Thème n°3: Interruptions et timers/compteurs

CELECT7 : Applications des microcontrôleurs

M. Zwingelstein

rev. 2020

Travail préparatoire

Lectures de la partie préparatoire :

A. Lectures sur les interruptions

- Datasheet de l'ATmega 328p : [lien]
  - section 12 : External Interrupts (l'introduction + chercher ce qui permet de répondre au questionnaire préparatoire)
  - section 6.7 : Reset and Interrupt Handling
- Documentation arduino sur les interruptions : [lien]

B. Lectures sur les timers/compteurs

- Datasheet de l'ATmega 328p : [lien]
  - section 14 (Timer/Counter0 with PWM : introduction + chercher ce qui permet de répondre au questionnaire)
- Vous pouvez également vous aider des ressources suivantes :
  - Timers et leur mise en oeuvre : [lien]
  - Tutoriel video sur les timers de l'ATmega 328p : [lien]

Questionnaire de la partie préparatoire :

A. Questionnaire sur les interruptions

1. Questions générales :
   - qu'est-ce qu'un *vecteur d'interruption* ? (réponse consise et précise SVP)
   - dans quel espace mémoire du microcontrôleur ATmega328p trouve-t-on les vecteurs d'interruption ?
   - comment connait-on le niveau de priorité d'une interruption par rapport à une autre interruption?
   - à quoi correspond le préfixe *volatile* pour une variable, et pourquoi a-t-il un intérêt quand on écrit des routines d'interruption ?
2. Questions en rapport avec l'ATmega328p et arduino
   - quel est le nom de l'interruption externe associée à la broche n°3 de la carte UNO ?

— à quelles valeurs doit-on forcer les bits ISC10, ISC11 (du registre **EICRA**) et INT1 (du registre **EIMSK**) afin qu'un changement d'état de la broche n°3 de la carte UNO déclenche une interruption ?

B. Questionnaire sur les timers/compteurs

1. Le *drapeau de débordement* (*overflow flag bit*) du timer0 de l'ATmega328p est mis à 1 au cycle d'horloge qui suit celui où le registre de comptage **TCNT0** atteint la valeur XXX.
   — donner la valeur de XXX en décimal (justifier)
   — cette mise à 1 est-elle faite de manière logicielle (software) ou matérielle (hardware) ?
   — quel est le nom et dans quel registre se trouve ce drapeau de débordement ?
2. Il est possible de déclencher une interruption lorsque le timer0 de l'ATmega328p subit un débordement, en mettant à 1 le bit YYYYY du registre ZZZZZZ.
   — donner les noms YYYYY et ZZZZZZ
   — comment s'appelle le vecteur d'interruption associé ?
   — quel est le mode de fonctionnement du timer0 correspondant à cette situation (normal, CTC ou PWM) ?
3. Après un débordement, il est nécessaire de remettre à 0 le drapeau de débordement.
   Qui se charge de cette remise à 0, dans les deux cas suivants (hardware ou software) ?
   — timer/compteur configuré pour qu'une interruption soit déclenchée en cas de débordement
   — timer/compteur configuré pour qu'une interruption ne soit pas générée en cas de débordement
4. En mode CTC, quelle est la condition qui entraîne la remise à zéro du registre de comptage **TCNT0** ?

Cours 1 : Interruptions

Principe

Les interruptions permettent à un programme de répondre à des évênements impromptus (asynchrones), sans perturber sont fonctionnement normal :

- évênements externes :
  - changement d'état d'une broche d'entrée
  - niveau d'alimentation trop bas
  - …
- évênement interne :
  - donnée envoyée ou reçue sur un bus de communication (USART, TWI/I2C…)
  - timer dont le registre de comptage atteint sa valeur maximale)
  - …

Lors de la survenue d'un tel évênement, une interruption est déclenchée, qui conduit à la pause du programme principal et à l'exécution d'un autre petit programme appelé *routine de service d'interruption* (ISR : Interrupt Routine Service).

