ESMT

École Supérieure Multinationale des Télécommunications

Pratique des Microprocesseurs

Projet Digital Humidity and Temperature (DHT11)

Membres du groupe 6 :

- DJIGUIMDE Oumaro Titans
- OLOKE Ezin Léandre
- PAPY Diandy
- BARA tiaw

Encadré par :

Pr Fabe Idrissa BARRO

Licence en science de l'ingénieur

INGC1

2024-2025

Sommaire

I. Sigles et abréviations 5

II. Présentation du projet 6

Objectif général :..... 6

Objectif spécifique :..... 6

III. Equipements utiliser..... 7

1. Atmega 2560 : 7

1.1. Caractéristiques : 7

1.2. Ports : 7

2. DHT11 8

2.1. Caractéristiques :..... 8

2.2. Brochages :..... 9

2.3. Principe de fonctionnement :..... 9

3. LCD LM0044L :..... 11

3.1. Caractéristiques principales : 11

3.2. Brochage : 11

3.3. Principe de fonctionnement : 12

4. Led : 12

5. Button poussoir : 12

IV. Câblage : 13

V. Code : 15

1. La fonction init_ports() : 15

2. La fonction mesure_temp_hum() : 16

3. La fonction reglage_alarme() : 17

4. La fonction alarme () : 20

VI. Diagramme d'état du système : 22

1. Etat 1 mode normal : 22

2. Etat 2 mode de configuration d'alarme : 22

3. Etat 3 mode alarme : 22

VII. Conclusion : 24

VIII. Références : 25

Table des illustrations

Figure 1 : ATMEGA 2560 8

Figure 2 : capteur de temperature dht11 8

Figure 3 : LCD 20*4 12

Figure 4 : leds 12

Figure 5 : button poussoir 12

Figure 6 : schémas de cablage 13

Figure 7 : diagramme d'etat du système 23

I. Sigles et abréviations

- AVR : Advanced Virtual RISC
- LCD : Liquid Crystal Display
- DHT11 : Digital Humidity and Temperature sensor
- LED : Light Emitting Diode.

II. Présentation du projet

Ce projet constitue l'évaluation final du chapitre sur les microcontrôleurs et microprocesseur. Il consiste en la réalisation sous Proteus d'un circuit capable de lire la température et l'humidité sur un capteur de température dht11 de l'afficher sur un écran LCD, le circuit devras permettre la configuration d'un seuil inferieur et supérieur de température afin de pouvoir gérer une alarme qui s'enclenchera dès qu'un des seuils est atteint que ce soit pour la température ou l'humidité.

Objectif général :

- Comprendre le fonctionnement des microcontrôleurs
- Apprendre à écrire du code C interprétable par des microcontrôleurs
- Se familiarisé avec la gestion de mémoire et de fréquence d'exécution
- Se familiarisé avec le logiciel Proteus

Objectif spécifique :

- Réaliser un câblage lisible et précis sous Proteus
- Ecrire du code c maintenable et exécutable par des microcontrôleurs
- Apprendre à gérer l'affichage sur des LCD.

III. Equipements utiliser

Pour réaliser ce projet nous avons eu recours aux équipements suivants :

1. Atmega 2560 :

L'ATmega2560 est un microcontrôleur populaire de la famille AVR fabriqué par **Microchip Technology**, qui a acquis Atmel Corporation (le fabricant original des microcontrôleurs AVR).

1.1. Caractéristiques :

- Architecture : AVR 8bits
- Mémoire Flash : 256 KO
- EEPROM : 4 KO
- SRAM : 8 KO
- Fréquence maximale de l’horloge : 16MHZ
- Tension de fonctionnement : 5V

1.2. Ports :

- E/S : 11 ports (PA-PL) de 8lignes chacune (0 à 7)

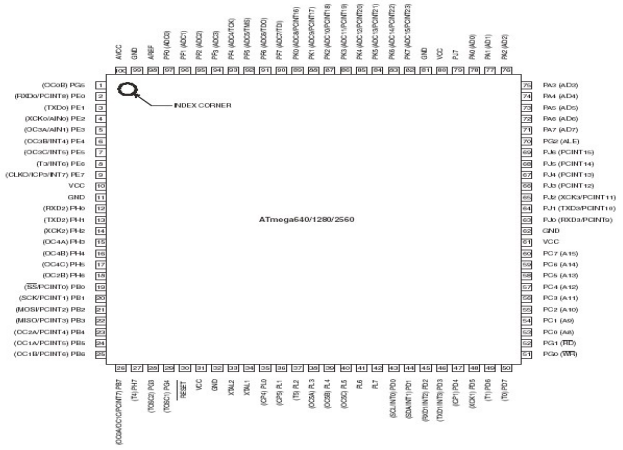


Figure 1 : ATMEGA 2560

2. DHT11 :

Le dht11 est un capteur de température et d'humidité à sortie numérique calibré. Sa technologie garantit une grande fiabilité et une excellente stabilité à long terme.

