Introduction to the H8(/3048)

Jean-Michel FRIEDT, June 2001-August 22, 2001

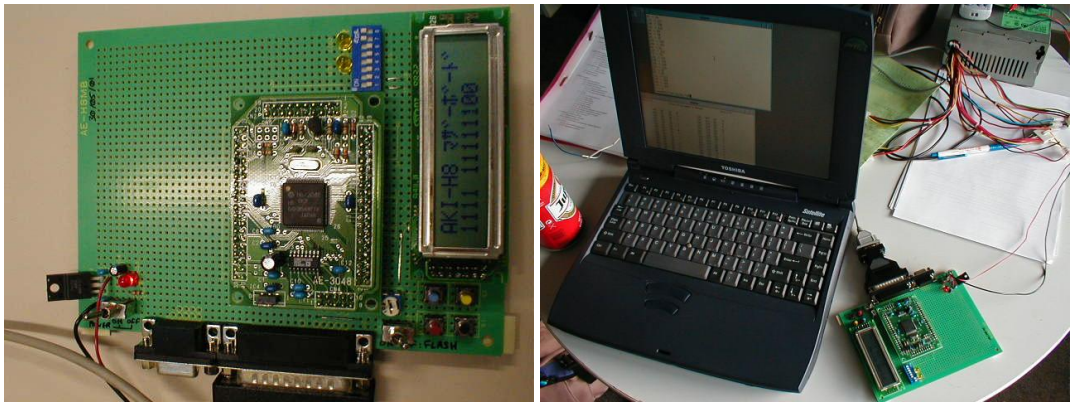
1 Introduction

After spending about 2 years developing circuits around the 68HC11(F1), one reaches the limits of this microcontroller and wishes to extend to a more powerful circuit. The Hitachi H8 offers a similar way of testing programs than the bootstrap mode of the 68HC11 (simply using a RS-232 cable and a MAX232 level converter) allowing low cost development as no specific programmer is required. Additionally, the former's hardware is much more developed, including two RS232 port, two 8 bits D/A converters and eight 10 bits A/D converters as well as much more RAM and EEPROM (in this case flash) memory.

GNU tools for compiling software are available, and will here be used under Linux. All developments and test programs described in this document were developed under Linux kernel 2.2.17 based on a Debian 2.1 distribution regularly upgraded (until June 2001).

2 Development tools

The hardware used for the applications described in this document is based on the Aki H8 board sold by the Akizuke shop in Akihabara, Tokyo, Japan (`www.akizuke.ne.jp`). It includes an H8/3048 based board with an RS-232/TTL levels converter chip, a voltage converter and all the passive components for running the CPU. An additional board is available for programming the flash memory, connecting a 2×16 characters LCD display and various LEDs and switches to the main board. Soldering is easy and well explained in the manual, even with little or no understanding of Japanese.



From the software point of view, all development were done using the gcc cross-compiler to the Hitachi H8300 series. Compilation of these tools requires the following packages with at least the following version number: `binutils-2.8.1`, `gcc-2.7.2.3`, newlib-1.8.1.tar.gz and eventually `crossgcc-gcc-2.7.2.3.patch` (which should no longer be necessary with newer versions of gcc). I found the program `h8comm` to be very instructive in learning how to write basic software for the H8 and how to communicate with it. Writing a new communication program was however necessary in order to be able send test-programs to RAM rather than to flash memory (which only has a limited life time of about 100 writings).

Compiling gcc and accompanying packages is well described by `hoso@ce.mbn.or.jp`: for each new package (.tar.gz file) run `./configure --target=h8300-hms --prefix=/usr/local/cross` followed by the usual make procedure (`make;make install`).

06-26-01: it appears that the following examples can be compiled with only the tools provided by `binutils-2.8.1`, as demonstrated by adding the `-v` flag to the `gcc` command line presented in chapter 4. Compilation can be done using only `as`, `ld` and `objcopy` (all of them being provided by `binutils`) using the following commands:

```
/usr/local/h8300-hms/bin/as -o $1.obj $1.S
/usr/local/h8300-hms/bin/ld -m h8300h -o $1 -T h8ram.x $1.obj
/usr/local/bin/h8300-hms-objcopy -S -O binary -R .stack $1 $1.bin
```

The resulting file `$1.bin` is the binary data to sent to the RAM of the H8.

3 Transferring a program to the H8/3048

A program, `h8comm`, was written by Muneyuki Iwata (kp9m-iwt@asahi-net.or.jp) for programming the flash memory of the H8 from Linux. Although this was an invaluable source of inspiration and knowledge, a new program was written for simply sending the binary file containing the list of opcodes of the program to be tested on the H8 to the board, without accessing the flash memory itself.

The program includes the basic RS-232 port initialization (all parameters are defined in `rs232.h`) as well as the basic communication between the PC and the H8 (`h8.c`). Compilation is done by `gcc -c rs232.c; gcc -o h8 h8.c rs232.o`:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>                 /* declaration of bzero() */
#include <fcntl.h>
#include <termios.h>
```

```
int init_rs232();
void free_rs232();
void sendcmd(int,char*);
struct termios oldtio,newtio;

// #define BAUDRATE B2400
#define BAUDRATE B9600
#define HC11DEVICE "/dev/ttyS0"
```

The header `rs232.h`

This header mainly defines which serial port is used (COM1 under DOS or /dev/ttyS0 under linux) and the communication speed (9600 baud can be used here as our H8 is clocked by a 16 MHz resonator).

This program is used by calling `./h8 program.bin` (the serial port to which the board is connected is defined in the RS232 header file). We will see later (chap. 4) how to generate `program.bin`.

```
/* All examples have been derived from miniterm.c                        */
/* Don't forget to give the appropriate serial ports the right permissions */
/* (e. g.: chmod a+rw /dev/ttyS0)                                        */

#include "rs232.h"

extern struct termios oldtio,newtio;

int init_rs232()
{int fd;
 fd=open(HC11DEVICE, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(HC11DEVICE); exit(-1); }
 tcgetattr(fd,&oldtio); /* save current serial port settings */
 bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings */
// newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
 newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;  /* _no_ CRTSCTS */
 newtio.c_iflag = IGNPAR | ICRNL |IXON;
 newtio.c_oflag = ONOCR|ONLRET|OLCUC;
// newtio.c_lflag = ICANON;
// newtio.c_cc[VINTR]    = 0;     /* Ctrl-c */
// newtio.c_cc[VQUIT]    = 0;     /* Ctrl-\ */
// newtio.c_cc[VERASE]   = 0;     /* del */
// newtio.c_cc[VKILL]    = 0;     /* @ */
// newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
 newtio.c_cc[VTIME]    = 0;     /* inter-character timer unused */
 newtio.c_cc[VMIN]     = 1;     /* blocking read until 1 character arrives */
```

```
// newtio.c_cc[VSWTC]    = 0;     /* '\0' */
// newtio.c_cc[VSTART]   = 0;     /* Ctrl-q */
// newtio.c_cc[VSTOP]    = 0;     /* Ctrl-s */
// newtio.c_cc[VSUSP]    = 0;     /* Ctrl-z */
// newtio.c_cc[VEOL]     = 0;     /* '\0' */
// newtio.c_cc[VREPRINT] = 0;     /* Ctrl-r */
// newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
// newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
// newtio.c_cc[VLNEXT]   = 0;     /* Ctrl-v */
// newtio.c_cc[VEOL2]    = 0;     /* '\0' */
 tcflush(fd, TCIFLUSH);tcsetattr(fd,TCSANOW,&newtio);
// printf("RS232 Initialization done\n");
 return(fd);
}

void sendcmd(int fd,char *buf)
{unsigned int i,j;
 if((write(fd,buf,strlen(buf)))<strlen(buf))
    {printf("\n No connection...\n");exit(-1);}
 for (j=0;j<5;j++) for (i=0;i<3993768;i++) {}
 /* usleep(attente); */
}

void free_rs232(int fd)
{tcsetattr(fd,TCSANOW,&oldtio);close(fd);} /* restore the old port settings */
```

The RS232 initialization routines `rs232.c`

```
#include "rs232.h"
#include <fcntl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>


void send_h8(FILE *f,int fd)
{unsigned char buf;char prg[3072];int i=0,status;unsigned int j;
 do {status=fscanf(f,"%c",&buf);prg[i]=(char)buf;i++;} /* read while !EOF */
    while ((status!=EOF) && (i<3072));
  if (i<3072) i--;printf("%d bytes read from file\n",i);
  buf=(char)(i/256);write(fd,&buf,1);read(fd,&buf,1);
      printf("%d -> %d \n",i/256,(int)buf&0x000000FF);
  buf=(char)(i%256);write(fd,&buf,1);read(fd,&buf,1);
      printf("%d -> %d \n",i%256,(int)buf&0x000000FF);
  for (j=0;j<i;j++) {write(fd,&prg[j],1);read(fd,&buf,1);
      printf("%d -> %d \n",(int)prg[j]&0x000000FF,(int)buf&0x000000FF);}
}

void init_comm(int fd)
{char buf;int i;
 fd_set rfds;
 struct timeval tv;
 int retval=0;
```

