H8/38076

LCD RTC and Power Mode Demonstration

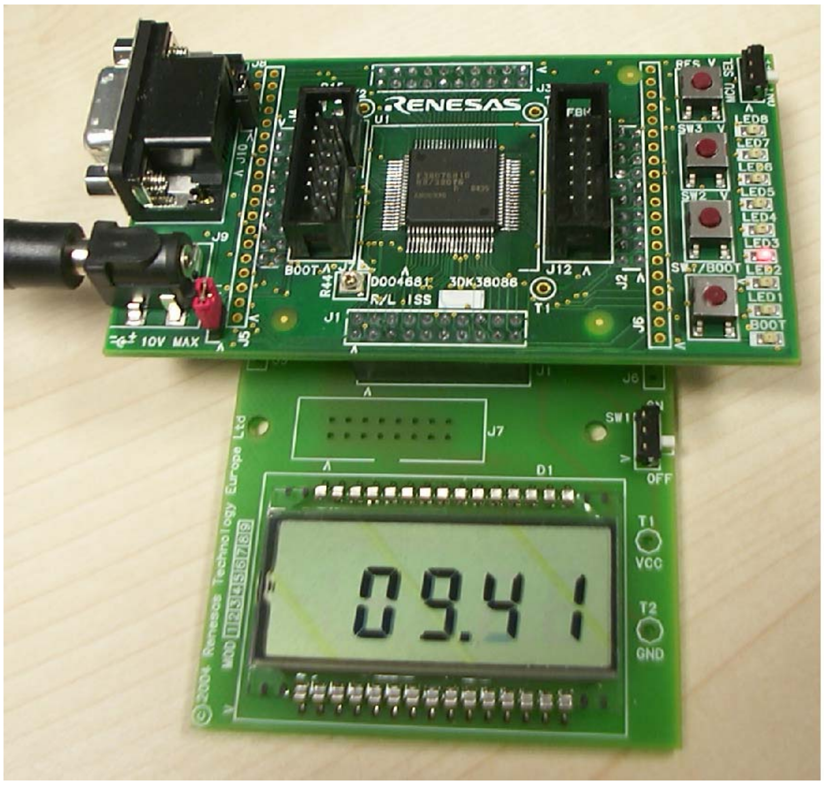
Introduction

The purpose of this application note is to provide an example of the H8S/38076 Real time clock and LCD peripherals in use. The application also demonstrates these peripherals functioning in seven power modes.

The application utilises a 3DK38076 board with LCD baseboard, a current meter and a power supply.

All code was developed using Renesas' High Performance Embedded Workshop (HEW) with the Renesas C/C++ Compiler, Ver.6.02. The code was developed on an H8/38076 using the On-Chip Debug functionality via an E7 Emulator.

A picture of the 3DK38076 with LCD base board and embedded software is shown below.



Contents

INTRODUCTION 1

CONTENTS 2

H8/SLP FAMILY OVERVIEW 4

SYSTEM OVERVIEW 4

REAL TIME CLOCK PERIPHERAL 6

1. RTCCR1&2: RTC CONTROL REGISTERS H'F06C & H'F06D 7
2. RTCCSR – RTC CLOCK SOURCE SELECT REGISTER H'F06F 9
3. RTCFLG – RTC INTERRUPT FLAG REGISTER H'F067 10
4. RSECDR - SECOND /FREE RUNNING COUNTER DATA REGISTER H'F068 11
5. RMINDR - MINUTE DATA REGISTER H'F069 12
6. RHRDR - HOUR DATA REGISTER H'F06A 13
7. RWKDR - WEEK DATA REGISTER H'F06B 14

LCD CONTROLLER PERIPHERAL 15

LCD RAM 16

8. LPCR – LCD PORT CONTROL REGISTER H'FFA0 18
9. LCR – LCD CONTROL REGISTER H'FFA1 19
10. LCR2 – LCD CONTROL REGISTER 2 H'FFA2 21
11. LTRMR – LCD TRIMMING REGISTER H'FFA3 22
12. BGRMR – BGR CONTROL REGISTER H'FFA4 23