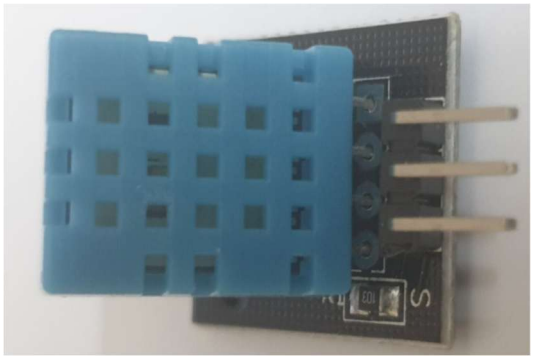


Figure 2 : capteur de temperature dht11

2.1. Caractéristiques :

- Plage de mesure de l'humidité : 20-90% HR (Humidité Relative)
- Précision de l'humidité : ±5% HR
- Plage de mesure de la température : 0-50°C

- Précision de la température : ±2°C
- Tension d’alimentation : 3.5V à 5.5V

2.2. **Brochages** :

- VCC : Alimentation
- Data : Broche de données
- NC : non connecté
- GND : Masse

2.3. **Principe de fonctionnement** :

Le capteur DHT11 est conçu pour mesurer la température et l'humidité de l'air. Il combine un capteur d'humidité capacitif et un thermistor pour mesurer les conditions de l'air et envoie les données via une interface numérique à un microcontrôleur.

- **Capteur d’humidité capacitif :**

  Le capteur d'humidité capacitif utilise un polymère diélectrique qui absorbe ou libère de l'eau en fonction de l'humidité ambiante. Le polymère est placé entre deux électrodes, formant un condensateur dont la capacité varie avec l'humidité relative (HR) de l'air. La capacité change proportionnellement à l'humidité, et cette variation est convertie en un signal électrique mesurable.

- **Thermistor :**

  Le thermistor est un composant électronique dont la résistance change avec la température. Le DHT11 utilise un thermistor pour mesurer la température ambiante. Les variations de résistance du thermistor dues à la température sont converties en un signal électrique mesurable.

- **Interface de communication numérique :**

  Le DHT11 communique avec un microcontrôleur via une interface numérique à une seule broche. La communication suit un protocole spécifique pour transmettre les données de température et d'humidité. Voici les étapes de la communication :

  - Signal de démarrage : Le microcontrôleur envoie un signal de démarrage en mettant la ligne de données à bas pendant au moins 18 ms, puis en la mettant à haut pendant environ 20-40 µs.
  - Réponse du capteur : Le DHT11 répond en mettant la ligne de données à bas pendant 80 µs, puis à haut pendant 80 µs.
  - Transmission des données : Le DHT11 envoie ensuite 40 bits de données au microcontrôleur. Les 40 bits sont divisés en cinq segments de 8 bits :
    - 8 bits pour l'humidité intégrale (partie entière)
    - 8 bits pour l'humidité décimale
    - 8 bits pour la température intégrale (partie entière)
    - 8 bits pour la température décimale
    - 8 bits pour le contrôle de parité (somme de contrôle)

    Chaque bit est transmis en deux phases :

Une phase de démarrage (50 µs à bas)

Une phase de données (26-28 µs à haut pour un bit 0, 70 µs à haut pour un bit 1).

3. LCD LM016L :

Le LM0044L est un module d'affichage à cristaux liquides (LCD) couramment utilisé dans les projets électroniques en raison de sa compatibilité avec le contrôleur HD44780 et sa facilité d'utilisation avec des microcontrôleurs comme l'ATmega2560.

