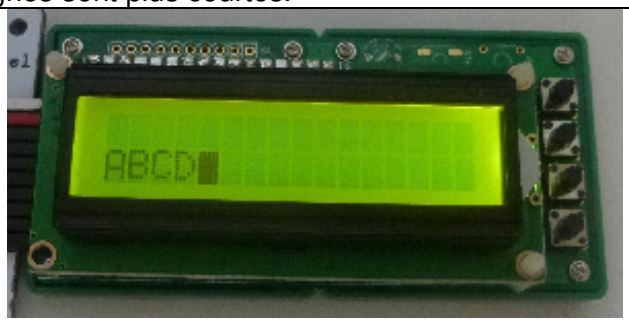
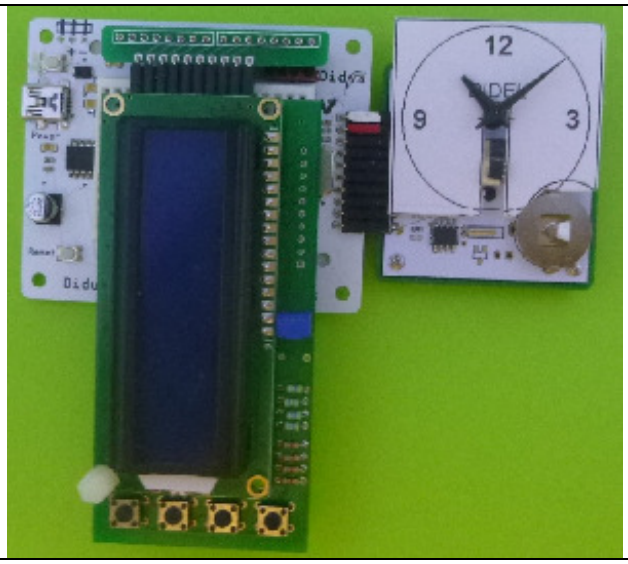
Af2x16 - Microdule-Kidule affichage 2 lignes de 16 caractères

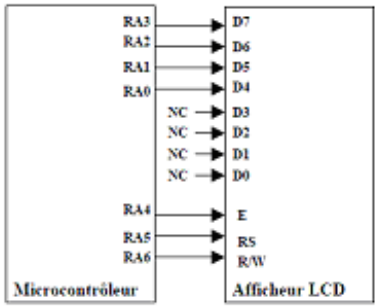
Pour mettre au point une application et pour afficher les valeurs des capteurs et régler les paramètres de l'application (par exemple l'évitement d'obstacles d'un robot) un affichage alphanumérique est très utile. L'affichage se branche sur le connecteur Kidule. Une variante logicielle permet de se connecter sur le port K2 qui utilise les pins 4 à 11. Le logiciel est compatible avec les afficheurs 2x8, mais les lignes sont plus courtes.





Les signaux sur le port K commandent directement les signaux du contrôleur Hitachi HD44780, utilisé dans tous ces types d'affichages. Quatre poussoirs sont multiplexés avec l'affichage. Plusieurs librairies Arduino sont accessibles sur le Web. Didel a sa propre librairie, qui ne supporte que les applications simples d'affichages de nombres et textes, mais cette librairie est simple et peut être facilement complétée pour exploiter les autres ressources du circuit, qui est le Hitachi HD44780 https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

Le circuit est utilisé en mode 4 bits, les données 8 bits sont multiplexées sur les 4 lignes de poids fort. 3 lignes de commande définissent la lecture et écriture des commandes et des caractères affichés.
Un bit commande le rétro-éclairage.



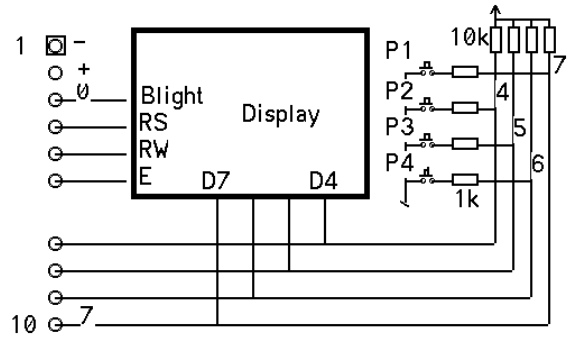
Pin Kidule	
1	Gnd
2	+5V
3 PortC bit0	BackLight
4	RS
5	RW
6	E
7 PortC bit4	Data 4
8 PortC bit5	Data 5
9 PortD bit2	Data 6
10PortD bit3	Data 7

