Cours 3
Les temporisations et temporisateurs.

Nous avons très souvent besoin d'une temporisation précise en informatique, plusieurs solutions peuvent être mise en œuvre, des solutions logicielles ou alors des solutions matérielles utilisant un circuit annexe au microprocesseur appelé Timer.

1. Création d'une temporisation logiciel

Un microcontrôleur exécute des instructions assembleurs de façon séquentielle. Chaque instruction assembleur est exécutée en un nombre de cycle machine fourni par le constructeur du circuit. La durée du cycle machine est fonction de l'horloge du microcontrôleur. Pour réaliser une temporisation logicielle, il suffit d'exécuter une série d'instructions correspondant à un certain nombre de cycle machine.

Si on écrit le programme en C, il faudra regarder le résultat de la compilation pour mesurer le temps d'exécution en analysant le code assembleur.

1.1. Instructions d'assembleur AVR

Exemple de temporisation écrite en assembleur :

On charge un registre (R18) 8bits avec une valeur et on effectue l'opération R18=R18-1 jusqu'à avoir R18=0

;Boucle d'attente de quelques dixième de seconde	
ldi r18, 250 ;Charge le registre R18	1cycle
Attente:	
dec r18 ;Décrément de 1 de le registre r18	1cycle
nop ;Attente d'un cycle d'horloge (125 ns)	1cycle
brne Attente ;Boucle sur Attente jusqu'à l'obtention d'un zéro dans r18	1cycle 2cycles si vraie

http://www.atmel.com/webdoc/AVRLibcReferenceManual/inline_asm_1gcc_asm.html

Durée de la temporisation :

La carte Arduino Uno a un oscillateur à la fréquence de 16Mhz

3cycles x période de l'horloge x 250 + 3cycles x période de l'horloge

Période de l'horloge = $\frac{1}{16Mhz}$ = 6,25 . 10^{-8}s Durée de la temporisation = 47 µs

Si l'horloge est précise, la temporisation sera précise. L'inconvénient de ce type de tempo est que le microprocesseur est exclusivement utilisé pour mesurer le temps.

Liste des instructions assembleur du microcontrôleur :

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0...6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3...0)←Rd(7...4),Rd(7...4)←Rd(3...0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1

2. Utilisation d'un TIMER matériel

Le microcontrôleur Atmel 328 possède 3 timers internes,

- 2 timers 8bits (ils sont prépositionnables et peuvent être utilisés en mode comparateur)
- 1 timer 16bits (il est prépositionnable, peut être utilisé en mode comparateur et utilisé en mode capture).

L'événement de comptage peut être un front de l'horloge du microcontrôleur, ce qui revient à mesurer l'écoulement du temps. L'événement de comptage peut aussi être un front sur une broche d'entrée du microcontrôleur (les broches T0 et T1 peuvent servir d'entrée de comptage).

Fonction Temporisateur : lorsque l'on compte des fronts de l'horloge qui cadence le microcontrôleur, on mesure le temps. Les modules Timers/Counters permettent de compter les fronts du signal d'horloge ou un signal de fréquence plus faible obtenu par un diviseur appelé prescaler.

Fonction Compteur : lorsque l'on compte des fronts sur une entrée de comptage (broches T0 ou T1), on utilise alors la fonction "compteur" du module.

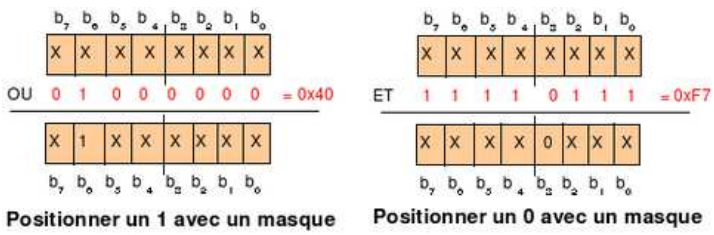
Le choix entre fonction de temporisation (avec prédiviseur de fréquence ou non) et fonction de comptage se fait par paramétrage de registres dédiés à la gestion des modules Timers/Counters.
Génération de signaux périodiques : les modules Timers/Counters sont assez complexes et chacun de ces modules peut générer deux signaux PWM dont le rapport cyclique est facilement modifiable.