```
  while (!retval) {
    FD_ZERO(&rfds);FD_SET(fd, &rfds);                  /* Watch stdin (fd 0) */
    tv.tv_sec = 1;tv.tv_usec = 00;                    /* Wait up to X seconds. */
    retval = select(fd+1, &rfds, NULL, NULL, &tv);
    retval=FD_ISSET(fd,&rfds);
    if (retval) printf("Data is available now.\n");
        else {printf(".");fflush(stdout);buf=0;
          for (i=0;i<255;i++) write(fd,&buf,1);}
    }
  read(fd,&buf,1);
  printf("received: 0x%x\n",(int)buf&0x000000FF); // read data received from H8
  buf=0x55;write(fd,&buf,1);                       // send acknowledgement
  read(fd,&buf,1);
  printf("received: 0x%x\n",(int)buf&0x000000FF); // read data received from H8
  printf("init finished \n");
}

int main(int argc,char **argv)
{int fd;FILE *f;
 if (argc<2) printf("%s filename\n",argv[0]); else {
    fd=init_rs232();init_comm(fd);
    f=fopen(argv[1],"r");send_h8(f,fd);fclose(f);
 /* free_rs232(); */
    } return(0);
}
```

Communication with the H8 board: `h8.c`

The hardware settings for the H8 to go to bootstrap mode when switched on (instead of executing a user program previously stored in flash memory) is to set MD1=MD2=+12 V while MD0=+5 V (this means closing the S7 switch on the Aki-H8 board). When the H8 boots in this mode, it will first check that no program is stored in flash memory (and will eventually erase any data stored in flash memory if necessary), then listen to the serial port to automatically detect the baud rate (as the host PC is continuously transmitting '0' on its RS232 port) and after a handshake procedure, will save the data coming from the PC's serial port in the H8's internal RAM. Once the transmission is finished, the program will jump to the beginning of the RAM address space and the program to be tested is executed.

4 Compiling an H8 program on a Linux system

Although `gcc` allows compiling full C programs targeted for the H8, and the use of `gdb` as a debugger, we here limit ourselves to writing programs in assembly language and use `gcc` as an assembler and address allocator. This limitation is due to the fact that we only aim at using the RAM for testing programs (3072 bytes available) and not the whole 32 KB flash memory in which the much larger C programs generated by gcc could be stored.

Compilation of the program prg.S is done by running the following commands:

```
h8300-hms-gcc -O -g -mrelax -mh -nostdlib -T h8ram.x prg.S -o prg
h8300-hms-objcopy -S -O binary -R .stack prg prg.bin
```

`prg.bin` is the actual binary file containing the opcodes and data to be sent to the H8 RAM for program execution. Its content can be analyzed using the `xxd` hexadecimal binary file viewer.

The file `h8ram.x` is used for memory allocation and should be as follows:

```
OUTPUT_FORMAT("coff-h8300")
OUTPUT_ARCH(h8300h)
ENTRY("_start")
MEMORY
{
/* 0xc4 is a magic entry.  We should have the linker just
   skip over it one day... */
vectors : o = 0x00000, l = 0xc4
/* vectors : o = 0x0000, l = 0xc4                jmfriedt */
magicvectors : o = 0xc4, l = 0x3c
ram    : o = 0xff300, l = 0x0c00 /* c00=3072 bytes */
/* The stack starts at the top of main ram.  */
topram : o = 0xffefc, l = 0x4
/* At the very to of the address space is the 8-bit area.  */
eight : o = 0xfff00, l = 0x10
}
SECTIONS
{
.vectors : {
/* Use something like this to place a specific function's address
   into the vector table.
SHORT(ABSOLUTE(_foobar)) */
*(.vectors)
        }  > vectors
.text : {
*(.rodata)
*(.text)
*(.strings)
     _etext = . ;
} > ram
.tors : {
___ctors = . ;
*(.ctors)
___ctors_end = . ;
___dtors = . ;
*(.dtors)
___dtors_end = . ;
}  > ram
.data : {
*(.data)
*(.tiny)
 _edata = . ;
} > ram
.bss : {
 _bss_start = . ;
*(.bss)
*(COMMON)
 _end = . ;
}  >ram
.stack : {
 _stack = . ;
*(.stack)
}  > topram
.eight : {
*(.eight)
}  > eight
.stab 0 (NOLOAD) : {
[ .stab ]
}
.stabstr 0 (NOLOAD) : {
[ .stabstr ]
}
}
```

The file `h8ram.x` defining memory allocation by the compiler

Additionally, any program to be run from the RAM of the H8 should start with the following lines:

```
.h8300h        ; H8 300H extended opcodes
.align 1       ; align on 8 bits boundaries
.global _start ; gcc requires a "_start" function
```

and the entry point of the program should be labeled `_start:`.

Considering our use of gcc as an assembler, it should be enough to install `h8300-hms-as` (the GNU assembler) instead of the full gcc packages. However, memory allocation could not be obtained using `as` only and full gcc was always used during development (ld is in fact implicitly called by gcc and re-locates code using the -T option file). See the addendum to section 2 on this subject.

5 Example programs

We here present some of the programs the authors wrote for getting used with the various hardware/software interaction aspects of this microcontroller. No complex algorithmic problem will be developed here, only very simple and short illustrations of each hardware characteristics of the microcontroller.

5.1 Basic I/O

The first basic program to get used to the gcc syntax will simply aim at blinking two LEDs connected to port 5.

```
led1:     file format coff-h8300

Disassembly of section .text:

000ff300 <_start>:
   ff300: 7a 07 00 0f  7a 07 00 0f fe fc mov.l #0xffefc,er7
   ff304: fe fc
   ff306: 07 c0        07 c0             ldc #0xc0,ccr
   ff308: f8 00        f8 00             mov.b #0x0,r0l
   ff30a: 6a a8 00 ff  6a a8 00 ff ff ed mov.b r0l,@0xffffed:32
   ff30e: ff ed
   ff310: f8 ff        f8 ff             mov.b #0xff,r0l
   ff312: 6a a8 00 0f  6a a8 00 0f ff c8 mov.b r0l,@0xfffc8:32
   ff316: ff c8

000ff318 <boucle>:
   ff318: 7f ca 70 00  7f ca 70 00       bset #0x0,@0xca:8
   ff31c: 7f ca 72 10  7f ca 72 10       bclr #0x1,@0xca:8
   ff320: 55 0e        55 0e             bsr .+14 (ff330)
   ff322: 7f ca 72 00  7f ca 72 00       bclr #0x0,@0xca:8
   ff326: 7f ca 70 10  7f ca 70 10       bset #0x1,@0xca:8
   ff32a: 55 04        55 04             bsr .+4 (ff330)
   ff32c: 5a 0f f3 18  5a 0f f3 18       jmp @0xff318:0

000ff330 <DELAI>:
   ff330: 79 04 00 2f  79 04 00 2f       mov.w #0x2f,r4

000ff334 <WAI1>:
   ff334: 79 05 4f ff  79 05 4f ff       mov.w #0x4fff,r5

000ff338 <WAI>:
   ff338: 1b 55        1b 55             dec.w #0x1,r5
   ff33a: 4a fc        4a fc             bpl .-4 (ff338)
   ff33c: 1b 54        1b 54             dec.w #0x1,r4
   ff33e: 4a f4        4a f4             bpl .-12 (ff334)
   ff340: 54 70        54 70             rts
```

`led1`: two blinking LEDs connected to port 5

This is the presentation of a program as it will always be used in this document. It is obtained by `h8300-hms-objdump -S name > name.lst` where the file `name` is the result of the compilation of `name.S` by gcc. In fact `name.S` in this example would like (as typed by the programmer in his favorite text editor):

```
.h8300h
; .section .start
.align 1
.global _start

_start:
       mov.l   #0xffefc,er7 ; =mov.l ...,sp : sett stack pointer
       ldc.b   #0xc0,  ccr             ; disable interrupt
       mov.b   #0x00,  r0l             ; 2-State Access(All area)
       mov.b   r0l,@0xffffed

       mov.b   #0xff,r0l   ; set P5 as output
       mov.b   r0l,@0xfffc8  ; |
boucle:
bset #0,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bclr #1,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bsr DELAI
bclr #0,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bset #1,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bsr DELAI
jmp  boucle

DELAI: mov.w   #0x002f, r4 ; Set program loop counter mais 1f ne marche pas
WAI1: mov.w   #0x4fff, r5 ; Set program loop counter     8fff
WAI: dec.w   #1, r5 ; Program
bpl WAI
dec.w   #1, r4
bpl     WAI1
rts
.end
```

`led1.S`: original source program

5.2 RS-232 communication

```
00000000 <_start>:
   0: 7f ba 70 40  7f ba 70 40       bset #0x4,@0xba:8
   4: 7f ba 70 50  7f ba 70 50       bset #0x5,@0xba:8
   8: 79 00 ff 00  79 00 ff 00       mov.w #0xff00,r0
   c: 6a a0 00 0f  6a a0 00 0f ff d4 mov.b r0h,@0xfffd4:32
  10: ff d4
  12: fa ff        fa ff             mov.b #0xff,r2l