Définitions

Le fichier définition doit déclarer le câblage
- signaux BackLight RS RW E sur les pins 0-3 de Ki, portC bits 0-3
- signaux Data 4-7 sur les pins 4-7 de Ki, portC bits 4,5 et port D bits 2,3

Poussoirs

Quatre poussoirs sont multiplexés avec les data. Des résistances empêchent les poussoirs d'interagir avec l'écriture sur l'affichage.
Les pousoirs sont nommés P1 P2 P3 P4 en commençant par le haut. Ils sont câblés sur les bits 7,4,5,6 du port Kidule, et ces 4 bits sont répartis sur 2 ports du processeur Avr328 du Diduino.



Le fichier de définition peut maintenant être écrit et testé avant d'être encapsulé dans le fichier inclus `KiAf2x16Def.h`. Tous les programmes ses trouvent sous www.didel.com/kidules/CKiAf2x16.zip

```
// KiAf2x15Def.h
#include <Arduino.h>

#define bLed 5 //PortB Arduino pin 13
#define LedOn bitSet (PORTB,bLed)
#define LedOff bitClear (PORTB,bLed)
#define LedToggle PORTB ^= 1 << bLed ;
// Câblage  BackLight RS RW En
#define bBackL 0 // port C
#define BackLightOff bitSet (PORTC,bBackL)
#define BackLightOn bitClear (PORTC,bBackL)
#define bRS 1
#define bRW 2
#define bEn 3
#define ModeData bitSet (PORTC,bRS)
#define ModeControl bitClear (PORTC,bRS)
#define ModeRead bitSet (PORTC,bRW)
#define ModeWrite bitClear (PORTC,bRW)

#define bPous1 3 //PortD top pin10 Ki8
#define bPous2 4 //PortC
#define bPous3 5 //PortC
#define bPous4 2 //PortD bot
#define Pous1On !(PIND & 1<<bPous1)
#define Pous2On !(PINC & 1<<bPous2)
#define Pous3On !(PINC & 1<<bPous3)
#define Pous4On !(PIND & 1<<bPous4)
```

```
 void KiModeOut () {
  DDRB |= 0b00100000 ; // led 13 out
  DDRC  = 0b00111111;  // bits 0-5
  DDRD  |= 0b00001100 ;  // bits 23>67
  PORTC = 0x00 ; // bits ctrl = 0
}
void KiModeIn () {
  DDRB |=  0b00100000 ; // led 13 out
  DDRC  =  0b00001111;  // bits 45 in
  DDRD &= ~0b00001100 ;  // bits 23>67
  delay (1);  // attente charge capa
}
void KiModeInOut () {  //0-3 out  4-7 in lect pouss
   DDRC  = 0b00001111 ; // bits  0-3 out 4 in
   DDRD &= 0b11110011 ; // bits 5 6 in =3 2
}

void KiWrite (int kk) {
  PORTC = kk;
  PORTD |= (kk>>4)& 0x0C; // force les 1
  PORTD &= (kk>>4)|~0x0C; // force les 0
}
byte KiRead () {  // lit 8 bits
  byte dd ;  // data to be read
  dd = PINC & 0x3F ; // keep 6 bits
  dd |= (PIND & 0x0C) << 4 ;  // add after a 4-bit shift
  return dd ;
}
```

Un premier programme de test vérifie que les définitions des poussoirs sont correctes.
On a remarqué dans la fonction KiModeIn() un délai de 1ms qui permet à l'entrée de prendre son état. C'est une pull-up de 10 kOhm qui fait monter les signaux des poussoirs inactifs; cela prend quelques microsecondes.

```
//KiAf2x16Pous.ino test poussoirs
#include "KiAf2x16.h"
void setup () {
  KiModeOut () ; // bits 0123=0
}
void loop () {
  KiModeIn();
  if (Pous1On) LedToggle;
  if (Pous2On) LedOn;
  if (Pous3On) LedOff;
  delay (100);
}
```

