Compte Rendu de cours et TP Systèmes Embarqués

EPSI Toulouse classe M1

TEISSIER Alexia
pour le 21/03/22

Introduction

Les systèmes embarqués sont des systèmes informatiques et électroniques qui ont comme particularité d'être autonomes, spécialisés, peu encombrant et pouvant fonctionner en temps réel.
Les objets munis de systèmes informatisés, ou équipements informatisés sont contraints dans leur puissance de calcul et leur mémoire. On doit donc leur fournir des fonctions spécifiques et délimitées. Ils ont une consommation électrique faible, et peuvent souvent fonctionner avec une simple batterie.

Ces équipements informatisés sont composés d'un processeur (un circuit intégré avec des broches de connexion) et d'un espace de stockage.
Le programme principal du système embarqué est stocké dans la mémoire et exécuté par le processeur.
Il collecte les données issues de capteurs et les transmet en signal numérique au processeur qui traite les données.
Une fois les données traitées, un autre signal numérique transmet les nouvelles données à un convertisseur qui les envoie vers un ou plusieurs actionneurs.

On peut retrouver des systèmes embarqués utilisant des algorithmes de contrôle dans des usines de production permettant de mesurer (exemple de capteur : thermomètre) et reporter le franchissement de seuils de tolérances (exemple d'actionneur : lampe, alarme), comme une surchauffe.
Enfin les systèmes embarqués utilisant le temps réel peuvent être retrouvés dans l'automobile sur des fonctionnalités critiques comme le système de freinage d'urgence.

Les systèmes embarqués sont donc utilisés dans une multitude de secteurs d'application, mais sont largement représentés dans l'automobile, le spatial, le ferroviaire, l'aéronautique, la défense ou encore la domotique.

Un objet connecté est un objet équipé d'un système embarqué et qui peut se connecter à un réseau et ainsi accéder à des données. Ils possèdent une adresse IP et grâce à des systèmes d'identification, on peut retrouver des appareils et les commander à distance.
L'IOT (Internet of things) est l'ensemble des objets connectés.

Les prédictions de nombreuses entreprises du marché permettent de dépeindre la trajectoire fructueuse de l'IOT.
Selon le Business Insider, "le chiffre d'affaires du marché de l'IOT est estimé à 212 milliards de dollars dans le monde en 2022".On apprend également avec cette statistique de l'entreprise Ericsson que "le nombre de connexions IoT cellulaires devrait atteindre 4,1 milliards en 2024, avec un taux de croissance annuel de 27 %." sources en Annexe.

Sommaire

Introduction **2**

Sommaire **3**

I - Préparation **4**

I.1 - Matériel 4

I.1.1 - Arduino Uno 5

I.1.2 - Utilisation de la documentation 6

I.2 - Langages et logiciel 7

I.2.1 - Le langage arduino 7

I.2.1.1 - Structure du langage 7

I.2.1.2 - Méthodes 7

PinMode(broche, mode) 7

DigitalWrite(broche, valeur) 8

DigitalRead(broche) 8

I.2.2 - La syntaxe du langage C (+annexes) 8

1.2.2.1 - Ecrire un nombre 8

1.2.2.2 - Ecrire un opérateur de manipulation des bits 8

1.2.2.3 - Le typage des variables 9

1.2.2.4 - Opérateur d'adressage 9

I.2.3 - VSCodium et PlatformIO 10

II - Objectifs des TP **10**

III - Réalisation **10**

1 - Allumer et éteindre une diode avec les fonctions du langage Arduino 10

2 - Allumer et éteindre une diode sans le langage Arduino 11

3 - Utilisation du clock prescaler 14

3.1 - Quelles sont les possibilités de division de la fréquence? 15

3.2 - Comment fonctionne l'ensemble de registre pour contrôler le fonctionnement du timer? 16

3.3 - Définir la valeur de comparaison 17

3.4 - Dans quel registre configurer le prescaler? 17

3.5 - Comment définir l'interrupt? 19

4 - Utiliser un capteur d'humidité DHT22 22

Annexes **24**

1 - Utilisation du binaire 24

2 - Conversion en hexadécimal 25

3 - Décalage de bits 27

4 - Manipulation de bits 28

4.1 - Mise à 1 du bit n 28

4.2 - Mise à 0 du bit n 28

5 - Liens 29

I - Préparation

I.1 - Matériel

Les systèmes embarqués peuvent être réalisés avec différents matériels comme des Raspberry Pi et des Arduino. Comparons ces produits.

Différence	Raspberry Pi	Arduino
Matérielle	basé sur un Microprocesseur.	basé sur un microcontrôleur.
Programmation	un logiciel qu'on installe sur l'OS	utilise un code compilé, nécessite un ordinateur pour téléverser.
Type de programmation	Linux embarqué : programmation s'appuyant sur un système Linux	Bare metal : Programmation sans os
Fonctionnalités	tâches diverses et complexes	tâches simples et spécifiques
Consommation	alimentation secteur nécessaire (~3A)	batterie (~100mA)
Connectivité	Ethernet, Wifi et/ou bluetooth nativement *-> Interaction haute*	USB, utilisation de pin pour faire un bus, port série (liaison bidirectionnelle) *-> Interaction faible*

Téléverser signifie transférer les données, dans le contexte, de l'ordinateur vers la carte.