00000014 <snd>:
  14: 7e bc 73 70  7e bc 73 70       btst #0x7,@0xbc:8
  18: 47 fa        47 fa             beq .-6 (14)
  1a: 3a bb        3a bb             mov.b r2l,@0xbb:8
  1c: 7f bc 72 70  7f bc 72 70       bclr #0x7,@0xbc:8
  20: 1a 0a        1a 0a             dec.b r2l
  22: 46 f0        46 f0             bne .-16 (14)
  24: fa ff        fa ff             mov.b #0xff,r2l
  26: 7a 07 00 0f  7a 07 00 0f fe fc mov.l #0xffefc,er7
  2a: fe fc
  2c: 07 c0        07 c0             ldc #0xc0,ccr
  2e: f8 00        f8 00             mov.b #0x0,r0l
  30: 6a a8 00 ff  6a a8 00 ff ff ed mov.b r0l,@0xffffed:32
  34: ff ed
  36: f8 ff        f8 ff             mov.b #0xff,r0l
  38: 6a a8 00 0f  6a a8 00 0f ff c8 mov.b r0l,@0xfffc8:32
  3c: ff c8
  3e: 6a a8 00 0f  6a a8 00 0f ff da mov.b r0l,@0xfffda:32
  42: ff da
  44: f8 00        f8 00             mov.b #0x0,r0l
  46: 6a a8 00 0f  6a a8 00 0f ff c5 mov.b r0l,@0xfffc5:32
  4a: ff c5
  4c: f8 00        f8 00             mov.b #0x0,r0l
  4e: 6a 28 00 0f  6a 28 00 0f ff c7 mov.b @0xfffc7:32,r0l
  52: ff c7
  54: 11 08        11 08             shlr r0l
  56: 11 08        11 08             shlr r0l
  58: 11 08        11 08             shlr r0l
  5a: 11 08        11 08             shlr r0l
  5c: 38 ca        38 ca             mov.b r0l,@0xca:8
  5e: 40 a0        40 a0             bra .-96 (0)
```

`rs_snd`: sends bytes 0xFF to 0x00 to the serial port at 9600 bauds, N81

```
00000000 <_start>:
   0: 7f ba 70 40  7f ba 70 40       bset #0x4,@0xba:8
   4: 7f ba 70 50  7f ba 70 50       bset #0x5,@0xba:8
   8: 79 00 ff 00  79 00 ff 00       mov.w #0xff00,r0
   c: 6a a0 00 0f  6a a0 00 0f ff d4 mov.b r0h,@0xfffd4:32
  10: ff d4

00000012 <rcv>:
  12: 7e bc 73 60  7e bc 73 60       btst #0x6,@0xbc:8
  16: 47 fa        47 fa             beq .-6 (12)
  18: 2a bd        2a bd             mov.b @0xbd:8,r2l
  1a: 7f bc 72 60  7f bc 72 60       bclr #0x6,@0xbc:8

0000001e <snd>:
  1e: 7e bc 73 70  7e bc 73 70       btst #0x7,@0xbc:8
  22: 47 fa        47 fa             beq .-6 (1e)
  24: 3a bb        3a bb             mov.b r2l,@0xbb:8
  26: 7f bc 72 70  7f bc 72 70       bclr #0x7,@0xbc:8
  2a: 40 e6        40 e6             bra .-26 (12)
```

`rs_rcv`: echoes on the serial port the characters received on this same port from the host PC (9600, N81 communication speed).

5.3 A/D and D/A

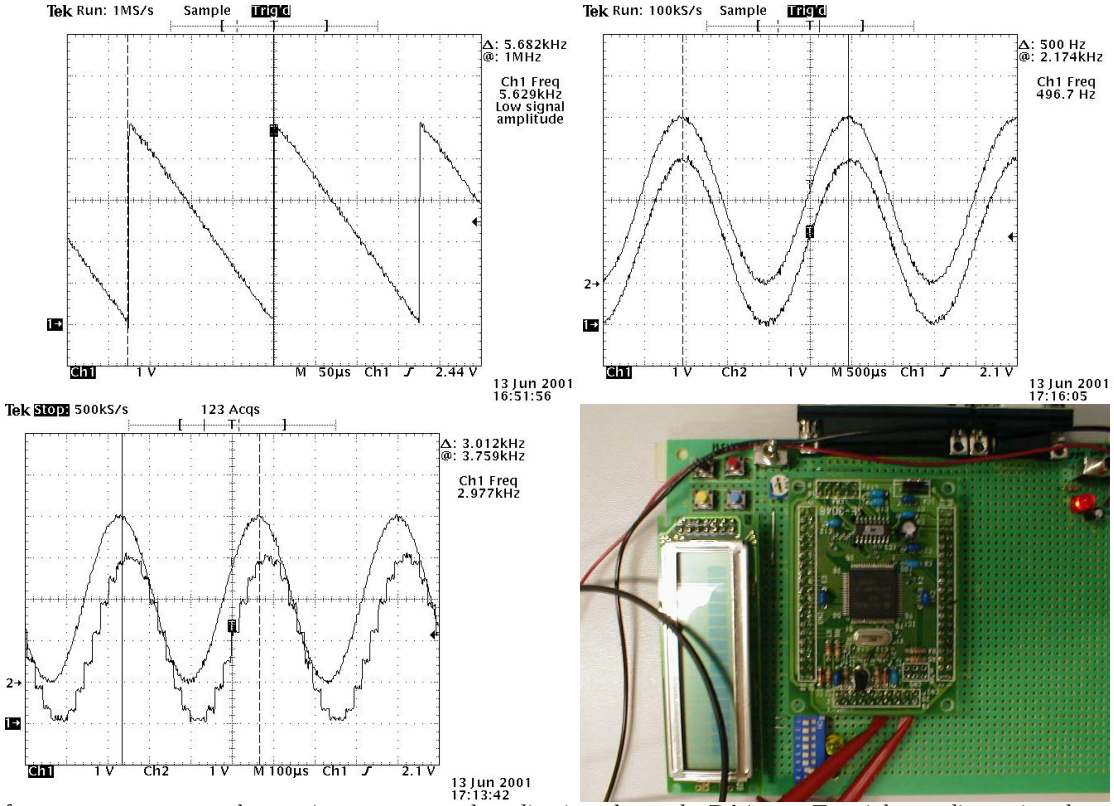
We will first display a saw-tooth shaped signal at the output of the digital to analog converter number 1 (DA1), and then echo on the D/A converter values read on the A/D converter.

```
000ff300 <_start>:
   ff300: 7a 07 00 0f  7a 07 00 0f fe fc mov.l #0xffefc,er7
   ff304: fe fc
   ff306: 07 c0        07 c0             ldc #0xc0,ccr
   ff308: f8 00        f8 00             mov.b #0x0,r0l
   ff30a: 6a a8 00 ff  6a a8 00 ff ff ed mov.b r0l,@0xffffed:32
   ff30e: ff ed
   ff310: f8 ff        f8 ff             mov.b #0xff,r0l
   ff312: 6a a8 00 0f  6a a8 00 0f ff c8 mov.b r0l,@0xfffc8:32
   ff316: ff c8
   ff318: 7f de 70 50  7f de 70 50       bset #0x5,@0xde:8
   ff31c: 7f de 70 70  7f de 70 70       bset #0x7,@0xde:8
   ff320: 7f ca 70 00  7f ca 70 00       bset #0x0,@0xca:8

000ff324 <boucle>:
   ff324: f8 ff        f8 ff             mov.b #0xff,r0l

000ff326 <ici>:
   ff326: 38 dd        38 dd             mov.b r0l,@0xdd:8
   ff328: 1a 08        1a 08             dec.b r0l
   ff32a: 46 fa        46 fa             bne .-6 (ff326)
   ff32c: 5a 0f f3 24  5a 0f f3 24       jmp @0xff324:0
```

`da`: sweeping a counter and sending its value on DA1.

```
000ff300 <_start>:
   ff300: 7a 07 00 0f  7a 07 00 0f fe fc mov.l #0xffefc,er7
   ff304: fe fc
   ff306: 07 c0        07 c0             ldc #0xc0,ccr
   ff308: f8 00        f8 00             mov.b #0x0,r0l
   ff30a: 6a a8 00 ff  6a a8 00 ff ff ed mov.b r0l,@0xffffed:32
   ff30e: ff ed
   ff310: f8 ff        f8 ff             mov.b #0xff,r0l
   ff312: 6a a8 00 0f  6a a8 00 0f ff c8 mov.b r0l,@0xfffc8:32
   ff316: ff c8
   ff318: 7f de 70 50  7f de 70 50       bset #0x5,@0xde:8
   ff31c: 7f de 70 70  7f de 70 70       bset #0x7,@0xde:8
   ff320: 7f e8 70 00  7f e8 70 00       bset #0x0,@0xe8:8

000ff324 <boucle>:
   ff324: 7f e8 70 50  7f e8 70 50       bset #0x5,@0xe8:8

000ff328 <cv>:
   ff328: 7e e8 73 70  7e e8 73 70       btst #0x7,@0xe8:8
   ff32c: 47 fa        47 fa             beq .-6 (ff328)
   ff32e: 28 e8        28 e8             mov.b @0xe8:8,r0l
   ff330: 7f e8 72 70  7f e8 72 70       bclr #0x7,@0xe8:8
   ff334: 29 e2        29 e2             mov.b @0xe2:8,r1l
   ff336: 39 dd        39 dd             mov.b r1l,@0xdd:8
   ff338: 5a 0f f3 24  5a 0f f3 24       jmp @0xff324:0
```

`ad_da` reads a signal on AD1 and echoes the 8 most significant bits of its 10 bit digital conversion to DA1.



Top left: D/A frequency output sweep by running a counter and sending its value to the DA1 port. Top-right: reading a sine-shaped signal from AD1 and sending the 8 most significant bits to DA1. No major difference between the input and output sine waves are visible for this oscillation frequency of 500 Hz. Bottom left: reading a sine-shaped signal from AD1 and sending the 8 most significant bits to DA1, this time to a 3 kHz input signal. The quantification error is easily visible in this case.

5.4 Connecting an LCD display

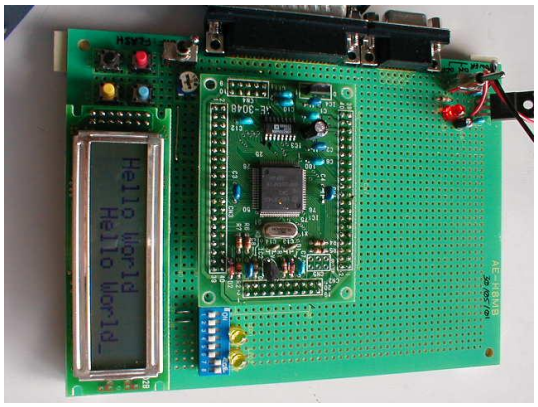
Communication with the LCD display is straight forward but requires being very careful about indicating to the compiler on how many bits the operation has to be performed (all our operations will be performed on 1 byte so most instructions will end with a `:8` suffix). Accessing pre-defined byte and characters arrays is also tested in this program.