3.1. Caractéristiques principales :

- Type d'affichage : à cristaux liquide
- Configuration : 16 caractères par ligne, 2 lignes
- Contrôleur intégré : Compatible avec le HD44780
- Mode de communication : Interface parallèle (4 bits ou 8 bits)
- Tension d'alimentation : 5V DC
- Taille des caractères : 5×8 points.

3.2. Brochage :

- VSS : Masse(0V)
- VDD : Alimentation(5V)
- VEE : ajustement du contraste (connecter à un potentiomètre)
- RS : Registre de sélection
- RW : Read/Write
- E : Enable

- D0-D7 : ligne de données.

3.3. **Principe de fonctionnement** :

Les LCD (Liquid Crystal Display) comme le LM016L 16x2 fonctionne en utilisant des cristaux liquides pour moduler la lumière et afficher des caractères.

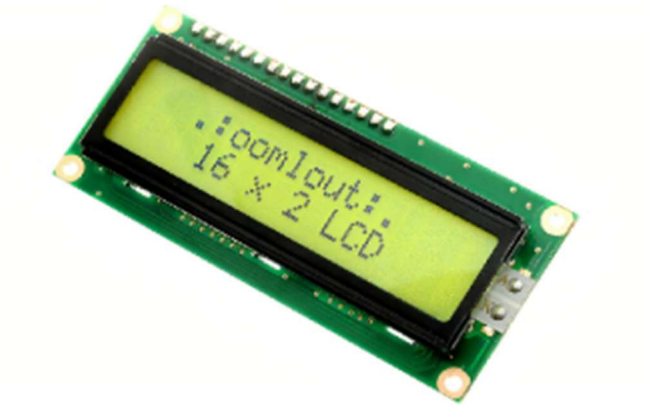


*Figure 3 : LCD 16*2*

4. Led :

Une LED est un type de diode qui émet de la lumière lorsqu'un courant électrique la traverse.

Figure 4 : leds

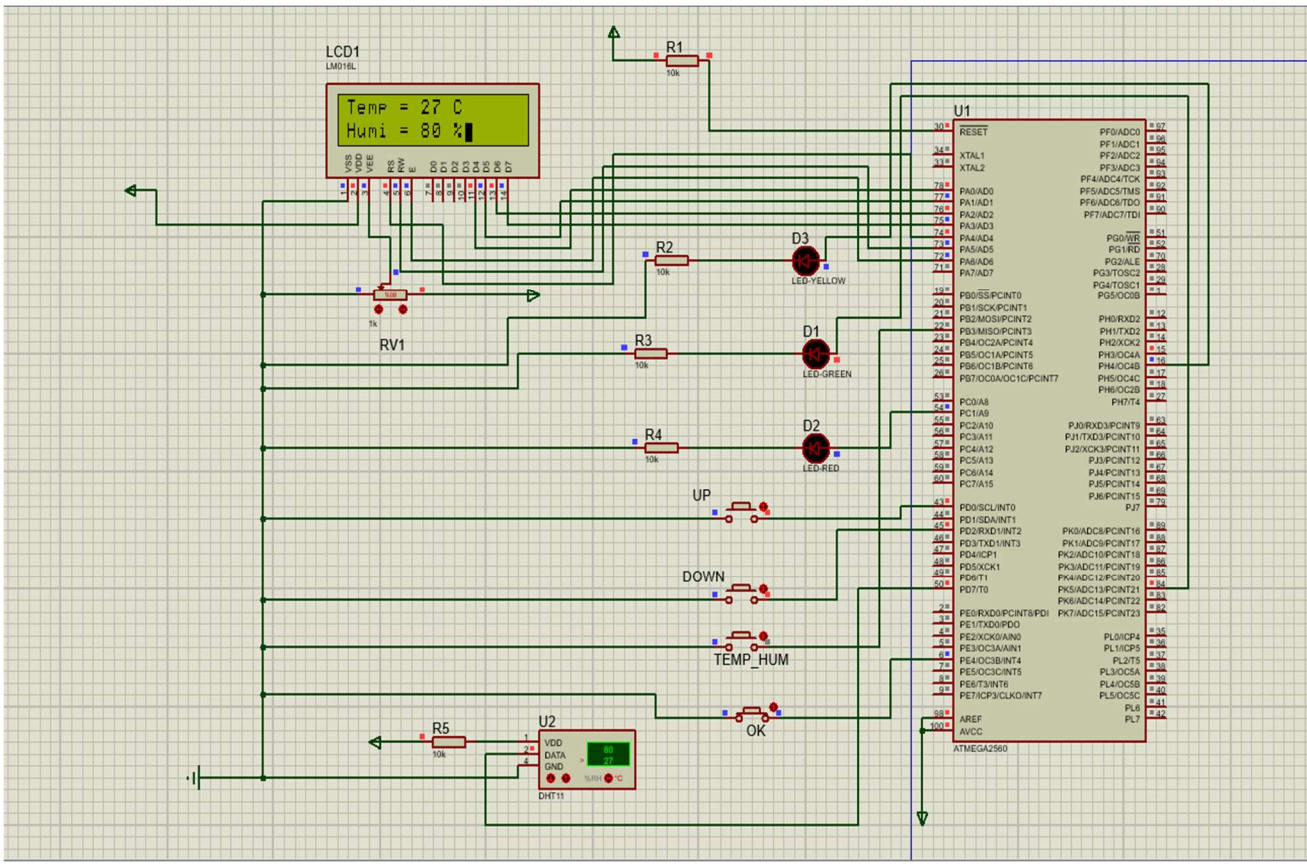
5. Button poussoir :