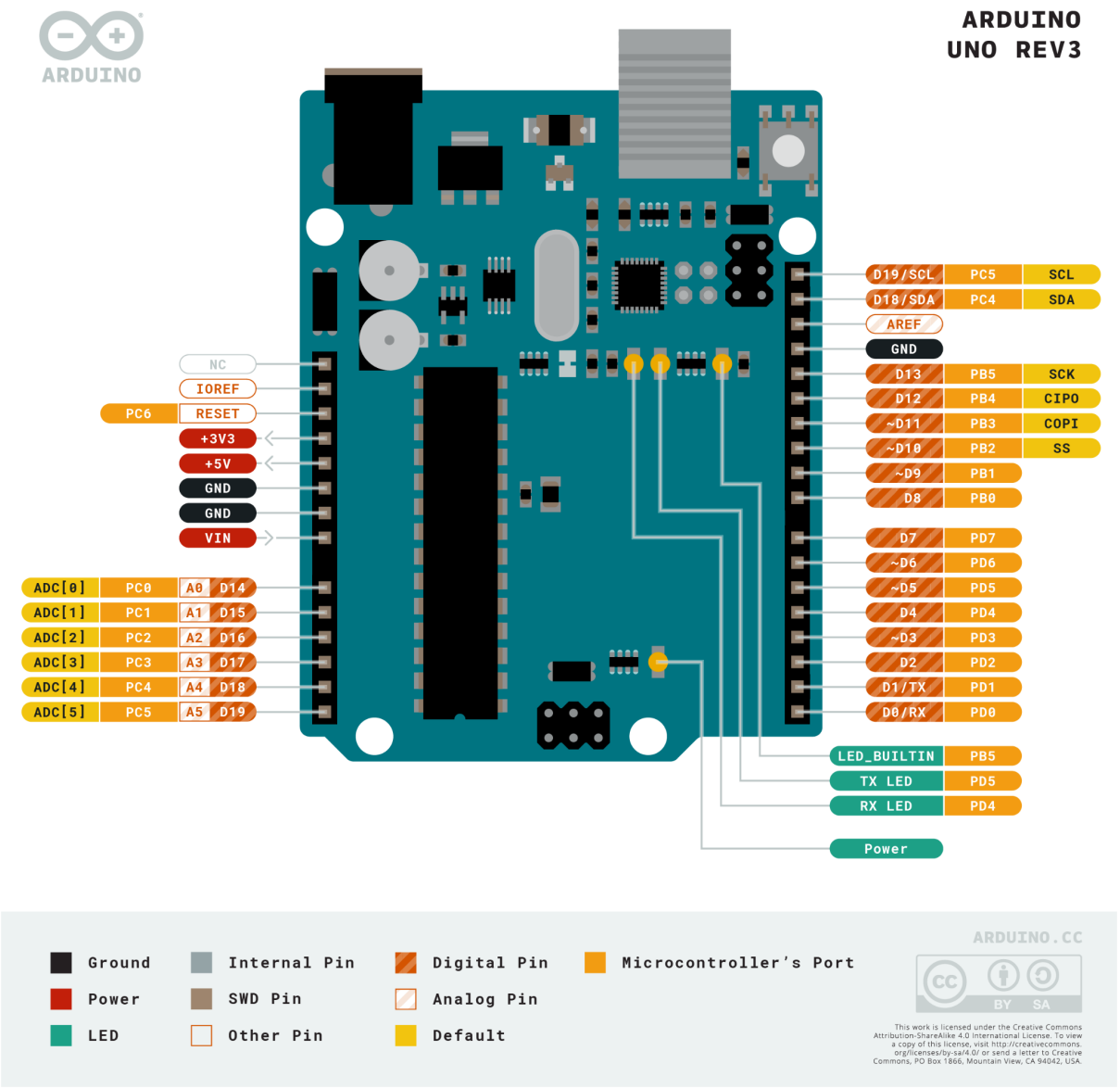
Comparons maintenant les caractéristiques matérielles de deux produits : Raspberry Pie 3 Modèle B+ et Arduino Uno.

	Raspberry Pi	Arduino
Architecture	ARM Cortex A53 (ARMv8) 64-bit	AVR 8-bit
Fréquence	1,4GHz	16MHz
Flash	SD	32Ko
RAM	1Go	2Ko

Arduino répond donc bien à la définition de système embarqué classique tandis que le raspberry pi peut être utilisé pour des tâches d'IOT plus complexes. Par exemple, on utilisera le raspberry pi pour créer un système de domotique et relier des capteurs IOT ou encore de créer un hub multimédia.

I.1.1 - Arduino Uno

Arduino Uno est un carte possédant les caractéristiques matérielles suivantes :



Il se compose de
La carte Arduino Uno est une carte à microcontrôleur basée sur l'ATmega328. Elle dispose des éléments suivants :

1. Un microcontrôleur,
2. d'une connexion USB,
3. d'un connecteur d'alimentation jack,
4. LED de visualisation (dont la LED_BUILTIN de pin 13, que l'on manipule dans les TP, ainsi que la Power qui permet d'informer du fonctionnement)
5. de 14 broches numériques d'entrées/sorties et de 6 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques),
6. d'un quartz 16Mhz,

7. d'un connecteur ICSP (programmation "in-circuit"),
8. un bouton de réinitialisation (reset en haut à droite de la figure).

I.1.2 - Utilisation de la documentation

Afin d'ajouter des fonctionnalités à cette carte Arduino Uno, il est nécessaire d'utiliser la documentation du microcontroleur qui la compose : l'ATmega328P. Vous trouverez cette documentation via le lien du constructeur microship ATMEL en annexe.

Il est nécessaire d'identifier les éléments suivants pour son projet pour obtenir les bonnes informations dans la documentation.

- définir les pins que nous allons utiliser
- définir les modes pour ces pins
- écrire l'algorithme pour recevoir la donnée de capteurs
- écrire l'algorithme pour réaliser un calcul
- écrire l'algorithme pour lancer une tâche
- identifier les éventuels modes qui puissent nous intéresser, et définir les paramètres de ces modes (comme le counter, et l'interupt pour le clock scaler par exemple)

Il est à savoir que dans la documentation, on retrouve la nomenclature suivante pour configurer l'entrée et et sortie :

- DDRxn (pour Data Direction Register x) : L'état de la broche ;
- PORTxn (pour Port x Data Register) : Le toggle de la broche ;
- PINxn (pour Port Input Pin x): La PIN à configurer.

Dans la documentation, on retrouve :

- le schéma des pins : *chapitre 1. Pin Configurations*

Le schéma représente l'ensemble des pins avec leur nom mais aussi les modes qui peuvent être utilisés. Par exemple, la pin 17 correspondant au PB5 (SCK/PCINT5). La valeur SCK correspond à Master Clock output, Slave Clock input pin.
Un autre schéma plus facile de lecture est disponible directement sur le site d'achat d'arduino Uno, ici la version rev 3, dans le paragraphe "Pinout Diagram" en annexe.

- la description des pins : *chapitre 1.1 Pin Descriptions*

Dans ce chapitre sont passées en revue les fonctionnalités qui peuvent être utilisées en fonction des pins. C'est dans ce chapitre que l'on se redirige vers les chapitres abordant les modes. On apprend par exemple que l'on peut accéder aux fonctions de l'horloge système depuis les bits du PORTB.