Etapes d'éxecution d'une interruption

1. mise en pause de l'exécution du programme principal
2. sauvegarde du contexte d'exécution du programme principal
3. traitement des actions spécifiques : routine de service d'interruption (Interrupt Service Routine - ISR)
4. restauration du contexte
5. reprise de l'exécution du programme principal qui avait été interrompu

Vecteurs d'interruption

- Un *vecteur d'interruption* est un emplacement mémoire dans lequel se trouve l'adresse de l'ISR associée à l'interruption.
- Chaque cause paussible de déclenchement d'une interruption, et donc chaque ISR, est associée à son vecteur d'interruption.
- En cas de conflit (deux évênements simultanés, c'est l'interruption la plus prioritaire qui est exécutée). L'ordre de priorité des interruptions dépend de l'adresse mémoire du vecteur d'interruption associé :
  - plus l'adresse est petite, plus l'interruption est prioritaire.

— Noter également qu'une interruption ne peut pas être interrompue par une interruption.

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
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
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
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

FIGURE 1 – Vecteurs d'interruption de l'ATmega 328p

Consignes pour l'écriture d'une ISR :

— Une ISR ne prend aucun paramètre et ne retourne rien
— Son temps d'éxecution doit être le plus court possible -> aucun calcul compliqué ni appel d'une fonction lourde
— Qualifier de `volatile` les variables globales modifiées par une ISR.
— Elle ne doit **en aucun cas** appeler une fonction qui est en attente d'une autre ISR (car une ISR n'est pas interrompue par une interruption).

Attention à l'utilisation de certaines fonctions arduino, qui utilisent des interruptions de manière non visible pour l'utilisateur (`delay()`, classe `Serial`, `tone()`, classe `Servo`...)

Variable *volatile*

On utilise des variables *globales* pour communiquer entre le programme principal et les routines d'interruption.

Ces variables doivent être qualifiées de *volatile* :

— cela indique au compilateur que la variable doit être rechargée depuis la SRAM à chaque fois qu'elle est lue dans une instruction (ne pas se contenter de prendre la valeur stockée dans un registre lors de la dernière lecture)
— nécessaire pour une variable modifiée par une ISR.

volatile

When you use the `volatile` qualifier in a variable declaration statement, this variable is loaded from the RAM instead of the storage register memory space of the board. The difference is subtle and this qualifier is used in specific cases where your code itself doesn't have the control of something else executed on the processor. One example, among others, is the use of interrupts. We'll see that a bit later.

Basically, your code runs normally, and some instructions are triggered not by this code, but by another process such as an external event. Indeed, our code doesn't know when and what **Interrupt Service Routine** (**ISR**) does, but it stops when something like that occurs, letting the CPU run ISR, then it continues. Loading the variable from the RAM prevents some possible inconsistencies of variable value.

FIGURE 2 – « C programming for arduino » - Packt publishing, UK, 2013. Page 75.

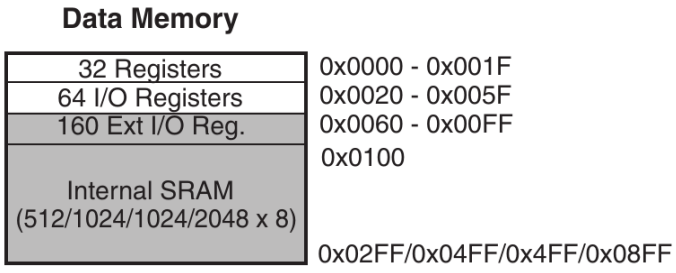


FIGURE 3 – Organisation de la mémoire des données dans un AVR.