Un bouton poussoir fonctionne en ouvrant ou fermant un circuit électrique lorsqu'il est pressé.

Figure 5 : button poussoir

IV. Câblage :

Cette partie présenteras le câblage sous Proteus :



❖ Details :

- Bouton UP sur PD0
- Bouton down sur PD2
- Bouton TEMP_HUM sur PB3(car pas d'interruption sur PH3)
- Bouton OK sur PE4

Figure 6 : schémas de cablage

- Led jaune sur PH4
- Led verte sur PK5
- Led rouge sur PC1

```
// === Définition des broches pour LEDs ===
#define led_rouge PC1
#define led_jaune PH4
#define led_verte PK5

// === Capteur DHT11 ===

#define DHT11_PIN PD7

// === Boutons physiques ===

#define BTN_UP PD0 // Augmenter les seuils
#define BTN_DOWN PD2 // Diminuer les seuils
#define BTN_TEMP_HUM PB3 // Car il n'y a pas d'interruption sur le port PH3 : // Changer entre température
#define BTN_OK PE4 // Valider / quitter le mode réglage
```

- Broche data du dht11 sur PD7
- D4 sur PA0
- D5 sur PA1
- D6 sur PA2
- D7 sur PA3
- RS sur PA4
- RW sur PA5
- E sur PA6

V. Code :

Le code de ce projet est articulé autour de 4 grosses fonctions :

```
// Variables globales

// === Variables globales pour état des boutons (modifiées dans les interruptions) ===

volatile uint8_t btn_up = 0;
volatile uint8_t btn_down = 0;
volatile uint8_t btn_temp_hum = 0;
volatile uint8_t btn_ok = 0;

// === États de fonctionnement ===

uint8_t mode_reglage = 0;
uint8_t type_selection = 0;
uint8_t seuil_selection = 0;

// === Valeurs lues ===

uint8_t temp_actuelle = 25;
uint8_t hum_actuelle = 60;


// === Seuils par défaut ===

uint8_t temp_sup = 30;
uint8_t temp_inf = 20;
uint8_t hum_sup = 70;
uint8_t hum_inf = 50;

// === Buffer pour affichage LCD ===
char buffer[32];

// === Prototypes de fonctions ===

void init_ports(void);
void mesure_temp_hum(void);
void reglage_alarme(void);
void alarme(void);
uint8_t lire_dht11(uint8_t *temp, uint8_t *hum);
```

1. La fonction init_ports() :

Cette fonction est relativement simple et est là pour initialiser les ports utilisés, c'est-à-dire mètre les ports en sortie ou en entrée et désactiver ou activer les résistances internes pour ces ports.

```
// === Initialisation des ports et des interruptions ===

void init_ports(void)
{
    // LEDs en sortie
    DDRC |= (1 << led_rouge);
    DDRH |= (1 << led_jaune);
    DDRK |= (1 << led_verte);

    // Éteindre les LEDs au départ
    PORTC &= ~(1 << led_rouge);
    PORTH &= ~(1 << led_jaune);
    PORTK &= ~(1 << led_verte);

    // Boutons en entrée avec pull-up
    DDRD &= ~((1 << BTN_UP) | (1 << BTN_DOWN));
    DDRE &= ~(1 << BTN_OK);
    DDRH &= ~(1 << BTN_TEMP_HUM);

    PORTD |= ((1 << BTN_UP) | (1 << BTN_DOWN));
    PORTE |= (1 << BTN_OK);
    PORTH |= (1 << BTN_TEMP_HUM);
```