- les registres : *chapitre 30. Register Summary*

Ils permettent de savoir comment modifier la valeur de la broche correspondant à la pin désirée, et ce pour chaque port. Le registre comprend bien 8 colonnes de 0bits à 7bits correspondant bien à l'architecture de l'arduino qui est sur 8 bits.
Par exemple, on retrouve le PCINT0 sur le bit0 du registre PCMSK0. Il nous redirige vers la documentation du registre sélectionné, par exemple ici la page 57. On apprend que

PCMSK0 signifie Pin Change Mask Register 0, que la broche peut être configurée en écriture ou lecture, que la valeur initiale est 0, etc.

Exercice
A l'aide du tableau d'adressage de la documentation Atmel328P p.280, lien en annexes, retrouvez le registre et les bits modifiés par le code :

instruction en C	registre / bits affectés	explications
((uint8_t *)0x25) |= 0xA0	PORTB PORTB7, PORTB5	0xA0 => 10100000 je garde que les 1 car je fais un |
((uint8_t *)0x23) = 0	PORTB PINB7 à PINB0	je garde toutes les pins du PORTB
((uint8_t *)0x24) &= 0xF2	DDRB DDB0, DDB2, DDB3	0xF2 => 11110010 je garde que là où j'ai 0 pcq c'est un &

I.2 - Langages et logiciel

I.2.1 - Le langage arduino

I.2.1.1 - Structure du langage

Le langage Arduino est basé sur les langages C/C++. Il comporte les éléments de programmation suivants :

- des déclarations et imports;
- une méthode de préparation, setup(), exécuté une fois au lancement du programme;
- une méthode permettant d'opérer les transformations de la données et le lancement de tâches, loop(), exécuté à l'infini.

Il est possible d'utiliser des variables et constantes prédéfinies. On retrouve les définitions de niveaux de broche HIGH et LOW, qui permettent de lire ou écrire sur une broche numérique. On peut également utiliser les définitions des constantes de broches numériques INPUT et OUTPUT.

I.2.1.2 - Méthodes

La carte Arduino Uno possède également des fonctions facilitant l'utilisation du matériel.

PinMode(broche, mode)

Cette fonction permet de configurer la broche (en donnant son numéro) pour qu'elle se comporte soit comme une entrée (INPUT) soit comme une sortie (OUTPUT).

Par exemple : pinMode(13, OUTPUT) pour configurer la LED_BUILTIN comme une sortie.

DigitalWrite(broche, valeur)

Cette fonction permet d'écrire un niveau logique HIGH ou LOW sur une broche numérique.
Par exemple : digitalWrite(13, HIGH) pour mettre la valeur 5V sur la broche 13.

DigitalRead(broche)

Cette fonction permet de lire l'état ou niveau logique sur une broche numérique. Il renvoie HIGH ou LOW
Par exemple : digitalRead(13) pour mettre la valeur sur la broche 13.

I.2.2 - La syntaxe du langage C (+annexes)

Il est possible de réaliser l'ensemble des tâches des fonctions arduino directement en langage C.

Vous trouverez en annexe plusieurs guides pour utiliser le binaire, opérer des conversions en hexadécimal, comprendre ce qu'est un décalage de bits.

1.2.2.1 - Ecrire un nombre

Afin de spécifier le typage des données en C il est nécessaire d'ajouter un suffixe aux valeurs décimales ou hexadécimales de cette façon :

	préfixe langage C	exemple
binaire	0b	0x4B
hexadécimal	0x	0b101101

1.2.2.2 - Ecrire un opérateur de manipulation des bits

Afin d'implémenter les algorithmes nécessaires à la programmation du système, nous utiliserons les opérateurs sous la syntaxe suivante :

opérateur	opérateur en C	exemple d'opération	
ET	&	0 & 1 = 0	1 & 1 = 1
OU	|	0 | 1 = 1	1 | 1 = 1

OU exclusif	^	0 ^ 1 = 1	1 ^ 1 = 0
NON	~	~ 1 = 0	~ 0 = 1

1.2.2.3 - Le typage des variables

Les variables doivent être typées lors de leur définition en C.

variable	syntaxe C
un entier 8bit signé	int8_t
un entier 16bit non signé	uint16_t

Pour que la variable ne soit pas modifiée, il est utile d'indiquer de ne pas optimiser une variable : volatile.
Par exemple : volatile uint8_t var

1.2.2.4 - Opérateur d'adressage

<table>
<tr><th></th><th></th><th>Définition succincte</th><th>Exemple d'utilisation</th></tr>
<tr><td>adresse de variable</td><td>&var</td><td>“Représente l'adresse de la variable var (l'adresse du premier octet de la zone mémoire qu'elle occupe).”
issu de iutenligne voir annexe</td><td rowspan="2">int i = 3;
int *p;
p = &i;

On se trouve dans la configuration
<table><tr><td>objet</td><td>adresse</td><td>valeur</td></tr><tr><td>i</td><td>4831836000</td><td>3</td></tr><tr><td>p</td><td>4831836004</td><td>4831836000</td></tr></table></td></tr>
<tr><td>valeur stockée à l'adresse</td><td>*var</td><td>“Il sera parfois nécessaire de connaître l'adresse d'une variable dans la mémoire, en particulier pour transmettre l'adresse d'une variable à une fonction.”
issu de inria voir annexe</td></tr>
</table>

Exemple de comment déclarer un pointeur sur entier 8-bit non signé : uint8_t *

I.2.3 - VSCodium et PlatformIO

VSCodium est un logiciel de programmation en licence gratuite issu d'un fork de Visual Studio Code. On se servira de ce logiciel pour coder le programme en C à téléverser vers la carte Atmel328P.
Nous utiliserons PlatformIO sur VSCodium afin de créer un nouveau projet C, compiler et exécuter le code avec le mode débogage. Lors de l'installation de cette extension, vous pourrez accéder à la fenêtre d'exécution de PlatformIO en cliquant sur l'icône de PlatformIO (une tête de fourmi) puis sur "start debugging" dans la section "Debug" dans l'encart "Quick Access".

II - Objectifs des TP

L'objectif général du TP est la réalisation d'une architecture générale logicielle d'un système embarqué : système bare métal et système linux embarqué.

Il s'agit également de concevoir une solution embarquée en temps réel en utilisant le système de Timer/Count interne d'une carte ATM328P.

Vous trouverez le lien du repository Github ouvert en public ci-dessous ainsi qu'en annexe :

https://github.com/AlexiaTdev/tpEmbarkedSystem/commits/main

III - Réalisation

Afin de réaliser ce TP, nous utiliserons une vm avec VSCodium et PlatformIO.

