Arduino

Condensé - Résumé fonctions

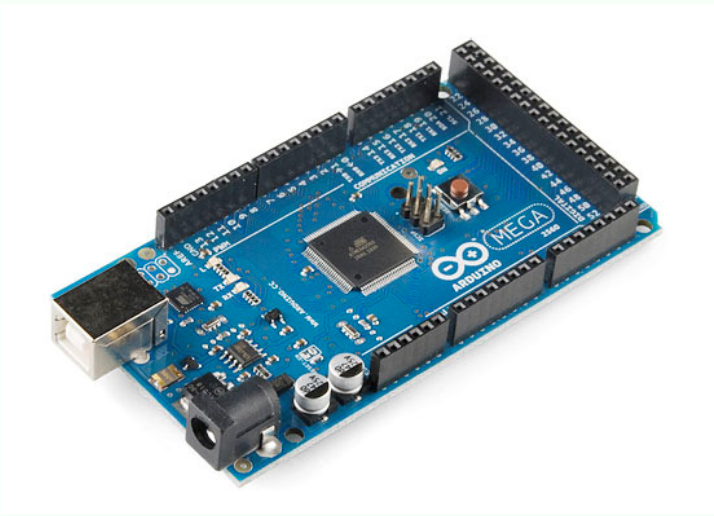


Table des matières

Cartes mères

Avant propos

Caractéristiques communes modèles standard - Electronique

Alimentation

Les cartes peuvent être alimentées soit par une source externe (7 a 18v) soit par l'interface USB. Cette dernière est protégée par un fusible réarmable de 500ma, de plus un vmos la déconnecte automatiquement a la détection de la présence de l'alimentation principale.

L'alimentation externe (Connecteur alim standard diamètre 2.1 mm) est transformée en 5v par un régulateur faible pertes (Mc332269 ou NCP1117) délivrant un maximum de 800mA (Attention a la dissipation thermique toutefois). La protection contre les inversions de polarité est assurée par une diode série.

Le 3v3 est généré a partir du 5v par un autre régulateur d'une capacité de 150ma (Lp2985 sur UNO ou MEGA).

Connecteur alimentation

Le connecteur Power 8 broches (6 dans e cas de certaines cartes IoRef et la broche de réserve n'étant pas implanté) a les fonctions suivantes :

1 : Reserve

2 : IoRef, relié au +5v
3 : Entrée ou sortie reset
4 : Sortie 3.3v
5 : Sortie 5v
6, 7 : Masse générale, 0v
8 : Vin, sortie avant régulateur et après la diode de protection, ne pas utiliser comme entrée d'alimentation a la place du connecteur jack coaxial d'alimentation.

Connecteurs E/S

Suivant les types de cartes celles-ci disposent d'un nombre de connecteurs plus ou moins importants, soit de type digital, soit de type mixte analogique / digital.

Ces entrées / sorties possèdent les caractéristiques suivantes :

- Courant d'entrée : Environ 1µA
- Valeur résistance de PullUp optionnelle : 20 a 50 KHo
- Courant de sortie : 40mA max, 200mA total µcontrôleur
- Tension d'entrée : Ne pas dépasser Vcc
- Caractéristique convertisseur AD : Par approximation successives, 10bits, temps de conversion 13 a 260µs.
- Référence de tension interne : 1.1 ou 2.56v

Leds externes

Les cartes standards sont généralement équipées de quatre leds.

- Vert : Alimentation 5v
- Jaune Tx : Commandée par l'interface USB, transmission au PC
- Jaune Rx : " " , réception du PC
- Jaune : Utilisateur, généralement connectée a la sortie digitale 13 (Pb7 sur Atm2560, Pb5/Sck sur Atm328)

Interfaces de programmation - ICSP - Convertisseur USB / série

Si les micro-processeurs Atmel se programment via leur interface SPI (Broches MOSI, MISO, SCK) les cartes Arduino utilisent le port série (Atm x28, Atm 1280/2560) ou USB (16U2) de la cpu. Le bootloader "Arduino" initialisé au reset de la carte se charge de cette fonction. Cette méthode permet outre la programmation d'obtenir des fonctions de débogage du programme utilisateur.

Dans le cas des processeurs Atm x28, 1280, 2560 c'est l'interface Usart 0 qui est utilisée, des résistances série de 1Ko protègent des conflits dans le cas d'un circuit utilisateur externe connecté.

Si sur les premières cartes Arduino le port série RS231 (Niveaux TTL 0/5v) était directement exploité un convertisseur USB/série a rapidement été intégré, au départ spécialisé avec le FT232RL de FTDI puis avec un autre processeur Atmel de type xxU2. Ces convertisseurs en outre permettent de commander la ligne Reset du processeur principal afin d'éviter une manipulation operateur lors d'une demande de téléchargement.

Afin de permettre les mises a jour ou récupérer un bootloader défaillant un connecteur ISCP est implanté permettant la programmation directe du ou des microprocesseurs (ICSP1 se chargeant de l'interface Usb/série basé sur xxU2).

Connecteur ICSP

MISO	1	o	o	2	Vcc5v
SCK	3	o	o	4	MOSI
Reset	5	o	o	6	Gnd

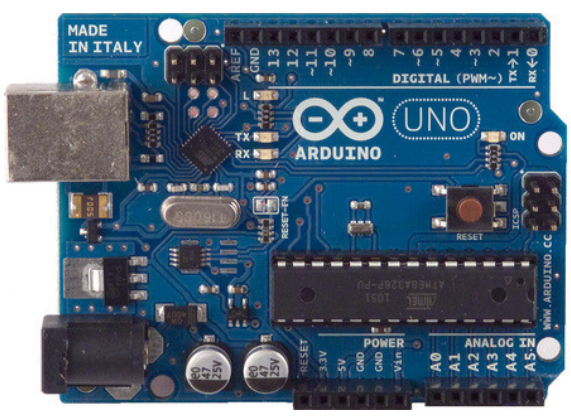
Versions cartes CPU et évolutions

Plusieurs cartes Cpu existent, de différents niveaux de puissance processeurs, de possibilités d'ES et sous facteur de forme. Outre les cartes standards acceptant les modules d'interface par empilement des cartes plus ou moins spécifiques existent pour une utilisation finale, sans connecteurs avec connexion par soudage directe des fils. L'interface USB pour l'IDE peut être aussi omise, la programmation du microprocesseur étant réalisé directement via l'entrée RS232.

Modèles cartes CPU début 2013

			Mémoire (Kio)				Ports E/S						Interface
Carte	**Processeur**	**F Mhz**	Flash	EEPROM	SRAM	Externe	Digital	Pwm	An. In	An. Out	Uart	OTG	**USB**
Arduino org	Atmega 8-16PI	16	8	0,5	1	-	14	3	8	0	1	0	Rs232 direct
							Connecteurs gigogne standard						
Diecimila	Atmega 168	16	16	0,5	1	-	14	6	6	0	1	0	FTDI
							Connecteurs gigogne standard						
Duemilanove	Atmega 168/328	16	16 / 32	0,5/1	1 / 2	-	14	6	6	0	1	0	FTDI
							Connecteurs gigogne standard						
Nano	Atmega 168/328 TQFP	16	16 / 32	0,5/1	1 / 2	-	14	6	8	0	1		FTDI FT232RL
							Format minimal, connecteurs non standard						
Pro	Atmega 168/328 TQFP	8 / 3v3 16 / 5v	16 / 32	0,5/1	1 / 2	-	14	6	6	0	1		Sans
							Sans connecteurs, sorties a souder						
LilyPad	Atmega 168/328	8	16 / 32	0,5/1	1 / 2	-	14	6	6	0	1	0	Sans
							Sans connecteurs, rond						
Mini	Atmega 368	16	32	1	2	-	14	6	8	0	1	0	Sans
							Sans connecteurs, rectangulaire format minimal						
Fio	Atmega 328	8	32	1	2	-	14	6	8		1		Sans
							Format mini, sorties a souder, alimentation 3v3						
Uno	Atmega 328	16	32	1	2	-	14	6	6		1		8U2 / 16U2
							Connecteurs gigogne standard						
Ethernet (Uno)	Atmega 328	16	32	1	2	-	14	4	6		1		8U2 / 16U2
							Uno avec interface ethernet et lecteur SD						
Leonardo	Atmega 32U4	16	32	1	2,5	-	20	7	12	0	1	1	32U4
Micro	Atmega 32U4	16	32	1	2,5	-	20	7	12	0	0	1	Sans
							Format minimal, connecteurs non standard						
LilyPad USB	Atmega 32U4	8	32	1	2,5	-	9	4	4	0	0	0	32U4
							Sans connecteurs, rond, Alimentation 3v3						
Esplora	Atmega 32U4	16	32	1	2,5	-	20	7	12		0	1	32U4
							Sans connecteurs standard, périphériques intégré Bp, Led, t°, joystick ...						
Mega	Atmega 1280	16	128	4	8	64	54	15	16	0	4	0	8U2 / 16U2
							Connecteurs gigogne standard						
Mega 2560	Atmega 2560	16	256	4	8	64	54	15	16	0	4	0	8U2 / 16U2
							Connecteurs gigogne standard						
Mega ADK	Atmega 2560	16	256	4	8	64	54	15	16	0	4	1	Max3421
							Connecteurs gigogne standard						
Due	At91Sam3X8E cortex	84	512	0	96	-	54	12	12	2	4	1	16U2
							Connecteurs gigogne standard						

Arduino UNO



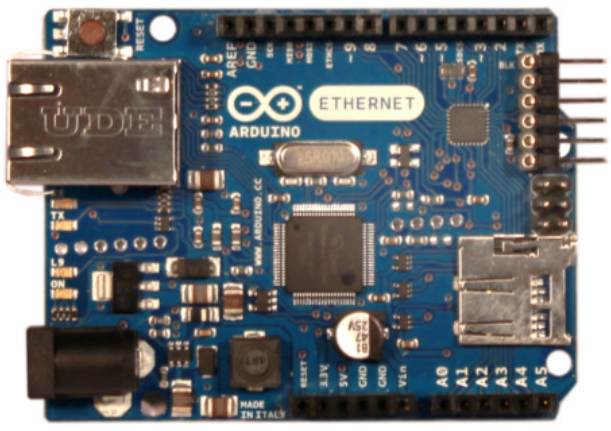
Version de base du système basé sur un Atmega 328 cette carte délivre un maximum de 14 points d'entrée / sortie digitaux et de 6 entrées analogiques. Existe avec un processeur en version Dil sur support ou SMD soudée.