Ce qui nous intéressera dans une application, c'est une fonction `GetPous ();` qui rend un nombre, 0,1,2,3,4 selon la touche appuyée.

```
KiAf2x16GetPous.ino
#include "KiAf2x16.h"

byte GetPous () {
  byte pp;
  byte tmp = ~KiRead(); // bits 7,4,5,6
  tmp &= 0xF0;
  if      (tmp & 1<<7) pp= 1;
  else if (tmp & 1<<4) pp= 2;
  else if (tmp & 1<<5) pp= 3;
  else if (tmp & 1<<6) pp= 4;
  else pp=0;
  return pp;
}
```

```
void setup () {
  KiModeOut () ;
}

void loop () {
  KiModeIn();
  Npous = GetPous ();
  for (byte i=0; i<2*Npous;i++) {
    LedToggle;
    delay (100);
  }
  delay (1000);
}
```

Initialisation du contrôleur Hitachi HD44780

L'initialisation doit respecter exactement la documentation, elle met l'affichage dans un mode avec le pointeur invisible au début de l'écran. Le bit RS sélectionne les commandes et l'écriture de textes, RW la lecture et l'écriture, mais on ne relira jamais; on sait ce que l'on a écrit et on veut écrire autre chose. Le bit E synchronise le transfert. Les premiers transferts sont spéciaux, car le circuit ne sait pas encore si on va utiliser le mode 8 ou 16 bits.

```
//KiAf2x16Lcd.h   Definitions et procedures pour
// l'affichage 2x16 Hitachi sur port Kidule

#define Reset 0x30
#define Control 0x20
#define Function 0x28
#define DisOnNC 0x0C       // init sans
curseur
#define Shift 0x14    // cursor right
#define Clear 0x01
#define Home 0x02
```

```
byte ReadChar (){
  byte cc ;
  KiModeInOut () ;
  WaitNoBusy;
  EnPulse ;
  cc =  (KiRead () & 0xF0) ;
  EnPulse ;
  cc |= ((KiRead()>>4) & 0x0F) ;
  KiModeOut () ;
  return cc ;
}
```

```
#define EntryMode 0x06

#define NoCursor  0x0C
#define Cursor  0x0D
#define CursorBlink 0x0F

void EnPulse () { // duree 1 us
  bitSet (PORTC,3);
  //bitSet (PORTC,3);  // ajoute 125ns
  delayMicroseconds (1);
  bitClear (PORTC,3);
}

void WaitNoBusy () {
  delayMicroseconds (50);
}
void WriteControl (byte cc) { // also set function
  byte tt;
  WaitNoBusy ();
  tt = (cc&0xF0) ; // |(PORTC&0x0F) ;
  KiWrite (tt);
  EnPulse ();
  tt = ((cc&0x0F)<<4) ; //| (PORTC&0x0F) ;
  KiWrite (tt);
  EnPulse ();
}
void WriteChar (byte cc) {
  byte tt;
  WaitNoBusy ();
  tt = (cc&0xF0) | 2 ; //(PORTC&0x0F) ;
  KiWrite (tt);
  EnPulse ();
  tt = ((cc&0x0F)<<4) | 2 ; //(PORTC&0x0F) ;
  KiWrite (tt);
  EnPulse ();
}
```

```
// first line address 0..0x0F  second line address 0x40 .. 0x4F
void SetAddress (byte aa)  {
  WriteControl (0x80 + aa) ;
}
void LcdClear ()  {
  WriteControl (0x01) ;
  delay (30) ; // busy flag not supported
}
void CursorHome ()  {  // cursor and pointer
  WriteControl (0x02) ;
}
void CursorOn ()  {   // noblink 0D
  WriteControl (0x0F) ;
}

void LcdInit ()  {  // pour le setup
  delay (30);  KiWrite (Reset) ; // 3 resets!
  delay (60);  KiWrite (Reset) ;
  delay (30);  KiWrite (Reset) ;
  delay (30);  KiWrite (Control) ;
  delay (30);  WriteControl (Function) ;
// 8 bits, 2 rows, 5 x 7 dots
  delay (30);  WriteControl (DisOnNC) ;
  // display off,cursor off,no blink
  delay (30);  WriteControl (Shift) ;  // right
  delay (30);  WriteControl (Home) ;  // home
  delay (30);  WriteControl (Clear) ;  // clear
  delay (30);  WriteControl (EntryMode) ;
// entry mode - auto-increment, shift cursor
  delay (30);  WriteControl (0x80) ;
// ready to write data 1st position
}
```