```
lcd_4bits:     file format coff-h8300

Disassembly of section .text:

000ff300 <_start>:
   ff300: 7a 07 00 0f  7a 07 00 0f fe fc mov.l #0xffefc,er7
   ff304: fe fc
   ff306: f8 00        f8 00             mov.b #0x0,r0l
   ff308: 6a a8 00 0f  6a a8 00 0f ff c1 mov.b r0l,@0xfffc1:32
   ff30c: ff c1
   ff30e: 6a a8 00 0f  6a a8 00 0f ff c5 mov.b r0l,@0xfffc5:32
   ff312: ff c5
   ff314: f8 ff        f8 ff             mov.b #0xff,r0l
   ff316: 6a a8 00 0f  6a a8 00 0f ff d8 mov.b r0l,@0xfffd8:32
   ff31a: ff d8
   ff31c: 6a a8 00 0f  6a a8 00 0f ff c4 mov.b r0l,@0xfffc4:32
   ff320: ff c4
   ff322: 6a a8 00 0f  6a a8 00 0f ff da mov.b r0l,@0xfffda:32
   ff326: ff da
   ff328: 6a a8 00 0f  6a a8 00 0f ff c8 mov.b r0l,@0xfffc8:32
   ff32c: ff c8
   ff32e: 5e 0f f3 cc  5e 0f f3 cc       jsr @0xff3cc:0
   ff332: f8 23        f8 23             mov.b #0x23,r0l
   ff334: 5e 0f f4 34  5e 0f f4 34       jsr @0xff434:0
   ff338: f8 23        f8 23             mov.b #0x23,r0l
   ff33a: 5e 0f f4 34  5e 0f f4 34       jsr @0xff434:0
   ff33e: f8 23        f8 23             mov.b #0x23,r0l
   ff340: 5e 0f f4 34  5e 0f f4 34       jsr @0xff434:0
   ff344: f8 22        f8 22             mov.b #0x22,r0l
   ff346: 5e 0f f4 34  5e 0f f4 34       jsr @0xff434:0
   ff34a: f8 28        f8 28             mov.b #0x28,r0l
   ff34c: 5e 0f f3 de  5e 0f f3 de       jsr @0xff3de:0
   ff350: f8 08        f8 08             mov.b #0x8,r0l
   ff352: 5e 0f f3 de  5e 0f f3 de       jsr @0xff3de:0
   ff356: f8 0e        f8 0e             mov.b #0xe,r0l
   ff358: 5e 0f f3 de  5e 0f f3 de       jsr @0xff3de:0
   ff35c: f8 06        f8 06             mov.b #0x6,r0l
   ff35e: 5e 0f f3 de  5e 0f f3 de       jsr @0xff3de:0
   ff362: fa 0b        fa 0b             mov.b #0xb,r2l
   ff364: 7a 03 00 0f  7a 03 00 0f f4 48 mov.l #0xff448,er3
   ff368: f4 48

000ff36a <SYOKI0>:
   ff36a: 6c 38        6c 38             mov.b @er3+,r0l
   ff36c: 5e 0f f3 8e  5e 0f f3 8e       jsr @0xff38e:0
   ff370: 1a 0a        1a 0a             dec.b r2l
   ff372: 46 f6        46 f6             bne .-10 (ff36a)
   ff374: f8 c4        f8 c4             mov.b #0xc4,r0l
   ff376: 5e 0f f3 de  5e 0f f3 de       jsr @0xff3de:0
   ff37a: fa 0b        fa 0b             mov.b #0xb,r2l
   ff37c: 7a 03 00 0f  7a 03 00 0f f4 48 mov.l #0xff448,er3
   ff380: f4 48

000ff382 <SYOKI02>:
   ff382: 6c 38        6c 38             mov.b @er3+,r0l
   ff384: 5e 0f f3 8e  5e 0f f3 8e       jsr @0xff38e:0
   ff388: 1a 0a        1a 0a             dec.b r2l
   ff38a: 46 f6        46 f6             bne .-10 (ff382)
   ff38c: 01 80        01 80             sleep

000ff38e <snddat8>:
   ff38e: 0c 89        0c 89             mov.b r0l,r1l
   ff390: 11 09        11 09             shlr r1l
   ff392: 11 09        11 09             shlr r1l
   ff394: 11 09        11 09             shlr r1l
   ff396: 11 09        11 09             shlr r1l
   ff398: e9 0f        e9 0f             and.b #0xf,r1l
   ff39a: 39 c6        39 c6             mov.b r1l,@0xc6:8
   ff39c: 7f c6 70 40  7f c6 70 40       bset #0x4,@0xc6:8
   ff3a0: 7f c6 70 50  7f c6 70 50       bset #0x5,@0xc6:8
   ff3a4: 5e 0f f4 1c  5e 0f f4 1c       jsr @0xff41c:0
   ff3a8: 7f c6 72 50  7f c6 72 50       bclr #0x5,@0xc6:8
   ff3ac: 5e 0f f4 1c  5e 0f f4 1c       jsr @0xff41c:0
   ff3b0: 0c 89        0c 89             mov.b r0l,r1l
   ff3b2: e9 0f        e9 0f             and.b #0xf,r1l
   ff3b4: 39 c6        39 c6             mov.b r1l,@0xc6:8
   ff3b6: 7f c6 70 40  7f c6 70 40       bset #0x4,@0xc6:8
   ff3ba: 7f c6 70 50  7f c6 70 50       bset #0x5,@0xc6:8
   ff3be: 5e 0f f4 1c  5e 0f f4 1c       jsr @0xff41c:0
   ff3c2: 7f c6 72 50  7f c6 72 50       bclr #0x5,@0xc6:8
   ff3c6: 5e 0f f4 1c  5e 0f f4 1c       jsr @0xff41c:0
   ff3ca: 54 70        54 70             rts

000ff3cc <DELAI>:
   ff3cc: 79 04 00 2f  79 04 00 2f       mov.w #0x2f,r4

000ff3d0 <WAI1>:
   ff3d0: 79 05 4f ff  79 05 4f ff       mov.w #0x4fff,r5

000ff3d4 <WAI>:
   ff3d4: 1b 55        1b 55             dec.w #0x1,r5
   ff3d6: 4a fc        4a fc             bpl .-4 (ff3d4)
   ff3d8: 1b 54        1b 54             dec.w #0x1,r4
   ff3da: 4a f4        4a f4             bpl .-12 (ff3d0)
   ff3dc: 54 70        54 70             rts

000ff3de <sndcmd8>:
```

```
   ff3de: 0c 89        0c 89            mov.b r0l,r1l
   ff3e0: 11 09        11 09            shlr r1l
   ff3e2: 11 09        11 09            shlr r1l
   ff3e4: 11 09        11 09            shlr r1l
   ff3e6: 11 09        11 09            shlr r1l
   ff3e8: e9 0f        e9 0f            and.b #0xf,r1l
   ff3ea: 39 c6        39 c6            mov.b r1l,@0xc6:8
   ff3ec: 7f c6 72 40  7f c6 72 40      bclr #0x4,@0xc6:8
   ff3f0: 7f c6 70 50  7f c6 70 50      bset #0x5,@0xc6:8
   ff3f4: 5e 0f f4 1c  5e 0f f4 1c      jsr @0xff41c:0
   ff3f8: 7f c6 72 50  7f c6 72 50      bclr #0x5,@0xc6:8
   ff3fc: 5e 0f f4 1c  5e 0f f4 1c      jsr @0xff41c:0
   ff400: 0c 89        0c 89            mov.b r0l,r1l
   ff402: e9 0f        e9 0f            and.b #0xf,r1l
   ff404: 39 c6        39 c6            mov.b r1l,@0xc6:8
   ff406: 7f c6 72 40  7f c6 72 40      bclr #0x4,@0xc6:8
   ff40a: 7f c6 70 50  7f c6 70 50      bset #0x5,@0xc6:8
   ff40e: 5e 0f f4 1c  5e 0f f4 1c      jsr @0xff41c:0
   ff412: 7f c6 72 50  7f c6 72 50      bclr #0x5,@0xc6:8
   ff416: 5e 0f f4 1c  5e 0f f4 1c      jsr @0xff41c:0
   ff41a: 54 70        54 70            rts