POWER MODES AND POWER MODE TRANSITIONS 24

SOFTWARE DESCRIPTION FOR DEMONSTRATION. 26

13. CODE PURPOSE 26
14. EXTERNAL CODE OPERATION 26
15. PROGRAM FLOW 27
16. 38076_CLOCK WORKSPACE 28

FILE NAMES AND PURPOSES 28

MEMORY AND CODE SECTIONS 29

SESSION FILES AND CONFIGURATIONS WITH THIS WORKSPACE 30

17. CODE FUNCTIONS 31

VOID RESETPRG(VOID) 31

void main(void) ... 32
void Hardware_Setup(void) ... 33
void Init_Port5(void) ... 33
void Init_Port9(void) ... 34
void Init_LCD(void) ... 34
void Init_RTC(void) ... 35
Void Set_Power_Mode(void) ... 37
Void UpDate_LCD(void) ... 40
Void Display(x, y) ... 42
18. ISR's ... 46
RTC Minute Interrupt ... 46
IRQ4 (SW3) ... 46
IRQAEC (SW2) ... 47
CURRENT CONSUMPTION AND MEMORY RESOURCE ALLOCATION ... 48
19. Current Measurement Technique ... 48
20. Current Measurement Results ... 49
21. Memory Map ... 49
CONCLUSION ... 51
WEBSITE AND SUPPORT ... 51

H8/SLP Family Overview

The H8/SLP range of devices are based around either the high-speed H8/300H or the high-speed H8/300L cores. The 300H based devices have an internal 16-bit architecture, Sixteen 16-bit general registers, 62 instructions and various peripheral functions. Those based around the H8/300L cores support 55 basic instructions and have eight 16 bit registers.

The low power consumption of these devices means that they are ideal for battery powered applications such as metering.

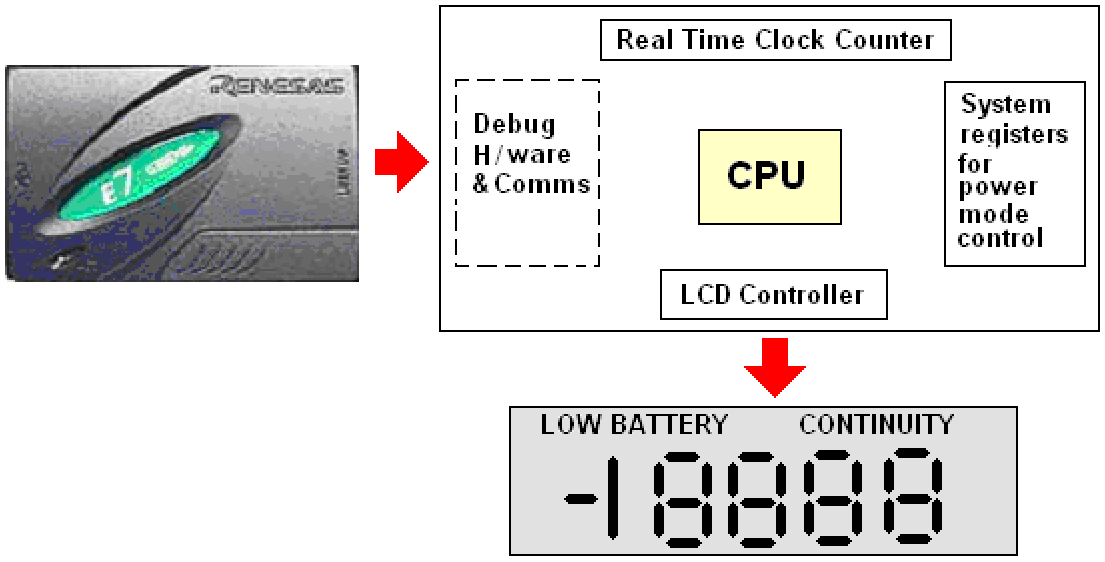
System Overview

The 38076 specifically is an SLP device based around the 300H core. It has the following peripherals:

