == -1) {
        std::cerr << "LCD initialization failed" << std::endl;
        return 1;
    }
    pinMode(INP_PIN, INPUT);
    pinMode(PWM_PIN, PWM_OUTPUT);

    // Initialize PWM
    softPwmCreate(PWM_PIN, 0, 100);

    // Initialize serial communication
    if (serial_fd < 0) {
        std::cerr << "Unable to open serial device" << std::endl;
        return 1;
    }
}
```

```

}

// Main loop
while (true) {
    lcdPosition(lcd_fd, 0, 0);
    lcdPuts(lcd_fd, "Bright");
    // Wait for GPIO pin interrupt
    if (digitalRead(GPIO_PIN) == HIGH) {
        // Send request to STM32
        serialPutchar(serial_fd, 'a');

        // Capture dt from measurement unit
        char dt_str[3] = {0};
        int i = 0;
        while (serialDataAvail(serial_fd) > 0 && i < 2) {
            dt_str[i++] = serialGetchar(serial_fd);
        }
        dt_str[i] = '\0';
        int dt = atoi(dt_str);

        // Simple filter application(Low pass)
        filter(dt);

        // Set PWM duty cycle based on filtered value
        softPwmWrite(PWM_PIN, static_cast<int>(filtered_duty_cycle));

        // Display behavior
        lcdPosition(lcd_fd, 0, 0);
        lcdPuts(lcd_fd, "Duty cycle is:");
        lcdPosition(lcd_fd, 0, 1);
        lcdPrintf(lcd_fd, "%.2f%%", filtered_duty_cycle);
        delay(100);
    }
}