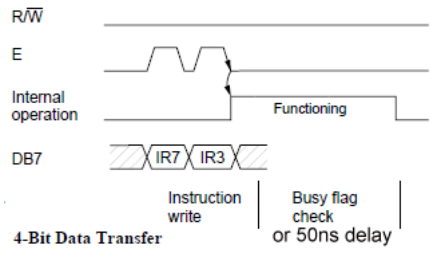
Après l'initialisation, on utilise essentiellement
Les procédures `LcdClear () et CursorHome ()`
La procédure `WriteChar (codeAscii) ;` écrit le caractère ('A' ou 0x41) et fait avancer le pointeur.
La procédure `WriteControl (commande);` pour les différents modes et positionner le curseur.
La procédure `SetAddresse (position)` doit tenir compte des adresses un peu particulières

Le premier caractère de la 2e ligne est en 40 (décimal).
Si on écrit trop de caractères sur une ligne, ils sont invisibles.

Display position	1	2	3	4	5	...	14	15
DDRAM address (hexadecimal)	00	01	02	03	04	...	14	15
	40	41	42	43	44	...	54	55

Commande

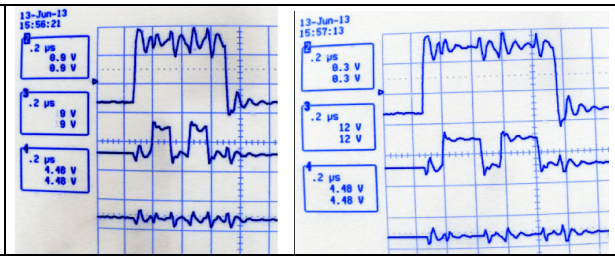
Les caractères et mots de 8 bits sont coupés et transmis avec 2 impulsions E. Le transfert interne prend un certain temps et un flag "Buzy" est activé pendant qu'un nouveau transfert est interdit. Le temps est inférieur à 50 ns et gagner quelques nanosecondes à ce niveau apporte peu. Dans les procédures WriteCommand et WriteChar, ce délai est inclus.
Pour les impulsions E notre souci était de ne pas écrire d'instructions inutiles.




```
void EnPulse ()  // durée pulse 0.125 us
{
  bitSet (PORTC,bEn);
  bitClear (PORTC,bEn);
}
```

Photo 1

Le programme de test appelle 2 fois de suite.
Une impulsion supplémentaire synchronise l'oscillo.
L'impulsion est trop courte (min 230 ns).



```
void EnPulse ()
{
  bitSet (PORTC,bEn);
  bitSet (PORTC,bEn);  // ajoute 0.25 us
  bitClear (PORTC,bEn);
}
```

Photo2

Write Operation — **Bus Timing Characteristics**

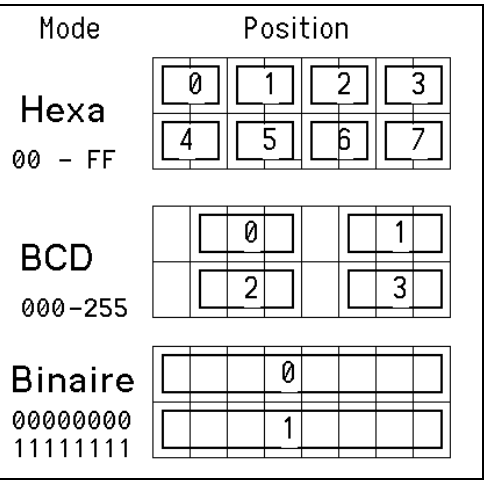
Item	Symbol	Min	Typ	Max	Unit
Enable cycle time	t_{cycE}	500	—	—	ns
Enable pulse width (high level)	PW_{EH}	230	—	—	
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	20	
Address set-up time (RS, R/$\overline{W}$ to E)	t_{AS}	40	—	—	
Address hold time	t_{AH}	10	—	—	
Data set-up time	t_{DSW}	80	—	—	
Data hold time	t_{H}	10	—	—	

Le logiciel est modulaire et importe deux librairies. Le fichier Lcd2x16.h contient les fonctions citées précédemment. Le fichier PortK.h transfère les mots 8 bits pour l'interface sur le PortK. Si on dispose d'un port complet ces fonctions sont triviales. Si on doit répartir les bits sur plusieurs ports, on a un bons exemples comment faire.
Les fonctions utilisées sont KiModeOut KiModeIn KiWrite
Le programme des test `KiAf2x16Af1.ino` montre comment écrire quelques caractères.

Comme exercice, on écrira les fonctions pour disposer des variables sur l'affichage

Pour afficher des nombres, voir
www.didel.com/C/AfficheNombres.pdf

Si on doit convertir de binaire en décimal ou inverse, voir
www.didel.com/C/ConvBCD.pdf.