1 - Allumer et éteindre une diode avec les fonctions du langage Arduino

Dans ce premier TP, nous utiliserons les fonctions du langage Arduino pour faire briller une diode de la carte. Les méthodes, vues précédemment, sont pinMode() et digitalWrite().
La diode sur la PIN13 est intégrée à la carte Atmel328P.

```arduino
#include <Arduino.h>
#define LED 13

void setup() {
  // put your setup code here, t
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
  Serial.println("blink");
}

void led_setup() {
  //utilisation du code arduino
  digitalWrite(LED, HIGH);
}

void loop() {
  led_setup();
  delay(1000);
}
```

2 - Allumer et éteindre une diode sans le langage Arduino

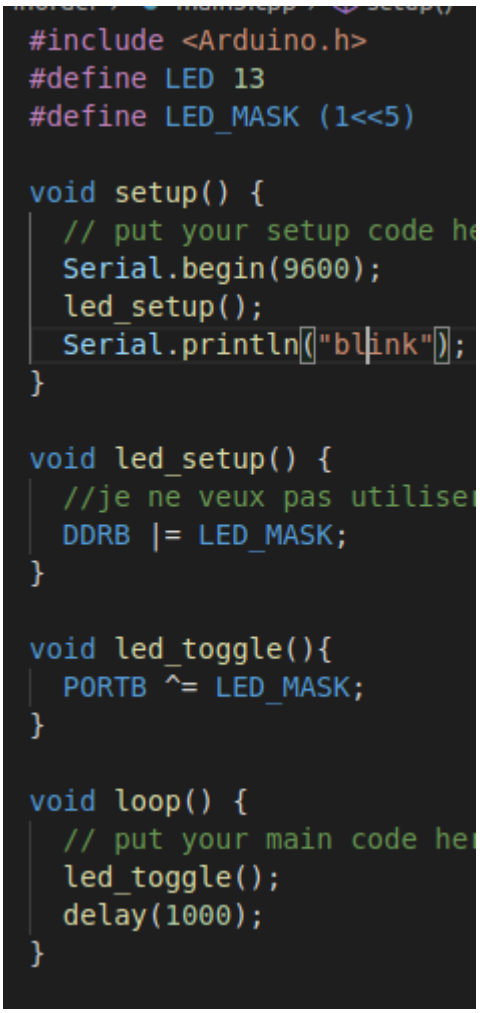
Ensuite nous allons faire clignoter la diode 13 avec un délai de 1000ms.
Pour cela nous utiliserons le décalage de 5 bit vers la gauche du bit 1, soit (1<<5).
En cherchant dans le registre, on retrouve bien que le PORTB5 est sur le bit 5.

0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	73
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	73
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	73
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	73
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	72
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	72
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	72

En appliquant ce masque avec un “ou” ou un “et”, on peut appliquer la valeur 0 ou 1 au bit 5 uniquement, et donc modifier la valeur “HIGH” ou “LOW”.

```ino
#include <Arduino.h>
#define LED 13
#define LED_MASK (1<<5)

void setup() {
  // put your setup code here
  Serial.begin(9600);
  led_setup();
  Serial.println("blink");
}

void led_setup() {
  //je ne veux pas utiliser l
  led_on();
}

 void led_on(){
  PORTB |= LED_MASK;
}
void led_off(){
  PORTB &= ~LED_MASK;
}

void loop() {
  // put your main code here,
  led_on();
  delay(1000);
  led_off();
}
```

Enfin nous pouvons réécrire la même commande avec une méthode led_toggle() permettant de switcher entre les deux modes, qui étaient pris en charge par led_on() et led_off().



```arduino
#include <Arduino.h>
#define LED 13
#define LED_MASK (1<<5)

void setup() {
  // put your setup code he
  Serial.begin(9600);
  led_setup();
  Serial.println("blink");
}

void led_setup() {
  //je ne veux pas utilise
  DDRB |= LED_MASK;
}

void led_toggle(){
  PORTB ^= LED_MASK;
}

void loop() {
  // put your main code her
  led_toggle();
  delay(1000);
}
```

On utilise le | directement sur le DDRB, puisque :
0 | 1 = 1 1 | 1 = 1
On utilise le OU exclusif directement sur le PORTB, puisque :
0 ^ 1 = 1 1 ^ 1 = 0
Ainsi la valeur du bit 5 pour la led 13 alternera entre 0 et 1 selon sa valeur initiale.

3 - Utilisation du clock prescaler

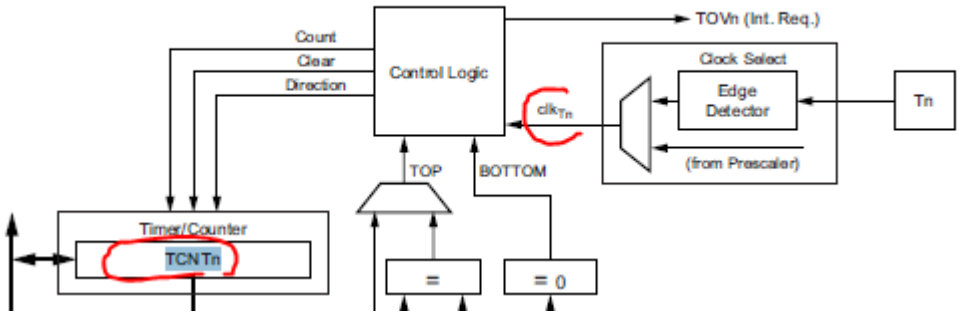
```
#include <Arduino.h>
#define LED 13
#define LED_MASK (1<<5)

void setup() {
  // put your setup code here, to
  Serial.begin(9600);
  clock_setup();
  Serial.println("blink");
}

void clock_setup() {
  //clock prescaler
  TCCR1B |= (1<<0)|(1<<2);
  //mode
  TCCR1A = OCR1A;
  //interrupt
  TIMSK1 =OCIE1A;
}

void led_toggle(){
  PINB = LED_MASK;
}

void loop() {
  // put your main code here, to r
  led_toggle();
  delay(1000);
}
```

Le timer est un périphérique matériel. A chaque période il incrémente de 1 : ça s'appelle le up up counting. La vitesse du timer dépend du matériel, il utilise n, qui est une valeur dans un registre. C'est comme du temps mais c'est pas du temps

On retrouve les détails suivants page 89 des spécifications Atmel328P:

num page	nom	valeurs possibles	remarques
74	8-Bit Time/counter 0 with PWM	on peut aller jusqu'à 255 valeurs	le timer 0 est déjà utilisé par arduino
89	16-Bit Time/counter 1 with PWM	on peut aller jusqu'à 65500 valeurs à peu pres	le registre Timer/Counter, soit TCNTn contient la valeur n; clk est l'entrée de l'horloge initiale, celle qui est 0 ; on peut configurer le matériel pour définir une vitesse de comptage (voir clock select)