000ff41c <time00>:
   ff41c: 01 00 6d f0  01 00 6d f0      mov.l er0,@-er7
```

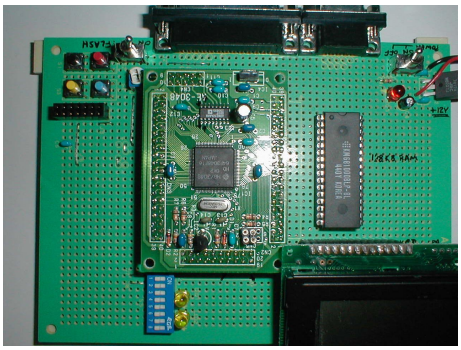
```
   ff420: 7a 00 00 00  7a 00 00 00 20 00 mov.l #0x2000,er0
   ff424: 20 00

000ff426 <time01>:
   ff426: 7a 30 00 00  7a 30 00 00 00 01 sub.l #0x1,er0
   ff42a: 00 01
   ff42c: 46 f8        46 f8            bne .-8 (ff426)
   ff42e: 01 00 6d 70  01 00 6d 70      mov.l @er7+,er0
   ff432: 54 70        54 70            rts

000ff434 <dlcd_4>:
   ff434: 38 c6        38 c6            mov.b r0l,@0xc6:8
   ff436: 5e 0f f4 1c  5e 0f f4 1c      jsr @0xff41c:0
   ff43a: 7f c6 72 50  7f c6 72 50      bclr #0x5,@0xc6:8
   ff43e: 7f c6 70 40  7f c6 70 40      bset #0x4,@0xc6:8
   ff442: 5e 0f f4 1c  5e 0f f4 1c      jsr @0xff41c:0
   ff446: 54 70        54 70            rts

000ff448 <MOJI>:
   ff448: 48 65        48 65            bvc .+101 (ff4af)
   ff44a: 6c 6c        6c 6c            mov.b @er6+,r4l
   ff44c: 6f 20 57 6f  6f 20 57 6f      mov.w @(0x576f:16,er2),r0
   ff450: 72 6c        72 6c            bclr #0x6,r4l
   ff452: 64 00        64 00            or.w r0,r0
```

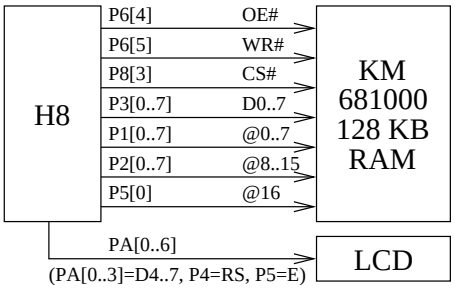
`lcd_4bits`: displaying data on an LCD display with a 4 bits (6 wires) interface.

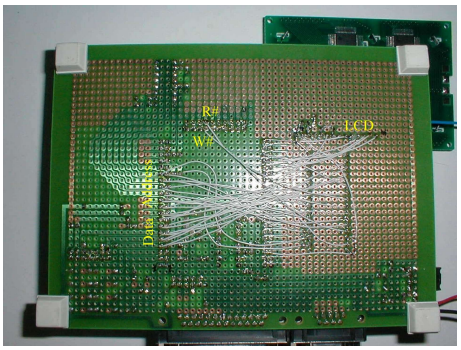


5.5 Connecting an external RAM

Once wired in mode 5 (short circuit the MOD1 pin to ground), the H8/3048 will behave as if some of its ports were data and address busses of a basic microprocessor. Thus, wiring and external RAM becomes obvious, especially since the H8 also provides banked chip select (CS#) signals for addressing multiple RAMs. We have used the connection as described on the following schematic:







128 KB RAM extension to the H8. The LCD screen had to be moved from port 3 to port A as the former was used for RAM access (data bus).

The address range triggering the CS1# signal from the H8 in mode 5 is 0x20000-0x3FFFF.

```
        .h8300h
; .section .start
        .align 1
        .global _start

_start:
        mov.l   #0xffefc,er7            ; =mov.l ...,sp : sett stack pointer
        ldc.b   #0xc0,  ccr             ; disable interrupt
        mov.b   #0x00,  r0l             ; 2-State Access(All area)
        mov.b   r0l,    @0xfffed

        mov.b   #0xff,r0l
        mov.b   r0l,@0xfffc0            ; set P1=@0..7 as output
        mov.b   r0l,@0xfffc1            ; set P2=@8..15 as output
        mov.b   r0l,@0xfffc8            ; set P5=@16.. as output
        mov.b   r0l,@0xfffec            ; set RAM to 8 bits wide access
        mov.b   #0xe8,r0l
        mov.b   r0l,@0xfffcd            ; set RAM to CS1#
        mov.b   #0x00,r0l
        mov.b   r0l,@0xfffed            ; all areas = 2 wait state
        mov.b   #0xf0,r0l
        mov.b   r0l,@0xfffee            ; WSC is programmable ???????
        mov.b   #0xff,r0l
```

```
        mov.b   r0l,@0xfffef            ; all areas use WSC   ???????

        bset    #4,@0xfffba:8   ; SCI initialization
        bset    #5,@0xfffba:8   ; |
        mov.w   #0xff00,r0      ; |
        mov.b   r0h,@0xfffd4:24 ; |

        bsr     rcv
        mov.l   #0x20000,er3    ; start of RAM
fill:   mov.b   @er3,r2l        ; read current RAM value
        bsr     snd             ; send it
        bsr     rcv             ; read new value
        mov.b   r2l,@er3        ; store it
mov.b   @er3,r2l        ; read current RAM value
        bsr     snd             ; send it

        add.l   #1,er3          ; next RAM cell ...
        jmp     fill

rcv:    btst    #6,@0xfffbc:8
        beq     rcv
        mov.b   @0xfffbd:8,r2l
        bclr    #6,@0xfffbc:8
```

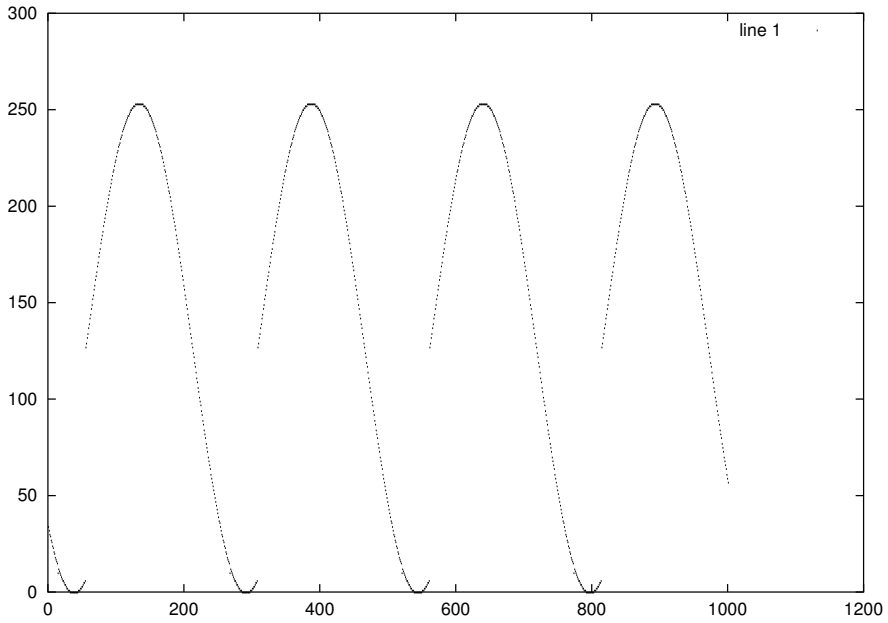
```
        rts

snd:    btst    #7,@0xfffbc:8
        beq     snd
        mov.b   r2l,@0xfffbb:8
        bclr    #7,@0xfffbc:8
        rts

        .end
```

This program sends the data originally stored in RAM (random value following power up sequence), reads data from the serial port, stores them in RAM and reads and sends back the resulting data.

```
// gcc -o ram_test.o ram_test.c rs232.o -lm
#include "rs232.h"
#include <fcntl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <math.h>

void test_h8(int fd)
{unsigned char buf;char prg[3072];int i=0,status;unsigned int j;
 write(fd,&buf,1);
 for (i=0;i<530;i++) // > 131072/253
    for (j=0;j<253;j++) // != 256 => on voit le saut a la fin
      {read(fd,&buf,1);printf("%d:old=%d\t",i*253+j,(int)buf&0x000000FF);
       buf=(int)( (sin ((double)j/(double)50.) +1.)*127. );
       printf("snd:%d\t->\t",buf);
       write(fd,&buf,1);read(fd,&buf,1);
         printf("%d \n",(int)buf&0x000000FF);}
}

int main(int argc,char **argv)
{int fd;
    fd=init_rs232();
    test_h8(fd);
 /* free_rs232(); */
    return(0);
}
```

Associated `C` program running on a linux-based PC for sending data (sine-wave sequence) and reading the result.