// Clean up
serialClose(serial_fd);
return 0;
}

```

a

Implémentation en Matlab Simulink :

Introduction :

Le module développer dans Simulink pour cette application se compose de 3 sous modules, le 1^{er} pour recevoir des données depuis le capteur, le deuxième pour l'envoi de la requête au capteur et le dernier pour commander la LED. La figure suivante montre le module global qui ressemble les trois sous modules.

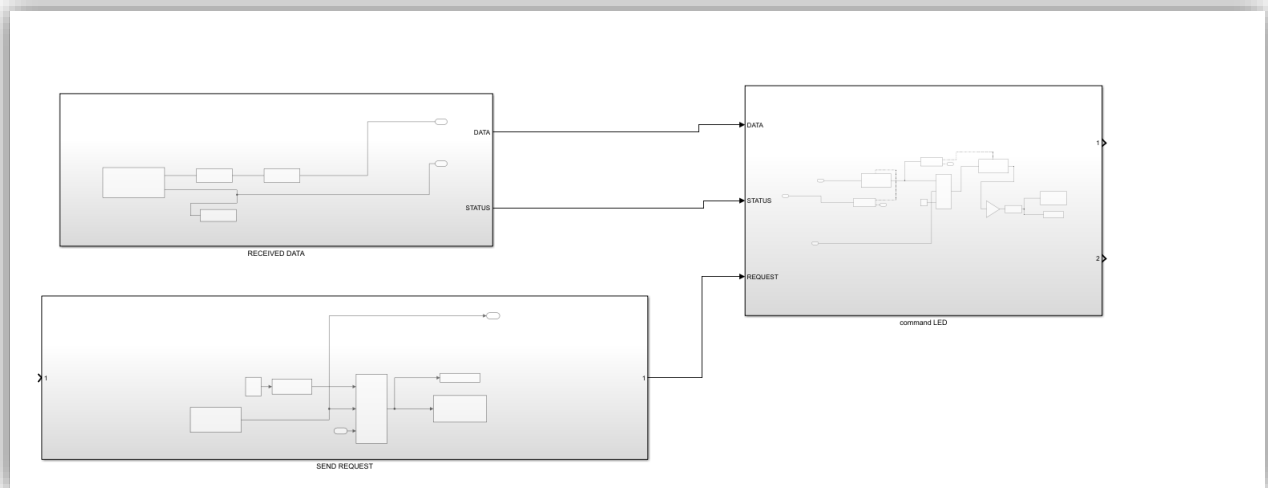


FIGURE 25: MODULE GLOBALE

Sous module 1 : RECEIVE DATA

Ce module reçoit des données depuis le capteur d'après le protocole UART, ces données sont converties depuis ASCII vers DECIMALE par les blocs spécifiques dans la figure suivant :

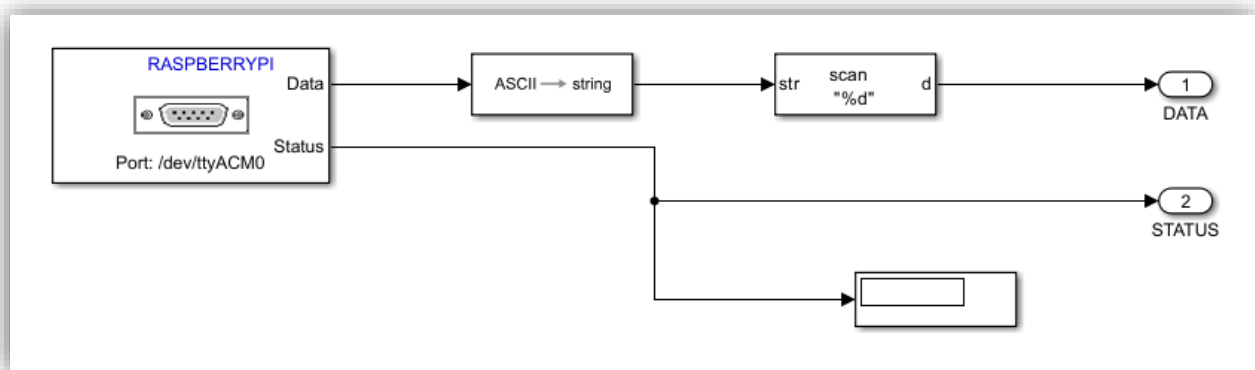


FIGURE 26: SOUS MODULE 1 : RECEIVE DATA

Les chemins des blocs utiliser pour le sous module 1 sont :

❖ **Bloc Serial Read :**

Library browser » Simulink Support Package for Raspberry Pi Hardware » communication » Serial Read.

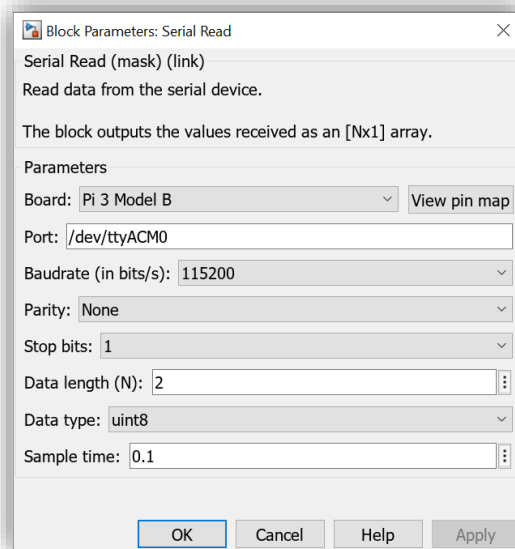


FIGURE 27: PARAMETRES DU BLOC SERIAL READ

❖ **ASCII -> STRING :**

Library browser » Simulink » String » ASCII TO String

❖ **STRING -> DECIMAL**

Library browser » Simulink » String » String to Double

En change double par %d pour sélectionner décimal.