Figure 15-1. 16-bit Timer/Counter Block Diagram[1]



extrait spécification Atmel328P p90

15.4 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the clock select logic which is controlled by the clock select (CS12:0) bits located in the Timer/Counter control register B (TCCR1B). For details on clock sources and prescaler, see Section 16. "Timer/Counter0 and Timer/Counter1 Prescalers" on page 114.

extrait spécification Atmel328P p94

The counting sequence is determined by the setting of the waveform generation mode bits (WGM13:0) located in the Timer/Counter control registers A and B (TCCR1A and TCCR1B). There are close connections between how the counter behaves (counts) and how waveforms are generated on the output compare outputs OC1x. For more details about advanced counting sequences and waveform generation, see Section 15.9 "Modes of Operation" on page 100.

extrait spécification Atmel328P p95

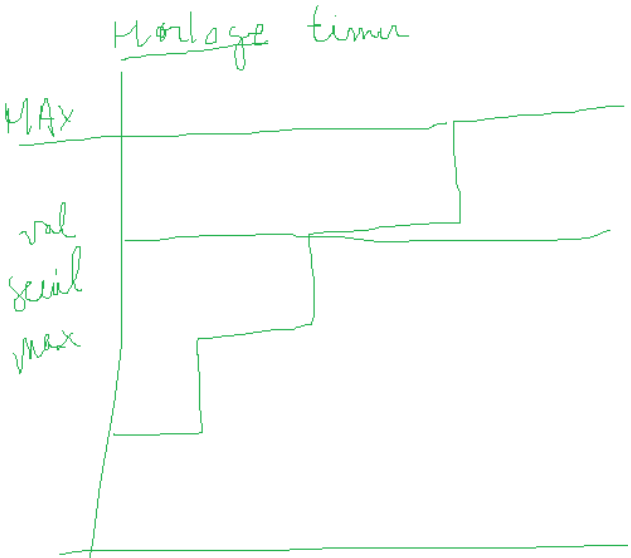
3.1 - Quelles sont les possibilités de division de la fréquence?

16.1 Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn2:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency ($f_{CLK\_I/O}$). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either $f_{CLK\_I/O}$/8, $f_{CLK\_I/O}$/64, $f_{CLK\_I/O}$/256, or $f_{CLK\_I/O}$/1024.

extrait spécification Atmel328P p112

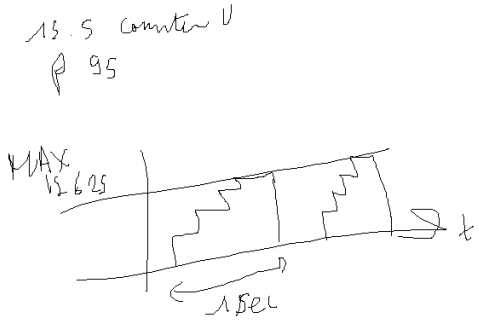
/8, /64, /256, /1024 c'est le préscaler, les valeurs sont des divisions possibles. Il compte par défaut à 16 mégahertz.

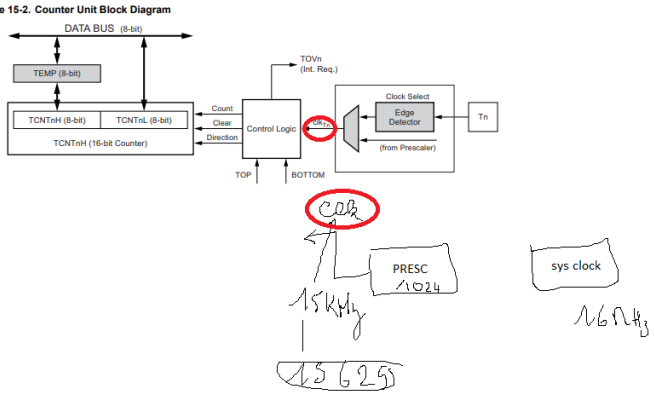


On utilisera la valeur 1024 en prescaler pour que la diode clignote tout doucement :
1/16/10^6x65535x256x4 = 4,19424
Le temps d'un pas de n sera :
1/16/10^6 = 0,000000063

En combien de temps le compteur va t'il déborder?
1/16/10^6 (car mégahertz) x65535 == 0,004095938

Le maximum étant :
1/16/10^6x65535x256 == 1,04856
Pour calculer la valeur du clk :
16x10^6 / 1024 == 15625
1/15625 == 0,000064 micro secondes





3.2 - Comment fonctionne l’ensemble de registre pour contrôler le fonctionnement du timer?

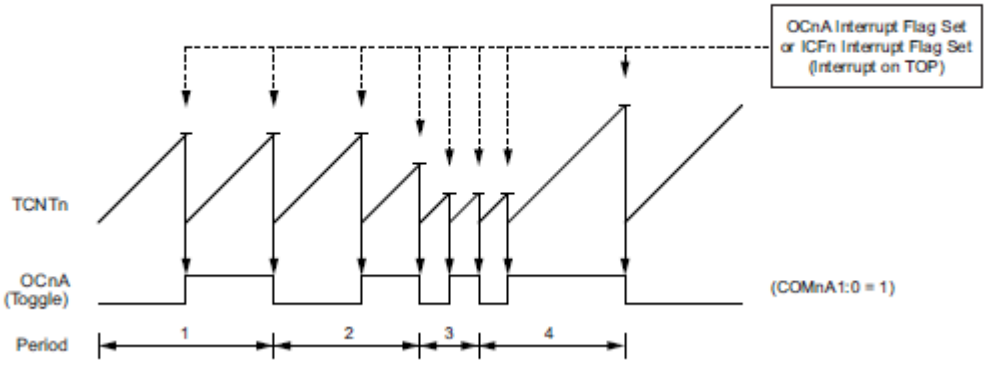
15.9.2 Clear Timer on Compare Match (CTC) Mode

In clear timer on compare or CTC mode (WGM13:0 = 4 or 12), the OCR1A or ICR1 register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT1) matches either the OCR1A (WGM13:0 = 4) or the ICR1 (WGM13:0 = 12). The OCR1A or ICR1 define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 15-6. The counter value (TCNT1) increases until a compare match occurs with either OCR1A or ICR1, and then counter (TCNT1) is cleared.

nous on veut configurer le timer en mode Clear Timer on Compare Match (CTC) avec une interruption toutes les secondes.