Outre le processeur, la principale évolution consiste en un remplacement du chipset de l'interface de programmation USB de la puce d'interface USB / Série FTDI en une puce Atmel 8U2 ou sur les dernières versions 16U2.

Versions

- Rev1 : Chipset USB Atmega 8U2
- Rev2 : Ajout d'une résistance de pull down sur la ligne HWB du 8U2 afin de faciliter le passage en mode DFU (Mise a jour firmware 8U2)
- Rev3 : Ajout de SDA et SCL sur le connecteur JP6, de IOREF et d'une broche libre sur le connecteur Power. Modification du circuit reset et remplacement du 8U2 par un Atmega 16U2.

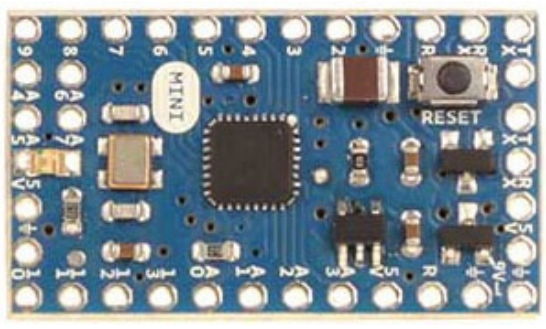
Arduino UNO Ethernet



Carte Uno 328 disposant d'une interface Ethernet intégrée, celle-ci pouvant être de type POE.

Le chipset Ethernet W5100 utilise l'interface SPI de l'Atm328 (Digital 10 a 13) ainsi que le port PD4 (Digital 4) pour la sélection du lecteur de carte micro SD.

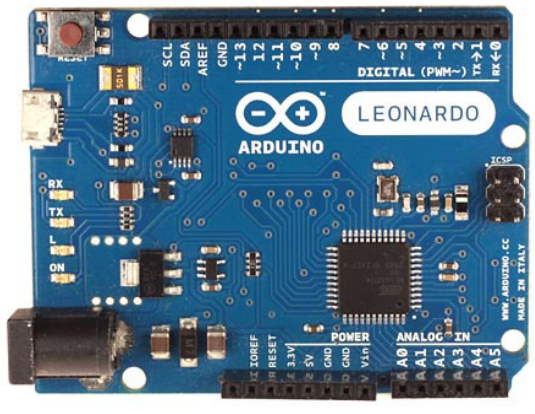
Arduino Mini



Carte Uno disposant d'un Atmega 328 au format TQFP ce qui lui permet d'obtenir 2 ports d'entrée analogique supplémentaire.

D'un facteur de forme minimal les connexions externes sont en fil a fil soudés, la programmation se fait directement en RS232 directement ou par un adaptateur externe.

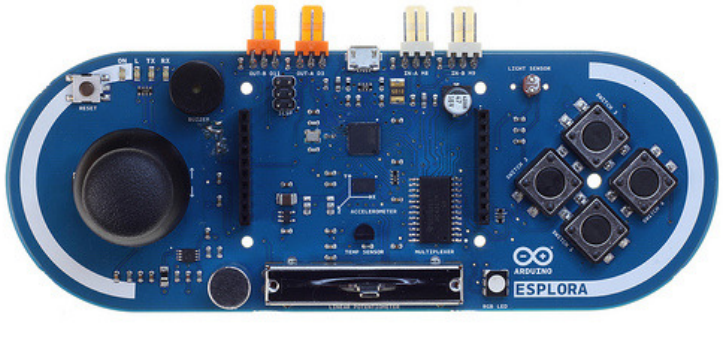
Arduino Leonardo



Modèle a peu près équivalent a l'Uno cette carte est basée sur un processeur Atmel 32U4.

Ce processeur interfaçant nativement l'Usb la gestion de la programmation est assurée directement sans passer par l'interface série. De plus le port usb permet le mode OTG pour la gestion de périphériques externes. (Connecteur au format micro Usb et non pas de type B).

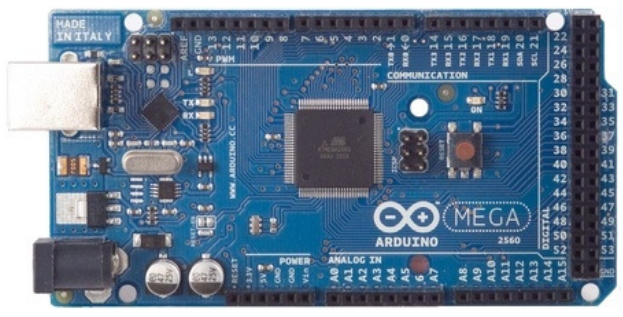
Arduino Esplora



Comme pour la Leonardo basée sur un processeur 32U4 cette carte intègre en direct des périphériques divers pour une utilisation autonome.

On trouve entre autre : Boutons poussoir, joystick digital, potentiomètre, microphone, capteur de lumière et de température et un connecteur pour écran couleur.

Arduino MEGA



Peu de différences de principe avec les cartes précédentes, basée sur un processeur Atmel 1280, cette carte est une des plus complètes de la gamme avec un nombre de ports E/S de 54 points digitaux et 16 analogiques. Les connecteurs d'extensions sont prévus pour accueillir les cartes filles des modèles Uno ou Duemilanove.

Arduino MEGA2560

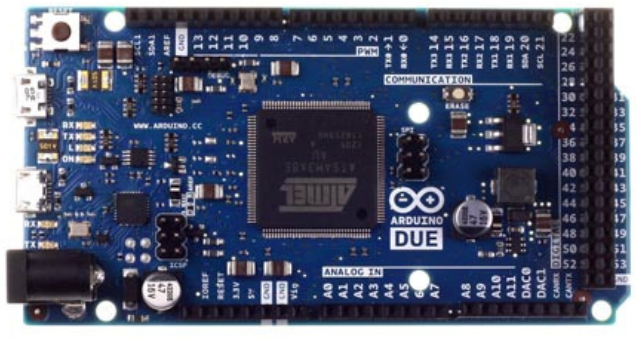
Identique à la Méga le processeur est remplacé par un 2560 disposant de plus de 128Kio de mémoire flash supplémentaire.

Comme pour la UNO l'interfaçage USB est confié a un chipset 8U2 ou 16 U2 suivant les révisions.

Versions

- Rev1 : Chipset USB Atmega 8U2
- Rev2 : Ajout d'une résistance de pull down sur la ligne HWB du 8U2 afin de faciliter le passage en mode DFU (Mise a jour firmware 8U2)
- Rev3 : Ajout de SDA et SCL sur le connecteur JP6, de IOREF et d'une broche libre sur le connecteur Power. Modification du circuit reset et remplacement du 8U2 par un Atmega 16U2.

Arduino Due



Carte actuellement la plus puissante du marche, elle est utilise un processeur Arm cortex M3. Ce dernier fonctionne en 32 bits contrairement aux processeurs précédents 8 bits.

Au niveau des entrées sorties elle reste comparable et compatible avec les modèles Mega.

