# Introduction to the Motorola 68HC908JB8

Jean-Michel FRIEDT, December 2001-October 7, 2004

### 1 Introduction

We decided to look at the 68HC908JB8 microcontroller from Motorola following interest in the development of USB enabled intruments. The 68HC908JB8 does not require any external programmer since it is based on flash memory, provides a convenient communication tool in ROM called the Monitor mode, includes one USB peripheral but no UART. It is available both is easy to handle DIP package and low volume SOIC package.

All development were done under Linux (kernel 2.2.19, although no kernel specific functions were used) using gcc 2.95.2 for generating the binaries running on the PC and as6808 v.03.10 provided by asxxxx v.3.10 (November 2001) for the programs running on the microcontroller.

The following script was used for generating the ASCII file containing the hexadecimal codes of the program to be transferred from the PC to the microcontroller:

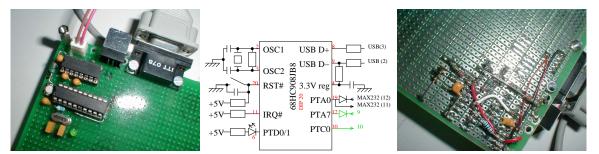
<pre>#!/usr/bin/tcsh if ( \$# == 0 ) then</pre>	./as6808 -	"->" \$nom".out"
---	------------	------------------

asm: script for compiling a text file containing the assembly program to an ASCII file containing the hexadecimal opcodes.

Once the .out ASCII file containing the hexadecimal opcodes is generated, the board is switched on and the program is uploaded using ./hc08 file.out where hc08 is a program described later developed for the purpose of learning how this microcontroller works. But first we must look at the hardware before being able to test our first program.

### 2 Electronic circuit

The electronic circuit around the 68HC908JB8 is quite simple and highly inspired from the development circuit provided by Motorola. It mainly consists of the oscillator circuit, pull up resistors to the interrupt (IRQ#) and PTA0 lines, and the reset switch. Since the PTA0 line, which is used for communication in the monitor mode with the PC through the MAX232, includes its own internal pull up resistor, adaptation between the TTL (5 V) logic of the MAX232 and the 3.3 V logic of the 68HC908JB8 is quite simple: a diode turned towards the highest voltage is enough. Indeed, if the MAX232 outputs a low level (pin grounded), the current can flow from the microcontroller to the MAX232 and PTA0 senses a ground level. If on the other hand the MAX232 pin is high (+5 V), the diode is blocking the current flow from the microcontroller to the MAX232 and the PTA0 pin senses a high (+3.3 V) level thanks to its pull up resistor.



Simple circuit for communicating with the 68HC908JB8 microcontroller. The green lines are optional connections between the microcontroller and the MAX232 in order to use a second software emulated UART. The red numbers are the pin numbers for a 20 pin DIP package.

### 3 The Monitor mode

First, we must find out how to program the 68HC908 and get familiar with its monitor feature. To this date (December 2001-January 2002) one program exists for programming two of the 68HC908 family: spgmr08, version 0.9. However, this software aims at integrating a lot of features in one bulky executable, including a GUI, which is not what I was looking for. And anyway, understanding every steps of the programming part of the microcontroller is interesting. So after building a basic board including a CPU, a MAX232 RS232 level converter and a few passive components as described earlier, I started putting together a few routines for getting familiar with the monitor mode.

Two tricks appeared:

- when sending the 8 security bytes at the beginning of the transmission, a delay between the received echo and the new

transmission is required

- the echo does not include one but two characters: the direct connexion through the protection diode of the RS232 transmission line with its reception line, followed by the echoed character by the microcontroller.

The monitor mode is otherwise implemented as described in section 10 of the Technical Data book. It allows reading and writing individual bytes or sequentially to any place in the microcontroller's memory, including to the I/O ports which makes testing simple circuits very easy. As a first example, let us make an LED connected to port D pin 0/1 (on the 20 pin DIP package) blink under computer control. We here write to the port D register from the PC: no program is running on the microcontroller itself (apart from the Monitor routine provided in ROM).

/* All examples have been derived from miniterm.c */	<pre>// newtio.c_cc[VSWTC] = 0; /* '\0' */</pre>
/* Don't forget to give the appropriate serial ports the right permissions $*/$	<pre>// newtio.c_cc[VSTART] = 0; /* Ctrl-q */</pre>
/* (e. g.: chmod a+rw /dev/ttyS0) */	<pre>// newtio.c cc[VSTOP] = 0; /* Ctrl-s */</pre>
	<pre>// newtio.c_cc[VSUSP] = 0; /* Ctrl-z */</pre>
#include "rs232.h"	<pre>// newtio.c_cc[VEOL] = 0; /* '\0' */</pre>
"Include Include Inclu	<pre>// newtio.c cc[VREPRINT] = 0; /* Ctrl-r */</pre>
extern struct termios oldtio, newtio;	<pre>// newtio.c_cc[VDISCARD] = 0; /* Ctrl-u */</pre>
extern struct termios ofutio, newtro,	
int init_rs232(int BAUDRATE)	<pre>// newtio.c_cc[VLNEXT] = 0;  /* Ctrl-v */</pre>
{int fd;	// newtio.c_cc[VEOL2] = 0; /* '\0' */
fd=open(HC11DEVICE, 0_RDWR   0_NOCTTY );	<pre>tcflush(fd, TCIFLUSH);tcsetattr(fd,TCSANOW,&amp;newtio);</pre>
if (fd <0) {perror(HC11DEVICE); exit(-1); }	<pre>// printf("RS232 Initialization done\n");</pre>
tcgetattr(fd,&oldtio); /* save current serial port settings */	return(fd);
bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings */	3
// newtio.c_cflag = BAUDRATE   CRTSCTS   CS8   CLOCAL   CREAD;	
newtio.c_flag = BAUDRATE   CS8   CLOCAL   CREAD; /* _no_ CRTSCTS */	void sendcmd(int fd,char *buf)
newtio.c_iflag = IGNPAR; //   ICRNL   IXON;	{unsigned int i, j;
newtio.c_oflag = IGNPAR; // ONOCR [ONLRET]OLCUC;	if((write(fd,buf,strlen(buf))) <strlen(buf))< td=""></strlen(buf))<>
// newtio.c_lflag = IGANON;	<pre>{printf("\n No connection\n");exit(-1);}</pre>
<pre>// newtio.c_cc[VIINTR] = 0; /* Ctrl-c */</pre>	for (j=0;j<5;j++) for (i=0;i<3993768;i++) {}
// newtio.c_cc[VQUIT] = 0; /* Ctrl-\ */	/* usleep(attente); */
// newtio.c_cc[VERASE] = 0; /* del */	3
// newtio.c_cc[VKILL] = 0; /* @ */	
<pre>// newtio.c_cc[VEOF] = 4; /* Ctrl-d */</pre>	void free_rs232(int fd)
<pre>newtio.c_cc[VTIME] = 0; /* inter-character timer unused */</pre>	{tcsetattr(fd,TCSANOW,&oldtio);close(fd);} /* restore the old port settings */
<pre>newtio.c_cc[VMIN] = 1; /* blocking read until 1 character arrives */</pre>	

rs232.c: basic RS232 initialization routine needed for all programs running under Linux requiring access to the serial port.

<pre>#include <stdio.h> #include <stdlib.h> #include <unistd.h></unistd.h></stdlib.h></stdio.h></pre>		<pre>int init_rs232(); void free_rs232(); void sendcmd(int,char*); struct termios oldtio,newtio;</pre>
<pre>#include <sys types.h=""> #include <sys stat.h=""> #include <string.h> #include <fcntl.h> #include <fcrtl.h></fcrtl.h></fcntl.h></string.h></sys></sys></pre>	<pre>/* declaration of bzero() */</pre>	// #define BAUDRATE B9600 // #define BAUDRATE B19200 #define HC11DEVICE "/dev/ttyS0"

rs232.h: header for the basic RS232 initialization routines.

```
// test hardware: make a diode blink from monitor (MON) mode
                                                                                                                                                                                            lo=0x03; hi=0; writ_hc08(fd,hi,lo,0x00);
                                                                                                                                                                                           sleep(1);
#include "hc08.h
                                                                                                                                                                                     3
void hc08_test(int fd)
{char lo,hi;
  lo=0x56; hi=0; read_hc08(fd,hi,lo);
                                                                                                                                                                                     void hc08_prg(int fd)
                                                                                                                                                                                     Vold neco_prg.ne 14;
{char lo,hi;
lo=0x00; hi=0x01; writ_hc08(fd,hi,lo,0x00);
iwrit_hc08(fd,0x22);
 lo=0x56; hi=0; read_hc08(fd,hi,lo);
lo=0x57; hi=0; read_hc08(fd,hi,lo);
lo=0x56; hi=0; read_hc08(fd,hi,lo);
lo=0x56; hi=0; read_hc08(fd,hi,lo);iread_hc08(fd);
lo=0x56; hi=0; writ_hc08(fd,hi,lo,0x50);
lo=0x56; hi=0; read_hc08(fd,hi,lo);iread_hc08(fd);
i=10; hc08(fd,0=20).
                                                                                                                                                                                       readsp_hc08(fd);
                                                                                                                                                                                     int main(int argc, char **argv)
  iwrit_hc08(fd,0x32);
                                                                                                                                                                                     {int fd:
  lo=0x59; hi=0; read_hc08(fd,hi,lo);
                                                                                                                                                                                            fd=init rs232(B9600):
 10=0x09; n1=0; real_nc00(rd,n1,10);
readsp_hc08(fd);
lo=0x07; h1=0; writ_hc08(fd,h1,l0,0xFF);
while(1) {
                                                                                                                                                                                            fawint_rs23(b900);
init_hc08mon(fd);fre_rs232();
fd=init_rs232(B9600);// forget the 10 stop bits
hc08_test(fd);
     lo=0x03; hi=0; writ_hc08(fd,hi,lo,0xFF);
                                                                                                                                                                                            free_rs232();
     sleep(1);
```

Test programming for controlling the blinking of an LED connected to port D pin 0/1 (20 pin DIP package) controlled from a Linux running PC.