```
131035:old=0 snd:0 ->0
131036:old=0 snd:0 ->0
131037:old=0 snd:0 ->0
131038:old=0 snd:0 ->0
131039:old=0 snd:0 ->0
131040:old=255 snd:0 ->0
131041:old=255 snd:0 ->0
131042:old=255 snd:0 ->0
131043:old=255 snd:1 ->1
131044:old=255 snd:1 ->1
131045:old=255 snd:1 ->1
131046:old=255 snd:2 ->2
131047:old=255 snd:2 ->2
131048:old=255 snd:3 ->3
131049:old=255 snd:3 ->3
131050:old=255 snd:4 ->4
131051:old=255 snd:5 ->5
131052:old=255 snd:5 ->5
131053:old=255 snd:6 ->6
131054:old=255 snd:127 ->127
131055:old=255 snd:129 ->129
131056:old=0 snd:132 ->132
131057:old=0 snd:134 ->134
131058:old=0 snd:137 ->137
131059:old=0 snd:139 ->139
131060:old=0 snd:142 ->142
131061:old=0 snd:144 ->144
131062:old=0 snd:147 ->147
131063:old=0 snd:149 ->149
131064:old=0 snd:152 ->152
131065:old=0 snd:154 ->154
131066:old=0 snd:157 ->157
131067:old=0 snd:159 ->159
131068:old=0 snd:162 ->162
131069:old=0 snd:164 ->164
131070:old=0 snd:166 ->166
131071:old=0 snd:169 ->169
131072:old=169 snd:171 ->171
131073:old=171 snd:174 ->174
131074:old=174 snd:176 ->176
131075:old=176 snd:178 ->178
131076:old=178 snd:181 ->181
131077:old=181 snd:183 ->183
131078:old=183 snd:185 ->185
131079:old=185 snd:187 ->187
131080:old=187 snd:190 ->190
131081:old=190 snd:192 ->192
131082:old=192 snd:194 ->194
131083:old=194 snd:196 ->196
131084:old=196 snd:198 ->198
131085:old=198 snd:200 ->200
131086:old=200 snd:202 ->202
131087:old=202 snd:204 ->204
```



Sample output of the previous `C` program, and plot of the same data. The columns show the sample number, the value initially stored in RAM, the value sent and to be stored in RAM, and finally the RAM read from RAM after being stored. Notice that at 131072=2^17, the RAM is no longer accessed (the originally stored value is no longer 0 or 255 but the datum previously sent to the data bus by our test routine, which is not updated by the following reading attempt).

5.6 Writing to flash memory (setting the IRQ vector)

Being able to write to the flash memory is required for being able to change the interrupt vectors when testing programs directly from RAM (*i.e.* without having changed the interrupt vectors table during the programming phase). The procedure for writing one byte to flash memory is described at section 20 of the H8/3048 manual (p. 607) and should be followed carefully in order to avoid damaging the chip. Make sure to fill register `EBR1` properly by setting to 1 the bit defining the block to be written. In our case we wish to write to the interrupt vector table, *i.e.* in the range below 0x04000, so we set `EBR1` to 1 in the `wrtbyte` routine.

In order to read the result from this program on the serial port (a copy of the output is given as comments at the end of the program listing), run `./h8rec` while the program is being transmitted to the H8. The resulting output will show whether the address of the ISR has indeed been saved in flash memory, and if this address is correct.

```
/* writebyte.S : write one byte to EEPROM */
; delay values for a 16 MHz quartz
;a=40 b=12 c=12 d=1516 e=12 f=15 g=15

; SCI RX1 interrupt vector is at 00E4-00E7 (4 bytes)
; MSB of @ must be stored at 00E4, LSB at 00E7 (cf memory format p. 25)

#include "h83048addr.h"

#define AAAA #0x28
#define GGGG #0x0F
#define BBBB #0x0C
#define WDTVAL #0xA57F
; A5=passwd for accessing TCSR (p. 447 of manual)
; writebyte : write 1 byte to EEPROM
; int writebyte(int addr,unsigned char* value)
;
.h8300h
; .section .text
.align 1
.global _start
_start:
        mov.l   #0xffefc,er7 ; =mov.l ...,sp : set stack pointer
        ldc.b   #0xc0,  ccr     ; disable interrupt
     ;   mov.b   #0x00,  r0l     ; 2-State Access(All area)
     ;   mov.b   r0l,@0xffffed

        mov.b   #0xff,  r0l     ; port 5 as output
        mov.b   r0l,@0xfffc8

bset #4,@0xfffba:8 ; SCI initialization
bset #5,@0xfffba:8 ; |
bset #6,@0xfffba:8 ; | <- enable Rx1 interrupt
mov.w #0xff00,r0 ; |
mov.b r0h,@0xfffd4:24 ; |

bset #0,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bclr #1,@0xfffca:8 ; P5DR et C8 -> FF pour out:
jsr DELAI
bclr #0,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bset #1,@0xfffca:8 ; P5DR et C8 -> FF pour out:
jsr DELAI
mov.l #0x000000e7,er2:32 ; END @ (ie we write @-3 to @)
mov.l #rx1isr,er3 ; data
mov  #04,r0l

continue:
push er0 ; no need to save er2/er3: not modified by wrtbyte
jsr wrtbyte
bsr snd
mov r3l,r0l
bsr snd
pop er0
dec.l #1,er2 ; WARNING: this is a DEC, not an INC
shlr.l er3
shlr.l er3
shlr.l er3
shlr.l er3
shlr.l er3
shlr.l er3
shlr.l er3
shlr.l er3
shlr.l er3
dec r0l
bne continue

mov.l @0x000000e4,er2 ; read content of 0x000000e4

mov r2l,r0l
bsr snd
mov r2h,r0l
bsr snd
mov  e2,r2
mov r2l,r0l
bsr snd
mov r2h,r0l
bsr snd

andc.b #0x7F,ccr ; I bit set to 0 => enable IRQs

mov.b #0,r0l ; why am I not allowed to bclr.b #1,@0x0ff300 ?
mov.b r0l,@0xff300 ; prepare register indicating an IRQ occured
mainloop:bset   #1,@0xfffca:8
jsr     DELAI
bclr    #1,@0xfffca:8
jsr     DELAI
mov.b @0xff300,r0l
cmp.b #0x00,r0l
beq mainloop ; if bit of r1l==0, no IRQ => no LED
bset    #0,@0xfffca:8 ; otherwise blink second LED
jsr     DELAI
bclr    #0,@0xfffca:8
jsr     DELAI
mov.b #0,r0l ; we received the char, now forget about it
mov.b r0l,@0xff300 ; |
jmp mainloop
sleep ; THE END
```

```
snd:  btst    #7,@0xfffbc:8
        beq     snd
        mov.b   r0l,@0xfffbb:8
bclr    #7,@0xfffbc:8
rts

DELAI: mov.w   #0x00ff, r4 ; Set program loop counter mais 1f ne marche pas
WAI1: mov.w   #0x0fff, r5 ; Set program loop counter     8fff
WAI: dec.w   #1, r5 ; Program
bpl WAI
dec.w   #1, r4
bpl     WAI1
rts

rx1isr: push r0
rcv: btst    #6,@0xfffbc:8 ; char received ?
        beq     rcv
mov.b   @0xfffbd:8,r0l ; yes: read it
bclr    #6,@0xfffbc:8
mov.b r0l,@0xff300 ; store the received byte in 1st RAM @
pop r0
rte

wrtbyte: ; ARGUMENTS: addr in er2, data in r3h
mov.w #0001,r0 ;vprg-vrfy count (manual, p.592)
mov.w GGGG,r1  ;variable 'g'
mov.w #0x4140,r4
mov.b r4l,@FLMCR:8

jmLOOP0:dec.w #1, r1 ;
bpl jmLOOP0
mov.b #01,r0h ; EBR1 MUST BE 1 => we can write in 1st block
mov.b r0h, @EBR1:8  ; Set EBR1 (r2h=0)      (1st blk=00000-03ffff)
mov.b r3l, @er2  ; Dummy write (er1=@, r2l=data)

mov.w AAAA, e4  ; Set initial program loop counter value
PRGMS: mov.w WDTVAL, r4  ; Start watchdog timer
mov.w r4, @TCSR:16  ;
mov.w e4,e5
mov.w #0x4140, r4  ;
mov.b r4h, @FLMCR:8  ; Set P bit

jmLOOP1:dec.w #1, e5  ; Program
bpl jmLOOP1
mov.w #0xA500, r4  ;
mov.w r4, @TCSR:16  ; Stop watchdog timer

mov.w BBBB , r1  ; Set program-verify loop counter
mov.b #0x44, r4h  ;
mov.b r4h, @FLMCR:8  ; Clear P bit
jmLOOP2:dec.w #1, r1  ; Wait
bpl jmLOOP2

mov.b @er2,r1h
cmp.b r3l,r1h

beq PVOK  ; Program-verify decision
PVNG: mov.b #0x40, r5h  ;
mov.b r5h, @FLMCR:8  ; Clear PV bit
cmp.b #06, r0l  ; Program-verify executed 6 times?
beq NGEND  ; If prg-vrfy executed 6 times, branch to NGEND
inc.b r0l  ; Program-verify fail count + 1
shll.w e4  ; Double program loop counter value
bra PRGMS  ; Program again
PVOK: mov.w #0x4000, r5  ;
mov.b r5h, @FLMCR:8  ; Clear PV bit
mov.b r5l, @EBR1:8  ; Clear EBR1
mov.b r5l, @FLMCR:8  ; Clear V PP E bit One byte programmed
sub r0,r0
bra EXIT

