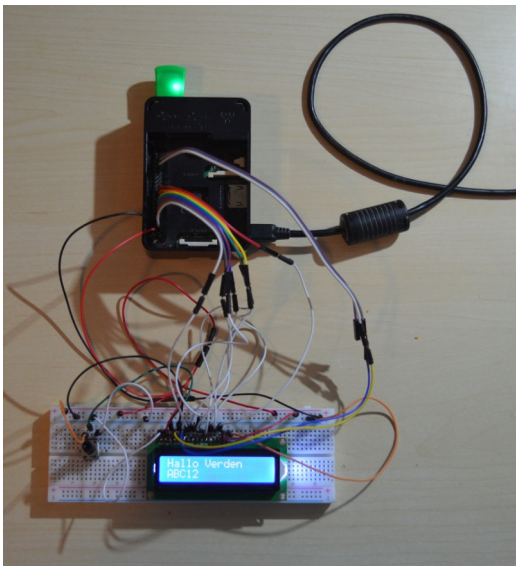
Test af 16x2 display

På en Raspberry Pi



Oversigt: Jeg beskriver i korte træk hvad displayet kan og hvordan man får det til det, hvilket demonstreres med et C program og en Raspberry Pi.

Denne og andre kan findes på: https://synkro.dk/bog
DB: Henrik Kressner
Dec 2020

Table of Contents

1. Indledning

Dette er mine private notater omkring 16x2 display, alle er velkommen til at lade sig inspirere.

Displayet er udstyret med en Hitachi HD44780U, der udgør interfacet mellem selve displayet og processoren. I HD44780U er der en karaktergenerator, så man skal ikke tænke på at generere bitmønstre, men man kan manipulere med karaktergenerator, hvis man har lyst til det.

Jeg har ikke eksperimenteret med karaktergeneratoren.

Testprogram (kildetekst) er vist i Bilag A

Henrik Kressner
December 2020
og senere rettelser

2. Konstruktionen

Displayet leveres på et print med 16 bens interface, jeg har forbundet dem således til en RPi.

Pin display	Pin RPi	Kommentar
1	6	Ground
2	2	Vcc (5V)
3	-	Kontrast (Til Ground, eller variabel modstand fra Vcc til Ground)
4	GPIO 13	R/S. R/S=0: Kommando - RS=1: Data
5	Ground !!!	Read/Write 1/0: **PAS PÅ** med Vcc = 5V må den ikke sende data til en RPi
6	GPIO 19	Clock. Der skiftes over når den går fra high til low.
7	GPIO 2	Databit 0
8	GPIO 3	Databit 1
9	GPIO 4	Databit 2
10	GPIO 17	Databit 3
11	GPIO 27	Databit 4
12	GPIO 22	Databit 5
13	GPIO 10	Databit 6
14	GPIO 9	Databit 7
15	Vcc	Anode (Baggrundslys) Både anode og katode skal tilsluttes for at tænde.
16	Ground	Katode (bagrundslys)

I denne konstruktion kan der kun sendes data til displayet, fordi displayet ikke kan lyse nok op når der vælges Vcc = 3,3V. Det betyder vi brænder GPIO af, hvis vi skriver fra display til RPi når den kører Vcc=5V. Derfor er R/W lagt til ground, så går det ikke galt, og gør testen nemmere.

Det eneste tidspunkt jeg har brug for at læse fra displayet, er for at se om det er optaget internt. Ved at holde pauser som oplyst i manualen, kan det ignoreres. Kan man ikke leve med pauserne, må man sikre output på datapindene ikke overstiger 3,3V, hvis man vil arbejde med en RPi.

På en Arduino Uno har jeg ladet det drive med 3,3V, der var guf nok i den til at den kunne lyse op med fin kontrast, uden en variabel modstand til at justere kontrasten. Den metode lever dog ikke op til spændingskravet i manualen.

Ellers er den nem at have med at gøre. Man sætter R/S alt efter om man vil sende en kommando til displayet, eller om man vil sende data der skal vises i displayet. Kommando/data sendes til eksekvering når man lader clocken gå fra high til low.