Once we have checked the basic circuitry around the microcontroller is operating properly, we can go on to the next step of performing the same task (blinking an LED) from a program stored on board the microcontroller. The slight additional difficulty is to figure out how to setup the memory in the microcontroller before sending a RUN command to the monitor. I found the answer in a comment to spgmr08, in mongp32.c (comment to the routine mon\_runpc()): after asking the monitor the current position of the stack pointer (SP), we simply store in SP+4 the high byte of our program's starting address and in SP+5 the lower byte of our program's starting address. Since I have decided to store my program in the beginning of RAM space which starts at 0x0040, (SP+4)=0x00 and (SP+5)=0x40 in my case. After reset, the monitor mode defines the stack pointer to be located at 0x00FF, and uses a few bytes so that just before executing the RUN command, SP always appears to be equal to 0x00FA (=250d).

#### 3.1 Storing and executing a program from RAM

#include "hc08.h"

j
int main(int argc,char \*\*argv)
{int fd;FILE \*f;
if (argc<2) {printf("%s filename\n",argv[0]);} else {
 fd=init\_rs232(); init\_hc08mon(fd);free\_rs232();fd=init\_rs232();// forget the 10 stop bits
 f=fopen(argv[1],"r");
 hc08\_prg(f,fd);
 fclose(f);
 free\_rs232();
 } return(0);
}</pre>

Store a program in RAM (address 0x0040) and execute it.

DEBUG("\n RUN executed \n");

<pre>start: ldhx #0x0140 ; TXS : (SP)&lt;-(H:X)-1 =&gt; STACK=0x013f txs ; reset stack pointer ; mov #0x01,CONFIG1 ; disable COP watchdog, CONFIG1=0x001f clra ; clear accumulator mov #0x0f,0x0007 ; DDRD: port D as output</pre>	delay: psha pshx clrx ; 256*0,9375ms=240ms loopz: clra ; 9*256*2304 (0,9375ms @ 2,4576MHz) loopa: nsa ; [3]
loop: eor #0x0f ; toggle diode	nsa ; [3] dbnza loopa ; [3]
sta 0x0003 ; store accumulator on PortD	dbnzx loopx ;
bsr delay	pulx
bra loop	pula
	rts

blink.asm: sample program for blinking a LED connected to PTD0/1.

The limitation of this programming method is quickly obvious: since the default location of the stack pointer (SP) is 0x00FF, the RAM is cut in two halves (0x0040 to 0x00FA approximately, and 0x00FF to 0x013F) which only allows uploading to the microcontroller programs 186 and 64 bytes long respectively (depending whether we store the program below or above the stack pointer). Since our aim is USB development and the sample (short) program from Motorola already needs 1.8 KB, being able to store the code to flash memory and execute it from there seems mandatory. Our next steps will thus be to develop a software UART (since the 68HC908JB8 does not include an hardware UART) so we can receive new data from our own programs, and then to learn how to store data (the new program we want to save) in flash memory. This way, we will be able to test programs up to 8 KB long, which should be enough to get us started with USB development.

## 4 Asynchronous communication (software emulation)

First, we wish to transmit characters from the microcontroller to the PC.

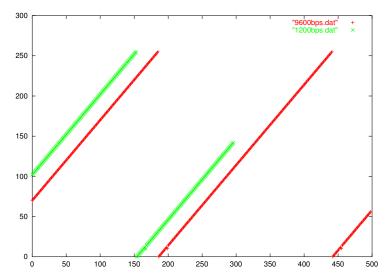
start: ldhx #0x0140 ; TXS : (SP)<-(H:X)-1 => STACK=0x013f	bcc bit0 ; 3 branch if carry is clear (is A&1=0)
txs ; reset stack pointer	mov #0x01,0x0000 ; 4 PTA0=hi
; mov #0x01,CONFIG1 ; disable COP watchdog, CONFIG1=0x001f	bra bit1 ; 3
-	bit0: mov #0x00,0x0000 ; 4 PTA0=10
mov #0x00,0x0003 ; PTD0/1 lo => LED lit	bit1: dbnzx looprs ; 3> sum=11 or 14
mov #0x03,0x0007 ; DDRD: PTD0/1 as output	fin: bsr delay
	bsr delay
mov #0x01,0x0000 ; PTA: PTAO hi	mov #0x01,0x0000 ; PTA: PTA0 hi : STOP bit
mov #0x01,0x0004 ; DDRA: PTAO as output	bsr delay
ldx #0h00	bsr delay
loop: incx ; increment counter	pulx
txa	rts
bsr send; send value of counter to serial port	
bra loop	delay: pshx ; 2+4 for bsr (12+2*((X*9)+15))*.3333=833
	ldx #0h0f ; 3 104
send: pshx	loopx: nsa ; 3 => Xinit=0x88 for 1200
ldx #0x08 ; snd through PTA0 the content of Acc (@9600)	nsa ; 3 =0x0f for 9600
mov #0x00,0x0000 ; PTA: PTA0 lo : START bit	dbnzx loopx ; 3
looprs: bsr delay ; X	pulx ; 2
bsr delay ; X	rts ; 4
rora : 1 rotate right Acc through carry	

rs\_snd.asm: sample program for sending the values of a free running counter to the serial port. The baud rate is defined by the delay value in the function delay (0x88 for 1200 baud, 0x0f for 9600 baud communication).

#include "rs232.h"
<pre>void read_osc(int fd) {ussigned char buf; while (1) {read(fd,&amp;buf,1); printf ("%u(\$%x) \n",buf&amp;Ox000000FF,buf&amp;Ox00000FF); fflush(stdout);}</pre>

void main(int argc,char \*\*argv)
{int fd;
fd=init\_rs232();
read\_osc(fd);
free\_rs232();
}

hc08rec.c: sample program for reading the values on the RS232 port and displaying their decimal and hexadecimal values.



Result of transmitting the values of a free running counter (increasing) at 1200 (green) and 9600 (red) bauds. All data were correctly transmitted from the microcontroller running the software UART to the PC.

Now that we understand how to transmit arbitrary values from the microcontroller to the PC, we also want the microcontroller to be able to read values from the serial port and process them (for example add 3 and send the result back).

start: ldhx #0x0140 ; TXS : (SP)<-(H:X)-1 => STACK=0x013f	fin: bsr delay
txs ; reset stack pointer	bsr delay
; mov #0x01,CONFIG1 ; disable COP watchdog, CONFIG1=0x001f	mov #0x01,0x0000 ; PTA: PTA0 hi : STOP bit
-	bsr delay
mov #0x00,0x0003 ; PTD0/1 lo => LED lit	bsr delay
mov #0x03,0x0007 ; DDRD: PTD0/1 as output	pulx
-	rts
; ldx #0h00	
	rcv: lda #0x80
loop: mov #0x00,0x0004 ; DDRA: PTA0 as output	rcvst: brset #0,*0x0000,rcvst; wait for START bit
bsr rcv	; mov #0x03,0x0003 ; START bit => switch LED off (debug)
; incx ; increment counter	bsr delay ; wait half a bit width
; txa	looprc: bsr delay ; X
inca	bsr delay ; X
inca	brclr #0,*0x0000,rcv0 ; 5 branch if bit is clr => branch if 0, C=bit
inca	rcv0: rora ; 1 after 8 ror, carry=1
mov #0x01,0x0000 ; PTA: PTAO hi (in order to avoid glitches)	; 2 uses of carry bit in these 2 lines: brclr sets the carry bit to the bit
mov #0x01,0x0004 ; DDRA: PTAO as output	; value, and we use the output of rora to the carry bit to check if we rotated
bsr send ; send value of counter to serial port	; 8 times already
bra loop	bcc looprc ; 3> sum=8 or 11 cycles
-	bsr delay
send: pshx	bsr delay ; wait for stop bit
ldx #0x08 ; snd through PTA0 the content of Acc (@9600)	rts
mov #0x00,0x0000 ; PTA: PTA0 lo : START bit	
loopsn: bsr delay ; X	
bsr delay ; X	delay: pshx ; 2+4 for bsr (12+2*((X*9)+15))*.3333=833
rora ; 1 rotate right Acc through carry	ldx #0h0f ; 3 104
bcc bit0 ; 3 branch if carry is clear (is A&1=0)	loopx: nsa ; 3 => Xinit=0x88 for 1200
mov #0x01,0x0000 ; 4 PTA0=hi	nsa ; 3 =0x0f for 9600
bra bit1 ; 3	dbnzx loopx ; 3
bit0: mov #0x00,0x0000 ; 4 PTA0=10	pulx ; 2
bit1: dbnzx loopsn ; 3> sum=11 or 14 cycles	rts ; 4

rs\_rcsn.asm: sample program for reading values on the RS232 port, adding 3 to them and sending them back.

#include "rs232.h"	<pre>printf ("%u(\$%x) \n",buf&amp;0x000000FF,buf&amp;0x000000FF); fflush(stdout);}</pre>
void read_osc(int fd)	}
{unsigned char buf,cpt=10;	
<pre>while (1) {buf=cpt;write(fd,&amp;buf,1);</pre>	void main(int argc, char **argv)
cpt++;	{int fd;
printf ("%u(\$%x) -> ",buf&0x00000FF,buf&0x000000FF);	fd=init_rs232();
// read(fd,&buf,1);	read_osc(fd);
<pre>// printf ("%u(\$%x) ",buf&amp;0x00000FF,buf&amp;0x00000FF);</pre>	free_rs232();
read(fd,&buf,1);	}
	•

hc08sndrec.c: accompanying C program for testing rs\_rcsn. This program sends values to the microcontroller and reads the processed result.