Figure 15-6. CTC Mode, Timing Diagram



extrait spécification Atmel328P p100

Pour la configuration du prescaler, on a besoin de définir les paramètres suivants :

- valeur de comparaison;
- mode;

- l'activation/ interruption.

Donc quand on atteint le pic, le max quoi, on atteint une interruption, donc qui active à son tour le led_toggle().

3.3 - Définir la valeur de comparaison

La valeur de comparaison, ou output compare, sera 15625. Si je l'écris en binaire j'obtiens 0B 0011 1101 0000 1001 .
Je voudrais ne garder que 0000 1001 pour le low.
Il faut ensuite le caster en uint8_t, cela permet de ne récupérer que la partie à droite.

The double buffered output compare registers (OCR1A/B) are compared with the Timer/Counter value at all time. The result of the compare can be used by the waveform generator to generate a PWM or variable frequency output on the output compare pin (OC1A/B). See Section 15.7 "Output Compare Units" on page 97. The compare match event will also set the compare match flag (OCF1A/B) which can be used to generate an output compare interrupt request.

The input capture register can capture the Timer/Counter value at a given external (edge triggered) event on either the input capture pin (ICP1) or on the analog comparator pins (see Section 22. "Analog Comparator" on page 202). The input capture unit includes a digital filtering unit (noise canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCR1A register, the ICR1 register, or by a set of fixed values. When using OCR1A as TOP value in a PWM mode, the OCR1A register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICR1 register can be used as an alternative, freeing the OCR1A to be used as PWM output.

extrait spécification Atmel328P p91

15.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

extrait spécification Atmel328P p111

Je vois que TCNTn est lié à la valeur de OCRnA. On retrouve deux types suivants de OCRnA :

- OCR1AH -> H pour haut
- OCR1AL -> L pour bas

Elle permet donc de configurer la valeur de OCRnA.
Les registres sont sur 8 bits, on retrouve donc 4 bits bits et 4 bits faibles.

3.4 - Dans quel registre configurer le prescaler?

Dans le registre TCCR1B, on veut utiliser le clock select avec le waveform generation Mode.

15.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

extrait spécification Atmel328P p110

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

extrait spécification Atmel328P p110

Si je veux utiliser le clk du prescaler, je reprends donc sa configuration soit /1024, je veux donc définir les valeurs suivantes : CS1 bit 0 à 1, CS1 bit 1 à 0, CS1 bit 2 à 1.

- **Bit 1:0 – WGM11:0: Waveform Generation Mode**

Combined with the WGM13:2 bits found in the TCCR1B register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 15-5. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), clear timer on compare match (CTC) mode, and three types of pulse width modulation (PWM) modes. See (Section 15.9 "Modes of Operation" on page 100).

Table 15-5. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP

extrait spécification Atmel328P p109

Je dois également définir les WGM1 bit 2 et bit 3 qui sont sur les bits 3 et 4 du registre TTCR1B.

Je lis que si je veux appliquer le mode 4 soit le mode CTC du Timer avec le TOP sur OCR1A, il faut définir WGM12 à la valeur 1.
Ces conditions donnent :
TCCR1B = (1<<0) | (1<<2) | (1<<3);

3.5 - Comment définir l'interrupt?

Un interrupt est une configuration permettant d'arrêter le système. On peut utiliser un flagset pour OCR1A, mais on peut aussi utiliser directement un masque sur le registre TIMSK1, le Timer/Counter1 Interrupt Mask Register.
En réalité ce système s'interrompt via un signal donné par une méthode "compare".

15.2.1 Registers

The Timer/Counter (TCNT1), output compare registers (OCR1A/B), and input capture register (ICR1) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the Section 15.3 "Accessing 16-bit Registers" on page 91. The Timer/Counter control registers (TCCR1A/B) are 8-bit registers and have no CPU access restrictions. Interrupt requests (abbreviated to Int.Req. in the figure) signals are all visible in the timer interrupt flag register (TIFR1). All interrupts are individually masked with the timer interrupt mask register (TIMSK1). TIFR1 and TIMSK1 are not shown in the figure.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T1 pin. The clock select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clkT1).

extrait spécification Atmel328P p90

15.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

extrait spécification Atmel328P p112

On choisit de configurer le OCIE1A car l'output compare était OCRnA : il correspond au bit1. Il faut donc faire la mise à 1 du bit 1 du registre TIMSK1.

- **Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the status register is set (interrupts globally enabled), the Timer/Counter1 output compare A match interrupt is enabled. The corresponding interrupt vector (see Section 11. "Interrupts" on page 49) is executed when the OCF1A flag, located in TIFR1, is set.

extrait spécification Atmel328P p112

Donc l'interruption que l'on programme correspond au "Timer/Counter1 compare match A" soit la TIMER1 COMPA, vector 12, d'adresse 0x0016.

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
		...	
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B

extrait spécification Atmel328P p49

Ainsi on retrouve pour la configuration du clock prescaler :

CONFIGURER LE PRESCALER	CONFIGURER LE	CONFIGURER

	MODE	L’INTERRUPT
il faut mettre à 0 les bits du registre TCCR1A et mettre à 1 le bit 3 du registre TCCR1B	il faut mettre à 1 le bit 0 et 2 du registre TCCR1B	il faut mettre à 1 le bit 1 du registre TIMSK1

Nous pouvons donc réécrire le code comme suit :

```arduino
#include <Arduino.h>
#define LED 13
#define LED_MASK (1<<5)
#define CS_PRESCALER_1024 ((1<<CS10) | (1<<CS12))
#define TICKS_IN_SECOND 15625
void led_setup() {
  //je ne veux pas utiliser les fonctions arduino
  DDRB |= LED_MASK;

}
void clock_setup() {
    //clock prescaler
  TCCR1A = 0; //WGM11 et WGM10 à 0
    //mode
  //TCCR1B = (1<<0) | (1<<2) | (1<<3); //WGM13 à 0, WGM12 à
  //la meme   TCCR1A = (1<<CS10) | (1<<CS12) | (1<<WGM12);
  //la meme   TCCR1A = CS_PRESCALER_1024 | (1<<WGM12);
  TCCR1B = CS_PRESCALER_1024 | (1<<WGM12);
    // valeur a comparer
  // je prends 15625 je le mets en binaire  0B 0011 1101 000
  //il faut le caster en Uint8_t ça récupère que la partie à
  //OCR1AL = (uint8_t) (TICKS_IN_SECOND & 0xFF);
  //OCR1AH = (uint8_t) (TICKS_IN_SECOND & (0xFF << 8)) >>8;
  OCR1A = TICKS_IN_SECOND;
    //interrupt
  TIMSK1 = (1 << OCIE1A);
}

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  led_setup();
  clock_setup();
  Serial.println("blink");
}


void led_toggle(){
  PINB = LED_MASK;
}

ISR(TIMER1_COMPA_vect) {
  led_toggle();
}

void loop() {
  // put your main code here, to run repeatedly:
  //led_toggle();
  delay(10000);
  TCCR1B &= ~(CS_PRESCALER_1024);
  delay(10000);
  TCCR1B |= (CS_PRESCALER_1024);
}
```