Comparatif détaillé UNO / MEGA

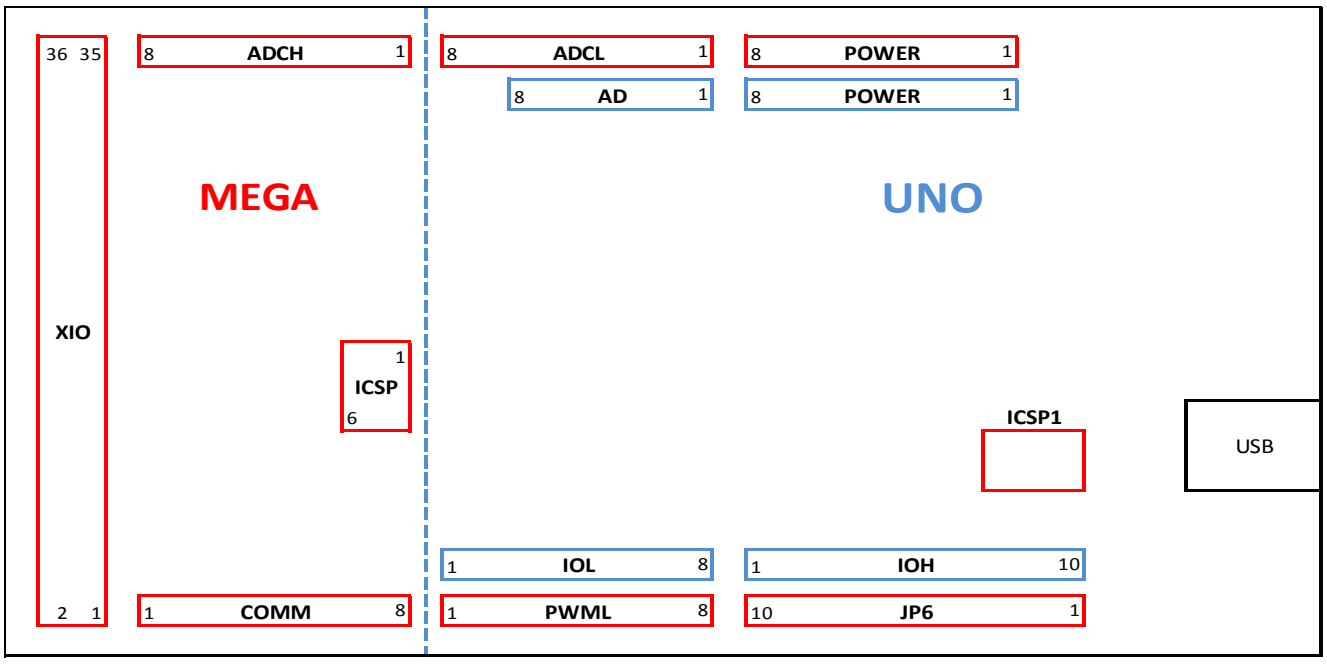
Caractéristiques processeurs

Caractéristiques générales				
	Sens	**Nom E/S**	**ATM 328**	**ATM 2560**
Entrées /sorties digitales	I/O	Pxx	23	86
Sorties Pwm résolution 8 bits / 16 bits	O	Pxx	6 / 0	4 /12
Adressage mémoire externe	I/O		-	Add 16 bits
Entrées analogiques 10bits	I	ADC x	6	16
Comparateur analogique	I	AIN+ , AIN-	1	1
Input capture pin for Timer Counter n	I	ICPn	1	4
Entrées comptage n	I	Tn	2	5
Entrée oscillateur de référence	I	Tosc 1, Tosc2	1	1
Sorties déclenchées par comptage	O	OC val cpt	2 x 3	2x2 + 3x4
Sorties horloge	O	Clk n	1	1
Port I2c	I/O	SDA, SCL	1	1
Port USART série synchrone / asynchrone	I/O	RXn, TXn, CLKn	1	4
Port master/slave SPI	I/O	MESI, MISO, SCK, SS	1	1
JTAG Interface	I/O			1
Interruptions	I	INT n	2	8
Timers / Compteurs 8bits avec comparaison			2	2
Timers / Compteurs 16bits avec comparaison			1	4
Fonctions			131	135
Horloge système			20 Mhz	16 Mhz
Registres généraux 8bits			32	32
Registres E/S 8bits			64 + 160	64 + 416
Mémoire SRam utilisateur			2Ki	8 Kio
Mémoire SRAM externe			-	56 Kio
Mémoire Eeprom			1Ki	4 Kio
Mémoire Flash programme			2Ki	256 Kio

Facteur de forme

Les connecteurs de sortie sont conçus pour que les cartes d'extension prévues pour les modèles de type Uno soient compatibles broche a broche avec celles du format des cartes Mega.

Si les dénominations logiques des broches (Analog n, Digital n) sont identiques entre ces cartes, sur schémas la numérotation des broches du connecteur IOH (Uno) est inversée par rapport a celle de JP6 (Mega).


36 35
8 ADCH 1
8 ADCL 1
8 POWER 1
8 AD 1
8 POWER 1
MEGA
UNO
XIO
1
ICSP
6
ICSP1
USB
1 IOL 8
1 IOH 10
2 1
1 COMM 8
1 PWML 8
10 JP6 1

Détail connecteurs - Affectation ports processeur

Divers système Interruptions	Pin Change interupt	USART	Serial peripheral interface	I2c Serial bus data I/O	Timer /Counter Comparateur AB	Timer / Oscillator	Entrée analogique Comparateur analogique	External memory Add/Data	Nom Port Digital Interne	N° broche ATM2560	MEGA Marquage Platine	MEGA Connecteur Schéma	UNO Connecteur Schéma	UNO Marquage Platine	N° broche ATM328	Nom Port Digital Interne	PWM Possible	Entrée analogique Comparateur analogique	Timer / Oscillator	Timer /Counter Comparateur AB	I2c Serial bus data I/O	USART	Serial peripheral interface	Pin Change interupt	Divers système Interuptions
	PCINT16						An in 8		PK0	89	An In 08	ADCH 1													
	PCINT17						An in 9		PK1	88	An In 09	ADCH 2													
	PCINT18						An in 10		PK2	87	An In 10	ADCH 3													
	PCINT19						An in 11		PK3	86	An In 11	ADCH 4													
	PCINT20						An in 12		PK4	85	An In 12	ADCH 5													
	PCINT21						An in 13		PK5	84	An In 13	ADCH 6													
	PCINT22						An in 14		PK6	83	An In 14	ADCH 7													
	PCINT23						An in 15		PK7	82	An In 15	ADCH 8													
							An In 0		PF0	97	An In 00	ADCL 1	ADCL 1	An In 0	23	16+7		ADC0						PCINT8	
							An In 1		PF1	96	An In 01	ADCL 2	ADCL 2	An In 1	24	PC1		ADC1						PCINT9	
							An In 2		PF2	95	An In 02	ADCL 3	ADCL 3	An In 2	25	PC2		ADC2						PCINT10	
							An In 3		PF3	94	An In 03	ADCL 4	ADCL 4	An In 3	26	PC3		ADC3						PCINT11	
Jtag Clk							An In 4		PF4	93	An In 04	ADCL 5	ADCL 5	An In 4	27	PC4		ADC4			SDA			PCINT12	
Jtag Mode							An In 5		PF5	92	An In 05	ADCL 6	ADCL 6	An In 5	28	PC5		ADC5			SCL			PCINT13	
Jtag Out							An In 6		PF6	91	An In 06	ADCL 7													
Jtag In							An In 7		PF7	90	An In 07	ADCL 8													
INT0				SCL					PD0	43	Dig 21 (SCL)	COMM-1													
INT1				SDA					PD1	44	Dig 20 (SDA)	COMM-2													
INT2		RXD1							PD2	45	Dig 19	COMM-3													
INT3		TXD1							PD3	46	Dig 18	COMM-4													
		RXD2							PH0	12	Dig 17 (PWM)	COMM-5													
		TXD2							PH1	13	Dig 16 (PWM)	COMM-6													
	PCINT9	RXD3							PJ0	63	Dig 15	COMM-7													
	PCINT10	TXD3							PJ1	64	Dig 14	COMM-8													
					ICP4				PL0	35	Dig 49	ICSP-1	ICSP-1		18	PB4							MISO	PCINT4	
											Vcc 5v	ICSP-2	ICSP-2	Vcc 5v											
	PCINT1		SCK						PB1	20	Dig 52 (PWM) (SDA)	ICSP-3	ICSP-3		19	PB5							SCK	PCINT5	
	PCINT2		MOSI						PB2	21	Dig 51 (PWM) (SCL)	ICSP-4	ICSP-4		17	PB3	x			OC2A			MOSI	PCINT3	
/Reset										30	RAZ	ICSP-5	ICSP-5	RAZ	1	PC6								PCINT14	/Reset
											GND	ICSP-6	ICSP-6	GND											
INT0				SCL					PD0	43	Dig 21 (SCL)	JP6-01	IOH 10	-	28	PC5		ADC5			SCL			PCINT13	
INT1				SDA					PD1	44	Dig 20 (SDA)	JP6-02	IOH 9	-	27	PC4		ADC4			SDA			PCINT12	
Analog référence										98	AREF	JP6-03	IOH 8	AREF	21	Analog référence									
											GND	JP6-04	IOH 7	GND											
	PCINT7				OC0A / OC1C				PB7	26	Dig 13 (PWM)	JP6-05	IOH 6	Dig 13	19	PB5							SCK	PCINT5	
	PCINT6				OC1B				PB6	25	Dig 12 (PWM)	JP6-06	IOH 5	Dig 12	18	PB4							MISO	PCINT4	
	PCINT5				OC1A				PB5	24	Dig 11 (PWM)	JP6-07	IOH 4	Dig 11 (Pwm)	17	PB3	x			OC2A			MOSI	PCINT3	
	PCINT4				OC2A				PB4	23	Dig 10 (PWM)	JP6-08	IOH 3	Dig 10 (Pwm)	16	PB2	x			OC1B			/SS	PCINT2	
					OC2B				PH6	18	Dig 09 (PWM) (TX1)	JP6-09	IOH 2	Dig 09 (Pwm)	15	PB1	x			OC1A				PCINT1	
					OC4C				PH5	17	Dig 08 (PWM) (RX2)	JP6-10	IOH 1	Dig 08	14	PB0			CLK0	ICP1				PCINT0	
											Nc	Power 1	Power 1	Nc											
Analog Vcc										100	IOREF Vcc 5v	Power 2	Power 2	IOREF Vcc 5v	20	Analog Vcc									
/Reset										30	RAZ	Power 3	Power 3	RAZ	1	PC6								PCINT14	/Reset
											3v3	Power 4	Power 4	3v3											
General Vcc										10, 31	Vcc 5v	Power 5	Power 5	Vcc 5v	7	General Vcc									
General GND										11, 32	GND	Power 6	Power 6	GND	8	General GND									
Analog GND										99	GND	Power 7	Power 7	GND	22	Analog GND									
											Alim 7-12v	Power 8	Power 8	Alim 7-12v											
PDI	PCINT8	RXD0							PE0	2	Dig 00 (PWM) (RX0)	PWML 1	IOL 1	Dig 00 (Rx)	2	PD0						RXD		PCINT16	
PDO		TXD0							PE1	3	Dig 01 (PWM) (TX0)	PWML 2	IOL 2	Dig 01 (Tx)	3	PD1						TXD		PCINT17	
INT4					OC3B				PE4	6	Dig 02 (PWM)	PWML 3	IOL 3	Dig 02	4	PD2								PCINT18	INT0
INT5					OC3C				PE5	7	Dig 03 (PWM)	PWML 4	IOL 4	Dig 03 (Pwm)	5	PD3	x			OC2B				PCINT19	INT1
					OC0B				PG5	1	Dig 04 (PWM)	PWML 5	IOL 5	Dig 04	6	PD4			T0			XCK		PCINT20	
					OC3A		AIN1		PE3	5	Dig 05 (PWM)	PWML 6	IOL 6	Dig 05 (Pwm)	11	PD5	x		T1	OC0B				PCINT21	
					OC4A				PH3	15	Dig 06 (PWM) (RX3)	PWML 7	IOL 7	Dig 06 (Pwm)	12	PD6	x	AIN0(+)		OC0A				PCINT22	
					OC4B				PH4	16	Dig 07 (PWM) (TX2)	PWML 8	IOL 8	Dig 07	13	PD7		AIN1(-)						PCINT23	
											Vcc 5v	XIO 01													
											Vcc 5v	XIO 02													
								A/D0	PA0	78	Dig 22	XIO 03													
								A/D1	PA1	77	Dig 23	XIO 04													
								A/D2	PA2	76	Dig 24	XIO 05													
								A/D3	PA3	75	Dig 25	XIO 06													
								A/D4	PA4	74	Dig 26	XIO 07													
								A/D5	PA5	73	Dig 27	XIO 08													
								A/D6	PA6	72	Dig 28	XIO 09													
								A/D7	PA7	71	Dig 29	XIO 10													
								A15	PC7	60	Dig 30	XIO 11													
								A14	PC6	59	Dig 31	XIO 12													
								A13	PC5	58	Dig 32	XIO 13													
								A12	PC4	57	Dig 33	XIO 14													
								A11	PC3	56	Dig 34	XIO 15													
								A10	PC2	55	Dig 35	XIO 16													
								A9	PC1	54	Dig 36	XIO 17													
								A8	PC0	53	Dig 37	XIO 18													
						T0			PD7	50	Dig 38	XIO 19													
								ALE	PG2	70	Dig 39	XIO 20													
								/RD	PG1	52	Dig 40	XIO 21													
								/WR	PG0	51	Dig 41	XIO 22													
									PL7	42	Dig 42	XIO 23													
									PL6	41	Dig 43	XIO 24													
					OSC5C				PL5	40	Dig 44 (PWM)	XIO 25													
					OSC5B				PL4	39	Dig 45 (PWM)	XIO 26													
					OSC5A				PL3	38	Dig 46 (PWM)	XIO 27													
						T5			PL2	37	Dig 47	XIO 28													
					ICP5				PL1	36	Dig 48	XIO 29													
	PCINT3		MISO						PB3	22	Dig 50	XIO 31													
	PCINT3		MISO						PB3	22	Dig 50	XIO 31													
	PCINT2		MOSI						PB2	21	Dig 51 (PWM) (SCL)	XIO 32													
	PCINT1		SCK						PB1	20	Dig 52 (PWM) (SDA)	XIO 33													
	PCINT0		/SS						PB0	19	Dig 53 (PWM) (RX1)	XIO 34													
											GND	XIO 35													
											GND	XIO 36													
					ICP1				PD4	47															
		XCK1							PD5	48															
						T1			PD6	49															
		XCK0					AIN0		PE2	4															
INT6						T3			PE6	8															
INT7					ICP3	CLK0			PE7	9															
						TOSC2			PG3	28															
						TOSC1			PG4	29															
		XCK2							PH2	14															
						T4			PH7	27															
	PCINT11	XCK3							PJ2	65															
	PCINT12								PJ3	66															
	PCINT13								PJ4	67															
	PCINT14								PJ5	68															
	PCINT15								PJ6	69															
									PJ7	79															
Xtal1										34					9	PB6			TOSC1					PCINT6	XTAL1
Xtal2										33					10	PB7			TOSC2					PCINT7	XTAL2