### 5 Storing a program in flash memory and execution

Flash memory starts at address 0xDC00. The limitation for the flash memory programmer is as follows: it must be stored in RAM between locations 0x8C (since RAM space from 0x40 to 0x8B is used by the flash programmer provided in ROM of the 68HC08JB8) and 0xF9 (since that is the lowest address accessed by the stack in monitor mode). Hence, the flash programmer cannot be longer than 0x8C-0xF9=109 bytes.

Due to this limitation, we strip our flash programmer to the bare minimum: no character is sent from the microcontroller to the PC during the programming step (so the UART transmit emulation can be removed). This limitation removes the ability to echo the received bytes in order to confirm what is happening on the microcontroller side.

<pre>idx #00140;TES:(SP)&lt;-(H:D)-1 ⇒&gt; STACK=0x013f txs:rest state phinter is or #0030,000FG1 : stable OD watchdog, ODFIG1=0x001f txs:rest state phinter is or #0030,000G3; FTMO/1 a &gt;&gt; LED it cor #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state phinter is or #0030,00007 : DED: PTD0/1 a output the state the stat</pre>		
<pre>tx : rest tack pointer ; sow #0x00,0x0003 ; PT0/10 -&gt; LED lit sov #0x00,0x0003 ; PT0/10 -&gt; LED lit sov #0x00,0x0007 ; DDDD: PTD0/1 as output ldx #0x400 start: sow #0x00,0x0007 ; DDDD: PTD0/1 as output ber for ; reseits bit to be written in A tack for a for a start ber for ; reseits bit to be written in A ther folds do 0, sow #0x00,0x0000 ; FTA: PTA0 hit : DTAD hit ber for ; reseits bit to be written in A ther folds do 0, sow #0x01,0x0000 ; FTA: PTA0 hit (norder to avoid glitches) sow #0x01,0x0000 ; FTA: PTA0 hit (norder to avoid glitches) sow #0x01,0x0000 ; FTA: PTA0 hit (norder to avoid glitches) sow #0x01,0x0000 ; FTA: PTA0 hit in FLCR lish in ber delay is #1; H:K=t ber sead sis #1; H:K=t flah: lda #0x01 sto 0,ff00 ; read FLDR is 0,ff00 ; read FLDR is</pre>		
<pre>is or #0x00,00003if i; disable OUP watchdag, CNNFIGL=0x001f</pre>		and, ash
<pre>nov #0x00,0x0000; PTA: PTAO ls : START bit nov #0x03,0x000; PTA: PTAO ls : START bit nov #0x03,0x000; DDAD: PTAO/l as output ldb foxdo bor row #0x00,0x0004; DDAD: PTAO as input bor row #0x00,0x0004; DDAD: DTAO as input bor row #0x00,0x0004; DDAD: PTAO as input bor row #0x00,0x0004; DDAD: DTAO as input bor row #0x00,0x0004; DDAD: DTAO as input bor row #0x00,0x0004; DDAD: DTAO as input bor row #0x00,0x0004; DDAD: PTAO as input bor row #0x00,0x0004; DDAD: PTAO as input bor row #0x00,0x0004; DDAD: PTAO as input bor row #0x00,0x0004; ATAO+ls ls to row #0x00,0x004; ATAO+ls ls t</pre>		
<pre>ioor #0x00,0x000; yTDD/1 is &gt;&gt; LED lit or #0x00,0x0000; yTDD/1 is output idx #0x00,0x0000; jDDA: PTA0 as input bar rev; receive but to be written in A sta 0x100 bar rev; receive but to be written in A sta 0x100 bar rev; receive but to be written in A sta 0x100 bar rev; receive but to be written in A sta 0x100 bar rev; receive but to be written in A sta 0x100 bar rev; receive but to be written in A sta 0x100 bar rev; receive but to be written in A sta 0x100 bar rev; receive but to be written in A sta 0x100 bar flamb ide 0,x sow #0x01,0x0000; PTA: PTA0 hi (in order to avoid glitches) sow #0x01,0x0000; PTA: PTA0 hi (in order to avoid glitches) sow #0x01,0x0000; PTA: PTA0 hi (in order to avoid glitches) sow #0x01,0x0000; PTA: PTA0 hi (in order to avoid glitches) sow #0x01,0x0000; PTA: PTA0 hi : STOP bit be dainy bra start flamb: ide #0x01 sta 0xfe00; read FLBRR ide 0x01 sta 0xfe00; read FLBRR ide 0x02 sta 0xfe00; FLCR: PON bit=1, HVEN=1 ide 80x0 sta 0xfe00; FLCR: PON bit=1, HVEN=1 ide 80x0 sta 0xfe00; FLCR: PON bit=0 ide 80x0 sta 0xfe00; FLCR: FUR bit=0</pre>	; mov #uxul,cumrigi; disable cup watchdog, cumrigi=uxuuli	
<pre>bor #0x03,0x0007 ; DDRD: FTD0/1 as output bar datay ; X core ; 1 rotate right Ac through carry in clear (is Ak1=0) the focal y ; X core ; 1 rotate right Ac through carry bc bit ( ; 3 branch if carry is clear (is Ak1=0) the focal y ; X bc bit ( ; 3 branch if carry is clear (is Ak1=0) the focal y ; X bc bit ( ; 3 branch if carry is clear (is Ak1=0) the focal y ; X bc bit ( ; 3 branch if carry is clear (is Ak1=0) the focal y ; X bc bit ( ; 3 branch if carry is clear (is Ak1=0) the focal y ; X br desay and focal y ; X br desay br desay ; X bc bit ( ; 3 branch if carry is clear (is Ak1=0) the focal y ; X br desay and focal y ; X br desay ; X br desay br desay ; X br desay br desay ; X br desay br</pre>		
<pre>rows i rotate right Ac through carry bcchio(xc000; DBA: PTA0 as input ber row; receive bit to be written in A at a Ox100 bar rows; receive bit to be written in A at a Ox100 bar rows; receive bit to be written in A at a Ox100 bar flash lds 0,x are of 0x00,0x0000; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000; stDA: PTA0 as output ber send aix #1; H:X=1 br a start flash: lds f0x01 sta 0xfs08; set PON bit in FLCR lds 0xfs08; read FLBPR sta 0xfs08; r</pre>		
<pre>idux #0xdc00 start: sov #0x00,0x0004 ; DDRA: PTA0 as input ber row ; receive bit to be written in A sta 0x1000 ber flanh ida 0,x sov #0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) sov #0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) sov #0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) sov #0x01,0x0000 ; DDRA: PTA0 hi (in order to avoid glitches) sov #0x01,0x0000 ; DTA: PTA0 hi (in order to avoid glitches) sov #0x01,0x0000 ; DTA: PTA0 hi (in order to avoid glitches) sov #0x01,0x0000 ; DTA: PTA0 hi (in order to avoid glitches) sov #0x01,0x0000 ; DTA: PTA0 hi : STOP bit ber send six #1, #1:**1 bra start flash. ida #0x01 sto 0rfe08 ; read FLBPR ida %0x1 sto 0rfe08 ; read FLBPR ida %0x2 sto 0rfe08 ; fLCR : PCM bit=1, HVEM=1 ida %0x1 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x1 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; fLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; FLCR : PCM bit=0 ida %0x2 sto 0rfe08 ; FLCR : PCM bit=0 ida %0x2 st</pre>	mov #0x03,0x0007; DDRD: PID071 as output	
<pre>star: now forc0.0c000; t DDRA: PTA0 as input bar row; receive bit to be written in A sta Ox100 bar flash lda 0,x now f0x01.0c0000; FTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; j DTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; rTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; rTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; rTA: PTA0 hi (in order to avoid glitches) mow f0x01.0c0000; read tLBPA it a f0x1600; read tLBPA it a f0x160; read tLBPA it a 0x1600; read tLBPA ; EEQUIRED it a 0x1600; read tLBPA ; EEQUIRED it a 0x1600; read tLBPA ; EEQUIRED it a 0x1600; read tLBPA ; IEQUIRED it a 0x1600; re have put th dature to be programmed on stack *** it a</pre>		
ber rov ; receive bit to be written in A ta 0x10 ber flamb lda 0,x now f0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches) mow f0x01,0x0000 ; PTA: PTA0 hi : STOP bit ber send mix #1; H:X+11 bra start flamb: lda 40x01 ta 0xfs09 ; read FLBPA ; REQUIRED ta 0xfs09 ; FLCR : PCM bit=1, HVEM=1 tda 40x6 flaß x0xfs0 ; FLCR : PCM bit=1, HVEM=1 tda 40x6 ta 0xfs08 ; FLCR : PCM bit=1, HVEM=1 tda 40x6 ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; FLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; FLCR : PCM bit=0   to check : *** ta 0xfs08 ; SLCR : PCM bit=0   to check : *** ta 0xfs08 ; FLCR : PCM bit=0   to check : *** ta 0xfs08 ; FLCR : PCM bit=0   to check : *** ta 0xf		
<pre>sta Ox100 bar flash lda 0,x now f0x01,0x0000; PTA: PTA0 hi (in order to avoid glitches) now f0x01,0x0000; PTA: PTA0 hi (un order to avoid glitches) now f0x01,0x0000; PTA: PTA0 hi (store to avoid glitches) now f0x01,0x0000; PTA: PTA0 hi : STUP bit ber send aix #1; #:X+=1 bra start flash: lda #0x01 sta Oxfe08; set PCM bit in FLCR flash (lda G0x16) sta Oxfe08; set PCM bit in FLCR flash (lda G0x16) sta Oxfe09; read FLEPR ida #0x16 dsu2; dsu2 adsu2; 3 cycles =&gt; 1 us/boucle: 5 us delay ta 0x00; vr us/but let be for a to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to the bit i, value, and we use the output of rora to the carry bit to check if we rotated i &amp; futine salready bcc loopre; 3&gt; sum=8 or 11 cycles bra delay; bar delay; salt for stop bit rts da #0x06 sta Oxfe08; FLCR : FOM bit=0 lda #0x06 sta Oxfe08; FLCR : FOM bit=0 lda #0x06 sta Oxfe08; FLCR : FWE bit=0 lda #0x06</pre>		
ber flach ida 0,x mov #0x01,0x0000; FTA: FTA0 hi (in order to avoid glitches) mov #0x01,0x0000; FTA: FTA0 hi (in order to avoid glitches) mov #0x01,0x0000; FTA: FTA0 hi (in order to avoid glitches) mov #0x01,0x0000; FTA: FTA0 hi (in order to avoid glitches) ber delay pulx flach: Ida #0x01 flach: Ida #0x02 flach: Ida #0x02 flach: Ida #0x02 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x05, read Flach flach: Ida #0x04 flach: Ida #0x05 flach: Ida #0x05 flach: Ida #0x05 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x04 flach: Ida #0x05 flach:		