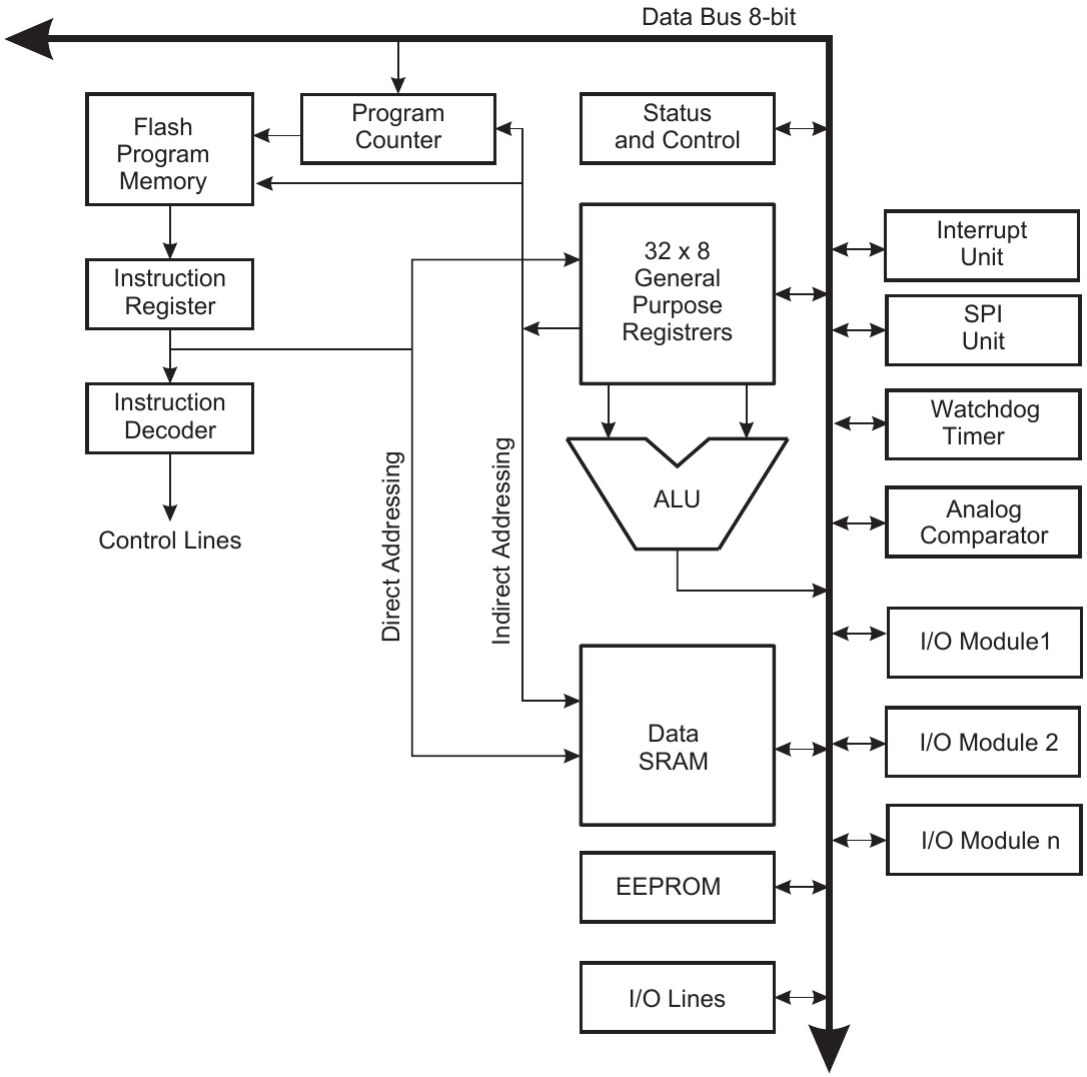


Figure 4 – Vue d'ensemble de l'architecture d'un microcontrôleur AVR.

