SGS-THOMSON MICROELECTRONICS

APPLICATION NOTE

ST9058 MICROCONTROLLER PLL CLOCK APPLICATION NOTE AND DEMOBOARD

By Olivier Garreau

INTRODUCTION

The objective of this Application Note is to present the technical features of the ST9058 clock generator based on a Phase Locked Loop (PLL) circuit.

Included is also the presentation of a simple application that features the PLL clock and illustrates the programming of this clock generator.

The hardware of this application note is the ST9058 PLL Demoboard that demonstrates the capabilities of the ST9058 running in low power mode and at low frequency.

NEW TECHNICAL CHARACTERISTICS OF THE ST9058

The ST9058 is the first ST9 to include a very low-power operating mode.

In addition to being the first microcontroller of the ST9 family to include a large internal ROM (62 Kbytes), the ST9058 also works in the 5-volt supply range. As in all microcontrollers, lower power consumption can only be achieved by lowering the draw of current and consequently the operating frequency.

In this special operating mode, all the ST9 core and peripherals run at a frequency in the range of 90 to 150 kHz. (90 for a 3Mhz external crystal and 150 for a 5Mhz one).

This can be compared to the former unique frequency range 2 to 16 Mhz.

The power consumption ratio between maximum and minimum speed can reach 50. For example, with an external crystal of 3.2 Mhz, speeds as low as 100 kHz and as high as 16 Mhz are attainable, which gives a ratio of 160 for the frequency.

INDUSTRIAL APPLICATIONS

Since this microcontroller can use a battery-based power supply, this very integrated and flexible ST9 (ST9058) is highly suitable for low-power applications like automotive applications or portable consumer applications. While executing an optimized application software, the ST9058 can require less than 4 milliwatts (average power consumption in sleep mode), which is easily compliant with consumer and automotive markets.

Table of Contents

1 DESCRIPTION OF THE ST9058 PLL

As described in the ST9058 Datasheet, the Phase Locked Loop circuit is part of the clock generator. This generator will feed the clock to the ST9 core and to the ST9 peripherals.

“CPUCLK” designates the core clock.

“INTCLK” designates the peripherals clock.

“CK” designates the output signal out of the clock multiplier/divider.

“XTAL” designates the frequency of the external crystal.

The PLL is the multiplying stage of the clock generator: it will multiply by a fixed preset value (6 or 10). Then a couple of dividers intervene to reduce the output frequency of the PLL.

Four (4) dividing stages will generate CPUCLK; three (3) dividing stages will generate INTCLK.

In total, there is one bit to multiply and 8 bits to divide. That gives theoretically 512 (2^9) different ST9 core operating speeds.

Of course, some of the speeds are not compatible with the performance of the PLL or the core, but the PLL works like a powerful “gearbox” that can be shifted anytime to any new ratio.

The two equations that govern CPUCLK and INTCLK are:

PLL ACTIVATED:

CK = XTAL / DIV2 * MX / DX / LP

INTCLK = CK

CPUCLK = (CK / PRS) * FLAG

PLL BEING ACTIVATED:

CK = XTAL / DIV2

INTCLK = CK

CPUCLK = (CK / PRS) * FLAG

PLL DEACTIVATED (WFI LOW POWER MODE):

CK = XTAL / DIV2 / 16

INTCLK = CK

CPUCLK = 0

with

3 Mhz < XTAL < 5 Mhz	(external crystal)
DIV2 = 2	(this is recommended to operate the PLL)
MX = {6,10}	(multiplying constant: 6 or 10)
DX = {1,2,3,4,5,6,7}	(PLL output clock divider)
LP = {1, 16}	(special divider for very low power mode)
CK < 16 Mhz	(speed limitation of the actual core & peripherals)
PRS = {1,2,3,4,5,6,7}	(Prescaler of the core clock)
FLAG = {0,1}	(Flag that can stop the core in WFI or ext bus req.)

Here is a concrete example with an 4.4112 Mhz external crystal:

PLL ACTIVATED: MODE PLL

XTAL = 4.4112 Mhz

DIV2 = 2

MX = 10

DX = 2

LP = 1

so **INTCLK = 11.028 Mhz**

PLL BEING ACTIVATED: MODE XTAL/2

DIV2 = 2

so **INTCLK = 2.2056 Mhz**

PLL DEACTIVATED (WFI LOW POWER MODE): MODE XTAL/32

DIV2 = 2

so **INTCLK = 0.13785 Mhz (137.85 kHz)**

2 ALGORITHM FOR USING THE PLL AND LOW POWER MODE.

We can distinguish three working modes for the ST9:

– The high speed mode is when the PLL is ACTIVATED and used by the core. In that mode, the user can set the multiplying factor to 6 or 10 (**MODE PLL**).

– The middle speed mode is when the PLL is being ACTIVATED and in the process of being locked, the core can then run at XTAL/2 (**MODE XTAL/2**).

– The low speed mode (like for the Wait For Interrupt low power mode), then the core and the peripherals can run as low as XTAL/32 (**MODE XTAL/32**).

These modes are precisely described page 62/202 of the ST9058 datasheet.

The most effective mode in terms of power consumption is the WFI (Wait For Interrupt) mode set with the Low Power option (WFI-LP).

What interests us in this application note is the way the ST9 switches from one mode to another. These transitions are given on page 64/202 of the ST9058 datasheet which gives the theoretical waveforms.

The algorithm for using the PLL is very simple:

1 configure the modes (low-power, Multipliers, Divisors etc...)
2 enable the PLL
3 wait for the PLL to be locked up by polling a status bit
4 run the application at the PLL speed.
5 enter anytime the WFI Low-Power mode if requested

A couple of transitions have to be triggered by the user, some other transitions are automatic in the ST9058. The state diagram below shows these transitions:

state 1: reset -> XTAL /2 (automatic) (XTAL/2 MODE)

state 2: the user can set the PLL up (PLL MODE)

state 3: the user can enter the WFI LP mode (XTAL/32 MODE)

state 4: one interrupt exit from WFI mode to XTAL/2 MODE (automatic)

state 5: after around 100 uS, the PLL MODE is selected (automatic)

state 6: the user should get back to state 3 with WFI instruction

3 PRACTICAL RESULTS OVER THE POWER CONSUMPTION

All evaluations have been performed on a prototype of the ST9058 PLL and I2C demoboard.

The crystal used is a 4.4112 Mhz crystal.

The power supply is a set of 4 battery sticks of 1.5 volt, 500mA/h (AA BATTERY).

As explained in the table below, this new clock generator, based on an improved oscillator and the PLL, is a large improvement for low-power applications. The reduction of supply current is dramatic as measured on a prototype board with a prototype ST9058.

This current was measured with an ammeter inserted between the battery and VDD; this is the purpose of the jumper JP1 (refer to the schematics page 14).

VDD=5.3 Volts
QX = 4.4112MHz
TEMP= 70F
	ST9 board
	LCD OFF
	WFI-LP(137khz)
		ST9 board
		LCD ON
		WFI-LP(137khz)
			ST9 in PLL
			mode (11.028MHz)

DC CURRENT(mA)	0.65	3.6	30
DC POWER (mW)	3.44	19	160

It is clear that automotive and portable applications can benefit from these new Low-Power modes. Even real-time applications can use the ST9058 as it has a fast interrupt response time in that mode: the application software can be serviced after a few micro-seconds after an interrupt (IT) occurs.

The ST9058 will also fit well in any automotive application: while the car engine is off, the ST9058 draws very minimal current and can service background slow tasks. While the car engine is on, the power consumption of the uC is not so critical, and the device can then execute fast tasks. The global power consumption will be the average of the time spent in low-power WFI mode and in PLL (high-speed) mode.

Below are shown current draw time waves for a simple application executed on the ST9058 PLL demoboard: a real-time clock with alarm capability.

These plot have been obtained by inserting a 1-ohm resistor instead of JP1, it represents the voltage through this resistor (one milli-volt means one milli-amp source by the power supply). This is represented by the bottom trace. The top trace represents CPUCLK.

Figure 1 shows the microcontroller is ON once per second and for approximately 200 uS. For that time, the current draw is important but is very minimal when averaged over one second. Figure 2 is a close-up of what happens during this 200 uS. From this figure, the Idle mode, then the XTAL/2 mode, and then the PLL MODE (higher consumption) can be easily distinguished. Figure 3 displays the actual difference between the mode at 2.2 Mhz and that at 11 Mhz.

Figure 1.

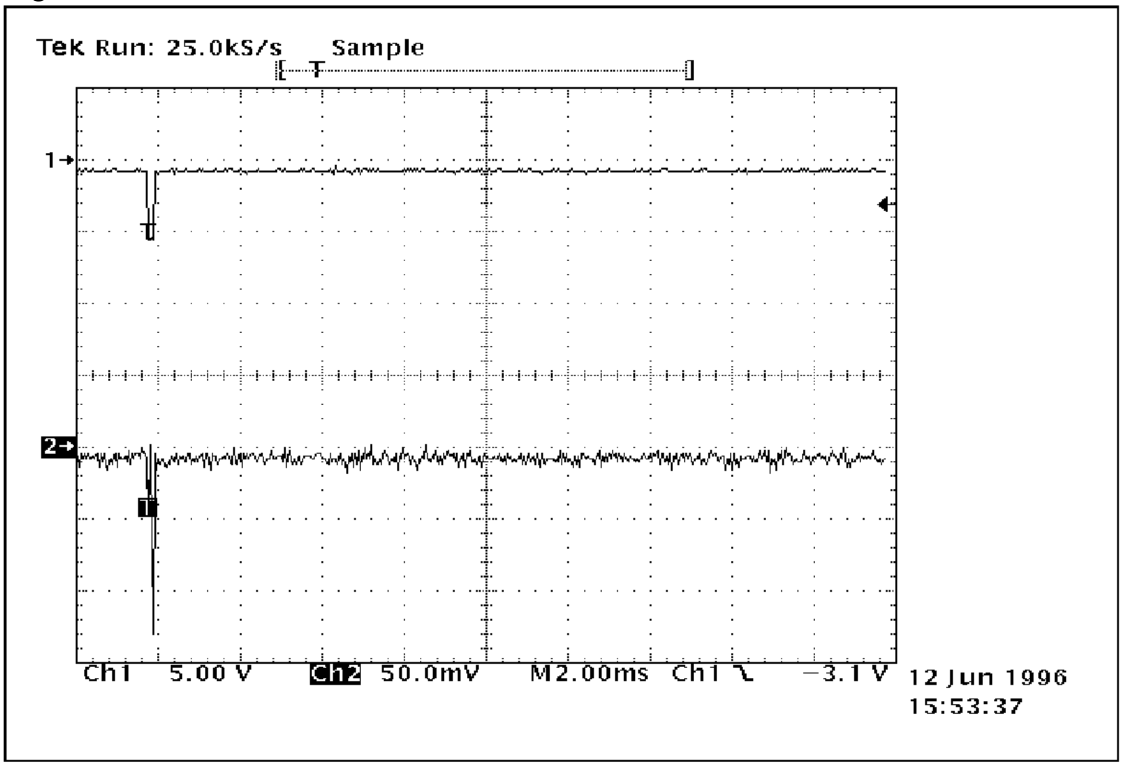


Figure 2.