RTC - Real Time Clock Counter which may also be used as a free running counter.
AEC - Asynchronous Event Counter
LCD controller/driver - For driving an LCD. It supports 32SEGS with 4 COMM lines.
Timer F - 16 bit counter
TPU - 16-bit timer pulse unit
PWM - 14-bit Pulse Width Modulator
WDT - 8 bit Watchdog timer
SCI - Serial Communication Interface
IIC2 - I^{2}C bus interface providing a subset of the Philips I^{2}C format.
ADC - 10-bit Analogue to Digital converter

Figure 1 shows the peripherals used in this demo

Figure 1: System overview



As can be seen from the previous figure (figure 1), the RTC and LCD peripherals are used in this demonstration. The on chip emulator was used in conjunction with an E7 for code development.

Development Environment

Figure 2 shows the environment (hardware and software) used to develop the application.

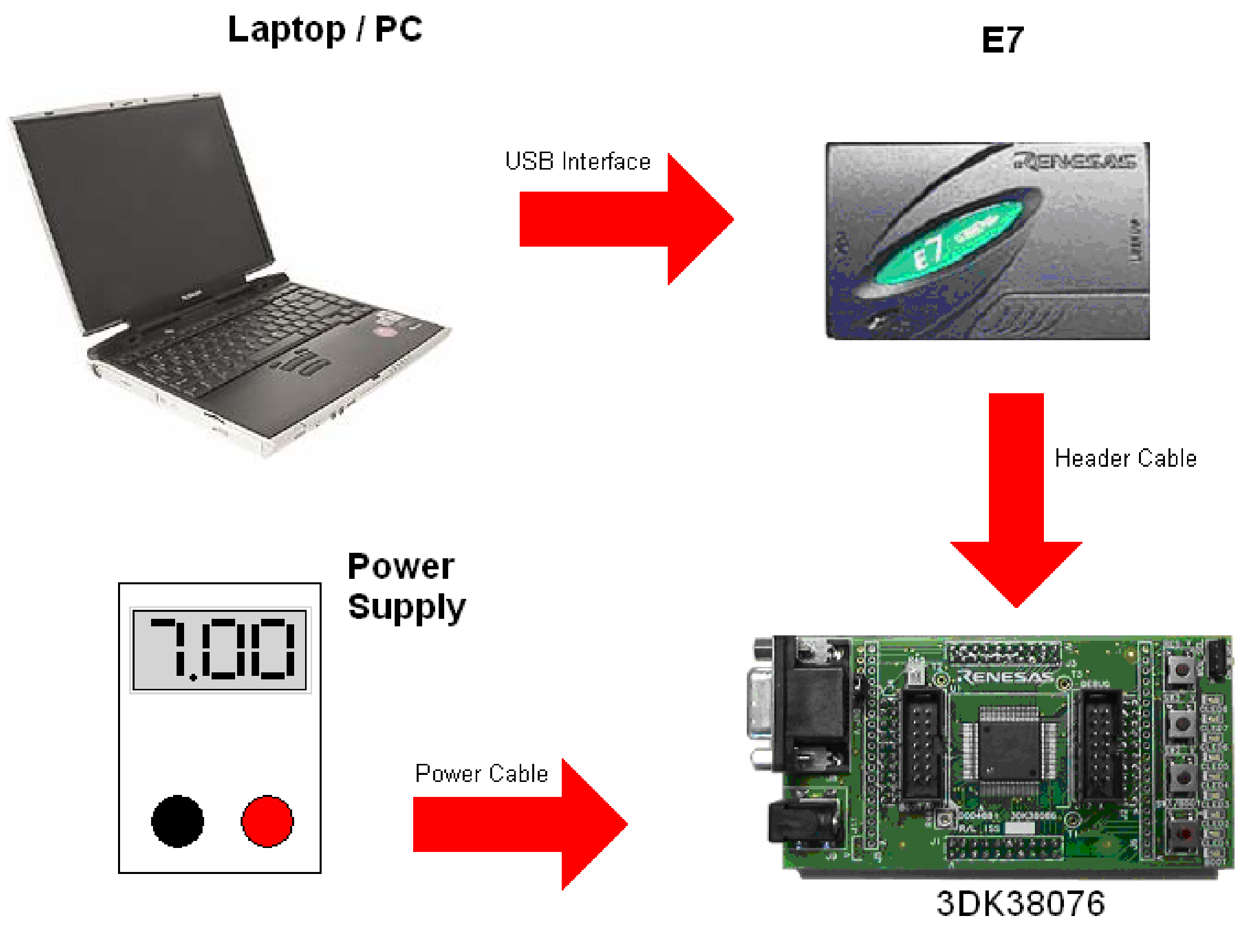


Figure 2: Application Set-up

The laptop PC runs HEW and has the E7 drivers installed. HEW provides a code developing environment, and also allows code to be debugged on the device using the E7.

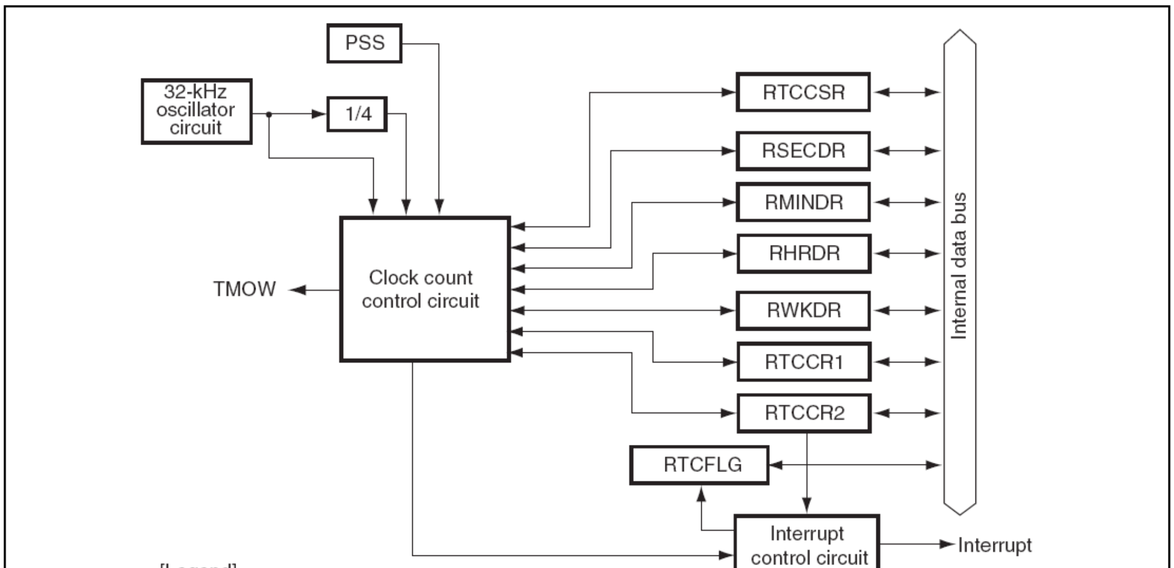
The 3DK38076 provides a hardware platform for the H8/38076 device and connects to the LCD baseboard below.

A centre positive 5V power supply is required to power the 3DK board when the final application is running without intervention with the E7. This ensures LCD clarity. If an E7 is used, provide a 6V power supply or higher (10V maximum) to guarantee flash programming.