lad 0.7file: ber delaynov #0x01,0x0000; FTA: PTA0 hi (in order to avoid glitches)file: ber delaynov #0x01,0x0004; DDRA: PTA0 as outputber delayber sendber delayix #1; H:*+1ber delaybra startfile: ber delayflash: lda #0x01file: ber delaysta 0xfe06; set FOM bit in FLCRrcv: lda #0x80lda 0xfe09; read FLEPRrcv: lda #0x80sta 0xfe06; set FOM bit in FLCRrcv: lda #0x80lda 0xfe09; read FLEPR ; REQUIREDts 0xfe09; read fuely : Xdsus: dbnza dbus1; 3 cycles >1 us/boucle: 5 us delayber delay ; Xdsus: dbnza dbus2; 3 cycles >1 us/boucle: 10 us delayber delay ; Xlda 0x10is delayis chereaddbus2; dbnza dbus2; 3 cycles >1 us/boucle: 20 us delayis delaylda 40x00is dorfe08; FLCR: FOM bit=1, HEEM=1is dorfe08; fLCR: POM bit=0lda 40x00is dorfe08; fLCR: FCR bit=0is delayis dorfe08; fLCR: FCR bit=0iu /boucle: 20 us delayber delaylda 40x00sta 0xfe08; fLCR: FCR bit=0iu /boucle: 5 us delaylda 40x00is dorfe08; fLCR: FCR bit=0iu /boucle: 20 us delaylda 40x00sta 0xfe08; fLCR: FCR bit=0iu /boucle: 20 us delaylda 40x00sta 0xfe08; fLCR: FCR bit=0iu /boucle: 5 us delaylda 40x00sta 0xfe08; fLCR: FCR bit=0iu /boucle: 5 us delaylda 40x00sta 0xfe08; fLCR: FCR bit=0iu /boucle: 20 us delaylda 40x0sta 0xfe08; fLCR: FCR bit=0iu /boucle: 5 us delaylda 40x00sta 0xfe0		
<pre>mov #0x01,0x0000; FTA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) mov #0x01,0x000; DBA: PTA0 bi (in order to avoid glitches) fish: 1da #0xf1 sta 0xfe00; set PCM bit in FLCR 1da 40xf1; EQUIRED sta 0xfe00; read FLEPR; EQUIRED sta 0xfe00; FLCR : PCM bit=1, HVEN=1 1da 40x6 dfus1; dbnza dfus2; 3 cycles &gt;&gt; 1 us/boucle: 5 us delay 1da 40x14 dfus3; dbnza dfus2; 3 cycles &gt;&gt; 1 us/boucle: 20 us delay 1da 40x04 dfus3; dbnza dfus3; 3 cycles &gt;&gt; 1 us/boucle: 5 us delay 1da 40x05 sta 0xfe00; FLCR : PCM bit=0 1da 40x06 sta 0xfe00; FLCR : HEN bit=0 1da 40x06 sta 0xfe00; FL</pre>		
<pre>mov #0x01.0x0004; DDRA: PTA0 as output bsr send aix #1; H:X+=1 br start flash: lds #0x01 frash: lds #0x01 flash: lds #0x01 flash: lds #0x01 flash: lds #0x01 flash: lds #0x02 flash: lds #0x04 flash: lds #0x</pre>		
ber send aix #1 ; H:X+1 bra start flash: lda #0x01 sta 0xfe08 ; set PGM bit in FLCR lda %0xfe08 ; set PGM bit in FLCR lda %0xfe08 ; read FLBPR lda %0xfe08 ; read FLBPR lda %0xfe08 ; read FLBPR lda %0xfe08 ; read FLBPR ; REQUIRED sta 0xfe09 ; read FLBPR ; REQUIRED sta 0xfe08 ; FLCR : PGM bit=1, HVEN=1 lda #0x0a dSus2 : dbrzz dSus2 ; 3 cycles => 1 us/boucle: 10 us delay lda #0x0a dSus2 : dbrzz dSus2 ; 3 cycles => 1 us/boucle: 20 us delay lda #0x6 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x06 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x06 sta 0xfe08 ; S cycles => 1 us/boucle: 20 us delay lda #0x06 sta 0xfe08 ; S cycles => 1 us/boucle: 5 us delay lda #0x06 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x06 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x06 sta 0xfe08 ; S cycles => 1 us/boucle: 20 us delay lda #0x06 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x07 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x06 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x07 sta 0xfe08 ; FLCR : FGM bit=0 lda #0x06 sta 0xfe08 ; FLCR : FWM bit=0 lda #0x06 sta 0xfe08 ; FLCR : FWM bit=0 lda #0x07 sta 0xfe08 ; FLCR : FWM bit=0 lda #0x06		
aix #1; H:X*=1ber delaybra startpulxflash: !da #0x01rtsflash: !da #0x1rcv: !da #0x80ta 0xfe08; set PGM bit in FLCRrcv: !da #0x80!da 0xfe09; read FLEPRrcv: lda #0x80ta 0xfe09; read FLEPRrcv: lda #0x80sta 0xfe08; FLCR: PGM bit=1, HVEN=1loopr: bsr delay; Xlda #0x6bsr delay; Xdsu2: dbuza d5us1; 3 cycles => 1 us/boucle: 10 us delaybsr delay; Xlda #0x0a's times alreadydbus2: dbuza d5us2; 3 cycles => 1 us/boucle: 10 us delaysta times alreadylda #0x14bsr delay; is utif for stop bitdfus3: dbuza d5us2; 3 cycles => 1 us/boucle: 20 us delaytrtslda #0x04delay: pshx; 2+4 for bsr (12+2*((X*9)+15))*.3333=833lda #0x05lda #0x06; FLCR: FQM bit=0lda #0x06id #0x06; S cycles => 1 us/boucle: 5 us delaylda #0x06flcR: FQM bit=0lda #0x06id #0x06sta 0xfe08; FLCR: FQM bit=0lda #0x06id #0x06; S cycles => 1 us/boucle: 5 us delaylda #0x06id #0x06; S cycles => 1 us/boucle: 5 us delaylda #0x06id #0x06sta 0xfe08; FLCR: FQM bit=0lda #0x06id #0x06sta 0xfe08; FLCR: FQM bit=0lda #0x06id #0x06; S cycles => 1 us/boucle: 5 us delaylda #0x06id #0x06; S cycles => 1 us/boucle: 5 us dela		
bra startpulxflash: lda #0x01rtssta 0xfe08; set FQM bit in FLCRrtslda 0xfe09; read FLBPRrcv: lda #0x80lda 40xfe08; read FLBPRrcv: lda #0x80sta 0xfe09; read FLBPR; mov #0x03,0x0003,crvst; wait for START bitis 40xrfe09; read FLBPR; mov #0x03,0x0003,crvst; wait for START bitlda #05dsuf: dbmza d5us1; 3 cycles >> 1 us/boucle: 5 us delayda #05dsuf: dbmza d5us2; 3 cycles >> 1 us/boucle: 10 us delaylda #0x0; value, and we use the output of rora to the carry bit to the bitlda #0x0; value, and we use the output of rora to the carry bit to check if we rotatedd5us2: dbmza d5us2; 3 cycles >> 1 us/boucle: 10 us delay; bar delay;lda #0x06; dclay; xi for stop bitsta 0xfe08; FLCR: PGM bit=0index stop carry bit in these 2 lines: bor 11 cyclesbar delay; is at for stop bitbar delay;lda #0x06; dclay; si for stop bitrtsrtssta 0xfe08; FLCR: PGM bit=0delay: shr, 2*4 for bsr (12*2*((X*9)+15))*.3333=833lda #0x06id #0x06sta 0xfe08; FLCR: HVEN bit=0delay: pshr, 2*4 for bsr (12*2*((X*9)+15))*.3333=833lda #0x06id #0x06sta 0xfe08; FLCR: HVEN bit=0dbmzr loopr; 3lda #0x06idx #0x07sta 0xfe08; FLCR: HVEN bit=0dbmzr loopr; 3lda #0x01max, loopr; 3sta 0xfe08; FLCR: HVEN bit=0dbmzr loopr; 3lda #0x01gmark		
flash: lda #0x01''tssta 0xfe08 ; set PGM bit in FLCRrcv: lda #0x80lda 0xfe09 ; read FLBPR''rcv: lda #0x80lda 0xfe09 ; read FLBPR''rcv: lda #0x80sta 0xfe09 ; read FLBPR ; REQUIRED''rcv: lda #0x80sta 0xfe09 ; read FLBPR ; REQUIRED''rcv: lda #0x80sta 0xfe08 ; rLCR : PGM bit=1, HVEN=1''rcv: lda #0x16lda #0x6''rcv: lda #0x000, rcv0; 's branch if bit is clr => branch if 0, C=bitrcv: roa; i after 8 ror, carry=1''rcv: roa; i after 8 ror, carry=1sta 0xfe08 ; FLCR : PGM bit=1, HVEN=1''rcv0; roa; i after 8 ror, carry=1lda #0x14''rcv0; ''rcv0; ''rrcv0; ''r		
<pre>flash: lda #0x10 sta 0xfe08 ; set Qh bit in FLCR tda 0xfe08 ; set Qh bit in FLCR tda 0xfe08 ; set Qh bit in FLCR tda 0xfe09 ; read FLEPR tda 40xfe ; REQUIRED sta 0xfe09 ; read FLEPR ; REQUIRED sta 0xfe09 ; read FLEPR ; REQUIRED sta 0xfe09 ; read FLEPR ; REQUIRED sta 0xfe08 ; FLCR : PGM bit = 1, HVEN=1 tda 40xf  dsus: dbnza d5us2 ; 3 cycles =&gt; 1 us/boucle: 10 us delay tda 0xfe08 ; FLCR : PGM bit = 0 tda 40xfe sta 0xfe08 ; FLCR : PGM bit=0 tda 40xfe dsus2 ; dbnza d5us3 ; 3 cycles =&gt; 1 us/boucle: 20 us delay tda 40xfe sta 0xfe08 ; FLCR : PGM bit=0 tda 40xfe</pre>	bra start	
sta Oxfe08 ; set PGM bit in FLCRrcv: lda #0x60lda 0xfe09 ; read FLBPArcvst: brset #0, 40x0000, rcvst; wait for START bitlda #0xfe09 ; read FLBPA ; REQUIREDbsr delay ; wit balf a bit widthsta 0xfe09 ; read FLBPA ; stpUneDbsr delay ; wit balf a bit widthlda #0x60is or delay ; Xlda #0x60bsr delay ; Xlda #0x60bsr delay ; Xlda #0x60bsr delay ; Xlda #0x60bsr delay ; Xlda #0x9bsr delay ; Xsta 0xfe08 ; FLCR : PGM bit=1, HVEN=1bsr delay ; Xlda #0x0absr delay ; Xlda #0x14bsr delay ; Xlda #0x14bsr delay ; Xlda #0x60is times 2 lines: brcl sets the carry bit to the bitis ta 0xfe08 ; StCR : PGM bit=0los delaylda #0x60is times alreadylda #0x60bsr delay ; with for stop bitdfus3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 20 us delaylda #0x60rtssta 0xfe08 ; FLCR : PGM bit=0delay: pshr; 244 for bsr (12+2*((X*9)+15))*.333=833lda #0x64is taroff is ; 2 cycles => 1 us/boucle: 5 us delaylda #0x65id #0x66 ; 3 cycles :> 1 us/boucle: 5 us delaylda #0x66id #0x66 ; 3 cycles :> 1 us/boucle: 5 us delaylda #0x66id #0x66 ; 3 cycles :> 1 us/boucle: 5 us delaylda #0x66id #0x66 ; 3 cycles :> 1 us/boucle: 5 us delaylda #0x66id #0x66 ; 3 cycles :> 1 us/boucle: 5 us delaylda #0x66 ; 1d #0x66 ; 5 cycles :> 1 us/boucle: 5 us delaylda #0x66 ; 5 chnza d5us5 ; 3 cycles :> 1 us/boucle: 5 us delayl		rts
<pre>lda Oxfe09; read FLBPR lda 40xff ; REQUIRED sta Oxfe09; read FLBPR ; REQUIRED sta Oxfe08; FLCR : PGM bit=1, HVEN=1 lda 40x9 lda 40x0 lda 40x0a lda 40x0a sta Oxfe08; FLCR : PGM bit=1, HVEN=1 lda 40x0a sta Oxfe08; FLCR : PGM bit=0 lda 40x10 lda 40x0a sta Oxfe08; FLCR : PGM bit=0 lda 40x0a sta Oxfe08; FLCR : HVEN bit=0 lda 40x0b sta Oxfe08; FLCR : HVEN bi</pre>		
<pre>lda #0xf; REQUIRED sta 0xfe09; read FLBFP ; REQUIRED sta 0xfe09; read FLBFP ; REQUIRED sta 0x; write to any area of row *** lda #05 dfus1; dnza dfus1; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x0 dfus2; dnza dfus2; 3 cycles =&gt; 1 us/boucle: 10 us delay lda #0x0 dfus2; dnza dfus2; 3 cycles =&gt; 1 us/boucle: 10 us delay lda #0x0 dfus2; dnza dfus2; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dfus2; dnza dfus3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dfus2; dnza dfus3; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x04 dfus2; dnza dfus3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dfus2; dnza dfus3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x06 sta 0xfe08; FLCR: PGM bit=0 lda #0x06 dfus5; dnza dfus5; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x06 sta 0xfe08; FLCR: HVEN bit=0 lda #0</pre>		
