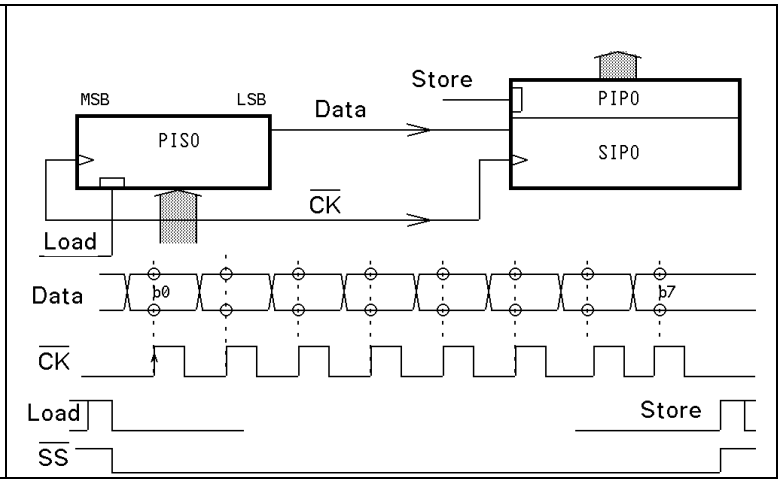
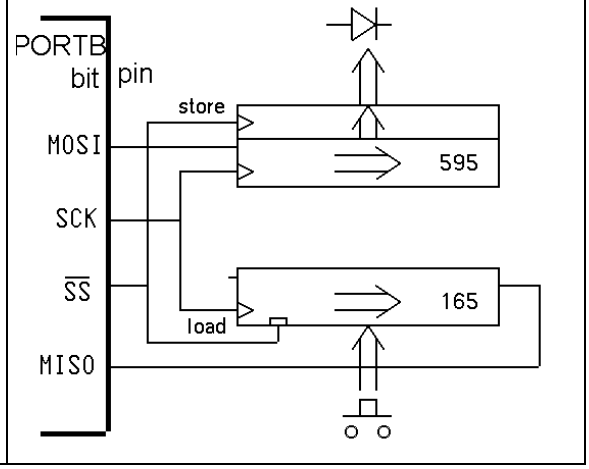
Transferts série et SPI

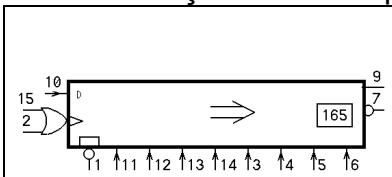
Le transfert d'information entre deux registres peut se faire en série. Les bits sont transférés en synchronisme avec une horloge. Un signal Load charge le registre PISO (Parallet In Serial Out). A chaque front montant de l'horloge Ck, un bit est transféré dans le SIPO (serial In Parallel Out) et après 8 impulsions d'horloge, l'information est transfélée par une impulsions Store dans le PIPO (Parallel In Parallel Out).

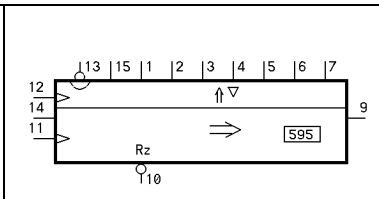


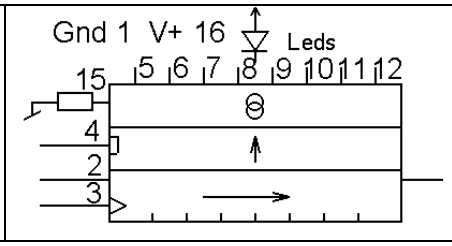
Le registre PISO peut être programmé, il existe en général dans les microcontrôleurs sous le nom des SPI (Serial Peripheral Interface). Las signaux sont appelés
MOSI Master Out Slave In
SCK Serial ClocK
/SS Slave Select
Le SPI peut aussi recevoir de l'information.
MISO Master In Slave Out
Le signal /SS jour à la fois le rôle de load et store pour l'esclave. S'il y a plusieurs périphériques SPI, on multiplie les signaux de sélection.



Quantité de capteurs, interfaces, affichages ont une interface SPI. On utilise aussi beaucoup des circuits TTL ou HC pour commander des Leds ou lire des entrées. C'est une façon de démutiplier les entrées-sorties du microcontrôleur.







Transfert programmé

On peut transférer de l'information en série par programmation en utilisant n'importe quelles pins. Dans les exemples ci-dessous les pins SPI usuelles ont été utilisées. Un oscilloscope a permis de mesurer les temps pour transférer 8 bits.

