WATCHDOG TIMER

Réalisé par Bruno PIQUEMAL

24 septembre 2020

Ce guide vous permettra de comprendre les trois types de fonctionnements du WATCHDOG Timer sur l'ATMEGA 328P.

Introduction

Un Watchdog est logiciel ou circuit électronique utilisé pour s'assurer qu'un automate ou un ordinateur ne reste pas bloqué à une étape particulière du traitement qu'il effectue. C'est une protection qui peut redémarrer le système, ou lui faire faire autre chose, si une action définie n'est pas exécutée dans un délai imparti par le programmateur.

Le Watchodog Timer (WDT) est un timer inclus dans la puce ATMEGA 328P. Il peut être utilisé de trois manières différentes qui sont :

- Reset mode
- Interruption and Reset mode
- Interruption mode

Table des matières

⋆ ⋆ ⋆

A Matériel

Pour ce tutoriel nous allons avoir besoin d'un Arduino UNO, d'une LED et d'une résistance de polarisation de 10K Ohms.

B Foncionnement

Le WDT est un compteur de cycles qui se fait sur un oscillateur de 128 kHz dans l'ATMEGA 328P. Il émet un signal d'interruption, ou il redémarre l'Arduino, dès que le temps présélectionné s'est écoulé.

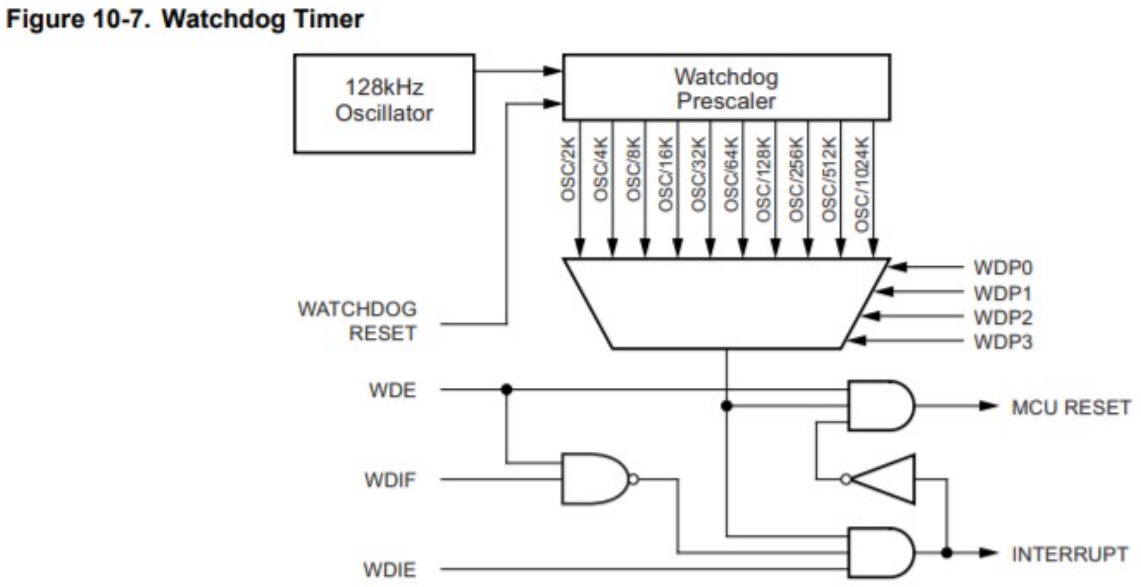

FIGURE 1 – Schéma de fonctionnement du Watchdog Timer

Ci-dessous la table des différentes valeurs de temporisation pour l'utilisation du WDT.

Table 10-3. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16ms
0	0	0	1	4K (4096) cycles	32ms
0	0	1	0	8K (8192) cycles	64ms
0	0	1	1	16K (16384) cycles	0.125s
0	1	0	0	32K (32768) cycles	0.25s
0	1	0	1	64K (65536) cycles	0.5s
0	1	1	0	128K (131072) cycles	1.0s
0	1	1	1	256K (262144) cycles	2.0s
1	0	0	0	512K (524288) cycles	4.0s
1	0	0	1	1024K (1048576) cycles	8.0s
1	0	1	0	Reserved	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

FIGURE 2 – Choix des prescalers du Watchdog Timer

B.1 Reset mode

Codage

```
#include <avr/wdt.h>
void setup() {
  wdt_disable();
  Serial.begin(9600);
  Serial.println("Setup started :");
  // make a delay before enable WDT
  // this delay help to complete all initial tasks
  delay(2000);
  wdt_enable(WDTO_4S);
}
void loop() {
  Serial.println("LOOP started ! ");
  for (int i = 0; i <= 5; i++) {
    Serial.print("Loop : ");
    Serial.print(i);
    Serial.println();
    delay(1000);
    wdt_reset();
  }
  // infinity loop to hang MCU
  while (1) {
    Serial.println("I am in while loop");
  }
}
```

Ce programme montre l'effet du WDT dans une boucle infinie. Dès que le temps atteint les 4 secondes demandées, malgré le fait que le programme soit dans la boucle « while », il redémarre.

Il est important de mettre la fonction wdt_disable() en première ligne du void setup() en cas de problèmes survenus lors du redémarrage.