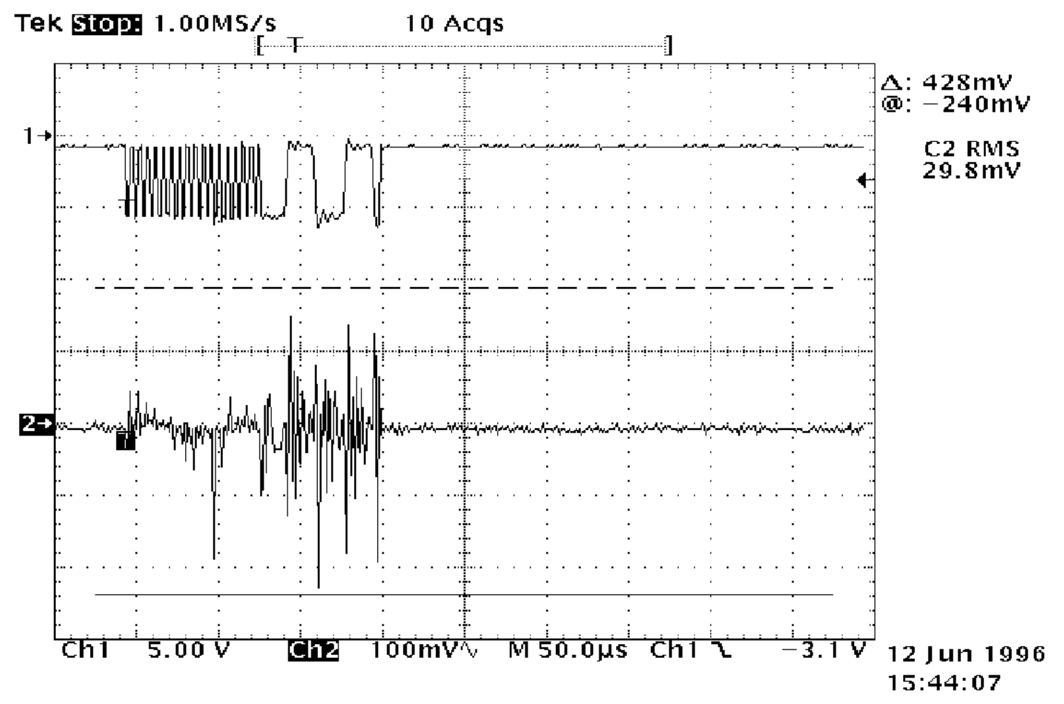
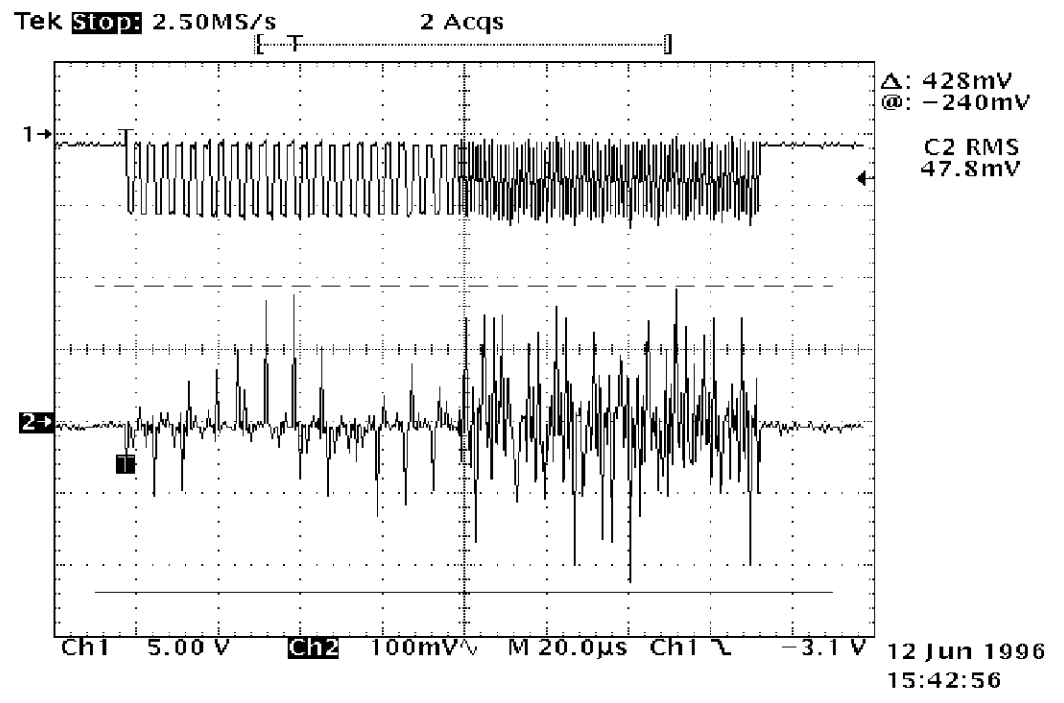

Tek Stop: 1.00MS/s 10 Acqs
Δ: 428mV
@: −240mV
C2 RMS
29.8mV
1→
2→
Ch1 5.00 V Ch2 100mV∿ M 50.0µs Ch1 ⌍ −3.1 V
12 Jun 1996
15:44:07

Figure 3.


Tek Stop: 2.50MS/s 2 Acqs
Δ: 428mV
@: −240mV
C2 RMS
47.8mV
1→
2→
Ch1 5.00 V Ch2 100mV∿ M 20.0µs Ch1 ⌍ −3.1 V
12 Jun 1996
15:42:56

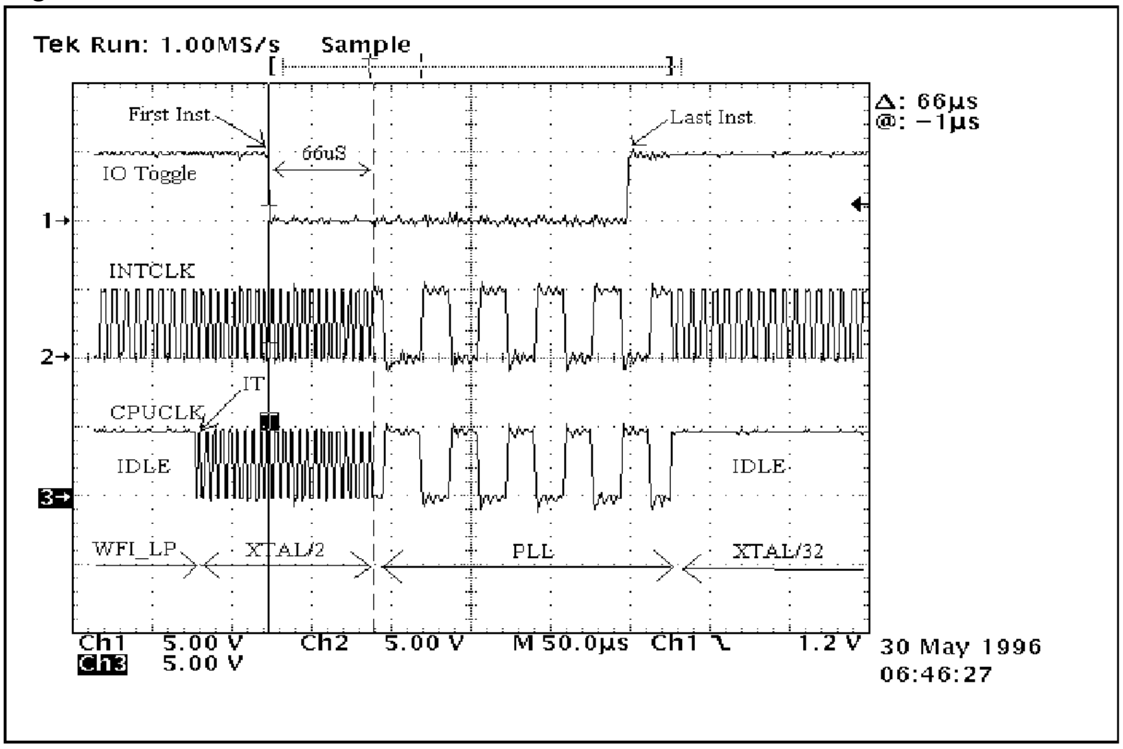
4 REAL BEHAVIOUR OF THE ACTUAL PLL

The ST9058 Datasheet describes on page 64/202, figure 6-4, how the ST9058 gets out of the WFI -LP mode and how the PLL achieves synchronization.

In the next paragraph, we expose more in details the sequences to get in or out of any of the three modes (PLL, XTAL/2, XTAL/32)

Figure 4 gives a snapshot of the behaviour of the PLL in practical conditions (with an external crystal of 4.4112 Mhz). It shows an interrupt service routine in WFI Low-Power mode, then in XTAL/2 mode, then in XTAL/32 mode, and then back in Low-Power mode:

Figure 4.



The three traces show an IO port toggled by software (trace 1), INTCLK (trace 2) and CPUCLK (trace 3). We can recognize the three phases:

1) WFI_Low-Power: the core is stopped, the peripherals run at 137.85 Khz,

2) one IT is generated by the Multi Function Timer, simultaneously the core and peripherals switch to XTAL/2 (2.2 Mhz). Then the first software instruction (First Inst.) is executed and toggles the IO (trace 1),

3) after 66 micro-seconds, the PLL is locked up and the core runs at high speed (11.028Mhz), then the last instruction before the WFI instruction toggles the IO back

to level '1'. Finally the core stops and the peripheral clock (INTCLK) switches to XTAL/32.

(In figure 4 the mode PLL has been sub-sampled by the oscilloscope and takes on the appearance of low rather than high frequency). The figures 5 and 6 describe the beginning and the end of this interrupt service routine and give the precise times for entering the awake mode and returning to the sleep mode of the core:

Figure 5 shows the effect of an IT on the core.

Figure 6 shows the process of returning to the WFI Low-Power mode.

Figure 5.