<pre>sta Oxfe00; read FLBPA ; EEQUIRED sta 0,x ; write to any area of row *** lda #05 dsu0; rbmza dsus1 ; 3 cycles =&gt; 1 us/boucle: 5 us delay ddsu1: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 5 us delay ddsu2: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 10 us delay lda #0x04 dsu2: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x14 dsu32: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dsu32: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dsu32: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dsu32: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x14 dsu32: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dsu32: dbmza dsus2 ; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x04 dsu32: dbmza dsus5 ; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x04 dsu32: dbmza dsus5 ; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x05 lda #0x06 dsu2 dsu5 ; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x06 dsu 0; w have put he datum to be programmed on stack *** lda #0x04 dsu32: dbmza dsu5 ; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x06 lda #0x06 dsu 0; w have put he datum to be programmed on stack *** lda #0x14 dsu63: dbmza dsu5 ; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x06 lda #0x66 lda #0x66 lda #0x66 lda #0</pre>		
<pre>sta 0,x; write to any area of row *** lda #05 ddus1; dbnza d5us1; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x9 sta 0xfe08; FLCR : PGM bit=1, HVEN=1 lda #0x0 ddus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 10 us delay lda #0x0 ddus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 10 us delay lda #0x0 ddus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x0 ddus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x6 ddus3: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x6 ddus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x0 ddus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x0 ddus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x06 ddus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x06 lda #0x06 dfus5; dbnza d5us5; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0x06 sta 0xfe08; FLCR : HVEN bit=0 lda #0x00 sta 0xfe08; FLCR : HVEN bit=0 lda #0x01 </pre>		
lda #05       bsr.delay; X         dSus1: dbnza dSus1; 3 cycles => 1 us/boucle: 5 us delay       bsr.delay; X         lda #0x9       bsr.delay; X         sta 0xfe08; FLCR: PGM bit=1, HVEN=1       bsr.delay; X         lda #0x0a       ; 2 uses of carry bit in these 2 lines: brch sets the carry bit to the bit         dSus2: dbnza dSus2; 3 cycles => 1 us/boucle: 10 us delay       ; 2 uses of carry bit in these 2 lines: brch sets the carry bit to check if we rotated         ida #0x14       dSus3; 3 cycles => 1 us/boucle: 20 us delay       bsr delay; bsr delay; wait for stop bit         ida #0x08       cdsus3; 3 cycles => 1 us/boucle: 20 us delay       bsr delay; bsr delay; is referred to the carry bit to check if we rotated         ida #0x04       cdsus3; 3 cycles => 1 us/boucle: 20 us delay       bsr delay; bsr delay; bsr delay;       bsr delay         ida #0x06       cdsus5; 3 cycles => 1 us/boucle: 5 us delay       is ta for 8 tor (12+2*((X*9)*15))*.3333=833       ida #0x06         ida #0x06       cdsus5; 3 cycles => 1 us/boucle: 5 us delay       ida #0x06       ida #0x06       ida #0x06         ida #0x06       sta 0xfe08; FLCR: HWEN bit=0       delay: pshr; 2*4 for bsr (12+2*((X*9)*15))*.3333=833       ida #0x06         ida #0x06       sa; 3 =0x0f for 9600       nsa; 3 =0x0f for 9600       nsa; 3 =0x0f for 9600         ida #0x01       dbazz loopz; 3       j.3       curve for 9600		
dSus1: dbmza dSus1 ; 3 cycles ⇒ 1 us/boucle: 5 us delay lda #0x9 sto Xr608 ; FLCR : PGM bit=1, HVEN=1 lda #0x0a dSus2: dbmza dSus2 ; 3 cycles ⇒ 1 us/boucle: 10 us delay lda 0x100 ; we have put the datum to be programmed on stack **** sta 0,x ; ADDRESS TO BE WRITEN *** lda #0x14 dSus3: dbmza dSus3 ; 3 cycles ⇒ 1 us/boucle: 20 us delay lda #0x66 sto Xr608 ; FLCR : PGM bit=0 dsus2: dbmza dSus3 ; 3 cycles ⇒ 1 us/boucle: 5 us delay lda #0x66 sta 0xfe08 ; FLCR : PGM bit=0 lda #0x06 sta 0xfe08 ; FLCR : HVEN bit=0 lda #0x01 sta 0xfe08 ; FLCR : HVEN bit=0 lda #0x05 s		
<pre>lda #0x9 st 0xfe08; FLCR: PGM bit=1, HVEN=1 lda #0x0 dfue2: dbnza dfue5; 3 cycles =&gt; 1 us/boucle: 10 us delay dfue2: dbnza dfue5; 3 cycles =&gt; 1 us/boucle: 10 us delay dfue2: dbnza dfue5; 3 cycles =&gt; 1 us/boucle: 20 us delay dfue3: dbnza dfue5; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x06 st 0xfe08; FLCR: PGM bit=0 lda #0x06 st 0xfe08; fLCR: HVEN bit=0 lda #0x60 st 0xfe08; fLCR: HVEN bit=0 lda #0x60 st</pre>		
<pre>sta 0xfe08; FLCR: PGM bit=1, HVEN=1 ida #0x0a d5us2; dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 10 us delay dda 0x100; we have put the datum to be programmed on stack *** sta 0x, z; ADDRESS TO BE WRITTEN *** ida 0x14 d5us3: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay ida #0x14 d5us3: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay ida #0x06 sta 0xfe08; FLCR: PGM bit=0 ida #0x06 d5us5; dbnza d5us5; 3 cycles =&gt; 1 us/boucle: 5 us delay ida #0x06 sta 0xfe08; FLCR: HVEN bit=0 ida #0x00 sta 0xfe08; FLCR: HVEN bit=0 ida #0x00 sta 0xfe08; GtCR: HVEN bit=0 ida #0x00 sta 0xfe08; GtCR: HVEN bit=0 ida #0x00 sta 0xfe08; GtCR: HVEN bit=0 ida #0x01 sta 0xfe08; gtCR: HVEN bit=0 id</pre>		
<pre>lda #0x0a d5us2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 10 us delay d6us2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 10 us delay da 0x100; we have put the datum to be programmed on stack *** sta 0,x; ADDRESS TO BE WRITEN *** lda #0x14 d5us3: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 20 us delay lda #0x08 sta 0xfe08; FLCR: PGM bit=0 lda #0x06 lda #0x06 sta 0xfe08; 5 cycles =&gt; 1 us/boucle: 5 us delay lda #0x06 sta 0xfe08; FLCR: HVEN bit=0 lda #0x00 sta 0xfe08; FLCR: HVEN bit=0 lda #0x06 sta 0xfe08; FLCR: HVEN bit=0 lda #0x01 sta 0xfe08; FLCR: HVEN bit=0 lda #0x01</pre>		
dSus2: dbnza dSus2; 3 cycles => 1 us/boucle: 10 us delay       ; 8 times already         lda 0x100; we have put the datum to be programmed on stack ***       bcc looprc; 3> sum=8 or 11 cycles         sta 0,x; ADDRESS TO EE WRITTEN ***       bsr delay         lda #0x14       bsr delay         dSus3: dbnza dSus3; 3 cycles => 1 us/boucle: 20 us delay       tsr         lda #0x06       rts         sta 0,x; 008       delay: pakr; 2+4 for bsr (12+2*((X*9)+15))*.3333=833         lda #0x05       ldx #0h0f; 3         dSus5; 3 cycles => 1 us/boucle: 5 us delay       loopr: nsa; 3 => Xinit=0x88 for 1200         lda #0x06       nsa; 3       =0x0f for 9600         sta 0xfe08; FLCR: HVEN bit=0       dbnz: 100x; 3       104         lda #0x06       pulx; 2       100x; 1200		
<pre>lda 0x100 ; we have put the datum to be programmed on stack *** sta 0x10 ; we have put the datum to be programmed on stack *** sta 0x10 da #0x14 dfua3: dbnza dfus3 ; 3 cycles ⇒ 1 us/boucle: 20 us delay dfua5: dbnza dfus3 ; 3 cycles ⇒ 1 us/boucle: 20 us delay dfua5: dbnza dfus5 ; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5: dbnza dfus5 ; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/boucle: 5 us delay dfua5: dbnza dfus5; 3 cycles ⇒ 1 us/bou</pre>		
sta 0,x ; ADDRESS TO BE WRITTEN *** lda #0x14 dbu3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 20 us delay lda #0x06 sta 0xfe08 ; FLCR : PGM bit=0 lda #0x05 lda #0x05 sta 0xfe08 ; FLCR : HVEN bit=0 lda #0x06 sta 0xfe08 ; FLCR : HVEN bit=0 lda #0x01 sta 0xfe08 ; FLCR : HVEN bit=0 lda #0xfe08 ; FLCR : HVEN bit=0 lda #		
lda #0x14     bsr delay; wait for stop bit       d5us3: dbnza d5us3; 3 cycles => 1 us/boucle: 20 us delay     rts       1da #0x06     rts       sta 0xfe08; FLCR : PCM bit=0     delay: pshx; 2*4 for bsr (12*2*((X*9)*15))*.3333=833       1da #0x05     1dx #0h0f; 3       d5us5: dbnza d5us5; 3 cycles => 1 us/boucle: 5 us delay     loopx: nsa; 3       1da #0x00     nsa; 3       sta 0xfe08; FLCR: HVEN bit=0     dbnz: 100       1da #0x01     pulx; 2		
dSus3: dbnza dSus3; 3 cycles => 1 us/boucle: 20 us delay     rts       lda #0x06     delay: pshz; 2+4 for bsr (12+2*((X*9)+15))*.3333=833       lda #0x05     ldx #0h0f; 3       lda #0x06     ldx #0h0f; 3       lda #0x06     sta 0xfe08; FLCR: HVEN bit=0       lda #0x06     sta 0xfe08; FLCR: HVEN bit=0       lda #0x06     sta 0xfe08; FLCR: HVEN bit=0       lda #0x06     msa; 3       sta 0xfe08; FLCR: HVEN bit=0     dbuzz loopx; 3       lda #0x01     pulx; 2		
lda #0x08 sta 0xfe08 ; FLCR : PGM bit=0 lda #0x05 dfsus5: dbnza d5us5 ; 3 cycles ⇒> 1 us/boucle: 5 us delay lda #0x00 sta 0xfe08 ; FLCR : HVEN bit=0 lda #0x01 sta 0xfe08 ; FLCR : HVEN bit=0 lda #0x01 da #0x01 da #0x01 da #0x01 da #0x02 dbnzx 100px ; 3 pulx ; 2		
sta 0xfe08; FLCR: PGM bit=0     delay: pshx; 2+4 for bsr (12+2*((X*9)+15))*.3333=833       lda #0x05     1dx #0h0f; 3       dsu5: dbnza d5u5; 3 cycles => 1 us/boucle: 5 us delay     logy: nsa; 3       lda #0x00     nsa; 3       sta 0xfe08; FLCR: HVEN bit=0     dbnz: logy; 3       lda #0x01     gpulx; 2		rts
lda #0x05     ldx #0h0f; 3     104       d5us5: dbnza d5us5; 3 cycles => 1 us/boucle: 5 us delay     loopx: nsa; 3     => Xinit=0x88 for 1200       lda #0x00     nsa; 3     =0x0f for 9600       sta 0xfe08; FLCR: HVEN bit=0     dbnzx loopx; 3       lda #0x01     pulx; 2		
d5us5: dbnza d5us5 ; 3 cycles => 1 us/boucle: 5 us delay     loopx: nsa ; 3     => Xinit=0x88 for 1200       lda #0x00     nsa ; 3     =0x0f for 9600       sta 0xfe08 ; FLCR : HVEN bit=0     dbnzx loopx ; 3       lda #0x01     pulx ; 2		
lda #0x00         nsa; 3         =0x0f for 9600           sta 0xfe08; FLCR: HVEN bit=0         dbzzx loopx; 3           lda #0x01         pulx; 2		
sta 0xfe08 ; FLCR : HVEN bit=0 dbnzx loopx ; 3 lda #0x01 pulx ; 2		
lda #0x01 pulx ; 2		
dbus6: dbuza dbus6 ; 3 cycles => 1 us/boucle: 1 us delay rts ; 4	d5us6: dbnza d5us6 ; 3 cycles => 1 us/boucle: 1 us delay	rts;4
rts	rts	