Lab 1 : interruptions externes

Présentation

— Partie A : mise en oeuvre d'une interruption externe sur **une** broche, *avec* puis *sans* abstraction arduino
— Partie B: mise en oeuvre d'une interruption externe sur **plusieurs** broches, sans l'abstraction arduino

---

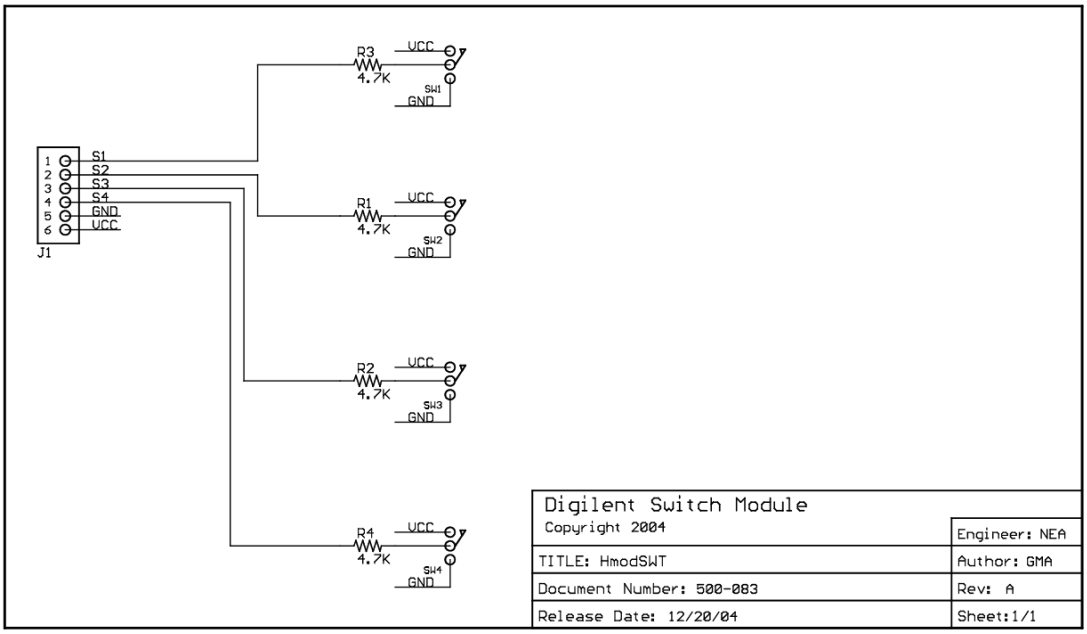
Partie A (une broche) : Mise en oeuvre de l'interruption externe sur la broche arduino n°3

Matériel utilisé

— carte arduino UNO
— platine d'essai
— module à 4 interrupteurs (« Digilent Switch Module »)
— fils de câblage
— PC avec l'IDE arduino

Travail à réaliser

1. Mise en évidence de l'intérêt des interruptions
   — Utiliser la platine d'essai attenante à la carte arduino UNO afin de câbler un des boutons du module « Digilent Switch Module » à la broche 3 de la carte UNO. Le schéma interne de ce module est le suivant :



- Dessiner le schéma électronique (pas le schéma de câblage) du circuit réalisé, faisant apparaître le symbole interrupteur, la broche 3 de la carte UNO, Vcc, GND, la résitance de 4,7 kΩ.
- Rechercher sur le schéma de la carte UNO [ici] le numéro arduino de la broche INT1 du microcontrôleur ATmega 328p.
- Compiler et téléverser le programme `seance3_a1.ino` fourni :
  - que réalise ce programme ? fonctionne-t-il bien ?
  - pourquoi la broche 3 n'a-t-elle pas été initialisée en `INPUT_PULLUP` ?
- Rajouter la ligne `delay(1000);` à la fin de la fonction `loop()` :
  - le programme fonctionne-t-il correctement ? Décrire vos observations et proposer une explication.
  - que pourrait permettre la mise en oeuvre d'une interruption externe déclenchée par un changement d'état sur la broche 3 ?