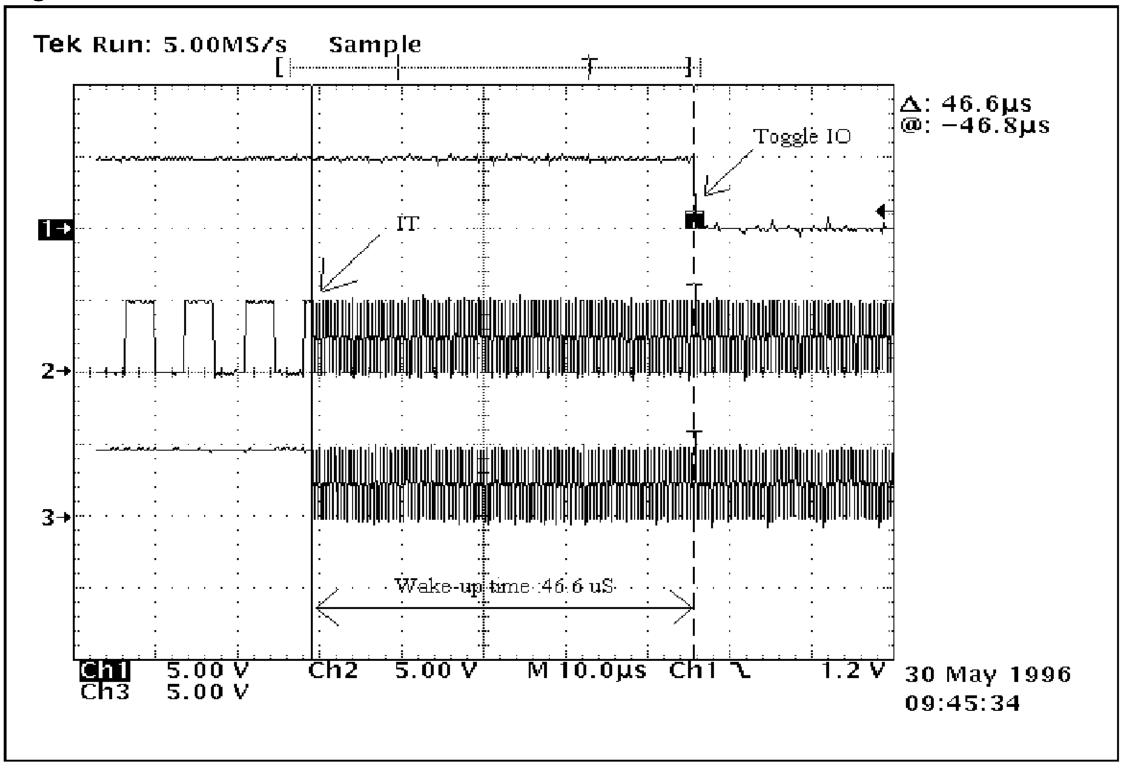
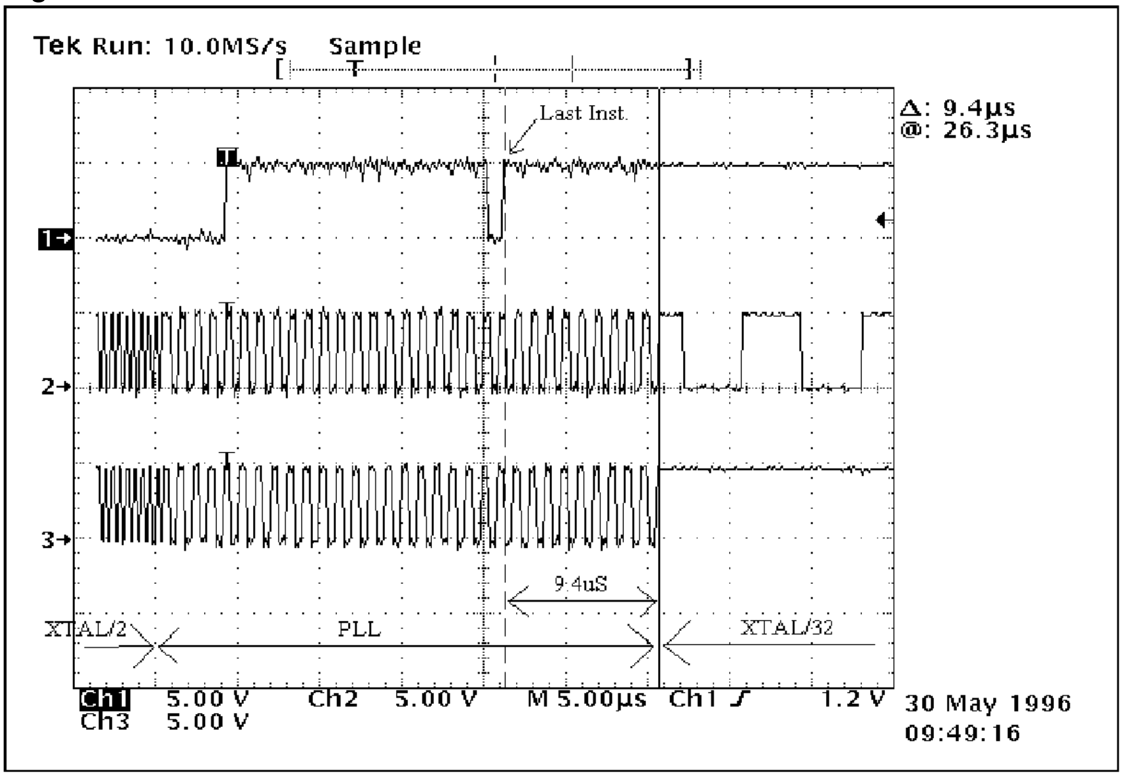


Figure 6.



The wake-up time mentioned above ('46.6uS') is not the interrupt response time given in the ST9 Datasheet (2.1 uS at 12Mhz internal frequency - time given for a program designed in assembly language and running at 12Mhz internal frequency).

This is a slower interrupt response time for these two reasons:

– difference of speed: the external crystal is different (4.4 Mhz instead of 24 Mhz)

– difference of language: the 'C' language is less quick than Assembly because of a few features such as saving the context and creating local variables.

5 DESCRIPTION OF A SIMPLE APPLICATION: REAL-TIME CLOCK

This simple application keeps track of the Time Of the Day (TOD) and monitors an alarm time. When the alarm time is reached, a beeper will buzz for one minute and one IO port will be toggled.

The TOD can be displayed at any time by pressing the ’refresh’ button.

A very simple algorithm is used to perform the following actions:

– by pressing the ’refresh’ button, which is an asynchronous event, an external IT is triggered.
– a flag is set up in that interrupt service routine to the attention of the timer IT.
– In the MFT timer IT, the time will be incremented once a second, then minutes etc...and be displayed for four seconds if the ’refresh’ flag has been set.
– finally the ST9058 core gets back to ’sleep’ in WFI Low-Power mode.

This ’refresh’ button is configured as an Interrupt input and will wake-up the core from its WFI mode.

This application software can be useful for time keeping applications: timers, Car Radio, VCR...

5.1 ALGORITHM TO KEEP THE TIME OF DAY

The algorithm for keeping the TOD uses a Multi-Function Timer, the MFT0, as a counter to generate one interrupt per second. The clock for this timer is INTCLK and varies in time. The clock speed is 137 Khz for almost one second and then 2.2 Mhz for almost 100uS and finally 11Mhz for the rest of the Interrupt routine.

This interrupt routine can access the LCD display to show the TOD, so it only lasts a few milli-seconds.

The only trick in the application of the PLL is to CHANGE THE PRESCALER OF THE TIMER REGARDING THE FREQUENCY OF INTCLK IN REAL-TIME.

As there are 3 different speeds, 3 prescalers values must be defined, which is coded onto 8 bits, between 0x00 and 0xFF. There are the 3 values: 1, 16, 80 as listed below.

```
 /* prescaling factors for the MFT0 function of INTCLK */

#define factor_wfi     0     /* operation at 137.85 kHz:%1 */
#define factor_xtal2   15     /* operation at 2.2056 MHz:%16 */
#define factor_pll     79     /* operation at 11.028 Mhz:%80 */
#define timer_period  45950  /* theoretical value for timer 0 */

/* 45950 * 3 * 2 * 16 = 4411200 cycles per second: ext. crystal */
/*    MFT DIV2 WFI-LP                          */
```

With this method, the counter is counting down from *timer_period* to 0 and then generates an interrupt. Before switching to any of the 3 modes, the software reconfigures the timer prescaler to the next value it should use to be accurate.

There is little drift as this commutation is performed by software, so a correction factor that trims the period of the timer was introduced.

```
#define offset_correct 37/* correction factor because earlier start of
PLL*/
```

After introducing this correction, the TOD is accurate and lasts as long as the battery (for 3 to 4 weeks for AA batteries)

5.2 USER MANUAL

The user can read and set up the times (current time and alarm time) through a set of 4 buttons.

The upper left one is the hardware reset and will reset the time to a default value.

The lower left one is the ’refresh key’ and validation key. In normal operation, it displays the current time for 4 seconds. In update mode, it allows validation of a keyboard entry.