D'autres explications, des librairies, se trouvent en cherchant ''LCD display'' sur le web.
En français il y a entre autre
http://gilles.berthome.free.fr/02-Syntheses/E-Conversion_electriques_physiques/Synthese_afficheur_LCD.pdf

jdn 130625/140115

2.2. Instruction set

Table 2.3. HD44780 instruction set

Instruction	Code										Description	Execution time**
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position (address 0).	1.64mS
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position (address 0). Also returns display being shifted to the original position. DDRAM contents remains unchanged.	1.64mS
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D), specifies to shift the display (S). These operations are performed during data read/write.	40uS
Display On/Off control	0	0	0	0	0	0	1	D	C	B	Sets On/Off of all display (D), cursor On/Off (C) and blink of cursor position character (B).	40uS
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM contents remains unchanged.	40uS
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N) and character font(F).	40uS
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address. CGRAM data is sent and received after this setting.	40uS
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address. DDRAM data is sent and received after this setting.	40uS
Read busy-flag and address counter	0	1	BF	CGRAM / DDRAM address							Reads Busy-flag (BF) indicating internal operation is being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).	0uS
Write to CGRAM or DDRAM	1	0	write data								Writes data to CGRAM or DDRAM.	40uS
Read from CGRAM or DDRAM	1	1	read data								Reads data from CGRAM or DDRAM.	40uS

Remarks:
- DDRAM = Display Data RAM.
- CGRAM = Character Generator RAM.
- DDRAM address corresponds to cursor position.
- * = Don't care.
- ** = Based on F_{osc} = 250kHz.

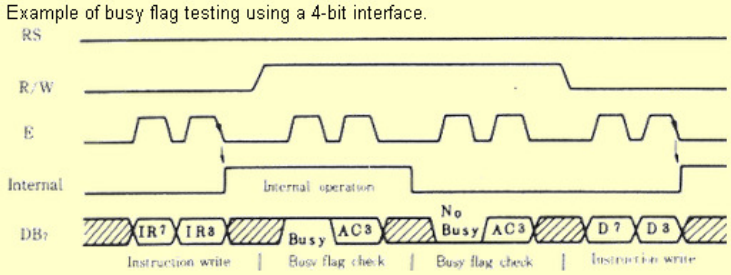
Table 2.4. Bit names

Bit name	Setting / Status	
I/D	0 = Decrement cursor position	1 = Increment cursor position
S	0 = No display shift	1 = Display shift
D	0 = Display off	1 = Display on
C	0 = Cursor off	1 = Cursor on
B	0 = Cursor blink off	1 = Cursor blink on
S/C	0 = Move cursor	1 = Shift display
R/L	0 = Shift left	1 = Shift right
DL	0 = 4-bit interface	1 = 8-bit interface
N	0 = 1/8 or 1/11 Duty (1 line)	1 = 1/16 Duty (2 lines)
F	0 = 5x7 dots	1 = 5x10 dots
BF	0 = Can accept instruction	1 = Internal operation in progress