2. Mise en oeuvre d'une interruption externe sur la broche 3, en utilisant l'abstraction arduino
   - Etudier le programme `seance3_a2.ino` fourni, afin de comprendre comment l'interruption externe sur la broche 3 a été mise en oeuvre afin de modifier l'état de la LED à chaque fois que l'état de la broche 3 est modifié.
     - quel est le nom de l'ISR ?
     - que renvoie le code `digitalPinToInterrupt(SW1)` ?
     - pourquoi le code `digitalWrite(LED, SW1State);` n'a-t-il pas été écrit dans la fonction `loop()` ?
     - à quoi sert la ligne 13 dans la fonction `setup()` : `SW1State = digitalRead(SW1);` ?
   - Compiler et téléverser le programme.

   - pourquoi fonctionne-t-il à présent correctement malgré la présence de la ligne `delay(1000);` dans la fonction `loop()` ?

3. Mise en oeuvre d'une interruption externe sur la broche 3, *sans* utiliser l'abstraction arduino.
   - Objectif : réaliser la même chose qu'en 2., sans utiliser l'abstraction arduino.
   - Enregistrer le programme précédent sous le nom `seance3_a3.ino`, puis apporter les modifications suivantes :
     - la ligne 25 (« `void SW1StateUpdate(){` ») sera remplacée par « `ISR(INT1_vect){` » (INT1_vect est le nom du vecteur d'interruption associé à l'interruption INT1).
     - la ligne 11 (« `attachInterrupt(digitalPinToInterrupt(SW1), SW1StateUpdate, CHANGE);` ») doit être remplacée par une configuration directe des registres liés à la mise en oeuvre de l'interruption INT1 :
       - registre **EICRA**
       - registre **EIMSK**

       Les valeurs à écrire dans ces registres seront déterminées à partir de la datasheet (voir partie préparatoire).

       Attention à bien « entourer » les lignes de configuration des registres avec les commandes `cli();` (avant) et `sei();` (après) afin qu'aucune interruption ne puisse s'éxécuter pendant la configuration des registres (idem séance n°1).

Partie B (broches multiples) : Mise en oeuvre de l'interruption externe sur les broches arduino n°8, 9, 10, 11

Matériel utilisé :

- carte arduino UNO
- platine d'essai
- module à 4 interrupteurs (« Digilent Switch Module »)
- trois leds simples
- résistance de 150Ω
- fils de câblage
- PC avec l'IDE arduino

Cahier des charges

- Objectif : traiter la mise en oeuvre des 4 boutons du module, toujours en faisant appel aux interruptions externes

- Câblage :
  - broches 8, 9, 10, 11 de la carte UNO respectivement reliées aux boutons SW1, SW2, SW3 SW4 du module
  - broches 5, 6, 7 de la carte UNO reliées aux trois leds. Utiliser une résistance de 150 Ohms pour limiter le courant dans les leds
- Fonctionnement final attendu :
  - changement d'état de SW1 : la led n°1 change d'état
  - changement d'état de SW2 : la led n°2 change d'état
  - changement d'état de SW3 : la led n°3 change d'état
  - changement d'état de SW4 : toutes les leds sont éteintes

Analyse du problème

- Les broches arduino n° 8, 9, 10, 11 correspondent à des **broches du port B** de l'ATmega328p
- Sur l'ATmega328p, seules deux interruptions externes se déclenchent en rapport avec un événement sur une broche particulière : INT0 (broche arduino n°2) et INT1 (broche arduino n°3)
  - Il nous en faudrait 4 !
  - Pas moyen de traiter le problème à 4 boutons avec cette méthode.
- Heureusement, l'ATmega328p possède une autre interruption externe nommée **PCINT0** (voir tableau des vecteurs d'interruption) qui se déclenche lorsque **l'une des broches du port B** change d'état (n'importe laquelle).
  - il faut programmer l'ISR afin de déterminer le numéro de la broche du port B qui a déclenché l'interruption, et le tour est joué !
  - le vecteur d'interruption s'appelle `PCINT0_vect`

Travail à réaliser :