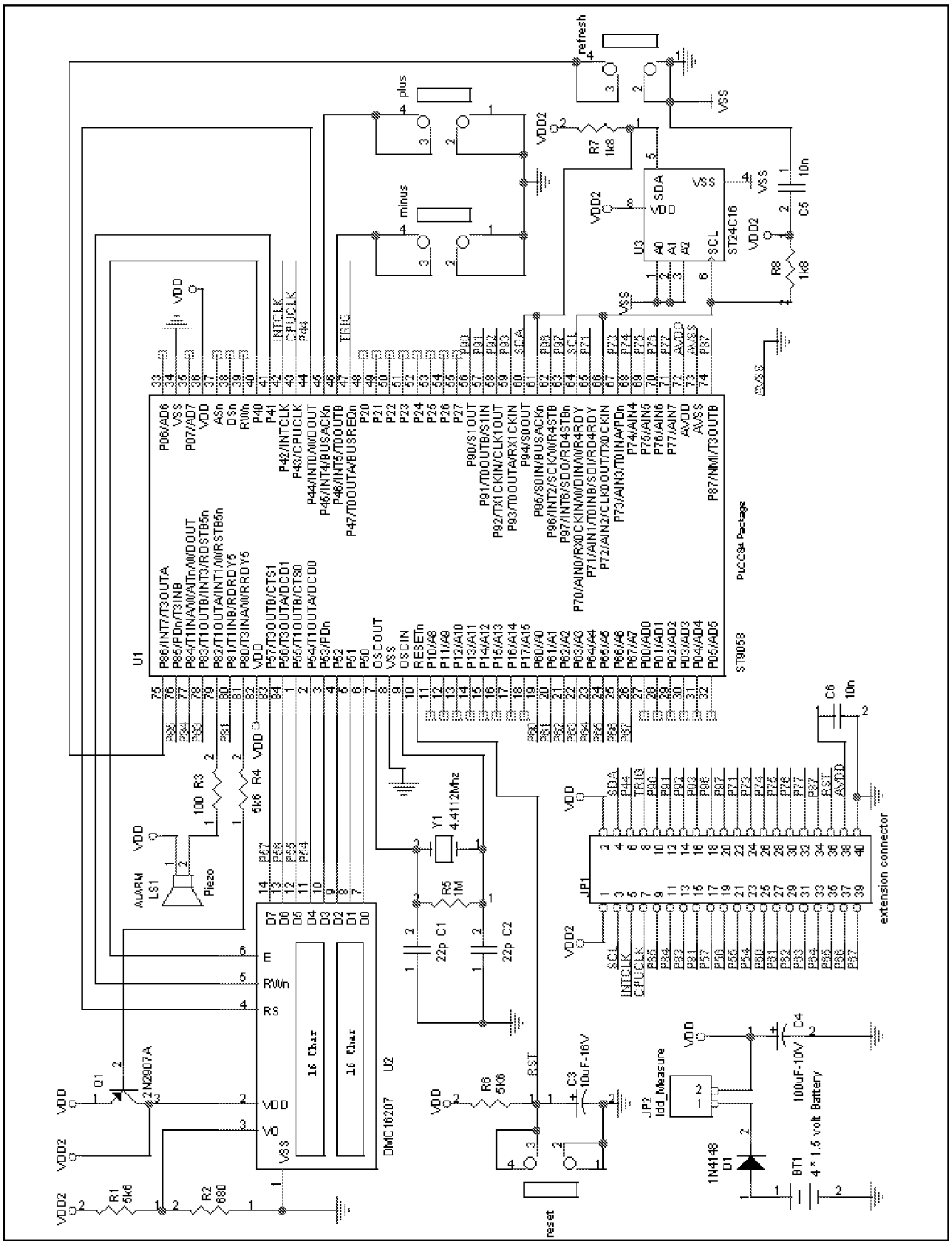
The lower right one is the ’decrement’ button, it decrements the current item on the LCD.

The upper right one is the ’increment’ button, it increments the current item on the LCD.

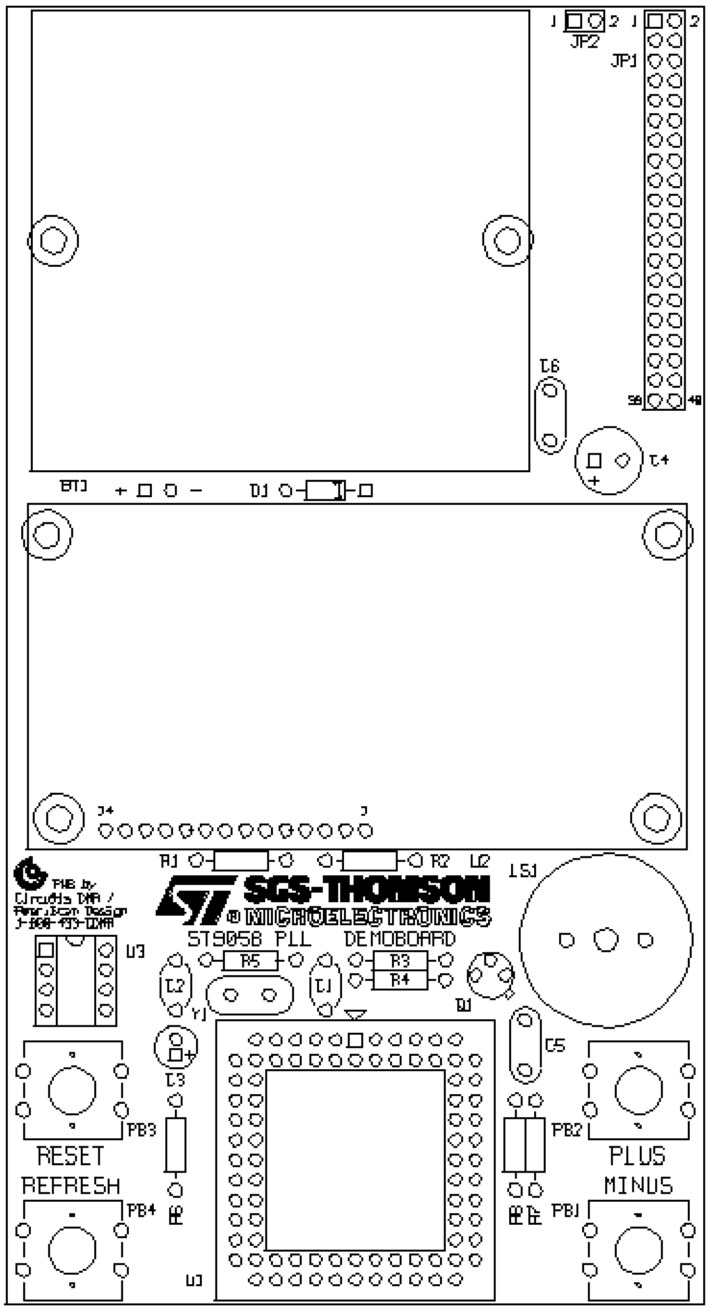
To enter the update mode, the two right keys are held down while pressing the ’refresh’ key. Then any item (second, minutes, hour, day and month) can be updated with the increment/decrement key. Any item will be entered by press the ’refresh’ button.

After setting up the current time, the alarm time prompt will appear.

5.3 APPLICATION HARDWARE



5.4 ST9058 PLL DEMOBOARD LAYOUT



5.5 APPLICATION SOFTWARE

REM BATCH FILE TO COMPILE AND LINK THIS APPLICATION SOFTWARE

```
gcc9 crt9.asm -c -g -o crt9.o
pause
gcc9 st9fmtp.c -Ic:\gnu9\include.st9 -c -g -O -Wall -o st9fmtp.o
pause
gcc9 st9putc.c -Ic:\gnu9\include.st9 -c -g -O -Wall -o st9putc.o
pause
gcc9 st9print.c -Ic:\gnu9\include.st9 -c -g -O -Wall -o st9print.o
pause
gcc9 pll.c -Ic:\gnu9\include.st9 -c -g -O -Wall -o pll.o
pause
ld9 -I -i -m -Tdata 0x1000f800 -Tuserstacksize 0 -o pll.u crt9.o st9print.o
st9putc.o st9fmtp.o pll.o -llibreg9 -lstdreg9
intel9 pll.u>pll.hex
```