Extensions - Shields - Modules périphériques

Shields

Afficheur LCD

Modèle courant constitué d'un afficheur LCD 2 lignes de 16 caractères, et d'une série de touches codées en analogique.

L'afficheur utilise les entrées suivantes :

- *Enable* : Out digital 9
- RS : Out digital 8
- D4 a D7 : Out digital 4 a 7
- Retro-Eclairage : Out digital 3

- Touches : In analog 0

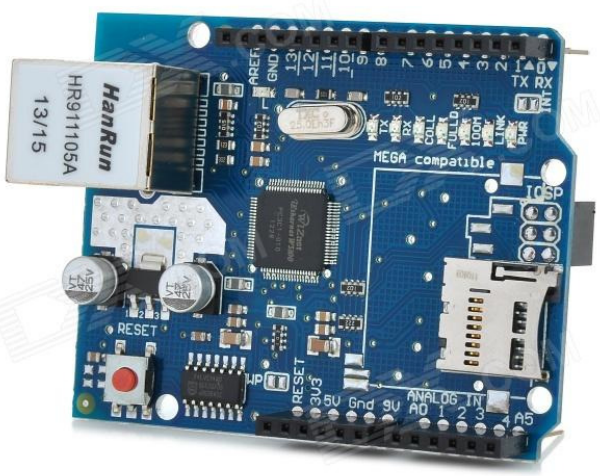
La déclaration de l'afficheur avec la bibliothèque LiquidCrystal.h utilise les valeurs suivantes :

- LiquidCrystal Nom_Lcd(8, 9, 4, 5, 6, 7);

La lecture des touches se fera par lecture de la valeur de l'entrée analogique A0 avec les valeurs suivantes (Une seule touche pouvant être actionnée simultanément) :

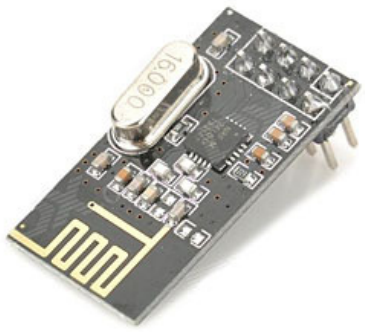
- None : 0x3FF
- Select : 0x2D0
- Gauche : 0x1D9
- Bas : 0x12C
- Haut : 0x082
- Droite : 0x000

Interface Ethernet - Lecteur carte SD



Modules périphériques

Modules Radio NrfRF24

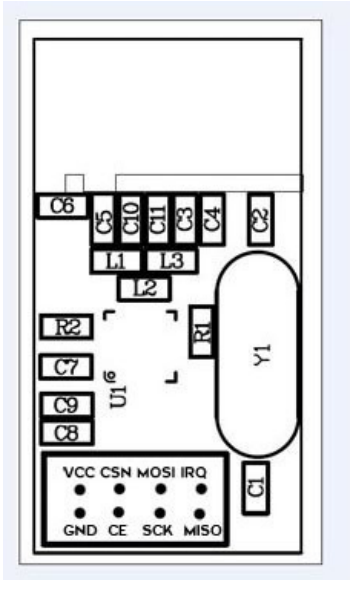


Module radio travaillant dans la bande des 2.4 Ghz et basé sur le chipset nRF24L01+ de Nordic semiconductor. Ce module a une portée d'environ 100m et un débit possible maximal de 2Mbs.

Alimenté exclusivement en 3v3, ce module utilise l'interface SPI de l'Arduino et deux E/S de contrôle. L'entrée Chip enable permet la mise en/hors service globale du circuit intégré et de ses fonctions émission/réception, l'entrée Spi CS ne sert que pour le dialogue sur le bus SPI.

Plusieurs bibliothèques permettent de commander ce module, celles-ci utilisent les bibliothèques communes SPI.h et nRF24L01.h.

Câblage avec bibliothèque RF24 maniac bug



	Module	Câblage	Arduino Uno	Arduino Mega
1	0v	Marron		
2	Vdd 3v3	Rouge		
3	Chip enable	Orange	Dig9	Dig49
4	Spi chip sel	Jaune	Dig10 / SS	Dig53 / SS
5	Spi clocl	Noir	Dig13 /Sck	Dig52 / Sck
6	Spi slave in	Blanc	Dig11 / Mosi	Dig 51 / Mosi
7	Spi slave out	Bleu	Dig12 / Miso	Dig 50 / Miso
8	Irq	Vert	Dig2 / Int0	Dig2 / Int0

Acronymes

ICSP	In circuit serial programming. Méthode de programmation de mémoire ou microcontrôleur sur leur circuit d'usage.
PWM	Pulse width modulation. Variation d'une valeur analogique par modification du rapport cyclique d'une sortie digitale.
SPI	Serial protocol interface. Liaison série asynchrone fonctionnant en maitre esclave
TWI	Two wire interface. Liaison deux fils, typiquement I^2C
USART	Universal synchronous serial receiver and transmitter

Interface de développement

Basé sur Java l'IDE Arduino nécessite donc sa présence sur la machine hôte, celle-ci devra disposer d'un port série (virtuel / usb) pour communiquer avec les cartes CPU. Les différentes versions sont téléchargeables librement sur le site du constructeur www.arduino.cc .