4 - Utiliser un capteur d’humidité DHT22

Le capteur de température et d’humidité fonctionne avec un bus 1-Wire. le bus contient :

- vdd : c’est le power supply ;
- data : c’est le signal que l’on souhaite récupérer ;
- gnd : c’est la terre.

Nous relieront le bus data à une PIN du DDRB. Le DDRB est un registre qui intervient dans la gestion du système.

Comment est-ce qu'on configure une entrée ou une sortie?

- x c'est le port
- n le numéro du bit

Quand on écrit dans DDRB bit 0 la valeur 1, c'est une sortie.
Quand on écrit dans DDRB bit 0 la valeur 0, c'est une entrée.

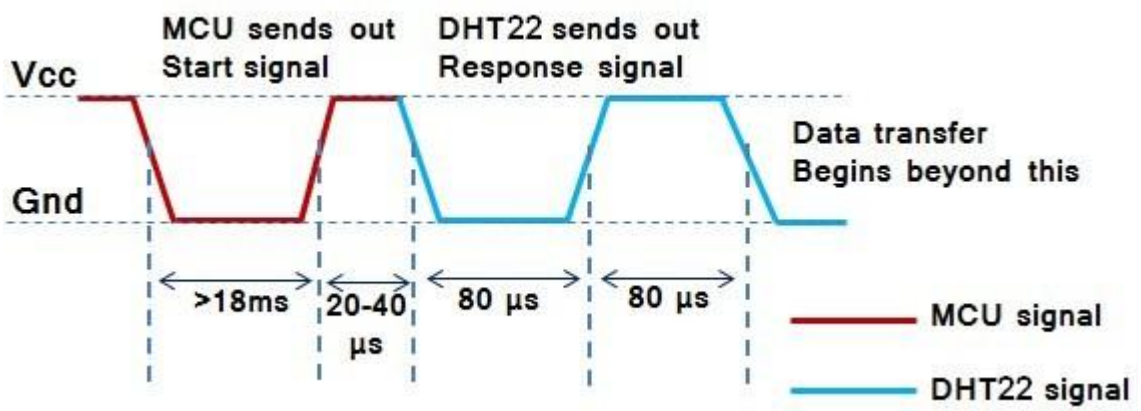
```
//dht22 sert à atmel le atm
void dht22_send_start() {
    DDRB |= (1<<DDB0);   // je set arduino  en état sortie sur le port DDRB
    PORTB &= ~(1<<DDB0); // je mets la val 0 sur le port 0 de DDRB
    delay(2000);
    PORTB |= (1 << DDB0); // je mets la val 1 sur le port 0 de DDRB
    delayMicroseconds(30);
    DDRB &= ~(1<<DDB0); //je remets l'arduino en écoute
}
```

13.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

extrait spécification Atmel328P p72

Ci-dessous le schéma représentant les échanges électriques aux bornes VCC et GND du DHT22 lors du démarrage du système.



extrait site simple-circuit voir annexes

Afin d'allumer le capteur, je dois mettre le bit 0 à 0 pendant 1 milliseconde, puis un 1 pendant 20 microsecondes pour activer le capteur DHT22.
Ensuite, j'attends deux fois 80 microsecondes pour commencer à recevoir des datas.

```
#include <Arduino.h>
//dht22 sert à alumer le atm
void dht22_send_start() {
    DDRB |= (1<<DDB0);   // je set arduino  en
    PORTB &= ~(1<<DDB0); // je mets la val 0 su
    delay(2000);
    PORTB |= (1 << DDB0); // je mets la val 1 s
    delayMicroseconds(30);
    DDRB &= ~(1<<DDB0); //je remets l'arduino e
}

void dht22_receive_ack() {
  //PINB & (1<<PINB0) en gros PINB ça va rendre
  // (1<<PINB0) on mets un 1 au niveau bit0
    while(PINB & (1<<PINB0)) {
      // Transmission des valeurs

    };
}

void setup() {
  // put your setup code here, to run once:
  dht22_send_start();
}

void loop() {
  // put your main code here, to run repeatedly
  dht22_receive_ack();
}
```

Annexes

1 - Utilisation du binaire

Les données reconnues par la machine sont des états 0 ou 1
Pour pouvoir renseigner les instructions nécessaires à la configuration d'un système embarqué, nous pouvons utiliser le binaire, langage sur une base de 2. Voici un tableau de conversion du décimal au binaire :

décimal	binaire
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Nous utilisons ici 4 bits, soit quatres "chiffres" pour écrire notre nombre binaire. Chaque bit est en fait la valeur d'une puissance de 2, voir tableau ci dessous :

exemple : convertir 0111 en décimal				
binaire en 4 bits	0	1	1	1
	0*2^3	1*2^2	1*2^1	1*2^0
	0	4	2	1
0111 (binaire) = 0 + 4 + 2 + 1 = 7 (décimal)				

On retrouvera une simplification d'écriture qui consiste à retirer les 0 à gauche du nombre binaire afin d'en simplifier la lecture :
0111 (binaire) pourra être écrit 0b111 en C.

On retrouve ainsi les valeurs binaires maximales sur 4 et 8 bits :

		plage de valeurs
Binaire en 4 bits	2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15 en plus de la valeur 0 => 16 valeurs possibles	[0;15]
Binaire en 8 bits soit 1 octet	2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7 = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255 en plus de la valeur 0 => 256 valeurs possibles	[0;255]

Remarque : il est possible d'utiliser des valeurs négatives avec l'utilisation de signes. Par exemple, un octet signé a une plage de valeurs répartie de part et d'autre de l'origine, soit [-128;127].

