http://www.didel.com/ info@didel.com

www.didel.com/coursera/AideMemoireArduinoEnergia.pdf

Aide-mémoire Arduino/AVR/MSP

Vous savez, mais vous n'êtes plus très sûr

Définitions, déclarations
utiliser des mots qui parlent
	#define Led 13 // ! pas de ;
	int Duree = 100;
	const int Led=13; // constante

variables byte nom = 35; 8 bits 16 bits	**byte** (non signé 0..255) **char** (signé –128..+127) **unsigned int** (0..65535) **int** (-32768..+32767)
void set-up () { configuration, initialisations }	pinmode (LedTop, OUTPUT); DDRC = 0b00000111; // 0 = in 1 = out

void loop () {
les instructions se terminent par un ;
}
	loop() {
	digitalWrite (LedTop,LOW);
	} ! LOW ne veut pas dire inactif !

Calcul aa=aa*2; ou aa *= 2; + - * / %(modulo) j++; ajoute 1 j--; soustrait 1 **bitwise** & (and) , | (or), ^ (xor) , ~ (not) >> , << décale	**Comparaison** == (égalité), !=(différent), < , > , <= , >= , **boolean** &&, || , ! (not) valeur =0 faux ou différent de 0 vrai

if (condition) { bloc d'instructions; **}**
si vrai on fait
On teste et on passe plus loin
	if (condition) {
	instructions;
	}

if (condition) { } else { } si vrai on fait, autrement on fait autre chose	if (condition) uneInstruction; else uneInstruction;
while (condition) { } on fait et refait tant que la condition est vraie	while (condition) { instructions; }
while (1) { bloc d'instructions; **}** on fait en boucle les instruction (comme loop)	while (1){} for(;;); même effet bloque l'activité
do { bloc d'instructions; **}** **while (condition) ;** on fait au moins une fois	do { a++ ;} // byte a=0; déclaré avant while (a < 5) ;

for (init ; condition ; modif)
{ bloc d'instructions; **}** on répète tant que la condition est vraie
	for (byte i; i<5; i++)
	{ Cligno ();
	}

byte etat;
switch (etat) {
case 0: instructions; break;
case 1: instructions; break;
default: instructions; // optionnel
}
	enum { Avance, Recule,..**} ; byte etat;**
	switch (etat) {
	case Avance: instructions;
	etat = recule; break;
	case Recule: . . . .
	}

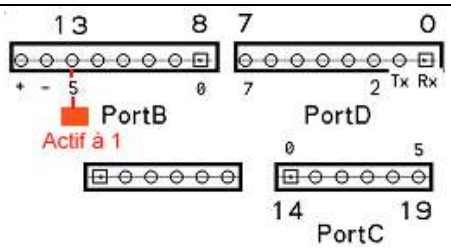
Commentaires // commentaire jusqu'à la fin de ligne	/* Commentaire sur plusieurs lignes */

Constantes: HIGH (maj) Variables: maVar (min-maj) Fonctions: FaireCeci (maj-min-maj)

Fonctions Arduino/Energia

pinMode(pin,b); digitalWrite (pin,b); digitalRead (pin);	boolean boolean boolean	INPUT=0 OUTPUT=1 LOW=0 HIGH=1
analogWrite (pin,pwm); analogRead (pin);	byte int (10 bits)	pins 5,6, 3,11, 9,10 0-100% PWM pins 14=A0 15,16,17,18, 19=A5
delay (ms); delayMicroseconds (us); millis (ms); micros (us); précision 4 us;	unsigned long unsigned int unsigned long unsigned long	1..~10^{10} millisecondes (~50 jours) 1.. 65535 us (~0.06 sec) boucle en 50 jours boucle en 70 minutes
min(a,b); max(a,b); abs(a);	all	
constrain (x,a,b);	all	
map (value,fromLow,fromHigh,toLow,toHigh) ; int		
random(max); random(min,max); *(>= min, <max)* randomSeed(value) ; int		
tone (pin,frequency); tone(pin,frequency,duration); notone(pin);		
Serial.begin (9600); Serial.print ("abc"); Serial.println (val,BIN);		

Correspondance pins Arduino – bits AVR 168/328
Les pins 0 à 7 vont sur les bits 0 à 7 du PortD. Les pins 0 et 1 ne sont pas utilisées pour ne pas interférer avec USB.
Les pins 8 à 13 vont sur les bits 0 à 5 du PortB
Les pins 14 à 19 vont sur les bits 0 à 5 du PortC; elles acceptent l'ordre analogRead (pin); (valeur 10 bits)
Les pins 3, 5, 6, 9, 10, et 11 acceptent l'ordre analogWrite (pin,pwm8bits);



Pour accéder aux pins, on a le choix entre les fonctions Arduino, ou l'accès direct aux bits des registres