Les fichiers de projet sont enregistrés au format .ino (anciennement .pde).

Installation

- Windows : Version portable, le zip source doit juste être décompressé dans son emplacement final (Env 250 Mo pour la version 1.04). Utilise les bibliothèques Sygwin.

  Les drivers des convertisseurs Usb/série des chipsets FTDI ou xU2 se trouvent dans le répertoire …\Arduino\Drivers et doivent être installés manuellement.

- Mac Os : Version portable également, ouvrir simplement le .dmg et glisser le logiciel Arduino.app dans le dossier application. Installer le driver .mpkg .

- Linux : Disponible sur les sources standards, installer via un "apt -get install Arduino", les dépendances seront mises à jour automatiquement.

Structure projets

Les projets ou croquis (sketch) sont constitués de un ou plusieurs onglets contenant le code utilisé lors de la compilation. Chaque onglet est enregistré a son nom dans un répertoire utilisant le nom du projet. Par défaut les fichiers d'un projet sont situes dans le répertoire Documents/Arduino de l'utilisateur et accessibles dans le menu Fichier / Carnet de croquis.

Les onglets sont gérés par le bouton  , les effacements sont définitifs, les importations de l'onglet d'un autre projet sont possibles par le menu croquis / ajouter un fichier, le fichier est alors copié dans le répertoire du nouveau projet.

Chaque fichier programme est en format texte brut et doté de l'extension .ino

Types de carte

Il est important de préciser le type de carte utilisée lors de la compilation, la vérification des paramètres d'instruction et surtout l'affectation des numéros de ports E/S sera réalisée en fonction de cette option. Il en sera de même avec certaines instructions, par exemple l'instruction PORTC adressera les ports 30 a 37 sur carte Mega et les ports analogiques 0 a 6 sur carte Uno.

Bibliothèques

Les bibliothèques sont des collections de fonctions, internes (Serial ..), externes fournies avec l'IDE (Eeprom, Ethernet, servo …) ou de source autre.

Les fichiers (.h ou . cpp) nécessaires a une bibliothèque doivent être stockés dans un répertoire au nom de cette bibliothèque dans le répertoire Documents/Arduino/Libraires. L'import automatique de ces fichiers a partir d'une archive .zip ou d'un répertoire les contenant a partir du menu Croquis / Import Bib.. / Add library.

La déclaration de l'utilisation d'une bibliothèque dans un projet se fait manuellement a l'aide de l'instruction #include <nom_fichier.h> ou automatiquement a partir de la liste de choix du menu Croquis / Importer bibliothèque.

Langage de programmation Arduino

Lors de la compilation le programme doit respecter la structure suivante en étant imbriqué dans ces deux fonctions obligatoires :

- **Setup** : Comprenant le code effectué une seule fois a l'initialisation (Configuration registres E/S, initialisation matériel ….)
- **Loop** : Boucle principale du programme répétée automatiquement.

```
void setup() {
        // Code initialisation
}

void loop() {
        //Code principal
}
```

En dehors de ces deux blocs on ne doit trouver que ceux des différentes fonctions et sous-programmes éventuels ainsi que la définition des variables globales.

Syntaxe générale

;	: Obligatoire a la fin de chaque ligne d'instruction, sauf sur fin de bloc de code }
{ *code* }	: Définition d'un bloc de code
' *Texte* '	: Définition d'une chaine de caractère
// *Texte*	: Commentaire sur une ligne
/* *Bloc texte*	: Commentaire sur plusieurs lignes

Attention, le nom des instructions, des variables ou des fonctions est sensible a la case. Le nom des variables ou des fonctions ne doivent pas contenir de caractères réservés (-, &, ! …)

Variables

Types de variables et constantes

Les types de variables doivent être définis obligatoirement une fois ainsi qu'éventuellement leur valeur initiale. Les deux formes suivantes peuvent être utilisées

```
Type Nom_Variable;
Type Nom_Variable=Valeur;
```

La portée des variables est limitée au bloc de code ou fonction dans lequel elles sont définies, les variables de portée globales doivent être définies en dehors des blocs Setup et Loop.

```
int Var1=0;
Void loop() {
        long Var2;
        For ( Int Var3; var3 < 10; Var3++)    { }
}

Void Fonction1 ()    { char Var4 }
```

```
// Var1 est utilisable partout
// Var2 est utilisable dans la boucle loop et For
// Var3 n'est utilisable que dans la boucle For
// Var4   ""                                   ""   Fonction1
```

	Valeurs min / max		Lg bits	Lg Octets
Int	-32 768	+ 32 767	16	2
Long	-2.14 EE 9	+2.14 EE 9	32	4
Char	-128	+128	8	1
Float Double	-3.4 EE 38 Limité à 6-7 chiffres significatifs	+3.4 EE 38	32	4
Unsigned char Byte	0	255	8	1
Unsigned int Word	0	65535	16	2
Unsigned long	0	4.29 EE 9	32	4
Boolean	0	1	1	1

Les formes standard uintXX_t utilisées en C peuvent aussi être employées pour déclarer une variable non signée avec xx représentant la longueur en bit de la variBable (8, 16, 32 ou 64).

Instructions de Conversion

Les instructions suivantes permettent de convertir une variable de n'importe quel type en valeur du type défini.

- Char (Variable) :
- Byte (Variable) :
- Int (Variable) :
- Long (Variable) :
- Float (Variable) :

Pointeurs

L'utilisation de &Nom_Variable dans une instruction force celle-ci à utiliser l'adresse de cette variable et non son contenu.

Tableaux de variables

La création de tableaux de variables a une seule dimension est possible lors de la déclaration de la variable par l'ajout de sa taille implicitement ou explicitement entre crochets.

Si la valeur de l'index commence a partir de zéro, la taille du tableau est réelle.

Si la taille du tableau est utilisée pour calculer son allocation mémoire lors de la compilation, aucun contrôle de l'index n'est effectué lors de l'exécution rendant possible les débordements mémoire.

```
Int Tableau1 [6];                 // creation d'un tableau vide d'une taille de 6 elements
Int Tableau1 [6] = {1,2,4};       // creation d'une taille de 6 elements et initialistion
Int Tableau1 [] = {1,2,4,8};      // creation et initialisation d'un tableau de 4 elements (taille auto)
Tableau1 [2] == 4;                // initialisation 3e cellule du tableau
```

Chaine de caractères

Les chaines de caractères sont considérées comme des tableaux de type char, la dernière valeur utile étant définie par la valeur zero.

```
Char String1[15];                    // Creation d'une chaine vide de 14 caracteres utilisables
Char String2[] = "azerty";           // Creation auto d'un tableau de 7 valeurs, la derniere contenant 0
Char String3[10] = { 'a', 'z', 'e', 'r', 't', 'y', '\0' }
```

Il est possible de créer des tableaux de chaine de caractères, l'utilisation d'une étoile après char indiquant l'utilisation d'un pointeur en place d'un accès direct.

```
Char* TableauS[] = { "Chaine1", "Chaine2", "Chaine3", ..... }       // Definition du tableau

.......... TableauS[ N ]..........                                   // Utilisation de la valeur N du tableau
```

Constantes : #Define, const

Les constantes peuvent être définies soit par l'instruction #define soit l'instruction Const type, cette dernière devant être préférée.

Les lignes utilisant #define ne doivent pas être terminées par ; ni comprendre de signe égal.

```
#Define Nom_Const Valeur
Const Type_Const Nom_const = Valeur;
```

Les valeurs utilisées peuvent être indiquées dans les bases suivantes :

- Décimal : Valeur (123)
- Binaire : B Valeur sur 8 bits (B110010)
- Octal : 0 Valeur (012) ! Tout chiffre commençant par zéro est considéré en octal
- Hexa : 0x Valeur (0x7B, 0x12, 0xaF .....)
- Exposant : e (1.2e3 = 1200) ou ^ (2.32^-1 = 0.23)

Constantes prédéfinies

true / false : 1 et 0, lors d'une comparaison toute valeur différente de 0 est considérée comme true.

LOW / HIGH : Utilisés avec les entrées / sorties. Dans le cas d'un port configuré en entrée toute valeur reçue inferieure a 2v sera considérée Low, supérieure a 3v High. Dans le cas d'un port de sortie sa valeur sera respectivement mise a une valeur de 0v et 5v.

INPUT / OUTPUT : Utilisés lors de la définition d'un port E/S avec l'instruction pinMode().

Operations de base sur variables

Initialisation variables

Les variables peuvent être initialisées avec n'importe quel chaine de caractère, valeur numérique, ou constantes prédéfinies (Ex : True, False).