On peut donc reprendre cette astuce pour convertir un nombre décimal en hexadécimal, avec les variables de x1 à x8 prenant les valeurs 0 ou 1 :

exemple : convertir 45 en binaire								
	45 = x1*128 + x2*64 + x3*32 + x4*16 + x5*8 + x6*4 + x7*2 + x8*1 45 = 0*128 + 0*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*1							
binaire	0	0	1	0	1	1	0	1
	0010 1101							

2 - Conversion en hexadécimal

La conversion du binaire en hexadécimal permet la simplification de l'écriture. Il est ainsi plus facile de manipuler les nombres dans le code et ainsi faciliter sa compréhension.

décimal	binaire	hexadécimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4

5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11

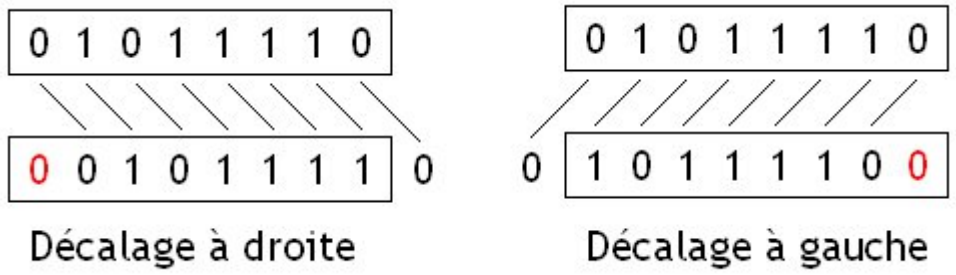
exemple : 0x4B | 0x0C = 0x4F								
	0x4B = 0b0100 1011 0x0C = 0b0000 1100							
binaire	0	1	0	0	1	1	1	1
	0b01001111 = 0x4F							
exemple : 0x67 & 0x20 = 0x20								
	0x67 = 0b0110 0111 0x20 = 0b0010 0000							
binaire	0	0	1	0	0	0	0	0
	0b00100000 = 0x20							
exemple : 0x8F ^ 0xA3 = 0x2C								
	0x8F = 0b10001111 0xA3 = 0b10100011							
binaire	0	0	1	0	1	1	0	0
	0b00101100 = 0x2C							

exemple : ~0x3D = 0xC2								
	0x3D = 0b00111101							
binaire	1	1	0	0	0	0	1	0
	0b11000010 = 0xC2							

3 - Décalage de bits

Le principe du décalage de bit est comparable à la multiplication ou la division par 10 en décimal. Elle permet de décaler la valeur des bits vers la gauche ou la droite : c'est un exemple d'opération bit à bit.

Regardons cet exemple issu du site zestedesavoir (lien en annexe) :



exemple : 0010 décalé de 1 bit à gauche				
binaire val initiale	0	0	1	1
décimal val initiale	0011 (binaire) = 1*2^0 + 1*2^1 + 0*2^2 + 0*2^3 = 1 + 2 = 3 (décimal)			
binaire val après opération	0	1	1	0
décimal val après opération	0110 (binaire) = 0*2^0 + 1*2^1 + 1*2^2 + 0*2^3 = 2 + 4 = 6 (décimal)			

Pour un nombre non nul :

- Lorsqu'on décale d'un bit vers la gauche, l'opération revient à multiplier par 2 en décimal.
- Lorsqu'on décale d'un bit vers la droite, l'opération revient à diviser par 2 en décimal
    - sauf lorsqu'il est décalé d'un nombre de bit égal ou plus grand que celui qui le compose après simplification d'écriture.
    - par exemple : 100(binaire) décalé de 3 bits vers la droite donne 0(décimal)

4 - Manipulation de bits

Si l'on veut modifier la valeur d'un bit en particulier, on peut utiliser la mise à 0 ou la mise à 1 du bit qui nous intéresse.

4.1 - Mise à 1 du bit n

La mise à 1 du bit n revient à dire "je veux remplacer la valeur au bit n par 1". Elle consiste donc à appliquer l'opération OU sur la valeur initiale, avec un "masque" qui permettra d'insérer la valeur du bit à coup sûr, en utilisant la syntaxe du décalage de bit au rang n qui nous intéresse.

Soit n le rang du bit que l'on veut modifier, i la valeur initiale et r le résultat, la mise à 1 du bit n s'écrit donc :
r = i | (1 << n)

exemple : mise à 1 du bit 2 de 0b1001				
binaire initial, i	1	0	0	1
valeur du "masque" (1<<2)	0	1	0	0
résultat i | (1 << 2)	1	1	0	1
on retrouve bien la valeur 1 au bit 2 de la valeur du résultat 0b1101				

4.2 - Mise à 0 du bit n

La mise à 0 du bit n revient à dire "je veux remplacer la valeur au bit n par 0". Elle consiste donc à appliquer l'opération ET sur la valeur initiale, avec un "masque" qui permettra d'insérer la valeur du bit à coup sûr.
On cherche maintenant à réaliser un "masque" avec la valeur 1 partout sauf au rang qui nous intéresse, on applique donc l'opération NON au décalage de bit au rang n qui nous intéresse .

Soit n le rang du bit que l'on veut modifier, i la valeur initiale et r le résultat, la mise à 0 du bit n s'écrit donc :
r = i & ~ (1 << n);

exemple : mise à 0 du bit 3 de 0b1101				
binaire initial, i	1	1	0	1

valeur de (1<<3)	*1*	*0*	*0*	*0*
valeur du masque ~(1<<3)	0	1	1	1
résultat i & ~(1 << 3)	0	1	0	1
on retrouve bien la valeur 0 au bit 3 de la valeur du résultat 0b101				

5 - Liens

Lien du github de réalisation du TP :
https://github.com/AlexiaTdev/tpEmbarkedSystem/commits/main

Spécification ATM328P
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Schéma pinOut ATM328P
https://store.arduino.cc/products/arduino-uno-rev3?queryID=undefined

Spécifications du DHT22
https://pdf1.alldatasheet.fr/datasheet-pdf/view/1132459/ETC2/DHT22.html?pseSrc=pgTutoDht22

Schéma de simplification démarrage du DHT22 de simple-circuit
https://simple-circuit.com/pic16f877a-dht22-sensor-proteus-simulation/

Statistiques sur les systèmes embarqués
https://www.ericsson.com/en/reports-and-papers/mobility-report
https://www.businessinsider.com/internet-of-things-report?r=US&IR=T

Langage C, les pointeurs - inria
https://www.rocq.inria.fr/secret/Anne.Canteaut/COURS_C/chapitre3.html