flash\_write.asm: program to be executed from the 68HC908JB8 RAM for reading values on PTA0 at 9600 bauds and store them in flash memory (starting at 0xDC00).

start: ldhx #0x0140 ; TXS : (SP)<-(H:X)-1 => STACK=0x013f	ldx #0x08 ; snd through PTA0 the content of Acc (@9600)
txs ; reset stack pointer	mov #0x00,0x0000 ; PTA: PTA0 lo : START bit
; mov #0x01,CONFIG1 ; disable COP watchdog, CONFIG1=0x001f	loopsn: bsr delay ; X
	bsr delay ; X
mov #0x00,0x0003 ; PTD0/1 lo => LED lit	rora ; 1 rotate right Acc through carry
mov #0x03,0x0007 ; DDRD: PTD0/1 as output	bcc bit0 ; 3 branch if carry is clear (is A&1=0)
	mov #0x01,0x0000 ; 4 PTA0=hi
<pre>mov #0x01,0x0000 ; PTA: PTA0 hi (in order to avoid glitches)</pre>	bra bit1 ; 3
mov #0x01,0x0004 ; DDRA: PTAO as output	bit0: mov #0x00,0x0000 ; 4 PTA0=10
	bit1: dbnzx loopsn ; 3> sum=11 or 14 cycles
lda #Ohff	fin: bsr delay
bcl: bsr delay	bsr delay
nsa	mov #0x01,0x0000 ; PTA: PTA0 hi : STOP bit
nsa	bsr delay
dbnza bcl	bsr delay
	pulx
ldhx #OhdcOO ; STARTING ADDRESS TO BE READ (fcOO or dcOO)	rts
loop: lda 0h00,x	delay: pshx ; 2+4 for bsr (12+2*((X*9)+15))*.3333=833
bsr send ; send value of counter to serial port	ldx #0h0f ; 3 104
aix #1 ; increment counter	loopx: nsa ; 3 => Xinit=0x88 for 1200
cphx #0xffff	nsa ; 3 =0x0f for 9600
bne loop	dbnzx loopx ; 3
end: bra end	pulx ; 2
	rts ; 4
send: pshx	

flash\_read.asm: program to be executed from the 68HC908JB8 RAM for reading values in the flash memory (from 0xDC00 to 0xFFFF) and send them on PTA0 at 9600 bauds.