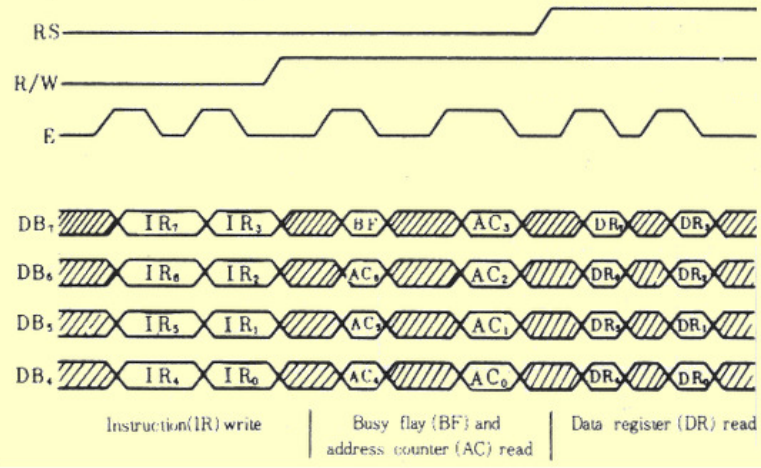
TOC

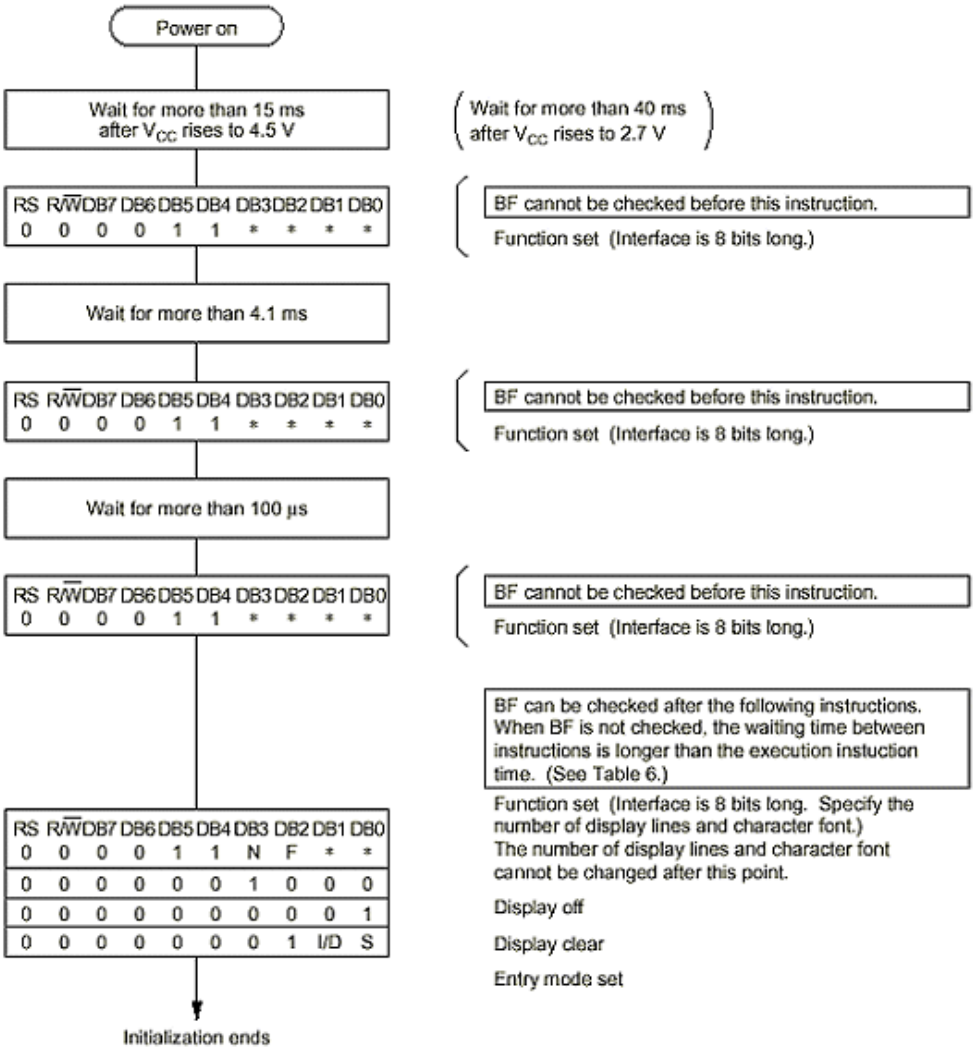
2.4.2. 4-bit interface

Example of busy flag testing using a 4-bit interface.



Example of data transfer using a 4-bit interface.




Power on
Wait for more than 15 ms after V_{CC} rises to 4.5 V
Wait for more than 40 ms after V_{CC} rises to 2.7 V
RS R/$\overline{W}$ DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 * * * *
BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)
Wait for more than 4.1 ms
RS R/$\overline{W}$ DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 * * * *
BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)
Wait for more than 100 µs
RS R/$\overline{W}$ DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 * * * *
BF cannot be checked before this instruction.
Function set (Interface is 8 bits long.)
BF can be checked after the following instructions.
When BF is not checked, the waiting time between instructions is longer than the execution instuction time. (See Table 6.)
RS R/$\overline{W}$ DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 0 1 1 N F * *
Function set (Interface is 8 bits long. Specify the number of display lines and character font.)
The number of display lines and character font cannot be changed after this point.
0 0 0 0 0 0 1 0 0 0
Display off
0 0 0 0 0 0 0 0 0 1
Display clear
0 0 0 0 0 0 0 1 I/D S
Entry mode set
Initialization ends