; FILE CRT9.ASM, MODIFIED START-UP FILE

```
INIT_CIC = 8fh          ; CIC = IT disabled + Nested Mode + CPL = 7
K_INITCLOCKMODE = 20h   ; R235 = both stacks in memory + clock divided
                        ; by 2
K_INITWCR = 40h         ; R252 = zero wait state + watchdog enabled
    .text
    .word  __Reset
    .word  __Reset
    .word  __Reset
    .blkb  10
ext_it_table::          ; this should be the address 10h
    .word  __Reset
    .word  __Reset
    .word  __Reset
    .word  __Reset
    .word  __Reset
    .word  __Reset
    .word  __Reset
    .word  user_IT
timer_it_table::       ; this should be the address 20h
    .word  timer_IT
    .word  __Reset
    .word  __Reset
    .word  __Reset

    .blkb  12          ; reserve room for the interrupt vectors

        .text
```

```
;---------------------------------------- ----------------
; Reset routine
;
; WARNING: it is important to set rr12 to something NOT zero.
; This is because for the debugger a frame pointer null means
; the function has no parent frame (ie cannot go up). This is OK
; for the main routine, but not for routines called by main.
; If -fomit-frame-pointer is used for compiling, rr12 could not
; be set in main, nor in functions called by main...
; Furthermore, it seems a good idea to initialize the current frame
;  pointer to the current stack pointer.
;
;-------------------- ---------------------------------- -----
        .global __Reset
__Reset:
    spp   #0                    ; page 0 to access WCR
    ld    R252,#K_INITWCR       ; WCR = zero wait state
    ld    R235,#K_INITCLOCKMODE ; init CLOCKMODE
                                 (both stack in memory)
    ld    R230,#INIT_CIC        ; CIC for our program

    srp   #0                    ; set register pointer to 0

    spp   #0
    sdm                         ; set data memory
    ldw   RR238,#_stack_end     ; setup stack
    ldw   RR236,#_user_stack_end ; setup user stack
    call  __K_InitDataBss       ; init data & bss section
    ei                          ; enable interrupts
    ldw   rr12,RR238            ; make sure rr12 is NOT zero
    call  main
    halt
```

```
;------------------------------------------------------------
; Function to initialize the data, bss and stack sections.
;
; input: none.
;         Data space is selected.
;
; output: none.
;          rr0,rr2,rr4,rr6 are trashed.
;          Data space is selected.
;
; Note: LD9 creates the following symbols:
;
;       _data_start points to start of run time data area,
;       _data_end points to end of run time data area,
;       _bss_start points to start of run time bss area,
;       _bss_end points to end of run time bss area,
;       _text_start points to start of text segment,
;       _text_end points to end of text segment.
;       _stack_start points to start of stack segment,
;       _stack_end points to end of stack segment.
;
; Note: option I must be used with LD9 in order to
;       get the initialize data information at the end of the text
;       section (in ROM).
;
;       In that case '_text_end - (_data_end - _data_start)' gives
;       the start address of the initialized data information in
;       the text segment.
;
; Note: be careful to save the return address before clearing
;       the BSS section, because the kernel stack belongs to the
;       BSS section.
;------------------------------------------------------------
        .global __K_InitDataBss
__K_InitDataBss:
    popw  rr6                  ; save return address in rr6
;
; Init data area
;
    ldw   rr0,#_data_start     ; start of run-time data area
    ldw   rr4,#_data_end       ; end of run time data area
    subw  rr4,rr0              ; rr4 = length of initialize data
    jrz   no_data              ; if empty
    ldw   rr2,#_text_end       ; end ROMed data area
    subw  rr2,rr4              ; start of ROMed data area
```

```
init_data:
    lddp  (rr0)+,(rr2)+      ; init data section
    dwjnz  rr4,init_data     ;
no_data:
;
; Init bss section
;
    xor   r4,r4              ; r4 = 0.
    ldw   rr0,#_bss_start    ; start of run-time bss area
    ldw   rr2,#_bss_end      ; end of run time bss area
    subw   rr2,rr0           ; rr2 = length of bss area
    jrz   no_bss             ; if bss is empty
init_bss:
    ld    (rr0)+,r4          ; clear all bss section (Data space)
    dwjnz  rr2,init_bss      ;
no_bss:
;
; Init stack section (not really necessary, but cleaner)
;
;    (here r4 = 0)
;
    ldw   rr0,#_stack_start  ; start of run-time stack area
    ldw   rr2,#_stack_end    ; end of run time stack area
    subw   rr2,rr0           ; rr2 = length of stack area
    jrz   endinit            ; if stack is empty
init_stack:
    ld    (rr0)+,r4          ; clear all stack section
                              (Data space)
    dwjnz  rr2,init_stack    ;

endinit:
    jp    (rr6)              ; return to caller.
/******************* ********************************** ********
*   Header file of PLL.C module                                 *
*   FILE: PLL.H                                                 *
*   (c) Olivier GARREAU / PPG / LINCOLN                         *
*   03/12/96                                                    *
*   07/03/96 last release                                       *
******************* ********************************** *******/

#include    <st905x.h>
#include    “PLLreg.h”
#include    “st9reg.h”
#include    “st9print.h”
#include    “st9putc.h”
```

```
#include     "hardware.h"

#define turn_LCD_ON()   spp(P8D_PG);power_IO=(refresh|LCD_ON);\
                        spp(P7D_PG);P7DR=0xFF; spp(P9D_PG);P9DR=0xFF;
#define turn_LCD_OFF()  spp(P8D_PG);power_IO=(refresh|L CD_OFF);\
                        data_LCD = 0;control_IO &= 0xec;\
                        spp(P7D_PG);P7DR=0x00; spp(P9D_PG);P9DR=0x00;

     /* prescaling factors for the MFT0 function of INTCLK */

#define factor_wfi     0      /* operation at 137.85 kHz :%1   */
#define factor_xtal2   15     /* operation at 2.2056 MHz :%16  */
#define factor_pll     79     /* operation at 11.028 Mhz :%80  */
#define timer_period   45950  /* theoretical value for timer 0 */

/* 45950 * 3 * 2 * 16 = 4411200 cycles per second : ext. crystal */
/*     MFT DIV2 WFI-LP                                           */

#define offset_correct 37/* correction factor 'cause earlier start of
PLL*/

/* 23 * 1 kHz * 2 * 3 = 137 kHz */
#define beep_divisor 23   /* generate a ~1kHz beep at 137 kHz */
#define ON 1
#define OFF 0

#define max_count 4

#define trigger_scope() control_IO ^= trig;

void display_time(void);
void display_alarm(void);
void delay(unsigned long);

/****************** ********************************* ********
*   Header file of PLL.C module                              *
*   PLLREG.H                                                 *
*   (c) Olivier GARREAU / PPG / LINCOLN                      *
*   03/13/96                                                 *
*   definition of new PLL registers                          *
******************* ********************************* *******/

register volatile unsigned char PCONF   asm("R251");
register volatile unsigned char SDCTL   asm("R252");
register volatile unsigned char SDRATH  asm("R254");
```

```
#define PLL_PG ((unsigned char)55)
#define R0P1    0x10  /* reference Clock/PLL clock */
#define MX      0x08  /* multiply factor */
#define DX2     0x04  /* MSB pll divider factor */
#define DX1     0x02  /* b1 divider factor */
#define DX0     0x01  /* LSB pll divider factor */
#define LPOWFI  0x20  /* Low Power WFI mode */
#define XT1RC0  0x04  /* Divider by 16 control bit */
#define PLLSTAT 0x08  /* PLL status register */
#define PLLCK1  0    /* PLL DIVIDER factors */
#define PLLCK2  1
#define PLLCK3  2
#define PLLCK4  3
#define PLLCK5  4
#define PLLCK6  5
#define PLLCK7  6
#define PLLOFF  7    /* disable the PLL */
#define MUL10   0    /* multiply by 10 */
#define MUL6    MX   /* multiply by 6 */

/* file ST9REG.H */
#define spm()  asm("spm");   /* set program memory */
#define sdm()  asm("sdm");   /* set data memory */
#define wfi()  asm("wfi");   /* enter WFI mode */

/******************* ********************************* **/
/* C macros for inline assembly code with an operand        */
/******************* ********************************* **/

/* set page pointer to value page   */
#define spp(page)        asm("spp %0":: "i" (page));

/* set working register pointer to value bank */
#define srp(bank)        asm("srp %0":: "i" (bank));

/* load a value to a working register */
#define ldw_rr_xx(reg,value)  asm("ldw rr%Q0,%1":: "i" (reg) ,"RR"
(value));

/******************* *****************/
/* SYSTEM REGISTERS DEFINITIONS          */
/******************* *****************/

/* system register */
```

```
register unsigned char STACK_POINTER  asm("RR238");
register unsigned char FLAGR          asm("R231");
register unsigned char PPR            asm("R234");
register unsigned char WCR            asm("R252");

/*   Header file of ST9PRINT.H      */

int st9_printf(char *fmt, ...);
int st9_sprintf(char *buf, char *fmt, ...);

/*   Header file of ST9PUTC.H      */

void wait (void);
unsigned char test_LCD (void);
void scroll(void);
void init_lcd(void);
void st9_putchar (unsigned char c);
void clear_screen (void);

#define     WS_VALUE     0x00 /* 0 Wait States in Data Space */

#define     BUSY         0x80
#define     TAB          0x03
#define     BLANK_CHAR   0x20
#define     ASCII_255    0xFF
#define     ASCII_BEL    0x07
#define     ASCII_CR     0x0D
#define     END_OF_FILE  0x00

#define     WRITE_DATA         0x80
#define     BEGIN_SECOND_LINE  0x40
#define     BEGIN_FIRST_LINE   0x00
#define     END_FIRST_LINE     0x0F
#define     END_SECOND_LINE    0x4F
#define     DISPLAY_ON         0x0C
#define     FUNCTION_SET       0x38
#define     CLEAR_DISPLAY      0x01
#define     PRESENT            0x80
#define     ABSENT             0x00
#define     RAM_PRESENT        0x01

/* specifics characters from 128 to 256 */

#define     FULL     '\xff'
#define     ROOT     '\xe8'
```

```
#define     EPSILON  '\xe3'
#define     POINT    '\xa5'
#define     OMEGA    '\xf4'
#define     TETHA    '\xf2'
#define     ALPHA    '\xe0'
#define     DIVIDE   '\xfd'
#define     INVERT   '\xe9'
#define     sigma    '\xe5'
#define     PI       '\xf7'
#define     SIGMA    '\xf6'
```