Real Time Clock Peripheral

- . The RTC may be started or stopped as required
- The RTC may be Reset
- An 8-bit register is used to hold each of the seconds, minutes, hours and day of the week values. These registers may be read or written to.
- The counter may be used as an 8-bit free running counter.
- The clock source is selectable from ϕ/8, ϕ/32, ϕ/128, ϕ/256, ϕ/512, ϕ/2048, ϕ/4096, ϕ/8192 and the 32.768KHz subclock.
- The RTC module may be placed in module standby mode

A block diagram of the Real time clock is shown below in figure 3



[Legend]

RTCCSR:	Clock source select register	RWKDR:	Day-of-week date register
RSECDR:	Second date register/ free running counter data register	RTCCR1:	RTC control register 1
RMINDR:	Minute date register	RTCCR2:	RTC control register 2
RHRDR:	Hour date register	RTCFLG:	RTC interrupt flag register
		PSS:	Prescaler S

Figure 3: RTC Block Diagram

The RSECDR stores the current values of seconds, the RMINDR holds current value of minutes, the RHRDR holds the value of hours and the RWKDR stores the day of the week. These are connected to the internal bus so that they may be read or written to. The RTCCSR selects the required clock source. The RTCCR1 determines the operation of the RTC peripheral and RTCCR2 enables or disables the RTC interrupts. RTCFLG holds the status of the RTC interrupts.

1. RTCCR1&2: RTC Control Registers H'F06C & H'F06D

RTCCCR1:

	7	6	5	4	3	2	1	0
	RUN	12/24	PM	RST	-	-	-	-
Initial Value:	-	-	-	0	0	0	0	0
Read/Write:	R/W	R/W	R/W	R/W	-	-	-	-

RTCCR2:

	7	6	5	4	3	2	1	0
	FOIE	WKIE	DYIE	HRIE	MNIE	1SEIE	05SEIE	025SEIE
Initial Value:	-	-	-	-	-	-	-	-
Read/Write:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The two 8-bit RTC Control read/write registers configure the RTC behaviours. A description of each of the bits in register RTCCR1 is given below:

Bit 7: **RUN – halts or starts RTC operation**

0: Stops / Halts RTC operation.

1: Starts RTC operation.

Bit 6: **12/24 – sets 12 or 24 hour time measurement**

0: The RTC will count in 12 hour time.
The PM bit in this register indicates whether the time is a morning or afternoon value. The values in the hour data register will be in the range 0 to 11.

1: The RTC will count in 24 hour time.
The values in the hour data register will be in the range 0 to 23.

Bit 5: **PM – indicates morning or afternoon time**

0: The time given by the RTC peripheral is AM

1: The time given by the RTC peripheral is PM

Bit 4: **RST – RTC Reset**

0: The RTC operates normally (i.e. no reset of the RTC is initiated)

1: All RTC registers apart from RTCCSR and this bit are reset. The bit must be

manually put to 0 after the reset is initiated. All control circuits within the RTC are also reset.

Bit 3 to 0: **Reserved**

The RTCCR2 register enables / disables any of the RTC interrupts. The RTC has 8 interrupt sources. A description of all the bits in RTCCR2 is given below:

Bit 7: **FOIE – Free Running Counter Interrupt Enable**

0: Disables the Free running counter overflow interrupt.

1: Enables the Free running counter overflow interrupt.

Bit 6: **WKIE – Week Periodic Interrupt Enable**

0: Disables a week periodic interrupt.

1: Enables a week periodic interrupt. (This will occur once every week).

Bit 5: **DYIE – Day Periodic Interrupt Enable**

0: Disables a day periodic interrupt.

1: Enables a day periodic interrupt. (This will occur once every day).

Bit 4: **HRIE – Hour Periodic Interrupt Enable**

0: Disables an hour periodic interrupt.

1: Enables an hour periodic interrupt. (This will occur once every hour).

Bit 3: **MNIE – Minute Periodic Interrupt Enable**

0: Disables a minute periodic interrupt.

1: Enables a minute periodic interrupt. (This will occur once every minute).

Bit 2: **1SEIE – 1 Second Periodic Interrupt Enable**

0: Disables a second periodic interrupt.

1: Enables a second periodic interrupt. (This will occur once every second).

Bit 1: **05SEIE – 0.5 Second Periodic Interrupt Enable**

0: Disables a 0.5 second periodic interrupt.

1: Enables a 0.5 second periodic interrupt. (This will occur once every half second).