lda #0xff ; REQUIRED     d5usa: dbnza d5usa ; 3 cycles >> 1 us/boucle: 250 us delay       sta 0xfe09 ; read FLBPR ; REQUIRED     sta 0x; write to any area of row ***		
<pre>; mov #0x01,CONFIG1; disable COP watchdog, CONFIG1=0x001f ida #0xfa d5us6; dbnza d5us6; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us6; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us6; dbnza d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us6; dbnza d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us6; dbnza d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us6; dbnza d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us6; dbnza d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us6; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 100 us delay lda #0xfa d5us2; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 100 us delay lda #0xfa d5us2; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us2; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us2; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us2; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa d5us7; dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa</pre>	ldhx #0x0140 ; TXS : (SP)<-(H:X)-1 => STACK=0x013f	lda #0xfa
dSus6: dbnza dSus6; 3 cycles >> 1 us/boucle: 250 us delay         nov #0x00,0x0003; PTD0/1 lo => LED lit         mov #0x03,0x0007; DDRD: PTD0/1 as output         ldm #0xffe0         start: lda #0xf6         start: lda #0x6         sta 0xfe08; set ERASE and MASS bit in FLCR         lda 0xfe09; read FLBPR         lda #0xfa         dSus9: dbnza dSus9; 3 cycles => 1 us/boucle: 250 us delay         lda #0xfa         dSus1: dbnza dSus1; 3 cycles => 1 us/boucle: 250 us delay         lda #0x6         sta 0xfe09; read FLBPR         lda #0x6         sta 0xfe09; read FLBPR         lda #0x6         sta 0xfe09; read FLBPR         lda #0x6         sta 0xfe08; FLCR: ERASE, MASS, HVEN=1         lda #0x6         dSus1: dbnza dSus2; 3 cycles => 1 us/boucle: 5 us delay         lda #0x6         dSus2: dbnza dSus2; 3 cycles => 1 us/boucle: 250 us delay         lda #0x6         dSus2: dbnza dSus2; 3 cycles => 1 us/boucle: 250 us delay         lda #0x6         ldsus2: dbn	txs ; reset stack pointer	d5us5: dbnza d5us5 ; 3 cycles => 1 us/boucle: 250 us delay
<pre>mov #0x00,0x0003; PTD0/1 lo =&gt; LED lit mov #0x03,0x0007; DDRD: PTD0/1 as output lda #0xfa dfus7; dbuza dfus2 dfus7; dbuza dfus7; dbuza dfus7; dbuza dfus8; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus8; set ERASE and MASS bit in FLCR dfus1; dbuza dfus9; set ERASE and MASS bit in FLCR lda #0xfa dfus9; read FLEPR ; REQUIRED sta 0xfe08; read FLEPR ; REQUIRED sta 0xfe09; read FLEPR ; REQUIRED sta 0xfe08; fLCR : ERASE, MASS, HVEN=1 lda #0xfa dfus2; dbuza dfus2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus2; dbuza dfus2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus2; dbuza dfus2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus2; dbuza dfus2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus2; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus2; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus2; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus2; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus3; dbuza dfus3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa</pre>	; mov #0x01,CONFIG1 ; disable COP watchdog, CONFIG1=0x001f	lda #0xfa
<pre>mov #0x03,0x0007; DDRD: PTD0/1 as output d5us7: dbnza d5us7; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus8: dbnza d5us8; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfus9; read FLBPR dfus1: dbnza d5us1; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf sta 0xf ov #v** lda #0xf dfus1: dbnza d5us1; 3 cycles =&gt; 1 us/boucle: 5 us delay lda #0xf dfus1: dbnza d5us1; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus1: dbnza d5us1; 3 cycles =&gt; 1 us/boucle: 100 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 100 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 100 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 100 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us2; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d5us3; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfus2: dbnza d</pre>		d5us6: dbnza d5us6 ; 3 cycles => 1 us/boucle: 250 us delay
<pre>lda #0xfa dfuss: dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 100 us delay lda #0xf dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa dfuss; dbnza dfuss; s cycles =&gt; 1 us/boucle: 250 us delay lda #0xfa</pre>	mov #0x00,0x0003 ; PTD0/1 lo => LED lit	lda #0xfa
ldh #0xffe0     d5u88: dbnza d5u88; 3 cycles >> 1 us/boucle: 250 us delay       start: lda #0x6     d5u82: dbnza d5u89; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 100 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 100 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 100 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     d5u82: dbnza d5us9; 3 cycles >> 1 us/boucle: 250 us delay       lda #0xfa     sta foxfe08; FLCR : MASS=1, HVEN=0, ERASE=0       lda #0xfa     mov #0x03,0x0003; PTD0/1 lo >> LED lit       lda #0xfa     end: bra end	mov #0x03,0x0007 ; DDRD: PTD0/1 as output	d5us7: dbnza d5us7 ; 3 cycles => 1 us/boucle: 250 us delay
start: lda #0x06 sta 0xfe08; set EASE and MASS bit in FLCR lda #0xfa lda #0xfa; set EASE and MASS bit in FLCR lda 0xfe09; read FLBPR lda 40xfa; REQUIRED sta 0xfe08; read FLBPR; REQUIRED sta 0xfe08; read FLBPR; REQUIRED sta 0xfe08; read FLBPR; REQUIRED sta 0xfe08; FLCR: EASE, MASS, HVEN=1 lda #0x6 dfus1: dhuza dfus1; 3 cycles => 1 us/boucle: 5 us delay lda #0x6 dfus1: dhuza dfus1; 3 cycles => 1 us/boucle: 5 us delay lda #0x6 sta 0xfe08; FLCR: EASE, MASS, HVEN=1 lda #0x6 dfus2; dhuza dfus2; 3 cycles => 1 us/boucle: 100 us delay lda #0x6 dfus2; dhuza dfus2; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dfus3; dhuza dfus2; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dfus3; dhuza dfus3; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dfus3; dhuza dfus3; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dfus3; dhuza dfus3; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dfus3; dhuza dfus3; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa	-	lda #0xfa
<pre>sta Oxfe08 ; set ERASE and MASS bit in FLCR lda 0xfe09 ; read FLBPR lda 0xfe09 ; read FLBPR lda #Oxfe09 ; read FLBPR  ; REQUIRED sta 0xfe09 ; read FLBPR ; REQUIRED sta 0xfe09 ; read fLBPR ; REQUIRED sta 0xfe09 ; read fLBPR ; REQUIRED lda #Oxf lda #Oxf lda #Ox lda #Ox lda #Ox lda #Oxfe08 ; FLCR : ERASE, MASS, HVEN=1 lda #Oxf lda #Oxf dSus2: dbnza dSus2 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus2: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250 us delay lda #Oxfa dSus3: dbnza dSus3 ; 3 cycles =&gt; 1 us/boucle: 250</pre>	ldhx #0xffe0	d5us8: dbnza d5us8 ; 3 cycles => 1 us/boucle: 250 us delay
lda 07fe09; read FLBPR     lda #0xfa       lda #0xff; REQUIRED     dfusa: dbnza d5usa; 3 cycles => 1 us/boucle: 250 us delay       sta 0xfe09; read FLBPR ; REQUIRED     lda #0xc       sta 0xfe09; read fLBPR ; REQUIRED     lda #0xc       lda #0x     lda #0xc       lda #0xe     lda #0xe       sta 0xfe08; FLCR : ERASE, MASS, HVEN=1     lda #100       dsus: dbnza d5us2 ; 3 cycles => 1 us/boucle: 250 us delay     lda #100       lda #0xfa     lda #0xfa       lda #0xfa     lda #0xfa       lda #0xfa     sta 0xfe08; FLCR : MASS, HVEN=1       lda #0xfa     mov #40x03,0x0003; FDD0/1 lo => LED lit       lda #0xfa     edi bra end	start: 1da #0x06	lda #0xfa
lda #0xff ; REQUIRED     d5usa: dbnza d5usa ; 3 cycles >> 1 us/boucle: 250 us delay       lda #0x6     ida #0xc       lda #0x6     sta 0xfe08 ; FLCR : MASS, HVEN=1, ERASE=0       lda #0x6     ida #100       d5us1: dbnza d5us2 ; 3 cycles >> 1 us/boucle: 250 us delay     ida #100       lda #0x6     ida #0x6       d5us2: dbnza d5us2 ; 3 cycles >> 1 us/boucle: 250 us delay     ida #0x6       lda #0x6     ida #0x6       ida #0x7a     ida #0x7       ida #0x7a     ida #0x6       ida #0x6     ida #0x6       ida #0x7     ida #0x6       ida #0x6     ida #0x6       ida #0x7a     ida #0x7       ida #0x7a     ida #0x8       ida #0x7a     ida #0x8	sta OxfeO8 ; set ERASE and MASS bit in FLCR	d5us9: dbnza d5us9 ; 3 cycles => 1 us/boucle: 250 us delay
sta 0xfe09; read FLBPR ; REQUIRED sta 0x ; write to any area of row *** lda #05 dGus1: dbnza d5us1 ; 3 cycles >> 1 us/boucle: 5 us delay lda #0xe sta 0xfe08 ; FLCR : MASS, HVEN=1, ERASE=0 lda #100 dSus2: dbnza d5us2 ; 3 cycles => 1 us/boucle: 100 us delay lda #0xfa dSus2: dbnza d5us2 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dSus3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dSus3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dSus3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa dSus3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa	lda OxfeO9 ; read FLBPR	lda #0xfa
sta 0,x ; write to any area of row ***     lda #0xc       lda #05     sta 0xfe08 ; FLCR : MASS, HVEN=1, ERASE=0       lda #0xe     lda #100       sta 0xfe08 ; FLCR : ERASE, MASS, HVEN=1     lda #100       lda #0xfa     d5usb: dbnza d5usb ; 3 cycles => 1 us/boucle: 100 us delay       lda #0xfa     lda #0xfa       d5us2: dbnza d5us2 ; 3 cycles => 1 us/boucle: 250 us delay     lda #0xfa       lda #0xfa     mov #0x03,0x0003 ; PTD0/1 lo => LED lit       lda #0xfa     end: bra end	lda #0xff ; REQUIRED	d5usa: dbnza d5usa ; 3 cycles => 1 us/boucle: 250 us delay
lda #05       sta 0xfe08; FLCR: MASS, HVEN=1, ERASE=0         d5us1: dbnza d5us1; 3 cycles => 1 us/boucle: 5 us delay       lda #100         da #0xfa       d5us2: dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay         lda #0xfa       lda #0xfa         d5us2: dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay       lda #0xfa         lda #0xfa       sta 0xfe08; FLCR: MASS=1, HVEN=0, ERASE=0         lda #0xfa       mov #0x03,0x0003; PTD0/1 lo => LED lit         lda #0xfa       end: bra end	sta OxfeO9 ; read FLBPR ; REQUIRED	
d5us1: dbnza d5us1; 3 cycles => 1 us/boucle: 5 us delay     Ida #100       Ida #0xe     d5us2: dbnza d5us2; 3 cycles => 1 us/boucle: 100 us delay       Ida #0xfa     Ida #0x4       d5us2: dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay     Ida #0xfa       Ida #0xfa     us4       d5us3: dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay     Ida #0xfa       Ida #0xfa     us0x1; dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay       Ida #0xfa     us0x1; dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay       Ida #0xfa     us0x1; dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay	sta 0,x ; write to any area of row ***	lda #0xc
lda #0xe     lda #100       sta 0xfe08; FLCR: ERASE, MASS, HVEN=1     d5usb: dbnza d5usb; 3 cycles => 1 us/boucle: 100 us delay       lda #0xfa     lda #0x4       d5us2: dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay     lda #0xfa       lda #3xfa     sta 0xfe08; FLCR: MASS=1, HVEN=0, ERASE=0       lda #3xia     mov     #0x03,0x0003; PTD0/1 lo => LED lit       lda #0xfa     end: bra end	1da #05	sta OxfeO8 ; FLCR : MASS, HVEN=1, ERASE=0
sta 0xfe08 ; FLCR : ERASE, MASS, HVEN=1 d5usb : 3 cycles => 1 us/boucle: 100 us delay lda #0xfa d5us2 : d5us2 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa d5us3 : d5us2 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa d5us3 : d5us3 ; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa mov #0x03,0x0003 ; PTD0/1 lo => LED lit end: bra end	d5us1: dbnza d5us1 ; 3 cycles => 1 us/boucle: 5 us delay	
lda #0xfa d5us2: dbnza d5us2; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa d5us3: dbnza d5us3; 3 cycles => 1 us/boucle: 250 us delay lda #0xfa mov #0x03,0x00003; PTD0/1 lo => LED lit end: bra end	lda #0xe	lda #100
d5us2: dbnza d5us2 ; 3 cycles => 1 us/boucle: 250 us delay     sta 0xfe08 ; FLCR : MASS=1, HVEN=0, ERASE=0       1da #0xfa	sta OxfeO8 ; FLCR : ERASE, MASS, HVEN=1	d5usb: dbnza d5usb ; 3 cycles => 1 us/boucle: 100 us delay
lda #0xfa         mov #0x03,0x00003; PTD0/1 lo => LED lit           da #0xfa         end: bra end	lda #0xfa	lda #0x4
d5us3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 250 us delay mov #0x03,0x0003 ; PTD0/1 lo => LED lit lda #0xfa end: bra end	d5us2: dbnza d5us2 ; 3 cycles => 1 us/boucle: 250 us delay	sta 0xfe08 ; FLCR : MASS=1, HVEN=0, ERASE=0
lda #0xfa end: bra end	lda #Oxfa	
	d5us3: dbnza d5us3 ; 3 cycles => 1 us/boucle: 250 us delay	mov #0x03,0x0003 ; PTD0/1 lo => LED lit
d5us4: dbnza d5us4 ; 3 cycles => 1 us/boucle: 250 us delay	lda #Oxfa	end: bra end
	d5us4: dbnza d5us4 ; 3 cycles => 1 us/boucle: 250 us delay	