2. La fonction mesure_temp_hum() :

Cette fonction est celle qui permet de récupérer les données fournis par le capteur d'identifier les valeurs de température et d'humidité et de les afficher :

```
// === Affichage des mesures sur LCD ===
void mesure_temp_hum(void)
{
    PORTK |= (1 << led_verte);// LED verte allumée (fonctionnement normal)

    if (lire_dht11(&temp_actuelle, &hum_actuelle)) {
        cls_LCD();
        pos_xy(1, 1);
        sprintf(buffer, "Temp = %d C", temp_actuelle);
        Write_LCD(buffer);
        pos_xy(1, 2);
        sprintf(buffer, "Humi = %d %%", hum_actuelle);
        Write_LCD(buffer);
    }
}
```

3. La fonction reglage_alarme() :

Cette fonction est sans doute la plus lourde de toutes. Elle a pour but de configurer les seuil inférieur et supérieur de température pour le déclanchement d'une alarme. Pour réaliser ce traitement nous avons opté pour une logique en boucle qui se présente comme ci :

```c
// === Mode réglage des seuils ===
void reglage_alarme(void)
{
    PORTK &= ~(1 << led_verte);// Éteindre LED verte pour indiquer réglage

    while(1)
    {
        // Quitter le mode réglage si bouton OK maintenu
        if(btn_ok) {
            btn_ok = 0;
            _delay_ms(1000);
            if(!(PINE & (1 << BTN_OK))) {
                mode_reglage = 0;
                return;
            }
        }
        // Bascule entre température et humidité
        if(btn_temp_hum) {
            btn_temp_hum = 0;
            type_selection = !type_selection;
        }
```

```c
        // Augmentation des seuils
        if(btn_up) {
            btn_up = 0;
            _delay_ms(1000);
            if(!(PIND & (1 << BTN_UP))) {
                seuil_selection = 0;// Sélectionner seuil haut
            } else {
                if(type_selection == 0) {
                    // Température
                    if(seuil_selection == 0 && temp_sup < 50) temp_sup++;
                    else if(seuil_selection == 1 && temp_inf < temp_sup) temp_inf++;
                } else {
                    // Humidité
                    if(seuil_selection == 0 && hum_sup <= 90) hum_sup += 10;
                    else if(seuil_selection == 1 && hum_inf <= hum_sup) hum_inf += 10;
                }
            }
        }
```

```
        // Diminution des seuils
        if(btn_down) {
            btn_down = 0;
            _delay_ms(1000);
            if(!(PIND & (1 << BTN_DOWN))) {
                seuil_selection = 1;// Sélectionner seuil bas
            } else {
                if(type_selection == 0) {
                    if(seuil_selection == 0 && temp_sup > temp_inf) temp_sup--;
                    else if(seuil_selection == 1 && temp_inf > 0) temp_inf--;
                } else {
                    if(seuil_selection == 0 && hum_sup >= hum_inf + 10) hum_sup -= 10;
                    else if(seuil_selection == 1 && hum_inf >= 10) hum_inf -= 10;
                }
            }
        }
```

```
        // Affichage LCD des seuils
        cls_LCD();
        pos_xy(1, 1);
        if(seuil_selection == 0) Write_LCD("Alarme Seuil_SUP");
        else Write_LCD("Alarme Seuil_INF");

        pos_xy(1, 2);
        if(type_selection == 0) {
            if(seuil_selection == 0)
                sprintf(buffer, "TEMP_SUP = %dC", temp_sup);
            else
                sprintf(buffer, "TEMP_INF = %dC", temp_inf);
        } else {
            if(seuil_selection == 0)
                sprintf(buffer, "HUMI_SUP = %d%%", hum_sup);
            else
                sprintf(buffer, "HUMI_INF = %d%%", hum_inf);
        }
        Write_LCD(buffer);

        _delay_ms(200);
    }
}
```

Le principe est simple si on est en mode configuration des seuils, nous entrons dans une dite boucle principale. A l'intérieur de boucle, nous vérifions quel seuil nous voulons configurer(inférieur ou supérieur). En fonction du seuil choisi, nous entrons dans une autre boucle ou nous décidons si nous voulons configurer l'humidité ou la température. Cela nous amène à une troisième boucle où nous ajustons la température ou l'humidité en augmentant ou diminuant leur valeur, tout en détectant un changement de seuil ou une validation.