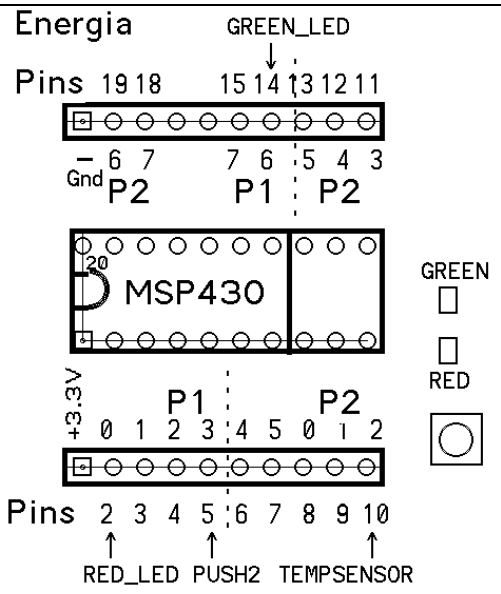
`pinMode (2,OUTPUT) ;`
`pinMode (2,INPUT) ;`
durée 3.1 µs
	`bitSet (DDRD,2);`
	`bitClear (DDRD,2);`
	durée 0.13 µs
		`DDRD = 1<< 2;`
		durée 0.06 µs
		`DDRD |= 1<< 2;`
		`DDRD &= ~(1<< 2);`
		durée 0.13 µs

`digitalWrite (2,HIGH);`
`digitalWrite (2,LOW);`
durée 3.8 /4.0 µs
	`bitSet (PORTD,2);`
	`bitClear (PORTD,2);`
	durée 0.13 µs
		`PORTD |= 1<< 2;`
		`PORTD &= !(1<< 2);`
		durée 0.13 µs

`digitalRead (2)`
durée 3.7 µs
`if (digitalRead(2)) {}`
durée 3.7 µs
	`bitRead (PIND, 2)`
	durée 0.13 µs
	`a= bitRead (PIND, 2);`
	durée 0.6 µs
	`if (bitRead (PIND,2)){}`
	durée 0.13 µs
		`PIND & (1<<2)`
		durée 0.06 µs
		`a = PIND & (1<<2);`
		durée 0.3 µs
		`if (PIND & (1<<2)){}`
		durée 0.13 µs

Correspondance pins Energia – bits MSP3420
Les pins 2 à 7 vont sur les bits 0 à 5 de P1.
Les pins 14 et 15 vont sur les bits 6 et 7 de P1
Les pins 2 et 3 sont utilisées pour les transferts série.
Les pins 19 et 18 vont sur les bits 6 et 7 de P7
si P2SEL=0; a été initialisé
Les pins 8 à 13 vont sur les bits 0 à 5 de P2
si un MSP 20 broches est inséré.
Toutes les pins de P1 acceptent l'ordre analogRead (pin); (valeur 10 bits)
Les pins 4,9,10,12,13,14 acceptent l'ordre analogWrite (pin,pwm8bits); (avec des restrictions).

Les timings ci-dessous correspondent au mode par défaut, 1MHz pour une consommation minimale.
Avec dans le setup l'instruction `BCSCTL1 = 0x0F;`
la fréquence est de 16MHz et les différences avec AVR sont nulles ou non significatives.



Pour accéder aux pins, on a le choix entre les fonctions Arduino/Energia, ou l'accès direct aux bits des registres.

`pinMode (2,OUTPUT) ;`
`pinMode (2,INPUT) ;`
durée 62µs 105µs si interrupt
	`bitSet (P1DIR,2);`
	`bitClear (P1DIR,2);`
	durée 4 µs
		`P1DIR = 1<< 2;`
		`P1DIR |= 1<< 2;`
		durée 4 µs
		`P1DIR &= ~(1<< 2);`
		durée 5 µs

`digitalWrite (2,HIGH);`
`digitalWrite (2,LOW);`
durée 62µs 105µs si interrupt
	`bitSet (P1OUT,2);`
	`bitClear (P1OUT,2);`
	durée 4 µs
		`P1OUT |= 1<< 2;`
		durée 4 µs
		`P1OUT &= ~(1<< 2);`
		durée 5 µs

`digitalRead (2) ;`
durée 31 µs
`if (digitalRead(2)) {}`
durée 31 µs
	`bitRead (P1IN, 2)`
	durée 3 µs
	`a= bitRead (P1IN,2);`
	durée 11.2 µs
	`if (bitRead (P1IN,2)){}`
	durée 3 µs
		`P1IN & (1<<2);`
		durée 3 µs
		`a = P1IN & (1<<2);`
		durée 3 µs
		`if (P1IN & (1<<2)){}`
		durée 3 µs