NGEND: mov.w #0x4000, r5 ;
mov.b r5l, @EBR1:8  ; Clear EBR1
MOV.B r5l, @FLMCR:8  ; Clear VPP E bit
mov.w #0xfe,r0

EXIT: ; r0=-1 if programming failed, 0 if programming succeeded
rts

.end

; output of h8rec:
; 170($aa)   <- left over from the communication protocol
; 0($0)      <- ack writing 1 byte (would be $fe otherwise)
; 156($9c)   <- value of the byte just written
; 0($0)      <- repeat another 3 times
; 243($f3)   <- |
; 0($0)      <- |
; 15($f)     <- |
; 0($0)      <- |
; 0($0)      <- last byte written
; 156($9c)   <- start reading address we have just written, LSB 1st
; 243($f3)   <- |
; 15($f)     <- |
; 0($0)      <- end of reading, MSB last => we read 00 0f f3 9c = ISR @
```

Example program for writing the address of a function (Interrupt Service Routine) in the interrupt vector table.

This program grew larger than expected for a simple example, as it includes everything for demonstating the use of interrupts triggered on receiving a new character on the serial port (SCI1). The various parts included are:

1. `wrtbyte`: as described in the H8 manual, this function takes a 32 bit address value (in ER2) and an 8 bit datum (in R3L) to be stored at that address. The correct timing has been taken care of and should not be changed too much. In our case, we want to store in address 0x00E7 (the interrupt vector related to the reception of a character on serial port 1) the address at which the Interrupt Service Routine, called `rx1isr`, is stored. Hence, we put in ER2 the end value of the adress storage space (as long data are stored in memory with the least significant byte at the end of the address space) and the address of the ISR, `#rx1isr`, in ER3. This latter register is shifted 4 times by 8 bits in order to successively store each byte at the right location.

2. `rx1isr`: this is the actual function called when a character is received on the serial port. As it uses register R0, we push this

register on the stack prior to changing it. We then quickly read the content of the serial port reception register (`0xfffbd`) and store in a know address location (beginning of RAM space, `0xff300`). Thus, the main routine can monitor this known RAM address and periodically check whether or not a new character has been received.

The main loop, running from `mainloop` to the (never executed) `sleep` instruction, keeps on blinking LED2, and perdiodically checks RAM address `0xff300` to see if a character has been received (*i.e.* when the value at that address location is not 00). If a character has been received, LED1 blinks and the RAM location is cleared. For a less trivial example, it might be useful to replace the LED blinking loop by the `sleep` instruction (for power saving): in that case, the microcontroller will only wake up when a character is received on the serial port, and perform its duty at that time.

Test characters are sent to the microcontroller simply by `cat >> /dev/ttyS0`.

6 Application examples

6.1 Serial (“SPI like”) communication and more LCD functions

```
; 01/08/2001, H8/3048 + DDS + 2x16 LCD
; 2min15sec pour un sweep de 6 a 4 MHz avec increments de 0xFF

; DDS connections: DATA=P4-0=1-31;     CK=P4-1=1-32;     FQ_UD=P4-2=1-33
.h8300h
; .section .start
.align 1
.global _start

_start:
       mov.l   #0xffefc,er7 ; =mov.l ...,sp : sett stack pointer
       ldc.b   #0xc0,  ccr             ; disable interrupt
       mov.b   #0x00,  r0l             ; 2-State Access(All area)
       mov.b   r0l,@0xffffed

       mov.b   #0xff,r0l   ; set P4 as output
       mov.b   r0l,@0xfffc5  ; |
       mov.b   r0l,@0xfffc8  ; set P5 as output
       MOV.B   r0l,@0xfffDa
       MOV.B   r0l,@0xfffC4
       MOV.B   r0l,@0xfffd8
       mov.b   #0x00,r0l   ; set P4 low
       mov.b   r0l,@0xfffc7  ; |

       bset    #0x0,@0xfffe8:8 ; select A/D 1 (fffe8=AD1 ctrl)

       bset    #4,@0xfffba:8   ; SCI initialization
       bset    #5,@0xfffba:8   ; |
       mov.w   #0xff00,r0      ; |
       mov.b   r0h,@0xffffd4:24 ; |

JSR     DELAI           ;15ms
jsr init_lcd

init_dds:
bset #1,@0xfffc7:8 ; CK=bit 1 of PORT 4
bclr #1,@0xfffc7:8 ; CK=bit 1 of PORT 4
jsr DELAI
bset #2,@0xfffc7:8 ; FQ_UD=bit 1 of PORT 4
bclr #2,@0xfffc7:8 ; FQ_UD=bit 1 of PORT 4
bsr DELAI

debut:
bset #0,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bclr #1,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bsr DELAI
bclr #0,@0xfffca:8 ; P5DR et C8 -> FF pour out:
bset #1,@0xfffca:8 ; P5DR et C8 -> FF pour out:

mov.l #0x33000000,er3 ; SET TO START VALUE to 6 MHz (5.976)
; mov.l #0xfaf5f6f0,er3 ; er3=freq  TEST PATTERN
; mov.l #0x02000000,er3 ; er3=freq  -> 234 kHz
; mov.l #0x01aaaaaa,er3 ; er3=freq  -> 227 kHz
; mov.l #0x01000000,er3 ; er3=freq  -> 117 kHz
; mov.l #0x11000000,er3 ; er3=freq  -> 1.98 MHz

mov #0x01,r1l ; suite de valeurs indiquant une nouvelle
jsr snd ; acquisition (nouveau sweep de freq)
mov #0xff,r1l ; RECHERCHER DANS LE FICHIER L_OCCURENCE DE
jsr snd ; LA SUITE : 01 FF 02 FE 80
mov #0x02,r1l ; POUR SAVOIR OU' COMMENCE UN BALAYAGE DE FREQ
jsr snd
mov #0xfe,r1l
jsr snd
mov #0x80,r1l
jsr snd

boucle:
; sub.l #0x00000100,er3 ; freq dec par 30E6/2^32*256=1.8 Hz
sub.l #0x00001000,er3 ; freq dec par 30E6/2^32*256*16=29 Hz
bsr envoie_donnees

; la condition d_affichage de la freq est er3==XXX00000
mov r3,r0:16 ; octet de poids faible de e3 dans r0h
bne contin ; on n_affiche que de temps en temps
mov e3,r0:16
and #0xf,r0:8 ; on en affiche encore un peu moins ...
bne contin ; on n_affiche que de temps en temps

push er3
jsr affich_fr ; affichage de la valeur courante en hexa
jsr my_affich_fr ; affichage de la frequence en Hz
pop er3

contin: bsr adread
cmp.l #0x22000000,er3 ; STOP VALUE at 4 MHz (3.984)
bne  boucle
bra debut

to_hz:  ; mov.w #0x8f,r0 ; division par 1/(30E6/2^32)=143(.16)
  mov.w #0x36,r0 ; division par 1/(80E6/2^32)=53.7=54
mov.w e3,r2 ; quand on affiche le long resultant,
extu.l        er2 ; on a la frequence en Hz
divxu.w       r0,er2
mov.w e2,e3
divxu.w       r0,er3
mov.w r2,e3
rts

DELAI: mov.w   #0x002f, r4  ; Set loop counter but 1f marche pas
WAI1: mov.w   #0x4fff, r5  ; Set loop counter
WAI: dec.w   #1, r5 ; Program
bpl WAI
dec.w   #1, r4
bpl     WAI1
rts

envoie_donnees: ; INPUT: relative freq in ER3 (32 bits value)
mov.b #0x20,r0l ; 32 bits to be sent (bit counter)
freqdds:btst #0,r3l:8 ; envoie de 32 bits=freq
bne   bas
bclr #0,@0xfffc7:8 ; DATA=bit 0 of PORT 4
bra suite
bas: bset #0,@0xfffc7:8
suite:bset #1,@0xfffc7:8 ; CK=bit 1 of PORT 4
bclr #1,@0xfffc7:8 ; CK=bit 1 of PORT 4
rotr er3:32
dec.b r0l
bne freqdds
footer:mov.b #0x08,r0l ; 32 bits to be sent
mov #0x0,r3l:8 ; r3l=footer
footdds:btst    #0,r3l:8 ; envoie de 32 bits=freq
bne     basfoot
bclr #0,@0xfffc7:8 ; DATA=bit 0 of PORT 4
bra suitefo
basfoot:bset #0,@0xfffc7:8
suitefo:bset #1,@0xfffc7:8 ; CK=bit 1 of PORT 4
bclr #1,@0xfffc7:8 ; CK=bit 1 of PORT 4
rotr r3l:8
dec.b r0l
bne footdds
bset #2,@0xffc7:8 ; set FQ_UD=ack
bclr #2,@0xffc7:8
rts

adread: bset    #5,@0xfffe8:8   ; A/D start conversion
cv:     btst    #7,@0xfffe8:8   ; conversion finished ?
        beq     cv       ; branch while bit Z=0 ie adf=0
        mov.b   @0xfffe8:8,r0l  ; reads ADCSR (?! ... p. 533)
        bclr    #7,@0xfffe8:8  ; datum read: ADF=0
        mov.b   @0xfffe2:8,r1l  ; read ADDRB=datum hi (lo in fffe1) for A/D1
snd:    btst    #7,@0xfffbc:8 ; NOW SEND TO RS232
        beq     snd
        mov.b   r1l,@0xfffbb:8
        bclr    #7,@0xfffbc:8
rts

init_lcd:MOV.B #0b00100011,r0l ; 4 bit mode
JSR @dlcd_4
MOV.B #0b00100011,r0l ;  4 bit mode
JSR @dlcd_4
MOV.B #0b00100011,r0l ;  4 bit mode
JSR @dlcd_4
MOV.B #0b00100010,r0l ; set 4 bit operations
JSR @dlcd_4