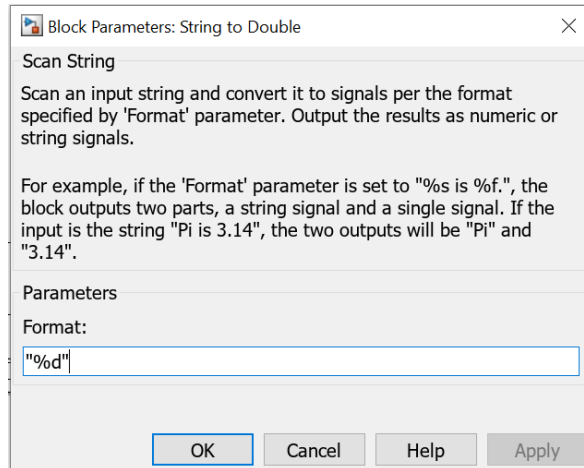


FIGURE 28: BLOCK STRING TO DECIMAL

Sous module 2: SEND REQUEST

Ce module transmet une requête qui est le caractère 'a' (49 en ascii) vers le capteur.

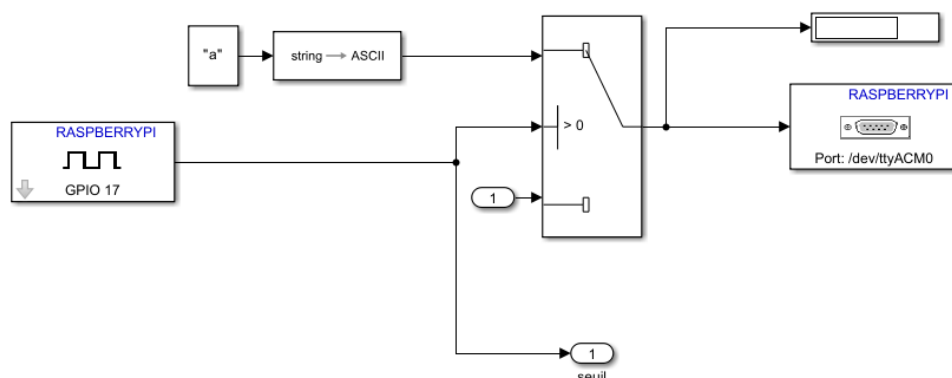


FIGURE 29: SOUS MODULE 2 : SEND REQUEST

Les chemins des blocs utiliser pour le sous module 2 sont :

❖ Bloc GPIO Read :

Library browser » Simulink Support Package for Raspberry Pi Hardware » Basic » GPIO Read.

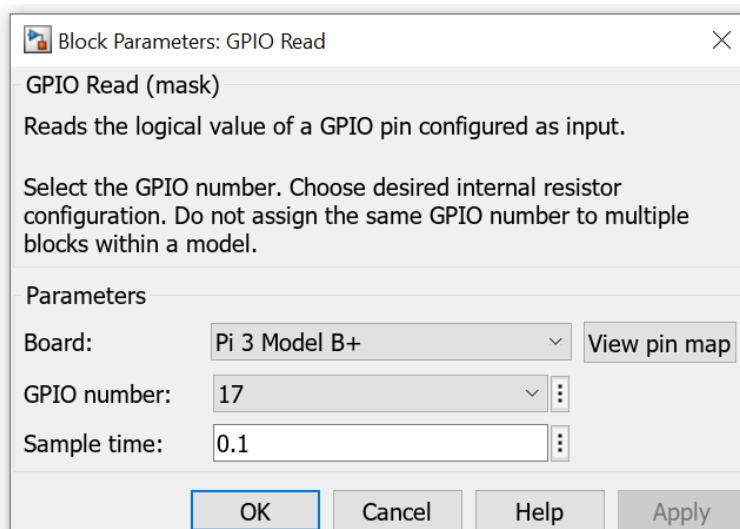


FIGURE 30: BLOCK GPIO READ

Ce block utilisé pour lire le seuil du module LDR depuis le pin 17 avant d'envoyer la requête vers stm32.

❖ Bloc Serial Write :

Library browser » Simulink Support Package for Raspberry Pi Hardware » communication » Serial Write.

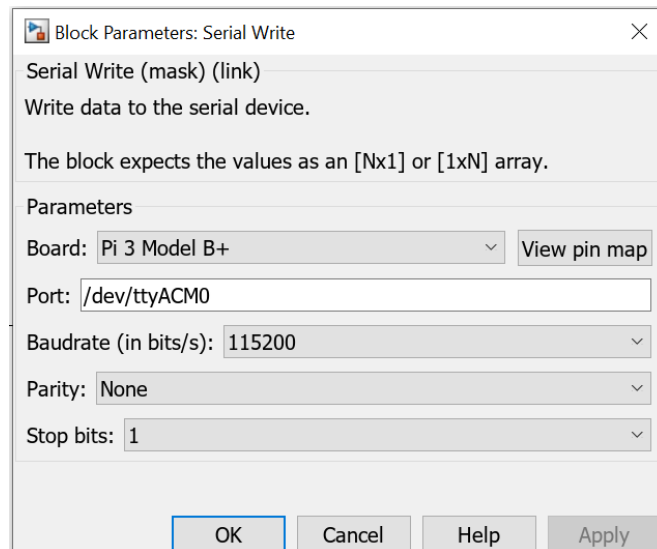


FIGURE 31:PARAMETRES DU BLOC SERIAL WRITE

Ce block utilise pour envoyer la requête 'a' vers stm32 pour recevoir une donnée qui présent le rapport cyclique du signal PWM afin de commander la LED

Sous module 3: COMMANDE LED

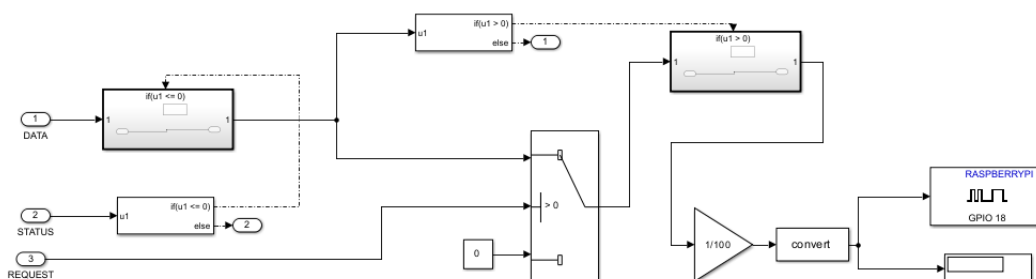


FIGURE 32: SOUS MODULE 3 : COMMANDE LED

Ce module reçoive le rapport cyclique se forme d'une donnée décimale pour commander la LED relia au pin 18.