1. Consulter le schéma de la carte arduino uno afin de repérer :
   - les numéros des broches du port B de l'ATmega328p correspondant aux numéros des broches arduino sur lesquelles sont connectés les boutons SW1, SW2, SW3 et SW4
   - les numéros des broches du port D de l'ATmega328p correspondant aux numéros des broches arduino sur lesquelles sont connectés les trois LEDs
2. Consulter la datasheet afin de déterminer comment doivent être configurés les registres **PCICR** et **PCMSK0** afin de déclencher l'interruption PCINT0 lorsque l'état d'un des boutons change.
3. Ecrire et tester un premier programme - nommé `seance3_1b.ino` - qui permette de vérifier que chaque bouton déclenche bien l'interruption PCINT0 (la routine d'interruption ne fera que

changer l'état de la led interne de la carte UNO, sans chercher à déterminer quel est le bouton qui a déclenché l'interruption).

4. Enregistrer le programme précédent sous le nom `seance3_2b.ino` puis modifier le code de la routine d'interruption afin de déterminer le bouton responsable du déclenchement de l'interruption et d'obtenir le comportement final attendu (voir cahier des charges).

   Vous pouvez si vous le souhaitez décomposer le problème :

   — Etape 1 : les boutons SWi allument ou eteignent les LEDs n°i ($i \in \{1,2,3\}$)
   — Etape 2 : idem + toutes les LEDs s'éteignent lorsque le bouton SW4 change d'état
   — Etape 3 : fonctionnement final attendu

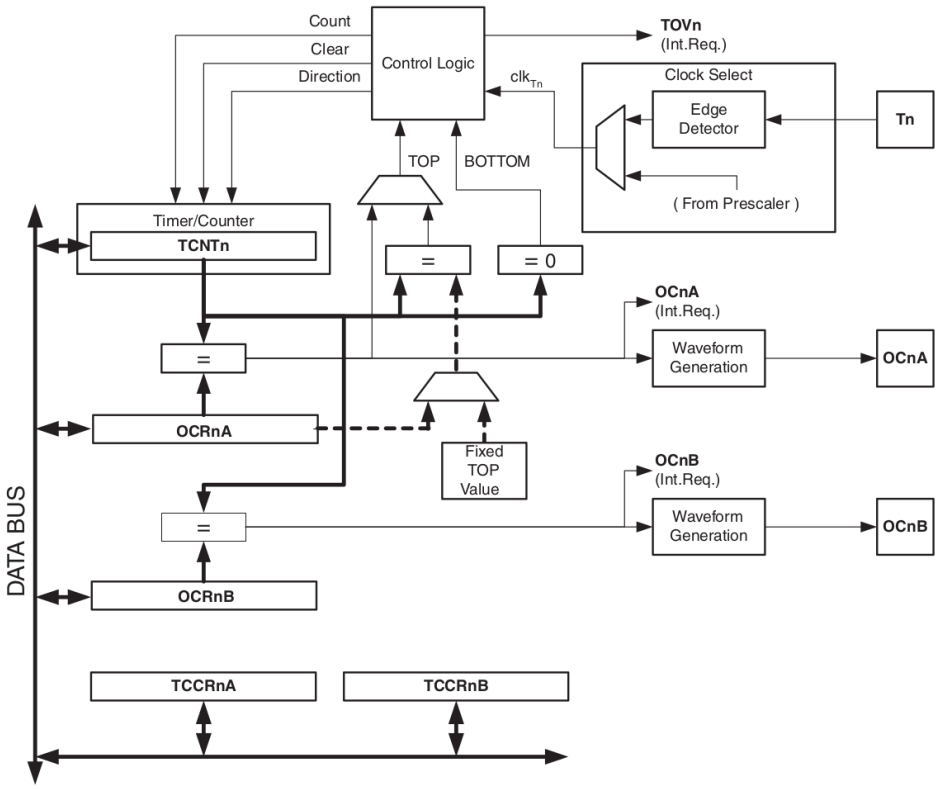
Cours 2 : Timers/Compteurs

Principe

- Un *timer* est composé d'un registre qui s'incrémente au rythme de l'horloge, associé à de la logique de contrôle et de comparaison
- Un *timer* va être utile pour gérer le temps (pendant que le timer compte, le programme peut faire autre chose)
  - exécution d'une tâche périodique
  - génération de signaux PWM
    - modulation en largeur d'impulsion - MLI
    - commande de servo moteur
  - …
- Un *timer* peut être configuré pour déclencher une interruption : exemple des vecteurs d'interruption associés au *timer/counter0*