MOV.B #0b00101000,r0l ; 11.1/2: 00101000 in 8 bit mode
jsr sndcmd8 ;           4 bits, 2 lines, 5x8 dots font
MOV.B #0b00001000,r0l ; 13.1/2: 00001000 in 8 bit mode
jsr sndcmd8                     disp. off, curs. off, blink off
MOV.B #0b00001110,r0l ; 15.1/2: 00001111 in 8 bit mode
jsr sndcmd8                     disp. on, curs. on, blink off
MOV.B #0b00000110,r0l ; 17.2 ; 00000110 in 8 bit mode
jsr sndcmd8                     inc. curs. pos., no disp. shift
rts

snddat8: ; send 8 bits data (!= cmds)
mov.b r0l,r1l:8
        SHLR.B  r1l:8
        SHLR.B  r1l:8
        SHLR.B  r1l:8
        SHLR.B  r1l:8
AND.B   #0x0F,r1l:8

MOV.B r1l,@0xfffC6:8
BSET #4,@0xfffC6:8 ; RS=0 -> 1 (datum)
BSET #5,@0xfffC6:8 ; E -> 1
JSR @time00
BCLR #5,@0xfffC6:8
JSR @time00

mov.b r0l,r1l:8
```

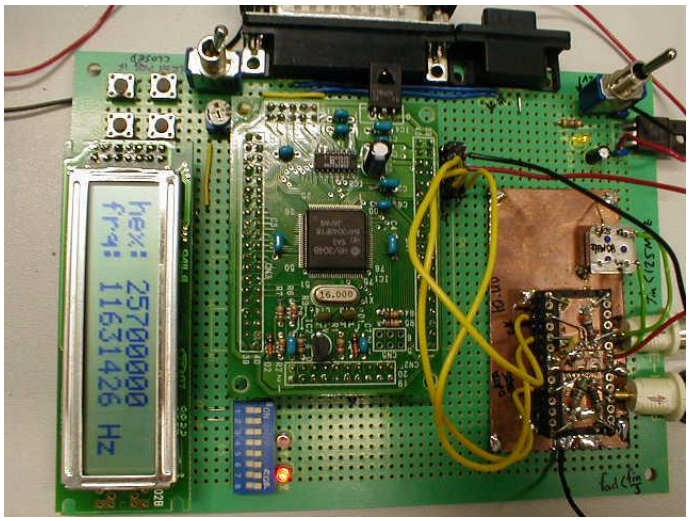
```
AND.B #0x0F,r1l:8
MOV.B r1l,@0xfffC6:8
BSET #4,@0xfffC6:8 ; RS=0 -> 1 (datum)
BSET #5,@0xfffC6:8 ; E -> 1
JSR @time00
BCLR #5,@0xfffC6:8
JSR @time00
RTS

sndcmd8: ; send 8 bits cmd
mov.b r0l,r1l:8
        SHLR.B  r1l:8
        SHLR.B  r1l:8
        SHLR.B  r1l:8
        SHLR.B  r1l:8
AND.B   #0x0F,r1l:8

MOV.B r1l,@0xfffC6:8
BCLR #4,@0xfffC6:8 ; RS-> 0 (cmd)
BSET #5,@0xfffC6:8 ; E -> 1
JSR @time00
BCLR #5,@0xfffC6:8
JSR @time00

mov.b r0l,r1l:8
AND.B #0x0F,r1l:8
MOV.B r1l,@0xfffC6:8
BCLR #4,@0xfffC6:8 ; RS=0 (cmd)
BSET #5,@0xfffC6:8 ; E -> 1
JSR @time00
BCLR #5,@0xfffC6:8
JSR @time00
RTS

time00: PUSH.w  r0 ; SHORT delay (avoids waisting too much time
        mov.w   #0x001f,r0 ; on display), must be > 0x000f
time01: SUB.w   #1,r0
        BNE     time01
        POP.w   r0
        RTS

dlcd_4: MOV.B r0l,@0xfffC6:8
JSR @time00
BCLR #5,@0xfffC6:8 ; RS -> 0
BSET    #4,@0xfffC6:8 ; E=hi
  JSR @time00
RTS

prn_msg:mov.b   @er3+,r0l
        jsr     @snddat8 ; (r2l contient le nbre de chars a afficher,
        dec.b   r2l ; er3 pointe sur la chaine de chars a printer)
        bne     prn_msg
rts

my_affich_fr: ; FREQ TO BE DISPLAYED IS XFERED IN ER3
        MOV.B  #0b11000000,r0l ; AFFICHAGE DE LA FREQ en decimal
jsr     sndcmd8         ; gotoxy (2nd line): pos 07

jsr to_hz ; convert to Hz (remove to simply display
; a _long_ in decimal format)
push er3
        MOV.B   #0x05,r2l       ; display string top-left
        mov.l   #M2,er3
jsr prn_msg
pop er3
; CONVERSION HEX -> DEC (pour 0 < N < 1E8)
mov.l #0x989680,er4 ; =1E7
deb: mov.b #0x00,r2l
suit: cmp.l er4,er3 ; test er3-er4
blt fini ; if er3<er4, then branch
sub.l er4,er3 ; er4=er4-er3
inc.b r2l
bra suit
fini: mov r2l,r0l
jsr toprint

cmp.l #0xf4240,er4 ; =1E6
ble fini1
mov #0xf4240,er4
bra deb
fini1: cmp.l #0x186a0,er4 ; =1E5
ble fini2
mov #0x186a0,er4
bra deb
fini2:cmp.l #0x2710,er4 ; =1E4
ble fini3
mov #0x2710,er4
bra deb
fini3:cmp.l #0x3e8,er4 ; =1E3
ble fini4
mov #0x3e8,er4
bra deb
fini4:cmp.l #0x64,er4 ; =1E2
ble fini5
mov #0x64,er4
bra deb
fini5:cmp.l #0xa,er4 ; =10
ble fini6
mov #0xa,er4
jmp deb
fini6: mov r3l,r0l
jsr toprint ; ... et on affiche les unites

        MOV.B   #03,r2l         ; _Hz_ en fin de ligne
        mov.l   #M3,er3
jsr prn_msg
rts

toprint:and     #0x0f,r0l ; only display lowest nibble (r0l in 0..9)
        cmp     #0x0a,r0l       ; hexa -> ASCII
        blt     base2
        add     #0x07,r0l
base2:  add     #0x30,r0l
        jsr     @snddat8
rts

affich_fr: ; FREQ TO BE DISPLAYED IS XFERED IN ER3
        MOV.B  #0b10000000,r0l ; AFFICHAGE DE LA FREQ en hexa
jsr     sndcmd8         ; gotoxy (2nd line): pos 07

push er3
        MOV.B   #0x05,r2l       ; display string top-left
        mov.l   #M1,er3
jsr prn_msg
pop er3

MOV.B   #0x08,r2l
snd_fr: rotl er3:32
rotl er3:32
rotl er3:32
rotl er3:32
mov r3l,r0l
and #0x0f,r0l
cmp #0x0a,r0l ; hexa -> ASCII
blt bbase
add #0x07,r0l
bbase: add #0x30,r0l
jsr @snddat8
DEC.B   r2l
BNE     snd_fr ; FIN AFFICHAGE FREQ SUR LCD
rts

.align 1
M1: .string "hex: "
M2: .string "frq: "
M3: .string " Hz"
.end
```

Program for sending orders (frequency) to an AD9850 frequency synthesizer, displaying the value on an LCD screen and the corresponding frequency in hertz.

From the hardware point of view, the AD9850 DDS chip is connected to port 4 of the H8/3048 in the following way: PA0 is the data line, PA1 is the clock line and PA2 is the `FQ_UD` line used for confirming the completion of a data transfer.

This program will:
- sweep the frequency between a minimum and maximum value, currently set to 4 and 6 MHz (for a 80 MHz DDS reference oscillator),
- display the frequency in Hz on a 2-line LCD display,
- read on the analog to digital converter (AD1) an 8 bit analog value of the signal transmitted at this frequency by the device under test,
- send on the serial port (SCI1) at 9600 bauds (N81) the result.

`init_dds` first initializes the AD9850 frequency synthesizer for serial communication. The frequency loop uses the counter stored in ER3 (32 bits value), which is initialized to its maximum value at `debut:` and is decremented until reaching its minimum value (set close to `contin:`). A pre-defined sequence, `01 FF 02 FE 80`, defines the beginning of a new frequency sweep and can easily be found in the file in which all results are stored. The frequency value (between 0x00000000 and 0xFFFFFFFF) is sent to the DDS by `envoie_donnees:`, which sends the 32-bits frequency followed by 0x00 (phase and control sequence) for a total of a 40-bits transfer. The signal `FQ_UD` is asserted at the end of the transfer. The value is displayed on the 2×16 characters LCD screen in hexadecimal format by `affich_fr:`, and in decimal format by `my_affich_fr:` after having been converted to hertz by `to_hz:` (which operates a division by $54 \simeq \frac{80MHz}{2^{32}}$



Circuit including the AD9850 frequency synthetizer.