Sans le wdt_reset() qui se trouve dans la boucle « for », le programme redémarre avant même d'avoir fini d'afficher les 5 valeurs de la variable i.

Avec le wdt_reset(), la boucle « while » ne s'exécute que pendant 4 secondes.

B.2 Interruption and Reset mode

Codage

```
#include <avr/wdt.h>
#define LED_PIN 4

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
  Serial.println("I am awake");

  wdt_enable(WDTO_8S);

  //Disable ADC - don't forget to flip back after waking up
  // if using ADC in your application ADCSRA |= (1 << 7);
  ADCSRA &= ~(1 << 7);

  //ENABLE SLEEP - this enables the sleep mode
  SMCR |= (1 << 2); //power down mode
  SMCR |= 1;//enable sleep
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(1000);
  Serial.println("Going to sleep");
  Serial.flush();
  digitalWrite(LED_PIN, LOW);


  //BOD DISABLE - this must be called right before the __asm__ sleep instruction
  MCUCR |= (3 << 5); //set both BODS and BODSE at the same time
  MCUCR = (MCUCR & ~(1 << 5)) | (1 << 6); //then set the BODS bit and
      clear the BODSE bit at the same time
  __asm__ __volatile__("sleep");//in line assembler to go to sleep
}

ISR(WDT_vect) {
  //DON'T FORGET THIS! Needed for the watch dog timer.
  //This is called after a watch dog timer timeout - this is the interrupt function
      called after waking up
}// watchdog interrupt
```

Le but de l'exemple est d'allumer une LED suite à l'endormissement de l'Arduino.

La fonction ISR(WDT_vect), qui signifie Interrupt Service Routine (ISR), permet de gérer les interruptions liées à plusieurs paramètres dont le WDT.

Ici, la LED reste allumée pendant 1 seconde avant que l'Arduino se mette en mode deep sleep et 7 secondes après il redémarre.

Serial.flush() est utilisé afin d'afficher ce que le Serial.prinln() envoie dans le moniteur série avant que l'Arduino s'endorme.

B.3 Interrupt Mode

Ce programme reprend les grandes lignes de la section précédente si ce n'est que dans notre cas, il n'y a pas de redémarrage. Ainsi, il n'y a pas de perte de temps machine dédié au réveil et donc des économies d'énergie sont faites. Ce type de programme peut être utile pour des applications de très basse consommation.

Ce code n'utilise pas la librairie <avr/wdt.h>, contrairement au précédent, mais plutôt il modifie les registres manuellement. En agissant de la sorte, on gagne de la place dans la mémoire.

La mise en place du WDTCSR se fait en 3 étapes car il nous est demandé de le faire de cette manière là dans la datasheet de l'ATMEGA 328P. Si on essaye de modifier ce paramètre en une seule fois, on ne pourrait pas obtenir le résultat souhaité qui est d'allumer la LED pendant 1 seconde puis de la rallumer 7 secondes après que l'Arduino se soit endormi.

Codage

```
#define LED_PIN 4

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
  Serial.println("I am awake");

  //SETUP WATCHDOG TIMER
  WDTCSR = (24);//change enable and WDE - also resets
  WDTCSR = (33);//prescalers only - get rid of the WDE and WDCE bit
  WDTCSR = (1<<6);

  //Disable ADC - don't forget to flip back after waking up
  //if using ADC in your application ADCSRA |= (1 << 7);
  ADCSRA &= ~(1 << 7);

  //ENABLE SLEEP - this enables the sleep mode
  SMCR |= (1 << 2); //power down mode
  SMCR |= 1;//enable sleep
}

void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(1000);
  Serial.println("Going to sleep");
  Serial.flush();
  digitalWrite(LED_PIN, LOW);

  //BOD DISABLE - this must be called right before the __asm__ sleep instruction
  MCUCR |= (3 << 5); //set both BODS and BODSE at the same time
  MCUCR = (MCUCR & ~(1 << 5)) | (1 << 6); //then set the BODS bit and
      clear the BODSE bit at the same time
  __asm__ __volatile__("sleep");//in line assembler to go to sleep
}

ISR(WDT_vect) {
  //DON'T FORGET THIS! Needed for the watch dog timer.
  //This is called after a watch dog timer timeout - this is the interrupt function
      called after waking up
}// watchdog interrupt
```

C Pour aller plus loin

Le WDT permet d'interrompre un processus ou de redémarrer le microcontrôleur suite à un problème d'état qui mettrait trop de temps à répondre. Cependant, le cycle de fonctionnement du WDT est imprécis et sa valeur maximale n'est que de 8 secondes, ce qui peut s'avérer peu pour beaucoup d'applications. C'est pourquoi on va préférer l'utilisation d'une horloge RTC qui contrôlera de manière très précise l'heure et les cadences des actions souhaitées.

Voici un tutoriel pour de la très basse consommation et du réveil par interruption :

https://www.youtube.com/watch?v=urLSDi7SD8M

D Bibliographie

http://gammon.com.au/interrupts

https://blog.frogslayer.com/creating-an-arduino-watchdog-timer/

https://create.arduino.cc/projecthub/rafitc/what-is-watchdog-timer-fffe
toc-example-code-4