Programmation Arduino	Fonction Arduino shitOut ()	Programmation C

//Decale1.pde| durée 115 us
#define Da 15
#define Ck 14
#define SS 16
void setup ()
{
pinMode (Da,1);
pinMode (Ck,1);
pinMode (SS,1);
}
void loop() {
int motif = 0x24 ;
	//Decale2.ino 128 us
	#define Da 11
	#define Ck 13
	#define Ld 10
	
	void setup () {
	pinMode (Da,OUTPUT);
	pinMode (Ck,OUTPUT);
	pinMode (SS,OUTPUT);
	}
	void loop() {
	byte motif = 0x24 ;
		//Decale3.ino 8 bits 8.1 us
		#define bSS 2 //pin 10 =0 pendant transfert
		#define bCk 5
		#define bDa 3
		#define SSOn bitSet (PORTB,bSS);
		#define SSOff bitClear (PORTB,bSS)
		#define CkOn bitSet (PORTB,bCk);
		#define CkOff bitClear (PORTB,bCk);
		#define DaOn bitSet (PORTB,bDa);
		#define DaOff bitClear (PORTB,bDa);
		void setup () {

```
  digitalWrite (SS,LOW) ;
  for (byte i=0;i<8;i++) {
    if (motif & 0x01) {
      digitalWrite(Da,1);
    }
    else digitalWrite(Da,0);
    motif >>=1 ;
    digitalWrite (Ck,HIGH) ;
    digitalWrite (Ck,LOW) ;
  }
  digitalWrite (SS,HIGH) ;
  delayMicroseconds (100);
}
```

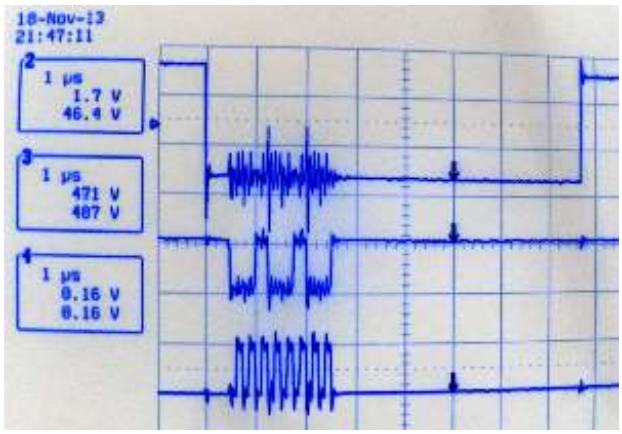
```
  digitalWrite (SS,0) ;
// shiftOut(dataPin, clockPin, bitOrder, value)
  shiftOut(Da, Ck, LSBFIRST, motif) ;
  digitalWrite (SS,1) ;
  delay (1);
}
```

```
  DDRB |= 1<<bSS | 1<<bCk | 1<<bDa ;
}
void loop()  {
  int motif = 0x24 ; //
  SSOff ;
  byte i=0;
  while (i++ < 8) {
    if (motif & 0x01) { DaOn ; }
    else              { DaOff ;}
    CkOn ; CkOff ; // durée 0.25 us
    motif >>= 1;  // détruit motif
  }
  SSOn ;
  delay (1);
}
```

Utiliser les registre SPI du processeur est facile. Des ordres Arduino utilisent des mots explicites, il suffit de lire la documentation.

```
//DecaleSPI.pde|    serie 8 bits sur Arduino
// durée ck 0.25us 4MHz  total 6.5 us
// ck(1)  1Mhz  tot 12.5 us  perdu 4us
// ck(2)  250k      36
// ck(3)  125       68
#include "SPI.h"
#define Da  11  // MOSI
#define Ck  13
#define Ld  10

int  motif ;
void setup ()
{
//  DDRC = 0b00000111; //out
pinMode (Da,1);
pinMode (Ck,1);
pinMode (Ld,1);
SPI.begin();
SPI.setBitOrder(MSBFIRST); // or LSB
SPI.setDataMode(0);
SPI.setClockDivider(0) ;
}
void loop()
{
  motif = 0x24 ;
  digitalWrite (Ld,0) ;
  SPI.transfer(motif);
  digitalWrite (Ld,1) ;
  delayMicroseconds (10);
}
```



En programmant directement les registres SPI, sans passer par des fonctions Arduino, on atteint le temps de transfert minimum. On a gagné sur l'activation et désactivation du signal SS. L'interaction est facile: L'écriture dans le PISO du SPI, appelé SPDR déclenche immédiatement le transfer, et il faut attendre le flag SPIF de fin de transfert.

```
//DecaleSPIjdn.pde| serie 8 bits sur AVR
// ck (0) 0.25us 4MHz  total 4.5 us

#define SpiControl 0b01110000
  // SPIE SPE DORD MSTR CPOL CPHA SPR1 SPR0
#define bDa  3   // MOSI 11
#define bCk  5   // pin 13
#define bLd  2   // SS pin 10

int  motif ;
void setup ()
{
  DDRB |= 1<<bLd | 1<<bCk | 1<<bDa ;
  SPCR = SpiControl ;
}
void loop()
{
  motif = 0x24 ;
  bitClear (PORTB,bLd);
  SPDR = motif;
  while (!(SPSR & 1<<SPIF)) {}
  bitSet (PORTB,bLd);
  delayMicroseconds (10);
}
```