3. Initialisering

Hvis enheden starter korrekt op, som beskrevet i manualen side 23, behøver man ikke tænke på initialisering. Det ser ud til at displayet der er fast forbundet med Rpi'en i min test, starter korrekt op når RPi'en starter op.

Jeg har alligevel skrevet en initialseringsfunktion, det er blot manualen side 45 (Initializing by Instruction), skrevet i C.

```
void Reset()
{
	Tx(0x30,0);
	usleep(4500);
	Tx(0x30,0);
	usleep(4500);
	Tx(0x30,0);
	usleep(4500);

	// Display 2 linier, 8 bit, 5x8
	Tx(0x38,0);
	usleep(4500);
	Tx(0x08,0);	// Display off
	usleep(4500);
	Tx(0x01,0);	// Display Clear
	usleep(4500);
	Tx(0x05,0);	// Entry Mode Set
	usleep(4500);
}
```

Tabel 3.1

Herefter kan der sættes bogstav A på displayet linie 1 ved at skrive:

```
Tx(0xC0,0);
Tx('A',1);
```

Eller på linie 2 ved at skrive:

```
Tx(0x80,0);
Tx('A',1);
```

Bemærk ettallet som parameter på den anden Tx, det betyder vi sender data.

Displayet kan noget mere, som flyt kurser, blink, og bruges som supplerende hukommelse, læs manualen.