/*   FILE **st9stdio.h**   */

```
/* prototypes */

#ifndef noproto
int st9_printf(char *, ...);
int st9_scanf(char *, ...);
int st9_sprintf(char *, char *, ...);
int st9_sscanf(char *, char *, ...);
void st9_gets(char *);
char st9_getchar(void);
char st9_getchav(void);
void st9_putchar(unsigned char);
#else
int st9_printf(/* char *, ...*/);
int st9_scanf(/* char *, ...*/);
int st9_sprintf(/*char *, char *, ...*/);
int st9_sscanf(/* char *, char *, ...*/);
void st9_gets(/*char **/);
char st9_getchar(/*void*/);
void st9_putchar(/*int*/);
#endif

/*************************************************
*   Header file of PLL.C module                  *
*   HARDWARE.H                                   *
*   (c) Olivier GARREAU / PPG / LINCOLN          *
*   03/13/96                                     *
*************************************************/
#include <st905x.h>

/* mapping of IO on port 4 */

#define control_IO    P4DR   /* P4 drive the control signals of LCD/KEYB*/
```

```
#define trig          0x80  /* one bit trigger (for the scope) */
#define minus_button  0x40  /* 'minus' button to increment parameter */
#define plus_button   0x20  /* 'plus' button to decrement parameter */
#define address_LCD   0x10  /* A0 address of LCD display */
#define cpu_clk       0x08  /* af output that duplicate CPUCLK P43 */

#define int_clk       0x04  /* af output that duplicate INTCLK P42 */
#define RWn_LCD       0x02  /* R/Wn signal to the LCD display */
#define enable_LCD    0x01  /* Enable IO access to LCD display */

/* mapping of IO on port 8*/

#define power_IO      P8DR  /* IO data register to control LCD Power Sup.*/
#define power_LCD     0x01  /* power up/down the LCD display */
#define LCD_ON        0x00  /* reset P80 will power up the LCD */
#define LCD_OFF       0x01  /* set P80 will power down the LCD */
#define refresh       0x40  /* P86 INT7 to refresh display/time */
#define T1OUTA        0x04  /* T1OUTA, output A of timer 1 */
#define IN            0     /* keep updating an item */
#define OUT           1     /* stop updating an item */

/* mapping of IO on port 5 */

#define data_LCD      P5DR  /* port 5 drive the data bus of the LCD */

/* mapping of IO on port 7 */

#define SCL         0x04  /* port P72 drive the I2C SCL */
#define SCLIN       0x01  /* port P70 receive the ST9 slave clock */

/* mapping of IO on port 9 */

#define S0OUT       0x10  /* port P94 drive the I2C SDA */
#define S0IN        0x20  /* port P95 read the I2C SDA */
#define SDA         S0OUT /* I2C signal SDA */
```

```
/*___________________________________________________________________
|
|Routine Name   : Monitor for ST9058B PLL DEMOBOARD
|File Name      : PLL.C
|Action         : Kernel of the application note, real-time clock
|Author(FN/LN)  : Olivier Garreau / PPG / LINCOLN
|First rev date : 03/12/96
|Last rev date  : 07/03/96
|Input paramet.: no
|Output paramet: no
|Modified var.  : no
|Global varia.  :
|Comments       : Use the low power mode of the 9058
|                 Try to keep time of the day with accuracy
|__________________________________________________________________*/
#include "PLL.h"
void  init_IO (void) {
/* __________________________________
  | b7  | b6 | b5 | b4  | b3 | b2 | b1  | b0  |
  | OUT| WP | WP | OUT| AF | AF | OUT| OUT|
  |__________________________________|
*/
 spp(P4C_PG);  /* select control register page of Port 4 */
 control_IO = plus_button|minus_button|cpu_clk|i nt_clk;
 /*default value for IO, all disabled*/
 P4C2R = 0x00;
 P4C1R =(trig+address_LCD+cpu_clk+int_clk+RWn_LCD+enable_LCD);
 P4C0R = (cpu_clk+int_clk);
/* ________________________________
  | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
  | x  | WP | x  | x  | x  | AF | x  | WP |
  |________________________________|
*/
  spp(P8C_PG);   /* select control register page of Port 8 */
  power_IO = (refresh+T1OUTA+LCD_OFF); /* prepare INT7 and turn LCD off */
  P8C2R = 0x00;
  P8C1R = T1OUTA;      /* P82 as AF PP, T1OUTA */
  P8C0R = T1OUTA;      /* P80 out WP, drive LCD power */
/* ________________________________
  | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
  | WP | WP | WP | WP | WP | WP | WP | WP |
  |________________________________|
*/

  spp(P5C_PG);    /* select control register page of Port 5 */
  data_LCD = 0x00;/* reset value of port 5 */
  P5C0R = 0x00;   /* set P5 as Bidir Weak Pull-up : port to LCD HW */
  P5C1R = 0x00;   /* idem */
  P5C2R = 0x00;   /* idem */

   }

void  init_PLL(void) {

  spp(PLL_PG);              /* select the PLL page */
  PCONF = DX0;              /* start the PLL, Mul. by 5, use Xtal2 */
  SDCTL |= (LPOWFI|XT1RC0);/* enable low power mode in WFI */
                            /* and disable SLOW2 mode */
  }

void  use_PLL(void) {

  spp(PLL_PG);        /* select the PLL page */
  PCONF |= R0P1;      /* switch from XTAL2 to PLL output */
  }

#define ext_it_table 0x10

void  init_ext_IT(void){

  spp(EXINT_PG);        /* set page 0 for ext interrupt page */
  EITR &= ~EIm_ted1m;   /* generate an IT on falling edge of INT7 */
  EIPR &= ~EIm_ipd1m;   /* reset any pending IT bit */
  EIMR |= EIm_id1m;     /* allow to service this IT INT7 */
  EIPLR&= ~0xc0;        /* set priority level to 1 for INT7 */
  EIVR = 0x06;          /* reset EIVR with reset value */
  EIVR |= ext_it_table; /* specify the location of the ext it table */
  }

#define timer_it_table 0x20

void  init_timer(void){

  spp(T0D_PG);  /* select page of MFT0 data registers */
  T_REG0R = timer_period+offset_correct;
  /* 1HZ with 4.4112 Mhz ext crystal */
  T_TCR = 0;      /* disable counter, count down, no clear */
  T_TMR = 0;      /* disable output, reload from REG0R, continous */
  T_PRSR = factor_xtal2;
  /* reset the MFT prescaler for XTAL/2 operation */
```

```
    T_FLAGR = 0;   /* reset any pending bit */
    T_IDMR = 0;    /* restore the reset value : 0 */
    T_IDMR |= (Tm_gtien|Tm_oui);   /* enable interrupt and OUF IT */
     spp(T0C_PG); /* select page of MFT0 data registers */
    T0_IVR = timer_it_table;/* specify the location of the timer it table */
    T0_IDCR &= ~0x07;  /* set timer IT with priority 0 */

    spp(T1D_PG);    /* select page of MFT1 data registers */
    T_REG0R = beep_divisor;
     /* correspond to ~1 kHZ with 137.85 khz wfi mode */
    T_TCR = 0;        /* disable counter, count down, no clear */
    T_TMR = 0x40;     /* enable output, reload from REG0R, continous */
    T_PRSR = 0;       /* no prescaling  */
    T_OACR = 0xf5;  /* toggle on OUF event and preset output T1OUTA */
    T_FLAGR = 0;     /* reset any pending bit */
    T_IDMR = 0;      /* restore the reset value : 0 */

    }

void   start_timer(void){

    spp(T0D_PG);        /* select page of MFT0 data registers */
    T_TCR |= Tm_cen;    /* enable counter, count down, no clear */
    }

unsigned char  display_mode,modify_mode,hold_time;
void   main (void) {

   init_IO();          /* initialize all IO ports */
   turn_LCD_OFF()      /* power down the LCD hardware, default mode */
   display_mode = OFF; /* flag for display mode reset */
   modify_mode = OFF;  /* don't modify the time at reset */
   hold_time = 0;      /* reset the counter */
   init_PLL();         /* initialize PLL registers */
   init_timer();       /* initialize TIMER 0 registers for 1 IT/sec */
   init_ext_IT();      /* initialize INT7 external interrupt */

    start_timer();     /* start timer for operation @ Xtal/2 = 1MHZ */
    use_PLL();         /* allow output of PLL to be INTCLK once locked-up*/

    while (1) {       /* core of the main routine : sleep mode */
       spp(T0D_PG);  /* select page of MFT0 data registers */
       T_PRSR=factor_wfi;
       /* prepare the MFT prescaler for XTAL/32 operation */
       wfi()          /* enter the Wait-For-Interrupt mode & low power */
     }

   }

#pragma interrupt (user_IT)
void   user_IT(void) {      /* the user did press INT7 button down */

    control_IO |= plus_button|minus_button; /*prepare IO for KB input */
    display_mode = ON;    /* flag set for display mode */
    if (control_IO & (plus_button|minus_button)) {
        modify_mode = OFF;    /* just display the tod */
        }
    else {
        modify_mode = ON;     /* modify time settings */
        }

    spp(EXINT_PG);       /* set page 0 for ext interrupt page */
    EIPR &= ~EIm_ipd1m;    /* reset pending bit of INT7 */
    }

struct {
    unsigned char second,minute,hour,day,month;
    }
    time,alarm;

char   * month[] = {
    “January”,”February”,”March”,”April”,”May”,”June”,
    “July”,”August”,”September”,”October”,”November”,”December”
    };

void   delay(unsigned long x) {
unsigned long y;
    for (y=0;y<x;y++);
    }

void   wait_next (void) {
unsigned char save_PPR;

    save_PPR = PPR;  /* save PPR locally */
    spp(P8D_PG);   /* select control register page of Port 8 */
    power_IO  |= refresh;  /* prepare INT7 input */
    while (power_IO & refresh) ; /* wait key down */
    while (!(power_IO & refresh)) ; /* wait key released */
    PPR=save_PPR;
    }
```

```
unsigned char exit;
unsigned char inc_dec(unsigned char value) {
unsigned char save_PPR, keyb;

    save_PPR = PPR;  /* save PPR locally */
    spp(P8D_PG);   /* select control register page of Port 8 */
    power_IO  |= refresh;  /* prepare INT7 input */
    control_IO |= plus_button|minus_button; /*prepare IO for KB input */
    keyb = control_IO & (plus_button|minus_bu tton);
    delay(4000);   /* anti-bouncing loop */
    if (keyb != (control_IO & (plus_button|minus_button))) {
        PPR=save_PPR;
        return (value);
        }
    switch  (keyb) {
        case plus_button:
        value ++;
        break;
        case minus_button:
        value --;
        break;
        default:
        if (power_IO & refresh) exit = IN;
         else exit = OUT;
        break;
        }
    PPR=save_PPR;
    return(value);
    }

void control_limits(void) {
    if (time.second>59) time.second = 0;
    if (alarm.second>59 ) alarm.second = 0;
    if (time.minute>59) time.minute = 0;
    if (alarm.minute>59 ) alarm.minute = 0;
    if (time.hour>23) time.hour = 0;
    if (alarm.hour>23) alarm.hour = 0;
    if ((time.day>31)||(time.day==0)) time.day = 1;
    if ((alarm.day>31)||(alarm.day==0)) alarm.day = 1;
    if ((time.month>12)||(time.month==0)) time.month = 1;
    if ((alarm.month>12 )||(alarm.month==0)) alarm.month = 1;
    }

void  modify_time(void){
```

```
unsigned char counter=1;

    turn_LCD_ON()  /* power up the LCD hardware */
    init_lcd();   /* init HW of the LCD */

    st9_printf("\vUpdate mode:\rpress refresh");
    alarm.second=time.second;/* update alarm with current time */
    alarm.minute=time.minute;
    alarm.hour=time.hour;
    alarm.day=time.day;
    alarm.month=time.month;
    wait_next();
    st9_printf("\v(c)SGS THOMSON \xE4\rOlivier GARREAU");
    wait_next();

    /* modify current time and alarm time*/
    while (counter<9) {
    switch (counter) {
        case 1: /* update the hour */
        time.hour=inc_dec(time.hour);
        break;
        case 2: /* update the minute */
        time.minute=inc_dec(time.minute);
        time.second=0;
        break;
        case 3: /* update the month */
        time.month=inc_dec(time.month);
        break;
        case 4: /* update the date */
        time.day=inc_dec(time.day);
        break;
        case 5: /* update the hour-alarm */
        alarm.hour=inc_dec(alarm.hour);
        break;
        case 6: /* update the minute-alarm */
        alarm.minute=inc_dec(alarm.minute);
        alarm.second=0;
        break;
        case 7: /* update the month-alarm */
        alarm.month=inc_dec(alarm.month);
        break;
        case 8: /* update the date-alarm */
        alarm.day=inc_dec(alarm.day);
        break;
        default:
        break;
        }

    control_limits( );
    if (counter<5) display_time(); else display_alarm();
    if (exit == OUT) {counter++;exit=IN;delay (10000);}

      }
    }