4. La fonction alarme () :

```
// === Gestion de l'alarme ===
void alarme(void)
{
    uint8_t compteur_temp = 0;

    // Vérification si la température est hors seuil pendant 3 mesures consécutives
    for (uint8_t i = 0; i < 3; i++) {
        lire_dht11(&temp_actuelle, &hum_actuelle);
        if (temp_actuelle > temp_sup || temp_actuelle < temp_inf)
            compteur_temp++;
        _delay_ms(200);
    }

    // Si température anormale confirmée
    if (compteur_temp >= 3) {
        PORTK &= ~(1 << led_verte);
        while (1) {
            lire_dht11(&temp_actuelle, &hum_actuelle);
            if (temp_actuelle <= temp_sup && temp_actuelle >= temp_inf)
                break;

            cls_LCD();
            pos_xy(1, 1);
            Write_LCD("!!!ALARME!!!");
            pos_xy(1, 2);
            sprintf(buffer, "TEMP = %d C", temp_actuelle);
            Write_LCD(buffer);
            PORTC ^= (1 << led_rouge);
            _delay_ms(300);
        }
        PORTC &= ~(1 << led_rouge);
        PORTK |= (1 << led_verte);
    }

    // Vérification humidité
    uint8_t compteur_hum = 0;
    for (uint8_t i = 0; i < 3; i++) {
        lire_dht11(&temp_actuelle, &hum_actuelle);
        if (hum_actuelle > hum_sup || hum_actuelle < hum_inf)
            compteur_hum++;
        _delay_ms(200);
    }


     if (compteur_hum >= 3) {
         PORTK &= ~(1 << led_verte);
         while (1) {
             lire_dht11(&temp_actuelle, &hum_actuelle);
             if (hum_actuelle <= hum_sup && hum_actuelle >= hum_inf)
                 break;

             cls_LCD();
             pos_xy(1, 1);
             Write_LCD("!!!ALARME!!!");
             pos_xy(1, 2);
             sprintf(buffer, "HUMI = %d %%", hum_actuelle);
             Write_LCD(buffer);
             PORTH ^= (1 << led_jaune);
             _delay_ms(300);
         }
         PORTH &= ~(1 << led_jaune);
         PORTK |= (1 << led_verte);
     }
}
```

Cette fonction surveille en permanence les valeurs de température et d'humidité. Dès qu'un des seuils est atteint, fait entrer le système en mode alarme et fait clignoter la led concerner :

VI. Diagramme d'état du système :

Cette partie résume en diagramme les différents états du système :

1. **Etat 1 mode normal :**
   C'est le mode par défaut dans ce mode, on mesure les valeurs de température et d'humidité, puis les affiche sur l'écran dans ce monde la LED verte est allumée.

2. **Etat 2 mode de configuration d'alarme :**
   Ce mode permet de configurer les valeurs de seuil d'alerte supérieur et inferieur de température et d'humidité. Pour y entrer, il faut appuyer simultanément sur les boutons **'up'** et **'down'**. Ensuite, un appui d'au moins une seconde sur l'une de ces boutons active soit en mode de configuration de seuil supérieur(up) ou en mode configuration de seuil inferieur (down) une fois dans l'un de ces modes un appuie sur le bouton temp_hum permet soit de configurer le seuil de température , soit le seuil d'humidité, le bouton **'ok'** enregistre les modifications et s'il reste appuyer pendant au moins une seconde on quitte le mode de configuration d'alarme et on retourne au mode normal. Dans le mode de configuration d'alarme, la LED la verte clignote.

3. **Etat 3 mode alarme :**
   On entre dans ce mode dès qu'un des seuils est atteint et on y reste tant qu'on ne quitte pas le seuil en fonction du seuil atteint (température ou humidité) les leds jaune (humidité) ou rouge (température) clignote.