Les caractères sont délimités entre guillemets simples, les chaines avec des guillemets doubles.

```
Nom_var_Int = 128;
```

```
Nom_var_int  = 0x6D
Nom_var_char = 'a';
Nom_String [0] = "Chaine";
```

Operations mathématiques

Les operateurs suivants peuvent être utilisés : + - * / et % (Modulo : reste de la division)

```
Var1 = Var2 + 2;
Var1 = Var2 / Var3;
```

L'auto incrémentation / décrémentation est aussi possible en doublant le signe + ou - , les formes suivantes donneront le même résultat.

```
Var1 ++;
Var1 = Var1 + 1;
```

Récursivité

L'utilisation de l'operateur avant = indique que l'on utilise la valeur cible comme premier argument. Utilisable avec : +, -, *, /, &, |

```
Var1 += Var2;      équivalent a Var1 = Var1 + Var2;
```

Operations logiques

Opération logique sur Booleans (Niveau logique True-False de la valeur)

! : Inversion logique (Identique a Var=1-Var pour une valeur 1bit)
|| : Ou de deux valeurs
&& : Et de deux valeurs

```
Var1 = ! Var2;
Var 1 = Var2  && Var3;
```

Operations bit a bit

& : ET logique de deux valeurs
| : Ou " "
^ : Xor " "
~ : Not " "
>> n : Décalage circulaire a droite de n bits
<< n : Décalage " " a gauche de n bits

```
Var1 = Var2 & Var3        // 0000 1011 & 0000 1001 = 0000 1001
Var1 = Var1 >> 2          // 0011 0001 >> 2 = 0100 1100
```

Fonctions mathématiques spéciales

Fonctions, blocs et sous-programmes

Operateurs conditionnels

==	: Egalite. *Attention un simple = provoque l'opération et non pas la comparaison*
!=	: Différence (Not egal)
< <=	: Inferieur, inferieur ou égal
> >=	: Supérieur, supérieur ou égal
&&	: Condition1 ET condition2
||	: Condition1 OU condition2
!	: Inversion, Not condition

Les conditions peuvent être imbriquées à l'aide de parenthèses (Toujours utilisées avec l'operateur d'inversion).

```
(Var1 < 100)                                  // Vrai tant que la variable 1 est inferieure a 100
( ( Var1 == n ) || ( Var2 == x  && Var3 == z ) )  // Vrai si var1=n ou var2 et var egals a x et z.
! ( Var1 == n)                                // Vrai si Var1 different de n
```

Fonctions et blocs : Void, Return

Fonction simple : Void

Les fonctions sans retour de résultat sont définies par l'instruction Void (Liste de variables). La liste de variables éventuelle et leur typage permet de passer des paramètres d'entrée a cette fonction. L'appel a la fonction simplement par son nom suivi des valeurs de paramètres entre parenthèses : nom_fonction();

```
Void Nom_Fonction (Type Var1, Type Var2)   {Bloc de Code fonction}

Nom_fonction(valeur1, valeur2);
```

Fonction typée avec retour : Return

L'instruction return permet d'avoir un retour du résultat du sous programme. Dans ce cas void est remplacé par le type de la variable retournée, sa valeur étant définie par l'instruction return.

```
Type Nom_Fonction () {
        Bloc de code du traitement
        Return variable_ou_valeur_resultat;
}

Utilisation = Nom_Fonction;
```

Sauts inconditionnels : goto Label

Comme habituellement l'instruction goto est à manier avec précautions, l'obtention de boucles infinies étant assez facile a réaliser. Le nom du label pointant sur une ligne du programme doit être suivi de **:** (sans ; de terminaison), l'appel se fait par un simple *goto label;* .

```
Label1 :
        ……..
        goto label1;
```

Sauts conditionnels: If , Else, Switch, Case

Initialisation conditionnelle du contenu d'une variable

La variable cible est chargée avec une des valeurs vraie ou fausse en fonction de la condition.

VarCible = (Condition) ? Valeur_Vraie Valeur_Fausse

Execution conditionnelle : if, else, else if

Ces instructions permettent d'exécuter un bloc de code en fonction d'une condition. Si la condition est vraie le bloc situé après If est exécuté, sinon ce sera le bloc après else. Else if permet d'imbriquer une seconde condition.

Chaque bloc de code doit être délimité par des accolades {}.

```
If ( Condition1 )
        {
        // Code execute si condition1 vraie
        }
else if (Condition2)
        {
        // Code execute si condition 1 fausse et condition 2 vraie
        }
else
        {
        // Code execute si condition 1 fausse et condition 2 fausse
        }
```

Sélection choix multiple : Switch - Case

Permet d'exécuter plusieurs blocs de code en fonction de la valeur d'une variable, la condition d'exécution de chaque bloc est définie par la fonction Case, la valeur Default est utilisé si aucune condition vraie n'a été rencontrée. Afin d'éviter d'exécuter tout les blocs Case situé après une réussite la fonction Break permet de sortir de l'ensemble Switch.

```
Switch (NomVariable_a_tester)
        {
        Case Valeur1:
                { //Bloc executé si variable=valeur1}
                Break;
        Case Valeur2:
                { //Bloc executé si variable=valeur2}
                Break;
        Case default:
```

```
                { //Bloc executé autrement}
                Break;
        }
```

Boucles : while, do, for, continue, break

Boucles conditionnelles : do - while

Deux structures sont possibles :

- Soit avec l'instruction While seule, le bloc de code situé après est exécuté tant que la condition est vraie.
- Soit sous la forme Do {Bloc de code] While, le bloc de code est exécuté au moins une fois puis bouclé tant la condition associée a While est vraie (Ne pas oublier ; a la fin de la ligne while)

```
while (Condition)    {Bloc de code}

do    {Bloc de code}
while (Condition);
```

Boucles incrémentales : for

L'instruction For (paramètres) permet d'exécuter le bloc de code situé après un certain nombre de fois définis par ses paramètres. Ces derniers pouvant être optionnels doivent être séparés par des points-virgules et sont :

- Initialisation : Instruction effectuée une seule fois. Généralement définition de la variable de travail, son type et sa valeur de départ.
- Condition : Condition requise pour l'exécution de la boucle
- Opération : Instruction exécutée à chaque boucle, généralement incrémentation de la variable de travail.

```
for (int var1=0; Var1 < 100; var1 ++)    {Bloc de code}

Int var1;
Var1=valeur;
for ( ; var1 > 100; )    {
        Bloc de code
        Var1 = var1 - 10;
        }
```

Sorties de boucles : Continue, Break

L'instruction Continue dans une boucle Do, While ou For permet d'empêcher l'exécution du code de la boucle situé après elle.

```
For (int var1=0; Var1 < 100; var1 ++)    {
        // Bloc de code toujours executé
        If (condition) { Continue; }
        // Bloc de code ignoré si la condition est vraie
}
```

L'instruction Break dans une boucle Do, While ou For permet une sortie immédiate de la boucle sans attendre les conditions normales de sortie.

```
While (Condition_Normale) {
        // Bloc de code
        If (Condition_Sortie) { Break; }
        // Bloc de code
}
```

Temporisations

Horloge interne : millis(), micros()

L'instruction millis () renvoi sous la forme d'une valeur de type long non signée le nombre de millisecondes depuis la dernière initialisation du programme, le débordement intervient environ au bout de 50 jours.

L'instruction micros() est identiques mais la valeur renvoyée est en microsecondes avec une résolution de 4 µs sur processeurs 16Mhz (le double sur 8Mhz) et un débordement après environ 70 minutes.

```
Heure_actuelle = millis();
```

Temporisation - Pause : delay(ms), delayMicroseconds(µs)

Les instructions Delay et DelayMicroseconds effectuent respectivement une pause dans l'exécution du programme du nombre de millisecondes ou microsecondes indiquées.

Lors de cette pause toute exécution est bloquée, les interruptions et les ports serie ne sont actifs que a priori avec l'instruction DelayMicroseconds.

```
delay (1000);       // Pause d'une seconde
```

Gestion ports entrées - sorties

Configuration des E/S : pinMode

L'instruction pinMode (Numéro de port, mode) avec l'utilisation pour la valeur mode les constantes Input et Output configure le port soit en entrée a haute impédance, soit en sortie totem pole.

La résistance interne de tirage au +5v de 20Kho peut être activée avec l'instruction DigitalWrite (Port, High) sur un port configuré préalablement en entrée.

Les numéros de port valides sont 0 a 13 pour les ports digitaux classiques, 14 a 19 pour les analogiques (0 a 5) sur carte de type Uno. Sur carte Mega la numérotation va de 0 a 53 pour les ports digitaux et de 54 a 69 pour les ports analogiques.

```
Int Port_Entree_A1 = 15;
```

```
pinMode ( Port_Entree_A1, Input);
```

Lecture / Ecriture entrée numérique : digitalWrite, digitalRead

L'instruction digitalRead (Port) renvoie sous forme d'une valeur Int la valeur lue sur le port indiqué.