flash\_erase.asm: program to be executed from the 68HC908JB8 RAM for block erasing the whole flash memory (data and interrupt vector table).

Commands: in order to erase the memory: ./hc08flash flash\_erase.out

in order to write the program blink.out to flash memory: ./hc08flash flash\_write.out blink.out in order to check that the data were properly written: ./hc08flash flash\_read.out > t, which reads flash and ROM memories (from 0xDC00 to 0xFFFF) and stores the result in file t (for further verification of the memory content).

Of course these three steps can be automated by being combined in a shell script.

The programs flash\_write.asm and flash\_erase.asm do not use the subroutines provided in the ROM of the 68HC908JB8 since I was not able to figure out how to make them work.

### 6 USB communication

Now that we know how to store a program in flash memory, we can consider testing the sample software provided by Motorola with its evaluation board. The program is provided both in source format and compiled to an S19 format, usb08.s19. After converting this S19 file to a raw list of opcodes and data in hexadecimal format (by removing the S1 header and the address word at the beginning of each line, as well as the checksum byte at the end of each line), we store this program to flash memory by executing:

hc08\_flash flash\_erase.out in order to clear the flash memory

hc08\_flash flash\_write.out usb08.flash in order to write the new program to flash memory

 $hc08_flash flash_read.out > usb08.dump$  in order to read the content of the flash memory and check that the new program is indeed stored there

hc08\_flash flash\_irq.out usb08.irq in order to program the interrupt vector.

flash\_irq.asm: program to be executed from the 68HC908JB8 RAM for programming the interrupt vector area (0xFFF0-0xFFFD).

For the last step, we have created a short file containing the hexadecimal values to be stored in the interrupt vectors located in 0xFFF0 to 0xFFFD. We must be careful *not* to write to 0xFFFE:FFFF or the microcontroller will no longer enter monitor mode when powered on (flash\_irq.asm should never write to locations 0xFFFE:FFFF, even if the interrupt vector for this location is included in the file usb08.irq).

Since we now wrote in the interrupt vectors space, we must adapt the security bytes sent during initialization of the monitor mode (as done by the function void init\_hc08mon(int ) in the HC08 library hc08.c). We could until now always send the security bytes 0xFF since the interrupt vectors were never set. However, if we now try to read the content of flash memory using hc08\_flash flash\_read.out, we end up only reading the same value (0xAD in my case). We must provide the proper values for the security bytes in order to be able to read the content of flash memory and execute the program stored there. Indeed, by adapting the function init\_hc08mon to send the same bytes as the ones read in usb08.irq, we can again read the content of the flash memory and thus execute it content (by running hc08\_flash without argument, which will automatically call the program starting at 0xDC00, beginning address of the flash memory).

#### 6.1 First USB tests under MS-Windows

Motorola provides with its USB development board a driver and software for controlling their microcontroller. This software requires a computer running MS Windows98 at least.

Running the demo version of USBIO provided by Thesycon while the board is connected through its USB link to the PC leads to an "USBIO Installation wizard" which properly lists our microcontroller as

Manufacturer: The sycon

Descritpion: USBIO Device: VID=0C70 PID=0000 Hardware ID: USB

VID\_0C70&PID\_0000&REV\_0100

after using the driver found in usbio\_lt.sys.

In the Control Panel/System Properties one can indeed see the USBIO Device with the same VID and PID appear each time the board is plugged in the USB port (while the Motorola software is running on the microcontroller).

However, when running the demo application provided by Motorola (io08usb.exe) I get the error message "Couldn't open port! Please restart application to retry!". The computer I tried this on was equipped with an Intel 82801AA USB Universal Host Controller.

#### 6.2 USB development under linux

Our objective was to provide a linux device driver and a user mode application running directly with the microcontroller USB example provided by MCT. We used the free evaluation version of the ICC compiler under Windows98 to generate the S19 file out of the C source code. The resulting S19 file was flashed in the HC908JB8 using the linux programming code following the procedure presented earlier in this document.

#### 6.2.1 The linux (kernel 2.4.x) driver

hc08.c: linux kernel 2.4.x driver for communicating with the 68HC908JB8 microcontroller running the example C program provided by MCT.

#### 6.2.2 The user client

start\_hc08: shell script for loading the module and generating the (two) appopriate /dev entries for communicating with the
microcontroller (up to two microcontrollers supported). This script is not to be run if devfs is used (ie the devfsd daemon is
running).

hc08\_user\_write: user programm for writing values to the microcontroller. The LED (connected to PTA0/1) should light if 0xff is sent, and should switch off is 0x00 is sent.

hc08\_user\_read.c: user programm for writing values to the microcontroller and at the same time displaying the status of keys connected to PTE.