Remarque : ces périphériques intégrés sont assez complexes (environ 70 pages du datasheet ATMega). Seule une vision simplifiée est fournie ici.

2.1. Utilisation des masques en C

On appelle **masque** une valeur définie qui va servir à l'aide d'un opérateur **ou** ou un opérateur **et** de positionner des bits sur un mot binaire.

Les opérations logiques effectuées sont des opérations bit à bit :



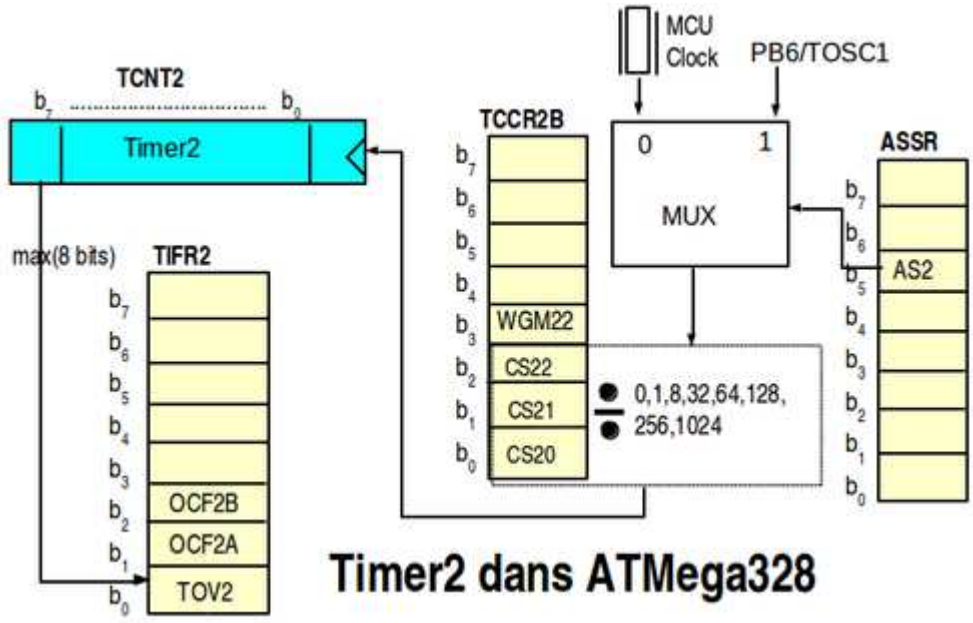
Autres solutions pour positionner un bit dans un registre :

Mise à 1 du 6éme bit du registre TWAR	**Mise à 0 du 6éme bit du registre TWAR**
TWAR |=(1<<6); Si le nom est déclaré au préalable #define TWA5 6 TWAR |=(1<<TWA5);	TWAR &= ~(1<<6); Si le nom est déclaré au préalable #define TWA5 6 TWAR &= ~(1<<TWA5);

2.2. Exemple : Timer/Counter 2 (comptage 8 bits)

C'est un module Timer/Counter avec registre de comptage 8 bits.

La structure générale du module Timer/Counter 2 est donnée dans le datasheet. Le registre de comptage est TCNT2 (registre 8 bits).



Le bit b0 du registre **TIFR2** appelé **TOV2** est mis à un automatiquement par le matériel. Ce ne sera pas le matériel qui le mettra à 0. C'est à vous le faire ! Mais pour le faire il vous faut écrire un 1 dans ce bit b0 !

Les bits de configuration de la division se trouvent dans le registre **TCCR2B**

Le registre **ASSR** sert à choisir la source de l'horloge du timer 2. Pour nous, ce sera toujours l'horloge du microcontrôleur.

Points importants de ce TIMER2:

- détection sur un débordement
- entrée de comptage interne = signal d'horloge interne avec une prédivision ou non
- possibilité de comparer TCNT2 à deux registres de comparaison OCR2A/OCR2B

- l'égalité TCTN2=OCR2A peut déclencher une IT
- l'égalité TCTN2=OCR2B peut déclencher une IT également
- Les broches OC2A(PB3) et OC2B (PD3) peuvent être activées par le Timer/Counter2 pour génération de signaux périodiques (PWM).