L'instruction digitalWrite (Port, Valeur) écrit sur le port sélectionné la valeur indiquée

Les valeurs peuvent être sous la forme numérique (0, 1) ou des constantes Low et High.

```
Int PortD4 = 4;
Int PortA1 = 15;
Int Var1;
Pinmode (PortD4, Output);
Pinmode (PortA1, Input);

Void loop () {
Delay (1000);
Var1 = DigitalRead (PortA1);
DigitalWrite (PortD4, Var1);
Delay (1000);
DigitalWrite (PortD4, Low);
}
```

Référence analogique : analogReference

L'instruction analogReference(type) configure la référence de tension utilisée par le convertisseur A/D interne. La valeur type peut prendre les valeurs suivantes

- DEFAULT : Valeur par défaut, utilise Vcc comme référence
- INTERNAL : Pour carte de type Uno, utilise la référence interne de 1.1v sinon 2.56
- INTERNAL1V1 : Pour carte de type mega, " "
- INTERNAL2v56 : " " " " de 2.56v
- EXTERNAL : Utilise l'entrée externe Aref

Entrées analogiques : analogRead

L'instruction analogRead(Port) renvoie une valeur de type Int contenant le résultat du convertisseur A/D du port indiqué.

Les ports 0 a 7 peuvent être utilisés. La résolution de 1024 points est d'environ 4.9mv par points. La durée de conversion étant d'environ 100µs la fréquence maxi d'échantillonnage est de 10Khz sans tenir compte bien sur du temps de traitement.

Les ports analogiques n'étant qu'en entrée l'instruction pinmode peut être omise.

```
Int PortAn = 3;
Val = analogRead (PortAn);
```

Sorties PWM : analogWrite

L'instruction analogWrite (Port, Valeur) permet d'envoyer une impulsion de largeur variable et d'une fréquence d'environ 490Hz sur le port sélectionné. La valeur de largeur est de 0 a 255 (100%).

Les cartes de type Atmega acceptent cette fonction pour les ports 3,5,6,9,10 et 11, les cartes de première génération basées sur un Atmega8 ne disposent que des ports 9,10,11.

```
Pinmode (3, Output);
analogWrite (3, 127);        // Signal rapport cyclique 50%
```

Sorties en Fréquence : Tone, NoTone

L'instruction Tone (Port, Fréquence, [Durée]) émet sur le port sélectionné un signal de rapport cyclique 50% a la fréquence indiquée en Hz. Le signal est émis le temps donné par la valeur Durée en ms, si celle-ci est omise jusqu'à exécution de la commande NoTone. Cette instruction ne peut être exécutée que sur un seul port a la fois.

L'instruction NoTone (Port) stoppe le signal éventuellement émis sur le port indiqué.

Mesure d'une impulsion : PulseIn

L'instruction PulseIn (Port, ValeurSeuil, [Delai]) renvoie sur une valeur de type long non signée la durée en µs d'une impulsion reçue sur le port sélectionné. La valeur Seuil détermine le type de l'impulsion (Low ou High), le Delai optionnel (1s par défaut) fixe le temps avant abandon de la mesure si aucun signal n'est reçu.

Accès direct aux ports CPU : DDR, PORT, PIN

Il est possible d'accéder directement aux registres des ports de la CPU. Si l'accès a une entrée particulière n'est pas possible les 8 bits étant modifiés simultanément l'accès est beaucoup plus rapide que par les instructions digital Read/write.

Les instructions sont suivies de la lettre du port Cpu en majuscule, A a D pour un Uno, A a L pour un Mega sous réserve de leur utilisation par le système et de la structure Cpu.

DDRn : Registre de direction, bit a 1 pour une sortie, 0 pour une entrée.
PORTn : Registre E/S, lecture ou écriture des données sur le port
PINn : Registre entrées, lecture seule des données.

La aussi les résistances de pull-up peuvent être activées sur un port en entrée en plaçant le registre de sortie a 1.

Ex :
```
DDRK = 0x00;
PORTK = 0xFF;
Lecture : PINK;
```

Communication

Sortie registre a décalage synchrone / SPI : shiftOut

L'instruction shiftOut (PortD, PortH, Sens, Valeur) transfère sur le PortD sous forme sérielle huit bits de la valeur indiquée. Les sens de lecture est déterminé par la valeur Sens en commençant par le bit de poids faible ou fort suivant celle ci (Respectivement LsbFirst ou MsbFirst).

Le cadencement est indiqué a chaque bit sur la sortie PortH.

Port série asynchrone

Les ports séries sont gérés par une bibliothèque intégrée ne nécessitant pas de déclaration. Celle-ci procure une collection d'instructions sous la forme NomPortCom.Instruction(), les cartes de type Uno ne disposant que d'un port série Nomport est toujours Serial, les cartes plus évoluées de type Mega utilisent Serial, Serial1, Serial2, Serial3 pour définir l'un de leurs 4 ports série.

- Initialisation : Nom_Com.begin (Debit, Config);

  Initialise le port com et configure les entrées sorties, interruptions du processeur.

  Débit: Vitesse en baud de transfert, de type long de valeur standard (300, 9600, 57600, 115200 ….) ou particulière au besoin.

  Config : Valeur optionnelle de la configuration du transfert, sous la forme SERIAL_BPS (Par défaut = SERIAL_8N1) avec :
  - B : Nombre de bits (5, 6, 7, 8)
  - P : Parité (N : sans, E : paire, O : impaire)
  - S : Bits de stop (1, 2)

- Libération ports : Nom_Com.end ();

  Libère les ports utilisés par Tx et Rx pour d'autres usages.

- Ecriture port série : Nom_com.write (val_int); Envoie l'octet sur le port
  Nom_com.write (String); Envoie la chaine de caractère sur le port
  Nom_com.write (buff, len); Envoie N octets du buffer (tableau)

  Nom_com.print (valeur) Envoi l'expression en ascii de la valeur
  Nom_com.print (val, format) "" " sous le format specifié (HEX, BIN, DEC) ou le nb de chiffres après la virgule pour une valeur float.
  Nom_com.println (valeur) Idem a print avec ajout auto d'un CrLf

- Etat buffer réception : var_int = Nom_com.available();

  Renvoie le nombre de caractères restant dans le buffer de réception (0 a 63)

- Lecture octet buffer : var_rec = Nom_com.peek ();

  Lit le premier caractère reçu sans l'effacer, renvoie 255 si vide.

- Extraction octet buffer : var_rec = Nom_com.read ();

  Extrait le premier caractère reçu et l'efface du buffer, renvoie 255 si vide.

- Extraction chaine buffer : Nom_com.readBytes (String, Longueur);

Extrait la chaine des Lg premiers caractères reçus dans String et les effaces du buffer.

- Extraction chaine limitée buffer : Nom_com.readBytesUntil (Val_arret, String, Lg_max);

  Extrait les caractères reçus jusqu'a découverte du caractère d'arrêt ou le nombre maxi et les effaces du buffer.

- Recherche octet ou caractère dans buffer : Ret = Nom_com.find (valeur);

  Renvoie une valeur true si la valeur a été trouvée dans le buffer, vide celui-ci jusqu'à la découverte de la valeur ou complètement si pas trouvée.

- Temps maximum de recherche : Nom_com.setTimeout (long_Tms);

  Configure le temps maxi d'attente de caractères avant abandon pour les fonctions readBytes et readBytesUntil. Par défaut = 1000 ms.

Temps d'exécution :

En fonction de la compilation les instructions Arduino peuvent parfois des temps d'exécution différents, le type (byte, word ….) des arguments utilisés peut avoir aussi son importance.

Dans les cas ou le temps d'exécution d'un boucle a son importance le choix des instructions peut être cruciale, entre un valByte / 2 et un valbyte >> 1 la seconde solution sera plus rapide.

Manipulation variables

1.7 µs : valWord = valByte
2 µs : valWord ++
1.75 µs : ValByte / 2

Accès aux ports

0.38 µs : PORTn = valByte
5,5 µs : DigitalWrite (Port, ValNyte)
5.3 µs : DigitalRead (Port)
10 µs : DigitalWrite (Port1, digitalRead(Port2))
110 µs : valInt = analogRead (0) (Independent de la valeur lue)
120 µ s : "" "" avec interruptions activées

Port communication

380 µs : Serial.println (ValeurInt);

Boucles

1.5 µs : if valByte == 1 {}

Temporisations

1.96 µs : delayMicroseconds(1)
2.7 µs : delayMicroseconds(2)
10.8 µs : delayMicroseconds(10)
50 µs : delayMicroseconds(50) avec leger jitter +/- 5µs

Memo instructions courantes Arduino

Type nom_variable = valeur; : Définition et initialisation d'une variable
Type nom_tb ***[N]*** *= {val, val,… };* : Définition et initialisation d'un tableau a une dimension
const *type nom_Ct* = valeur; : Définition d'une constante
Types : Boolean (0 / 1) Char (+/- 128) Byte (0 / 255)
Int (+/- 32768) Word (0 / 65535) Long (+/- $2.14^{e}9$)
Float (Virgule flottante 4 octets)
Constantes : true, false, HIGH, LOW, INPUT, OUTPUT, DEC, BIN, OCT, HEX

Operateurs mathématique : + - * / %
Operateurs bit a bit : & (Et), | (Ou), ^ (Xor), ~ (Not), >> << (Rotations)
Operateurs conditionnels : == , != (Different), < , > , >= , && (Et) , || (Ou) , ! (Not)

void*(type paramètre, ..) {Code}* : Défini une fonction et les paramètres associés
return *variable_valeur;* : Retourne la valeur a l'appelant de la fonction

if *(condition) {Code}* : Exécute le bloc de code si la condition est vraie
else if *(condition) {Code}* : Teste une autre condition et "" ""
else *{code}* : Exécute le code si les conditions précédentes sont fausses

switch *(Non_variable) {* : Associe un choix multiple a une variable
Case *valeur: {Code} }* : Exécute le code si la variable correspond a la valeur

while *(Condition) {Code}* : Exécute le code et boucle tant que la condition est vraie
do *{Code}* ***while*** *(Condition);* : Exécute au moins une fois "" ""

for *(Instruction_Initiale, Condition, instruction_de_boucle) {Code}* : Exécute le bloc et boucle tant que la condition est vraie, instruction de boucle exécutée a chaque tour.