D'abord si le STATUS égale 0 qui signifie que la communication UART est réussite le switch laisse passe la donnée du capteur après que la requête était envoyée pour éviter de lire des données incorrectes.

Après on divise la donnée par 100 pour obtenir des valeurs entre 0 et 1, Afin de commander la LED.

Les chemins des blocs utiliser pour le sous module 3 sont :

❖ Bloc if condition :

Library browser » Simulink » Port & subsystems » if.

Library browser » Simulink » Port & subsystems » if action subsystem.

Le principe de fonctionnement de ces deux blocks est lorsque la condition est respectée par le bloc « *if* » le block « *if action subsystem* » laisse passer la donner vers l'autre partie.

❖ Bloc PWM :

Library browser » Simulink Support Package for Raspberry Pi Hardware » basic » PWM.

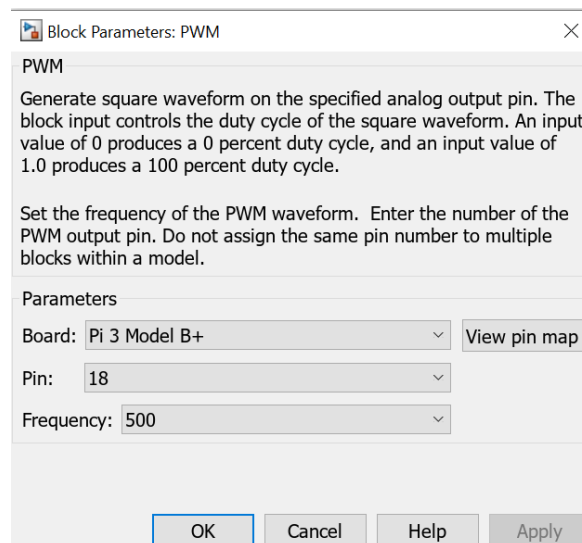


FIGURE 33: PARAMETRE DU BLOC PWM

Conclusion :

Ce projet utilise une combinaison de capteurs, microcontrôleurs et plateformes de traitement pour créer un système d'éclairage intelligent, efficace et écoénergétique. En exploitant les technologies modernes de l'IoT, il propose une solution innovante pour la gestion de l'éclairage public, avec des avantages significatifs en termes d'économie d'énergie et de réduction de l'empreinte carbone.