void  display_time(void){
unsigned char ampm=0;

    if (time.hour>12) {
        ampm = time.hour-12;
        }
    if (time.hour<13) {
        ampm = time.hour;
        }
    if (time.hour==0) {
        ampm = 12;
        }

    if (time.hour>=12) {
     st9_printf("\v%02d:%02d.%02dpm\r",ampm,time.minute,time.second);
        }
    else {
     st9_printf("\v%02d:%02d.%02d am\r",ampm,time.minute,time.second);
     }

    st9_printf("%s %02d",month[(time.month)-1],time.day);
    }

void  display_alarm(void){
unsigned char ampm=0;

    if (alarm.hour>12) {
        ampm = alarm.hour-12;
        }
    if (alarm.hour<13) {
        ampm = alarm.hour;
        }
    if (alarm.hour==0) {
        ampm = 12;
        }
    if (alarm.hour>=12) {
       st9_printf("\vAl %02d:%02d.%02d
       pm\r",ampm,alarm.minute,alarm.second);
        }
    else {
       st9_printf("\vAl %02d:%02d.%02d
       am\r",ampm,alarm.minute,alarm.second);
       }

    st9_printf("%s %02d",month[(alarm.month)-1],alarm.day);
    }

void  update_tod(void) {

    time.second++;
    if (time.second>=60) {
        time.second=0;
        time.minute++;
        }
    if (time.minute>=60) {
        time.minute=0;
        time.hour++;
        }
    if (time.hour>=24) {
        time.hour=0;
        time.day++;
        }

    switch (time.month) {
        case 1: /* january*/
        case 3: /* march */
        case 5: /* may */
        case 7: /* july */
        case 8: /* august */
        case 10:/* october */
        case 12:/* december */

            if ((time.day>=32)||(time.day==0)) {
            time.day=1;
            time.month++;
            }
        break;
        case 2: /* february */
            if ((time.day>=30)||(time.day==0)){
            time.day=1;
            time.month++;
            }
        break;
        default:/* other months */

            if ((time.day>=31)||(time.day==0)){
            time.day=1;
            time.month++;
            }
        break;

        }

    if ((time.month>=13 )||(time.month==0)){
        time.month=1;
        }

    }
#define stop_beep() spp(T1D_PG);T_TCR &= ~Tm_cen /* stop timer 1 */

void  test_alarm(void ){

    if(alarm.minute != time.minute) {stop_beep();return;}
    if(alarm.hour != time.hour) return;
    if(alarm.day != time.day) return;
    if(alarm.month != time.month) return;

    spp(T1D_PG);    /* select page of MFT1 data registers */
    T_TCR ^= Tm_cen; /* enable counter, toggle then disable etc.. */
    trigger_scope()
    }

#pragma interrupt (timer_IT)
void  timer_IT(void) {     /* wake-up from WFI is in XTAL/2 mode */

    /* from here on, the CPU speed up from 137.85 kHz to 2.2056 Mhz */

   spp(T0D_PG);       /* select page of MFT0 data registers */
   T_PRSR=factor_xtal2;/* reset the MFT prescaler for XTAL/2 operation */

  if ((display_mode == ON) && (modify_mode == ON)) {
        modify_time(); /* enter the procedure to modify the time */
        modify_mode = OFF;    /* go back to normal display mode */
```

```
      }
if (display_mode == ON) {    /* display was requested */
      if (hold_time == max_count) {  /*is display time over ? */
          display_mode = OFF;    /* yes, stop displaying */
          hold_time = 0;  /* reset the counter */
          turn_LCD_OFF() /* power down the LCD hardware */
          }
      else {
          hold_time++;   /* increment the counter */
          turn_LCD_ON()  /* power up the LCD hardware */
          init_lcd();   /* init HW of the LCD */
          }
      }

spp(PLL_PG);        /* select the PLL page */
while ( !(SDRATH & PLLSTAT) ); /* wait til the PLL is locked  */

/* from here on, the CPU speed up from 2.2056 to 11.028 Mhz */

spp(T0D_PG);       /* select page of MFT0 data registers */
T_PRSR=factor_pll;/* reset the MFT prescaler for PLL operation */

update_tod(); /* update time of the day, increment by 1 second */
if (display_mode == ON) {    /* display was requested */
      display_time();     /* send TOD to the LCD */
      }

test_alarm();      /* test if alarm time is reached */
spp(T0D_PG);       /* select page of MFT0 data registers */
T_FLAGR &= ~Tm_ouf;/* reset OUF IT pending bit */
T_PRSR=factor_wfi /* reset the MFT prescaler for WFI operation */
  }
```

```
/* FILE ST9PRINT.C, ST9 printf to generic hardware */
#include ”add_.h”
#include ”st9print.h”
#include ”st9stdio.h”

/* -------------------------------------------------------- --*/

int format( int (*)( int ), char *, void *);

/* -------------------------------------------------------- --*/

ADD_FUNCNAME(_st9_printf)
int st9_printf( char *fmt, ... )
{
    void *ap;
    ap = (void *)( (char *)( &fmt ) + sizeof( fmt ) );
    return format( (int (*)(int))st9_putchar, fmt, ap );
}

/* -------------------------------------------------------- --*/

static char *buff;

static int spsub( int c )
{
    return ( *buff++ = c );
}

ADD_FUNCNAME(_st9_sprintf)
int st9_sprintf( char *buf, char *fmt, ... )
{
    register int i; void *ap;

    ap = (void *)( (char *)( &fmt ) + sizeof( fmt ) );

    buff = buf;
    i = format( spsub, fmt, ap );
    *buff = ’\0’;
    return i;
}
```

```
/* FILE ST9FMTP.C , formating of PRINTF functions */
#include “c:\gnu9\include\ctype.h”
#include “add_.h”

int strlen( char *s );

#define MAXWIDTH 14    /* normal 200 ! */
#define NUMWIDTH 14

/* ------------------------------------------------------- --*/

void *fmtcvt( void *ap, int base, char *cp, int len)
{
    long val;
    static char digits[]=”0123456789abcdef”;

    if ( len == sizeof( long) )
        val = *( long * )ap;
    else if ( base > 0 )
        val = *(unsigned int * )ap;
    else
        val = *(int * )ap;

    len = 0;
    if ( base < 0 ) {
        base = -base;
        if ( val < 0L )
        {
            val = -val;
            len = 1;
        }
    }

    do {
        *--cp = digits[ (char)( val % base ) ];
    } while ( ( val /= base ) != 0);

    if ( len )
        *--cp = ’-’;
    return cp;
}

/* ------------------------------------------------------- --*/

int format( int (*putsub)( unsigned char ), register char *fmt, void *ap)
```

```
{
    register unsigned char c;
    int charcount, rj, fillc, maxwidth, width, i, k;
    char *cp; char s[MAXWIDTH+1];

    charcount = 0;

    while ( ( c = *fmt++ ) != 0 )
    {
        if ( c == ’%’ )
        {
            s[NUMWIDTH] = 0;
            rj = 1;
            fillc = ’ ’;
            maxwidth = 10000;
            if ( (c = *fmt++ ) == ’-’)
            {
                rj = 0;
                c = *fmt++;
            }
            if ( c == ’0’ )
            {
                fillc = ’0’;
                c = *fmt++;
            }
            if ( c == ’*’ )
            {
                width = *( (int *)ap );
                ap = (int *)ap + 1;
                c = *fmt++;
            }
            else
            {
                for ( width = 0 ; isdigit(c) ; c = *fmt++ )
                    width = width * 10 + c - ’0’;
            }
            if ( c == ’.’ )
            {
                if ( ( c = *fmt++ ) == ’*’ )
                {
                    maxwidth = *( (int *)ap );
                    ap = (int *)ap + 1;
                    c = *fmt++;
                }
                else
```

```
                    {
                        for ( maxwidth = 0 ; isdigit(c) ; c = *fmt++ )
                            maxwidth = maxwidth * 10 + c - '0';
                    }
                }
                i = sizeof( int );

                if ( c == 'l' )
                {
                    c = *fmt++;
                    i = sizeof(long);
                }
                else if ( c == 'h' )
                    c = *fmt++;

                switch ( c )
                {
                    case 'o':
                        k = 8;
                        goto do_conversion;

                    case 'u':
                        k = 10;
                        goto do_conversion;

                    case 'x':
                        k = 16;
                        goto do_conversion;

                    case 'd':
                        k = -10;
        do_conversion:
                        cp = (char *)fmtcvt( ap, k, s + NUMWIDTH, i );

                        ap = (char *)ap + i;
                        break;

                    case 's':
                        cp = *( ( char **) ap );
                        ap = (char **)ap + 1;
                        i = strlen( cp );
                        goto havelen;
                    case 'c':
                        c = *((char *)ap + 1);  /* low byte !*/
                        ap = (int *)ap + 1;
                    default:
                        *( cp = s + NUMWIDTH - 1 ) = c;
                        break;
                }

                i = ( s + NUMWIDTH    ) - cp;
havelen:
                if ( i > maxwidth )
                    i = maxwidth;

                if ( rj )
                {
                    if ( ( *cp == '-' || *cp == '+') && fillc == '0' )
                    {
                        --width;
                        if ( putsub( *cp++ ) == -1 )
                            return -1;
                    }
                    for (; width-- > i ; ++charcount)
                        if ( putsub( fillc ) == -1 )
                            return -1;
                }
                for ( k = 0 ; *cp && k < maxwidth ; ++k )
                    if ( putsub( *cp++ ) == -1 )
                        return -1;
                charcount += k;

                if ( !rj )
                {
                    for ( ; width-- > i ; ++charcount )
                        if ( putsub( ' ' ) == -1 )
                            return -1;
                }
            }
            else
            {
                if ( putsub( c ) == -1)
                    return -1;
                ++charcount;
            }
        }
        return charcount;
}
```

```
/* FILE ST9PUTC.C : primitives to manage the LCD display */
#include ”add_.h”
#include ”st9putc.h”
#include ”st9reg.h”
#include ”hardware.h”

unsigned char cst9_stdout;
unsigned char ascii_type;
unsigned char lcd_control;

register volatile unsigned char LCD_CONTROL   asm(”R229”);
register volatile unsigned char LCD_DATA      asm(”R229”);

/*  declare a few simple macros */

#define LCD_read_control()
LCD_DATA=0xff;control_IO&=~address_LCD;control_IO|=RWn_LCD+enable_LCD;
#define LCD_write_control()
control_IO&=~(address _LCD+RWn_LCD);control_IO|=enable_ LCD;
#define LCD_read_data()
LCD_DATA=0xff;control _IO|=address_LCD+RWn_LCD+enable_L CD;
#define LCD_write_data()
control_IO&=~RWn_LCD; control_IO|=address_LCD+enable_LC D;
#define LCD_end_cycle()
control_IO&=~(enable_ LCD+address_LCD);control_IO|=RWn_ LCD;

 /***************************************** **************/
/*   TITLE : WAIT                                         */
/*   Author(s) : GARREAU Olivier                          */
/*   Date : 05/22/96                                      */
/*   Description : Wait until the LCD is ready            */
 /***************************************** **************/

void wait (void)
{
    LCD_read_contro l()
    while (LCD_CONTROL & BUSY);  /* Test BUSY BIT */
    LCD_end_cycle()
}

unsigned char test_LCD (void)
{
        unsigned char wcr_back,TYPE;

    /* config of enough Wait States */
    wcr_back = WCR;        /* backup WCR */
    WCR|= WS_VALUE;        /* put enough wait states (1) on DM */

    LCD_CONTROL = CLEAR_DISPLAY;/* Clear display to check LCD presence */
    TYPE = LCD_CONTROL;
        WCR = wcr_back;        /* backup WCR */

        switch (TYPE) {
            case 0xff :              /* case of real silicon : no LCD */
            case 0x00 :              /* case of emulator    : no LCD */
            TYPE = ABSENT;
            break;

            case CLEAR_DISPLAY :  /* case of RAM present           */
            TYPE = RAM_PRESENT;
            break;

            default :               /* LCD  present */