continue*;* : Ignore le code restant dans la boucle while ou for
break; : Sortie de boucle while, for ou de test switch

var_long = ***millis*** *();*: Renvoie le nb de millisecondes depuis le dernier reset
var_long = ***micros*** *();* : "" "" microsecondes "" ""

delay *(Nb_ms);* : Fait une pause de Nb_ms millisecondes
delayMicroseconds *(Nb_µs);* : Fait une pause de Nb_µs microsecondes

pinMode *(Port, Etat);* : Configure l'état du port en entrée ou sortie
var_int = ***digitalRead*** *(Port);* : Lit l'état low ou high du port (valeur de type int)
digitalWrite *(Port, Valeur);* : Met a l'état low ou high le port
var_int = ***analogRead*** *(Port) ;* : Renvoie la valeur de l'adc du port (valeur int 0 a 1023)
analogWrite *(PortPwm, Valeur) ;* : Positionne l'impulsion du port Pwm a la valeur (0 a 255)

tone *(Port, Fhz, [Duree_ms]);* : Envoi un signal carré de fréquence F sur le port spécifié
noTone *(Port);* : Stoppe le signal précédent

shiftOut *(PortD, PortH, Sens, valeur);* : Envoie la valeur sous forme série synchrone

Serial, Serial1, Serial2, Serial 3 : Noms ports coms
NomCom.begin *(Debit, [Config]);* : Initialisation port com, config SERIAL_8N1 par défaut
NomCom.write *(valeur);* : Envoie la valeur ou la variable sur le port
NomCom.write *(tableau, Nb);* : Envoie N octets
Ret = ***NomCom.available*** *( );* : Donne le nombre d'octets restant dans le buffer de réception
Ret = ***NomCom.peek*** *( );* : Lecture sans effacement octet 0 du " "
Ret = ***NomCom.read*** *( );* : Extraction et purge " "
NomCom.begin *(Tableau, Nb)* : Extraction N octets du buffer dans le tableau

---

LiquidCrystal.Nom *(Rs,En,D4,D3,D2,D1);* : Définit les ports utilisés par l'afficheur
NomLcd.begin *(Col,Ln);* : Définit les caractéristiques afficheur
NomLcd.clear *();* : Efface l'afficheur
NomLcd.setCursor *(Col,Ln);* : Positionne le curseur aux lignes et colonnes indiqués
NomLcd.print *(Valeur, [Base]);* : Affiche la valeur (Texte, variable) dans la base indiquée
NomLcd.write *(Val_Ascii);* : Affiche le caractère correspondant a la valeur ascii.

Bibliothèques

L'emploi des bibliothèques de fonction se fait a l'aide de l'instruction #include <nom_bibliotheque.h>

```
#include <Servo.h>;
```

Commande de servo moteur standard 3 fils : Servo.h

Permet de commander un servo-moteur classique commandé en largeur d'impulsions.

- Déclaration d'un servo : Servo Nom_Servo;

- Configuration du servo : Nom_Servo.attach (port, [Min], [Max]);
  Port : Numéro de port utilisé
  Min : Valeur optionnelle en µs de la largeur d'impulsion de l'angle 0 (544 par défaut)
  Max : " " 180 (2400 par défaut)

- Positionnement servo : Nom_Servo.write (angle);

  Angle : Valeur de positionnement voulu du servo de 0 a 180°.

- Lecture de la position du servo : Nom_Servo.read();

  Renvoie la mémorisation de la dernière valeur envoyée par servo.write

```
#Include <Servo.h>
Servo SvMoteur1;

Void setup() {  SvMoteur1.attach(9); }
Void loop() {
SvMoteur1.write(90);
}
```

Commande d'affichage LCD standard Hitachi 4 bits : LiquidCrystal.h

Commande d'un afficheur lcd alphanumérique standard de plusieurs lignes de caractères en câblage 4 bits, le mode complet 8 bits est aussi possible.
La déclaration LiquidCrystal.nom et la définition de l'afficheur Nom.begin (x,y) sont obligatoires.

- Déclaration Afficheur mode 4 bits : LiquidCrystal Nom_Lcd(Rs, En, D4, D5, D6, D7);

  Défini un afficheur LCD et les ports de données utilisés, l'entrée Rw du Lcd doit être connecté au niveau 0, les datas 0 a 3 ne sont pas utilisés.

- Définition caractéristiques afficheur : Nom_Lcd.begin (Nb_Colonnes, Nb_Lignes);

- Extinction / allumage afficheur : Nom_Lcd.nodispaly ();
  Nom_Lcd.display ();

- Effacement texte afficheur et positionnement curseur en 0,0 : Nom_Lcd.clear ();

- Positionnement curseur en 0,0 : Nom_Lcd.home ();

- Positionnement curseur en X,Y : Nom_Lcd.setCursor (Colonne, Ligne);

  Positionne le curseur aux ligne et colonnes indiquées, le comptage commence a 0 avec un référentiel en haut a gauche. Le texte est écrit a partir de la position du curseur courante.

- Affichage / masquage curseur : Nom_Lcd.cursor ();
  Nom_Lcd.noCursor ();

- Clignotement barre soulignement curseur : Nom_Lcd.blink ();
  Nom_Lcd.noBlink ();

- Ecriture chaine sur afficheur : Nom_Lcd.print ("Texte") / (Variable) / (Variable, Base);

  Ecrit sur l'afficheur, un texte ou le contenu d'une variable (char, byte, int, long, string). Avec ces dernières l'option base permet de choisir le mode d'affichage (DEC, BIN, OCT, HEX).
  Le passage a la ligne est effectué en cas de dépassement du tampon de ligne de l'afficheur.

- Ecriture caractère sur afficheur : Nom_Lcd.write (Val_Ascii_Int)

  Ecrit sur l'afficheur le caractère équivalent au code ascii transmis et avance le curseur d'une position.

- Décalage ligne a droite ou a gauche : Nom_Lcd.scrollDisplayLeft () .scrollDisplayRight

  Effectue une permutation circulaire d'un caractère a droite ou a gauche de la fenêtre d'affichage physique par rapport a son buffer de ligne.

- Détermination sens écriture chaine : Nom_Lcd.leftToRight () .rightToLeft

  Détermine le sens de l'affichage à l'aide de la fonction print (Par défaut a gauche). Le contenu des variables est aussi affiché à l' envers.

- Création caractère personnalisé Nom_Lcd.createChar (No_Car, Tableau)

  Configure un des caractères personnalisable de l'afficheur (0 a 7) a l'aide d'un tableau de 8 octets (bits 4 a 0) représentant la matrice de 5x8 pixels du caractère.

```
#include <LiquidCrystal.h>
LiquidCrystal Lcd1 (8,9,4,5,6,7);

Void setup() {
        Lcd1.begin (16, 2);
}
Void loop () {
        Lcd1.clear();
        Lcd1.setCursor (5,0);
```

```
        Lcd1.print ( "Texte Test");
}
```

Temps d'exécution

Sur carte AtMega la durée d'exécution des commandes suivantes avec un affichage standard de 2x16 caractères sera d'environ :

- .clear 2.5ms
- .setcursor 0.4ms
- .print (1 caractère) 0.4ms
- .print (16 caractères) 6ms

Gestion télécommande infrarouge : IRremote.h

Permet l'envoi ou le décodage des trames IR aux standards Nec, Sony, Rc5, Rc6.

Bibliothèque crée à l'origine par Ken Shirriff : http://arcfn.com

Les résultats sont stockes dans une variable de type decode_results collection qui doit être déclarée obligatoirement.

VariableRes.decode_type : Type de codage trouvé (Nec : 1, Sony : 2, RC5 : 3, RC6 : 4, DISH : 5, SHARP : 6 …… Inconnu : -1)
VariableRes.value : Valeur du code, 0 en cas de code inconnu
VariableRes.bits : Nombre de bits du code
VariableRes.rawbuf : Trame brute reçue
VariableRes.rawlen : Longueur de la trame

- Déclaration de l'objet receiver : IRrecv irrecv(Receive_Pin);

  Crée l'objet récepteur et définit le port d'entrée (A placer en section déclaration)

- Initialisation réception : irrecv.enableIRIn ();

  Démarre, configure l'interruption et le scan toute les 50µs (Section setup)

- Lecture de code : Int_Ret = irrecv.decode (Var_Res);

  Essaye de lire un code valide, renvoie la valeur True si une valeur valide a été enregistrée dans Var_Res.

- Reset réception : irrecv.resume ();

  Réinitialise les buffers de réception, a utiliser après chaque lecture.

- Autorisation voyant réception : irrecv.blink13 (True / False);

  Autorise le clignotement du port 13 en cas de réception d'un signal IR

```
#include <IRremote.h>
IRrecv irrecv (11);
Decode_results ValeurIR;

Void setup () {
        irrecv.enableIRIn ();
        irrecv.blink13 (true);
}
Void loop () {
        If ( irrecv.decode (&ValeurIR) ) {
                Code = ValeurCode.value;
                Irrecv.resume ();
        }
}
```

Langage machine Atmel