   Tout ce ceci est représenter sur le diagramme suivant :

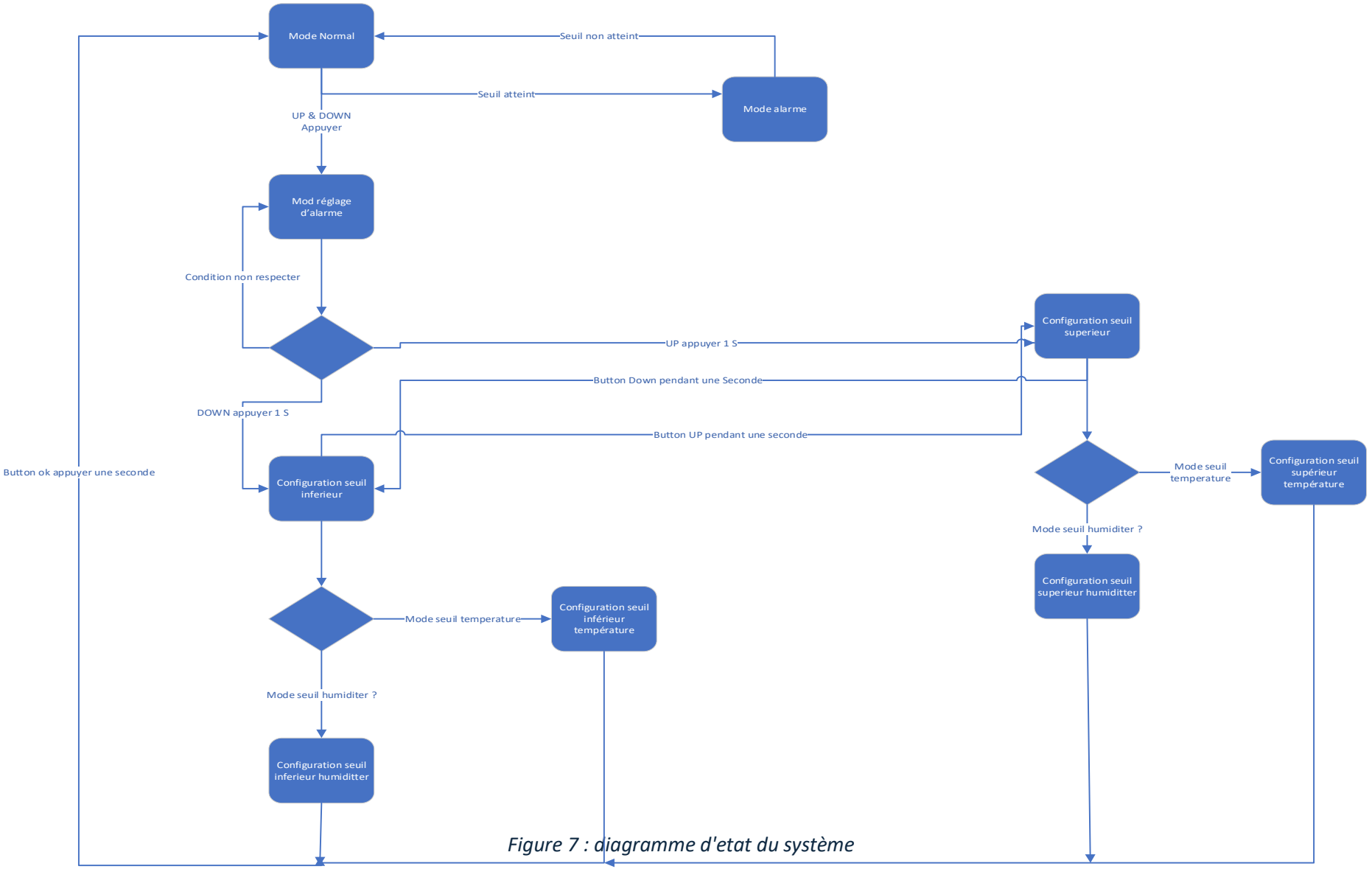


Figure 7 : diagramme d'etat du système

VII. Conclusion :

En conclusion ce projet nous as permis de nous familiariser avec le monde des microcontrôleurs. Nous avons pu simuler un montage réel avec l'atmega 2560 et des composant reconnus tels que DHT11 ou le LCD, nous avons écrit du code C compréhensible et exécutable par le microcontrôleur, et avons observé le résultat en simulant sous Proteus.

VIII. Références :

Site web de microchip : www.microchip.com

Datasheet dht11: DHT11-Data-Sheet

LCD wiki : wikipedia.org