            TYPE = PRESENT;
            break;
        }
    return (TYPE);
}

/******************************************* **************/
/*   TITLE : SCROLL                                         */
/*   Author(s) : GARREAU Olivier                            */
/*   Date : 05/22/96                                        */
/*   Description : scroll LCD                              */
/******************************************* **************/

void scroll(void)
{

unsigned char i,second_line_char;

for (i=0; i <=15 ; i++)
   {
   lcd_control = ((BEGIN_SECOND_LINE | WRITE_DATA) + i);  /* place cursor
   on the second line */
   wait();
   LCD_CONTROL = lcd_control;
   LCD_write_control( )
   LCD_end_cycle()
```

```
        wait();
        LCD_read_data()
        second_line_char = LCD_DATA;                /* save LCD_DATA */
        LCD_end_cycle()
        lcd_control = ((BEGIN_FIRST_LINE | WRITE_DATA) + i); /* place cursor on
        first line */
         wait();
         LCD_CONTROL = lcd_control;
         LCD_write_control()
         LCD_end_cycle()
         wait();
         LCD_DATA = second_line_char;  /* fill first line with 2nd line */
         LCD_write_data()
         LCD_end_cycle()
         lcd_control = (BEGIN_SECOND_LINE | WRITE_DATA);  /* restore cursor at
         2nd line */
         wait();
         LCD_CONTROL = (lcd_control + i);
         LCD_write_control()
         LCD_end_cycle()
        wait();
        LCD_DATA = BLANK_CHAR;       /* fill second line with blank */
        LCD_write_data()
        LCD_end_cycle()
        }
    lcd_control = (BEGIN_SECOND_LINE | WRITE_DATA);
    wait();
    LCD_CONTROL = lcd_control ;
    LCD_write_control()
    LCD_end_cycle()
    }

     /***************************************** **************/
    /*   TITLE :INIT_LCD                                      */
    /*   Author(s) : GARREAU Olivier                          */
    /*   Date : 05/22/96                                      */
    /*   Description : Initialize LCD                         */
     /***************************************** **************/

    void init_lcd(void) {

        wait();
        LCD_CONTROL = DISPLAY_ON;      /* Display ON\OFF control */
                                /* cursor mode = increment */
                                /* cursor display shift  */
```

```
                    /* accept instruction   */
    LCD_write_control()
    LCD_end_cycle()

    wait();
    LCD_CONTROL = FUNCTION_SET;    /* function set */
                    /* number of line = 2 */
                    /* access 8 bits */
                    /* fonts = matrix(5,7) */
    LCD_write_control()
    LCD_end_cycle()

    clear_screen();          /* clear display */
    wait();
    }

void clear_screen(void) {

    wait();
    LCD_CONTROL = CLEAR_DISPLAY;    /* clear display */
    LCD_write_control()
    LCD_end_cycle()
    }

/***************************************** **************/
/*   TITLE : ST9_PUTC                                   */
/*   Author(s) : VERA Laurent & GARREAU Olivier         */
/*   Date : 19/09/94                                    */
/*   Description : Display character on the LCD        */
/***************************************** **************/

ADD_FUNCNAME(_st9_put char)
void st9_putchar (unsigned char c)

{
unsigned char wcr_back, flagr_back, ppr_back;

    /* config of enough Wait States */

    ppr_back = PPR;      /* backup PPR */
    spp(0);              /* select page of WCR */
    wcr_back = WCR;      /* backup WCR */
    flagr_back = FLAGR   /* backup FLAGR */
    WCR|= WS_VALUE;      /* put enough wait states on DM */
```

```
cst9_stdout=c;
ascii_type = 3;
wait();

if ( (c >= BLANK_CHAR) && (c<=ASCII_255) )   /* if ascii 0x20 to 0xFF */
     {
     ascii_type = 1;
     }

if ( c == END_OF_FILE )              /* if end of file */
     {
     ascii_type = 0;
     }
if ( (c >= ASCII_BEL) && (c <= ASCII_CR) ) /* if ASCII control */
     {
     ascii_type = 2;
     }


switch (ascii_type)
     {
     /* ASCII 0x20 to 0xFF */
     case 1:
         {
         wait();
         LCD_read_control()
         if (LCD_CONTROL == (END_FIRST_LINE+1))  /* if cursor position
         = end of first line */

               {
               LCD_end_cycle()
               lcd_control = (BEGIN_SECOND_LINE | WRITE_DATA);
               wait();
               LCD_CONTROL = lcd_control;   /* place cursor to the second
               line */
               LCD_write_control()
               }
          LCD_end_cycle()
          wait();
          LCD_read_control()
          if (LCD_CONTROL > END_SECOND_LINE)      /* if cursor position
         = end of second line */
               {
               LCD_end_cycle()
               scroll();             /* call scroll procedure  */
```

```
            }
        LCD_end_cycle()
        wait();
        LCD_DATA = c;                  /* write data to LCD */
        LCD_write_data( )
        LCD_end_cycle()
        break;
        }

    case 0:
    /* ASCII 00 : END OF FILE */
        {
  /*      wait();
        if (LCD_CONTROL >= BEGIN_SECOND_LINE)
            {
            scroll();
            }
        wait();
        if (LCD_CONTROL <= END_FIRST_LINE)

            {
            lcd_control = (BEGIN_SECOND_LINE | WRITE_DATA);
            wait();
            LCD_CONTROL = lcd_control;
            }  */

        break;
        }

    case 2:
    /* CONTROL CHARACTERS  */
        {
        switch (c)
            {
            wait();
            /************ \a **************/
            case 7:
                      /* beep sound, not implemented */
                {
                break;
                }
            /************ \b **************/
            case 8:       /* back */
               {
               wait();
```

```
LCD_read_control()
if (LCD_CONTROL <= END_FIRST_LINE)      /* if cursor
on first line */
 {
 LCD_end_cycle()
 wait();
 LCD_read_control()
 if (LCD_CONTROL != BEGIN_FIRST_LINE)  /* if cursor
not in first position */
     {
     LCD_end_cycle()
     wait();
     LCD_read_control()
     lcd_control = ((LCD_CONTROL -1) | WRITE_DATA);
     LCD_end_cycle()
     wait();
     LCD_CONTROL = lcd_control;         /* cursor
position = previous position - 1 */
     LCD_write_control()
     }
 }
LCD_end_cycle()
wait();
LCD_read_control()
if (LCD_CONTROL == BEGIN_SECOND_LINE)  /* if cursor
on first position of second line */
 {
 LCD_end_cycle()
 lcd_control = (END_FIRST_LINE | WRITE_DATA);  /*
cursor position = last position of first line */
 wait();
 LCD_CONTROL = lcd_control;
 LCD_write_control()
 }
LCD_end_cycle()

wait();
LCD_read_control()
if (LCD_CONTROL > BEGIN_SECOND_LINE)     /* if cursor
on second line */
 {
 LCD_end_cycle()
 wait();
 LCD_read_control()
 lcd_control = ((LCD_CONTROL -1) | WRITE_DATA);  /*
```

```
               cursor position = previous position */
                LCD_end_cycle()
                wait();
                LCD_CONTROL = lcd_control;
                LCD_write_control()
                }
               LCD_end_cycle()

               break;
               }

            /************ \t **************/
            case 9:         /* tab */
               {
               wait();
               LCD_read_control()
               if (LCD_CONTROL <= END_FIRST_LINE)        /* if cursor
               on first line */
                   {
                   LCD_end_cycle()
                   wait();
                   LCD_read_control()
                   if (LCD_CONTROL <= (END_FIRST_LINE - TAB)) /* if
                   enough place to tab */
                       {
                       LCD_end_cycle()
                       wait();
                       LCD_read_control()
                       lcd_control = ((LCD_CONTROL + TAB) |
                      WRITE_DATA);  /* cursor position = previous
                      one + TAB */
                       LCD_end_cycle()
                       wait();
                       LCD_CONTROL = lcd_control;
                       LCD_write_control()
                       LCD_end_cycle()
                       }
                   else
                       {
                       /* if not enough room to insert tab */
                       LCD_end_cycle()
                       scroll();
                       /* call scroll procedure */
                       break;
                       }
```

```
        }
    LCD_end_cycle()
    wait();
    LCD_read_control()
    if (LCD_CONTROL >= BEGIN_SECOND_LINE)
    /* if cursor on second line */
        {
        LCD_end_cycle()
        wait();
        LCD_read_control()
        if (LCD_CONTROL <= (END_SECOND_L INE - TAB))
        /* if enough space to tab */
            {
            LCD_end_cycle()
            wait();
            LCD_read_control()
            lcd_control = ((LCD_CONTROL + TAB) |
           WRITE_DATA);   /* tab */
            LCD_end_cycle()
            wait();
            LCD_CONTROL = lcd_control;
            LCD_write_control()
            LCD_end_cycle()
            }
        else
            {
            /* if not enough space */
            LCD_end_cycle()
            scroll();
            /* call scroll procedure */
            }
        }
    LCD_end_cycle()
    break;
    }
/************ \n **************/
case 10:  /* new line , keep current position */
     {
     wait();
     LCD_read_control()
     lcd_control = LCD_CONTROL;
     LCD_end_cycle()
     if (lcd_control >= BEGIN_SECOND_LINE)
    /* if second line */
         {
         scroll();           /* call scroll procedure */
         LCD_CONTROL = lcd_control;
         LCD_write_contro l()
         LCD_end_cycle()
         }
     wait();
     if (lcd_control <= END_FIRST_LINE)
    /* if first line */
         {
         lcd_control |= (BEGIN_SECOND_LINE |
         WRITE_DATA);  /* cursor position = second line */
         wait();
         LCD_CONTROL = lcd_control;
         LCD_write_contro l()
         LCD_end_cycle()
         }

     break;
     }
/************* \v ***************/
case 11:     /* clear the 2 lines LCD display */
     {
     clear_screen();
     break;
     }
/************* \f ***************/
case 12:     /* go to beginning of current line */

    {
    wait();
    LCD_read_control()
    lcd_control = (LCD_CONTROL & ~END_FIRST_LINE);
    LCD_end_cycle()
    LCD_CONTROL = lcd_control;
    LCD_write_control()
    LCD_end_cycle()

    break;
    }
/************** \r ***************/
case 13:  /* carriage return */
    {
    wait();
    LCD_read_control()
    if (LCD_CONTROL >= BEGIN_SECOND_LINE)
```

```
                          /* if cursor on second line */
                               {
                               LCD_end_cycle()
                               scroll();
                               }
                           LCD_end_cycle()
                          wait();
                          LCD_read_control()
                         if (LCD_CONTROL <= END_FIRST_LINE)
                         /* if cursor on first line */
                                  {
                                  LCD_end_cycle()
                                    lcd_control = (BEGIN_SECOND_LINE |
                                    WRITE_DATA);
                                  wait();
                                  LCD_CONTROL = lcd_control;    /* set cursor on
                                 first position */
                                 LCD_write_control()
                                 }
                         LCD_end_cycle()
                         break;
                         }
                      }

                         break;
                         }
             }

        /* restore user application Wait States */

        FLAGR = flagr_back;   /* backup FLAGR */
        WCR = wcr_back;       /* backup WCR */
        PPR = ppr_back;       /* backup PPR */
    }
```