Exemple utilisant une scrutation du flag de débordement du Timer:

```
// ARDUINO UNO - Timer 2 avec scrutation
//LED SUR LA Patte 2 ARDUINO PORTD.2

volatile unsigned char cpt=0;   // compteur d'IT
// Fonction de traitement = Timer 2 OverFlow
void TIMER2_OVF(){
 cpt++;
 if(cpt==61){
   //PORTD ^=(1<<PORTD2);
   PORTD = PORTD ^ B100;
   cpt=0;
   }
TIFR2&= ~(1<<TIFR2);    //force le flag TOV2 à  0
}
void setup(){
          // Configuration Timer 2
 TCCR2A=0; // Mode Normal (pas PWM)
 TCCR2B=B111; // Prescaler 1024 (Clock/1024)
//Configuration PORTD.2
 DDRD |= B100; // PD2 en sortie
 PORTD = PORTD & B11111011;
 TIFR2 &= ~(1<<TIFR2);   //force le flag TOV2 à  0
}
void loop() {
if ((TIFR2 & 0x01) == 0x01) TIMER2_OVF() ;
}
```

Exemple utilisant une interruption:

L'exemple suivant illustre l'utilisation du timer 2 et de l'interruption associée à son débordement.

Une fonction d'interruption est une fonction appelée par un événement extérieur au microprocesseur, dans le programme ci-dessous se sera un débordement du TIMER2. L'interruption stoppe le programme pour s'exécuter la fonction ISR(TIMER2_OVF_vect) et à la fin de la fonction le programme reprend son exécution.

```
// ARDUINO UNO - INTERRUPTION Timer 2
//LED SUR LA Patte 2 ARDUINO PORTD.2

volatile unsigned char cpt=0;   // compteur d'IT
// Fonction de traitement IT n°10 = Timer 2 OverFlow
ISR(TIMER2_OVF_vect){
 cpt++;
 if(cpt==61){
 PORTD ^=(1<<PORTD2);
 cpt=0;
 }
}
void setup(){
          // Configuration Timer 2
 TCCR2A=0; // Mode Normal (pas PWM)
 TCCR2B=B111; // Prescaler 1024 (Clock/1024)
 TIMSK2=0x01; // IT Timer2 Over Flow Active
//Configuration PORTD.2
 DDRD |= B100; // PD2 en sortie
 PORTD &= ~(1<<PORTD2); // PORTD.2 <-0
sei(); // activation des IT (SREG.7=1)
}
void loop() { // aucun traitement
}
```

Paramètres Timer 2:

Mode 0 (Normal) : WGM2=0 WGM1=0 WGM0=0 [TCCR2A=0]

Prescaler = 1024 : CS22=1 CS21=1 CS20=1 [TCCR2B=0x07=(111)b]

Interruptions

Masque d'IT : Interruption sur Overflow = TIMSK2.0 =1

Autorisation générale des IT : SREG.7=1 [activé par sei()]

Digital I/O = Broche PORTD.2 en sortie.

Principe : après la fonction setup(), le registre TCNT2 (8bits) est incrémenté à chaque front du signal périodique clock/1024. A chaque débordement du registre TCNT2, le débordement déclenche l'interruption n°10 appelée "Timer 2 Over Flow". Tous les 60 appels de cette fonction, la broche PB5 (LED) change d'état. La LED clignote.

Quelle sera fréquence de clignotement ?

Clock = signal périodique de 16MegaHz (16 millions de ticks par seconde)

Prescaler = 1024 -> la fréquence d'incrément de TCNT2 = (16/1024) MegaHz

Fréquence de débordement : TCNT2 ne déborde que tous les 256 incréments

c'est-à-dire à la fréquence de 16/(1024*256) MegaHZ ≈ 61 Hz

- Il y a 1 interruption Timer2 Over Flow toutes les 1/61 secondes.
- Il faut 61 Interruptions pour que la LED change d'état
- La LED change d'état (0->1 1->0) à intervalles de environ 1 seconde

Attention : En utilisant l'IDE Arduino, le timer 0 est implicitement utilisé par les fonctions de delay (ainsi que l'interruption correspondante)..