Bit 0: **025SEIE – 0.25 Second Periodic Interrupt Enable**

0: Disables a 0.25 second periodic interrupt.

1: Enables a 0.25 second periodic interrupt. (This will occur once quarter of a second).

2. RTCCSR – RTC Clock Source Select Register H'F06F

	7	6	5	4	3	2	1	0
	-	RCS6	RCS5	SUB32K	RCS3	RCS2	RCS1	RCSO
Initial Value:	0	0	0	0	1	0	0	0
Read/Write:	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The 8-bit R/W RTC Control Source Select register selects the clock source to be used for output on the TMOW pin and the clock source to be used when the RTC is used as a free running counter. If the RTC peripheral is to be used as an RTC clock, the 32 KHz clock must be selected. A description of each of the bits in register RTCCSR is given below:

Bit 7: **Reserved**

Bit 6 to 4: **RCS[6:5] & SUB32K – Clock output selection from TMOW pin when TMOW bit in PMR3 is 1.**

000: Clock output will be ϕ/4.

010: Clock output will be ϕ/8.

100: Clock output will be ϕ/16.

110: Clock output will be ϕ/32.

XX1: Clock output will be ϕw.

Bit 3 to 0: **RCS[3:0] – indicates morning or afternoon time**

0000: Free running counter operation…. ϕ/8.

0001: Free running counter operation…. ϕ/32.

0010: Free running counter operation…. ϕ/128.

0011: Free running counter operation…. ϕ/256.

0100: Free running counter operation…. ϕ/512.

0101: Free running counter operation…. ϕ/2048.

0110: Free running counter operation…. ϕ/4096.

0111: Free running counter operation…. ϕ/8192.

1XXX: 32.768 KHz for RTC operation

3. RTCFLG – RTC Interrupt Flag Register H'F067

	7	6	5	4	3	2	1	0
	FOIFG	WKIFG	DYIFG	HRIFG	MNIFG	SEIFG	05SEIFG	025SEIFG
Initial Value:	0	0	0	0	0	0	0	0
Read/Write:	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The RTCLFG register is an 8 bit R/W register which stores the status of all the interrupt flags. A description of all the flags within this register is given below:

Bit 7: **FOIFG – Free Running Counter**

When this bit is a 1, the free running counter interrupt has occurred. Write a 0 to this bit to clear the flag.

Bit 6: **WKIFG – Week Periodic Flag**

When this bit is a 1, the RTC has generated a week counter interrupt. Write a 0 to this bit to clear the flag.

Bit 5: **DYIFG – Day Periodic Flag**

When this bit is a 1, the RTC has generated a day counter interrupt. Write a 0 to this bit to clear the flag.

Bit 4: **HRIFG – Hour Periodic Flag**

When this bit is a 1, the RTC has generated an hour counter interrupt. Write a 0 to this bit to clear the flag.

Bit 3: **MNIFG – Minute Periodic Flag**

When this bit is a 1, the RTC has generated a minute counter interrupt. Write a 0 to this bit to clear the flag.

Bit 2: **1SEIFG – 1 Second Periodic Flag**

When this bit is a 1, the RTC has generated a second counter interrupt. Write a 0 to this bit to clear the flag.

Bit 1: **05SEIFG – 0.5 Second Periodic Flag**

When this bit is a 1, the RTC has generated a half second counter interrupt. Write a 0 to this bit to clear the flag.

Bit 0: **025SEIFG – 0.25 Second Periodic Interrupt Enable**

When this bit is a 1, the RTC has generated a quarter of a second interrupt. Write a 0 to this bit to clear the flag.

4. RSECDR - Second /Free Running Counter Data Register H'F068

	7	6	5	4	3	2	1	0
	BSY	SC12	SC11	SC10	SC03	SC02	SC01	SC00
Initial Value:	-	-	-	-	-	-	-	-
Read/Write:	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The RSECDR register is an 8 bit Read register that stores the seconds value of the RTC time value. A seconds value may be written to the register. See section 5.3 for details.