15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow

- Schéma bloc du *timer/counter0* de l'ATmega 328p :



Débordement (overflow) :

- lorsque la valeur du timer atteint sont maximum (255 pour un timer 8 bits), il *déborde*,
  - un *débordement* entraîne la mise à 1 d'un *drapeau* (flag bit) dans un registre de contrôle,
  - un débordement peut générer une interruption
- En jouant sur le paramètre de *facteur d'échelle* (prescaler), on peut modifier la cadence des débordements

Table 15-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

FIGURE 5 – Paramétrage du facteur d'échelle par les bits CS02:0 du registre **TCCR0B**

Modes de fonctionnement

- mode normal
- mode CTC (Clear Timer on Compare match)
- modes PWM (*fast* et *phase correct*)

Figure 15-3. Output Compare Unit, Block Diagram

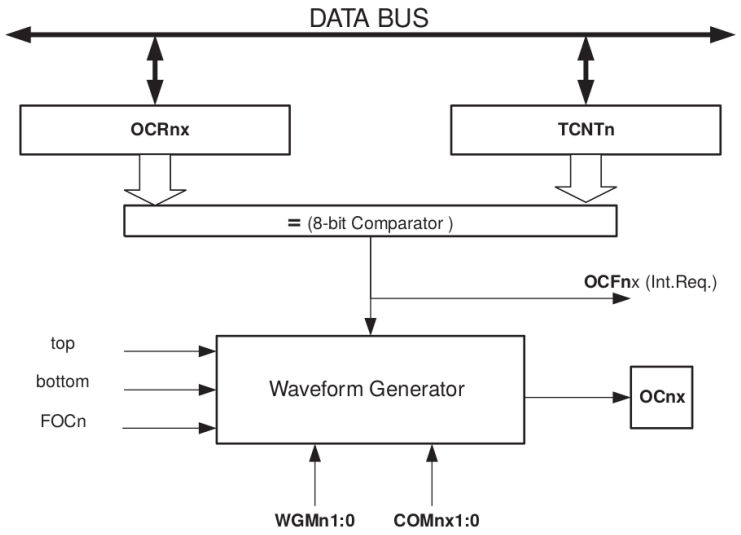


FIGURE 6 – Détail du comparateur

Registres de controle

Ils permettent de suivre et de modifier le comportement du timer0. Nous avons (en plus du registre **TCNT0** – **T**imer/**C**ou**NT**er **R**egister) :

- **TCCR0A** et **TCCR0B** – **T**imer/**C**ou**NT**er **C**ontrol **R**egister **A** (et **B**)
- **OCR0A** et **OCR0B** – **O**utput **C**ompare **R**egister **A** (et **B**)
- **TIMSK0** – **T**imer/Counter **I**nterrupt **M**a**SK** Register
- **TIFR0** – **T**imer/Counter 0 **I**nterrupt **F**lag **R**egister

Timers et abstraction arduino

ATTENTION : certaines fonctions arduino font appel aux timers de l'ATmega 328p :

- timer0 (8 bits) : `delay()`, `millis()`, `micros()`
- timer1 (16 bits) : génération de PWM (broches 9 et 10 de la carte UNO), bibliothèque *servo*
- timer2 (8 bits) : génération de PWM (broches 3 et 11 de la carte UNO), fonction `Tone()`

Lab 2

Matériel utilisé :

- carte arduino UNO
- platine d'essai
- potentiomètre 10 kΩ
- LED RVB
- résistance de 150Ω
- fils de câblage
- PC avec l'IDE arduino

Cahier des charges :

- L'objectif de cette partie est de mettre en oeuvre le timer2 de l'ATmega328p afin de :
  - réaliser la lecture d'un capteur analogique (potentiomètre) à intervalle de temps régulier (10 μs)
  - déclencher une alarme au cas où la valeur lue dépasse un certain seuil et allumer un témoin lumineux vert sinon
- Capteur analogique :
  - Le capteur analogique est « simulé » par un potentiomètre.