Bilag A Kildetekst

```
//  Use a RPi 2 med 16x2 LCD
//  Done by Henrik Kressner 2020
//  This is free software, get inspired and use at your own risc.

// ToDo: Nothing

//  To compile (must be root) :  cc 16x2.c -o 16x2
//  To run (must be root): ./16x2

#define VERSION "0.90"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
#include <time.h>
#include <math.h>



//  Access from ARM Running Linux
//  For Raspberry Pi 3 and later, change BCM2708_PERI_BASE to 0x3F000000
//  My RPi 2 runs with 0x200000 I think RPI3 should run at 0x3F000000
#define BCM2708_PERI_BASE       0x20000000
#define GPIO_BASE               (BCM2708_PERI_BASE + 0x200000) /* GPIO controller */
#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

# define RS 13          // Register Select: 0 = Write 1 = Data
# define RW 26          // Read/Write: Read = 1 Write = 0
# define CE 19          // Clock: Skifter på bagkant
# define D0 2
# define D1 3
# define D2 4
# define D3 17
# define D4 27
# define D5 22
# define D6 10
# define D7 9


int  mem_fd;
char *gpio_mem, *gpio_map;
char *spi0_mem, *spi0_map;
```

```
// I/O access
volatile unsigned *gpio;
// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x)
// or SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0
#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH


void setup_io()
{
   /* open /dev/mem */
   if ((mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
      printf("can't open /dev/mem \n");
      exit (-1);
   }
   /* mmap GPIO */
   // Allocate MAP block
   if ((gpio_mem = malloc(BLOCK_SIZE + (PAGE_SIZE-1))) == NULL) {
      printf("allocation error \n");
      exit (-1);
   }
   // Make sure pointer is on 4K boundary
   if ((unsigned long)gpio_mem % PAGE_SIZE)
     gpio_mem += PAGE_SIZE - ((unsigned long)gpio_mem % PAGE_SIZE);
   // Now map it
   gpio_map = (unsigned char *)mmap(
      (caddr_t)gpio_mem,
      BLOCK_SIZE,
      PROT_READ|PROT_WRITE,
      MAP_SHARED|MAP_FIXED,
      mem_fd,
      GPIO_BASE
   );

   if ((long)gpio_map < 0) {
      printf("mmap error %d\n", (int)gpio_map);
      exit (-1);
   }
   // Always use volatile pointer!
   gpio = (volatile unsigned *)gpio_map;
} // setup_io
```

```
void help()
{
       printf("*****************************\n");
          printf("\n16x2 display tester %s\n", VERSION);
          printf("DB Henrik Kressner 2020.\n\n");
          printf("*****************************\n");
          printf("");
          printf("");
          printf("");
          printf("");
}

void TikTak()
{
          GPIO_CLR = 1 << CE;
          usleep(1);
          GPIO_SET = 1 << CE;
          usleep(1);
          GPIO_CLR = 1 << CE;
          usleep(50);
}


void Rx(char *data)
// Husk tilpasning af spaending hvis input skal bruges på en RPi.
{
}
```

```
void Tx(char data, int kommando)
{
        GPIO_CLR = 1<<RW;
        if (kommando)
                GPIO_SET = 1<<RS;
        else
                GPIO_CLR = 1<<RS;

      GPIO_CLR = 1<<D0;
      GPIO_CLR = 1<<D1;
      GPIO_CLR = 1<<D2;
      GPIO_CLR = 1<<D3;
      GPIO_CLR = 1<<D4;
      GPIO_CLR = 1<<D5;
      GPIO_CLR = 1<<D6;
      GPIO_CLR = 1<<D7;

      if (data & 01)
                GPIO_SET = 1<<D0;
      if (data & 0x2)
            GPIO_SET = 1<<D1;
      if (data & 0x4)
            GPIO_SET = 1<<D2;
      if (data & 0x8)
            GPIO_SET = 1<<D3;
      if (data & 0x10)
            GPIO_SET = 1<<D4;
      if (data & 0x20)
            GPIO_SET = 1<<D5;
      if (data & 0x40)
            GPIO_SET = 1<<D6;
      if (data & 0x80)
            GPIO_SET = 1<<D7;

      TikTak();
}
```

```
void ClrLine(char line)
{
        int i;

        if (line)
            // Set Data Addr 2. linie
            Tx(0xC0,0);
        else
           // Set Data Addr 1. linie
           Tx(0x80,0);
        for (i=1;i<=16;i++)
                Tx(' ',1);

}

void WriteLn(char *line, char lineNo)
{
        int i = 0;;

        if (lineNo)
            Tx(0xC0,0);                 // Set Data Addr 2. linie
        else
                Tx(0x80,0);
        while (line[i] != '\0')   // Set Data Addr 1. linie
            Tx(line[i++],1);
}



void Reset()
{
      Tx(0x30,0);
      usleep(4500);
      Tx(0x30,0);
      usleep(4500);
      Tx(0x30,0);
      usleep(4500);

      // Display 2 linier, 8 bit, 5x8
      Tx(0x38,0);
      usleep(4500);
      Tx(0x08,0);       // Display off
      usleep(4500);
      Tx(0x01,0);       // Display Clear
      usleep(4500);
      Tx(0x05,0);     // Entry Mode Set
      usleep(4500);
}
```

```
void init()
{
	INP_GPIO(RS);
	OUT_GPIO(RS);
	INP_GPIO(RW);
	OUT_GPIO(RW);
	INP_GPIO(CE);
	OUT_GPIO(CE);

	INP_GPIO(D0);
	OUT_GPIO(D0);
	INP_GPIO(D1);
	OUT_GPIO(D1);
	INP_GPIO(D2);
	OUT_GPIO(D2);
	INP_GPIO(D3);
	OUT_GPIO(D3);
	INP_GPIO(D4);
	OUT_GPIO(D4);
	INP_GPIO(D5);
	OUT_GPIO(D5);
	INP_GPIO(D6);
	OUT_GPIO(D6);
	INP_GPIO(D7);
	OUT_GPIO(D7);

}
```

```
void main(int a, char *s[])
{
	char tmp[] = "ABC12";

	help();
	setup_io();
	init();
	switch (a)	// Nice to have
	{
		case 1: printf("0 %s\n",s[a-1]); break;
	    case 2: printf("1 %s\n",s[a-1]); break;
	    case 3: printf("2 %s\n",s[a-1]); break;
	    case 4: printf("3 %s\n",s[a-1]); break;
	}
	Reset();

	// Display & cursoer on
	Tx(0x0E,0);

	// Entry mode set ?
	Tx(0x06,0);

	// Curser off
	Tx(0x0C,0);
	sleep(1);
	ClrLine(0);
	ClrLine(1);

	WriteLn("Hallo Verden",0);
	WriteLn(tmp,1);

	printf("Slut\n");

}
```