  Celui-ci sera câblé afin que la tension capteur lue sur la broche A0 de la carte arduino soit comprise entre 0 et 5V.
  - La lecture du capteur se fera à l'aide de la fonction `analogRead()` d'arduino.
- Témoins lumineux et alarmes :
  - Alarme basse si tension capteur < 1,5 V
  - Alarme haute si tension capteur > 3,5 V
  - LED RVB à cathode commune, utilisée comme suit :
    - led R (rouge) : pin 11 de la carte arduino uno. S'allume en cas d'alarme haute,
    - led V (verte) : pin 12 de la carte arduino uno. S'allume lorsque la valeur lue est dans l'intervalle voulu,
    - led B (bleue) : pin 13 de la carte arduino uno. S'allume en cas d'alarme basse.
- Timer :
  - Le timer2 de l'ATmega 328p sera configuré de manière à déclencher une interruption toutes les 10 μs pour lire l'état du potentiomètre
  - Il sera utilisé en mode CTC (Clear Timer on Compare Match)
  - Les registres à configurer pour la mise en oeuvre du timer sont les suivants :

  - **TCCR2A** et **TCCR2B** (voir table 17-8 *Waveform Generation Mode Bit Description* et 17-9 *Clock Select Bit Description* de la datasheet)
  - **OCR2A** (voir section 17.11.4 de la datasheet : *OCR2A - Output Compare Register A*)
  - **TIMSK2** (voir section 17.11.6 de la datasheet : *TIMSK2 - Timer/Counter2 Interrupt Mask Register*)
- La routine d'interruption `ISR(TIMER2_COMPA_vect)` réalisera :
  - la lecture du capteur analogique (fonction `analogRead()`)
  - la mise à jour de la led RVB en fonction de la valeur lue du capteur (fonction `digitalWrite()`)

Travail à réaliser

1. Câbler le potentiomètre sur la broche arduino n° A0 et la led RVB
   - dessiner le schéma électrique du circuit réalisé
   - s'assurer de savoir allumer les trois couleurs indépendamment avec la fonction 'digitalWrite
   - apprendre à réaliser la lecture de la tension analogique délivrée par le potentiomètre avec la fonciton `analogRead()`. Vous pouvez vous inspirer de l'exemple *Basics/03.Analog*
     - déterminer les valeurs retournées par `analogRead()` correspondant aux tensions déclenchant les alarmes basse et haute
2. Préparer sur papier la configuration des registres :
   - registre **TCCR2A**
     - bits WGM21 et WGM20 (voir table 17-8)
   - registre **TCCR2B**
     - bit WGM22 (voir table 17-8)
     - bits CS22:0 (voir table 17-9)
   - registre **TIMSK2**
     - bit OCIE2A (voir section 17.11.6)
   - registre **OCR2A** (voir section 17.11.4)

   Expliquer de manière concise et précise le rôle de chacune de ces configurations.
3. Ecrire le code global répondant au cahier des charges (partir du cannevas du fichier fourni `seance3_2.ino`) :
   - Bien noter que la fonction `loop()` sera vide, mise à part la ligne pour afficher sur le port série la valeur lue.
     En effet, tout se fait dans la configuration des registres associés au timer2 ainsi que dans la routine d'interruption `ISR(TIMER2_COMPA_vect)`.
   - Ne pas oublier que les variables globales modifiées dans la routine d'interruption doivent être qualifiées de **`volatile`**

4. (Bonus) Modifier le programme de façon à faire la lecture du potentiomètre toutes les 500 ms (fichier `seance3_2Bonus.ino`).

   Bon amusement !