Bit 7: **BSY – Busy: Indicates when seconds value should/ should not be read.**

When this bit is a 1, the second value contained in the register should not be read. Wait until the BSY bit has cleared before reading the value.

Bit [6:4]: **SCI[2:0] – Tens value of seconds**

This holds the current tens value of seconds. The value stored here will be within the range 0:5. Only read this value when the BSY bit is 0.

Bit [3:0]: **SC0[3:0] – Units value of seconds**

This holds the current unit value of seconds. The value stored here will be within the range 0:59. Only read this value when the BSY bit is 0.

5. RMINDR - Minute Data Register H’F069

	7	6	5	4	3	2	1	0
	BSY	MN12	MN11	MN10	MN03	MN02	MN01	MN00
Initial Value:	-	-	-	-	-	-	-	-
Read/Write:	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The RMINDR register is an 8 bit Read register that stores the minute value of the RTC time value. A minute value may be written to the register. See section 5.3 for details.

Bit 7: **BSY – Busy: Indicates when seconds value should/ should not be read.**

When this bit is a 1, the minute value contained in the register should not be read. Wait until the BSY bit has cleared before reading the value.

Bit [6:4]: **MNI[2:0] – Tens value of minutes**

This holds the current tens value of minutes. The value stored here will be within the range 0:5. Only read this value when the BSY bit is 0.

Bit [3:0]: **MN0[3:0] – Units value of minutes**

This holds the current unit value of minutes. The value stored here will be within the range 0:59. Only read this value when the BSY bit is 0.

6. RHRDR - Hour Data Register H'F06A

	7	6	5	4	3	2	1	0
	BSY	-	HR11	HR10	HR03	HR02	HR01	HR00
Initial Value:	-	0	-	-	-	-	-	-
Read/Write:	R	-	R/W	R/W	R/W	R/W	R/W	R/W

The RHRDR register is an 8 bit Read register that stores the hour value of the RTC time value. An hour value may be written to the register. See section 5.3 for details.

Bit 7: **BSY – Busy: Indicates when seconds value should/ should not be read.**

When this bit is a 1, the hour value contained in the register should not be read. Wait until the BSY bit has cleared before reading the value.

Bit 6: **Reserved**

Bit [5:4]: **HR1[1:0] – Tens value of hours**

This holds the current tens value of hours. The value stored here will be within the range 0:2. Only read this value when the BSY bit is 0.

Bit [3:0]: **HR0[3:0] – Units value of hours**

This holds the current unit value of hours. The value stored here will be within the range 0:9. Only read this value when the BSY bit is 0.

7. RWKDR - Week Data Register H'F06B

	7	6	5	4	3	2	1	0
	BSY	-	-	-	-	WK2	WK1	WK0
Initial Value:	-	0	0	0	0	-	-	-
Read/Write:	R	-	-	-	-	R/W	R/W	R/W

The RWKDR register is an 8 bit Read register that stores the hour value of the RTC time value. An hour value may be written to the register. See section 5.3 for details.

Bit 7: **BSY – Busy: Indicates when seconds value should/ should not be read.**

When this bit is a 1, the minute value contained in the register should not be read. Wait until the BSY bit has cleared before reading the value.

Bit [6:3]: **Reserved**

Bit [2:0]: **WK[2:0] –Indicates the day of week**

This holds the current day of the week as a value.

000:	Sunday
001:	Monday
010:	Tuesday
011:	Wednesday
100:	Thursday
101:	Friday
110:	Saturday
111:	Setting prohibited

The value stored here will be within the range 0:7. Only read this value when the BSY bit is 0.

LCD Controller Peripheral

- The display capacity is 32 SEG using static, ½, 1/3 or ¼ duty.
- LCD RAM capacity of 16bytes. This may also be accessed in words.
- Unused segment pins may be used as IO pins in groups of four.
- There is a choice of 11 frame frequencies.
- Either an A or a B waveform can be selected via software as the output.
- There is an on chip power supply split resistor.
- LCD display is possible in all power modes other than Standby mode.
- The device can supply a 3V constant voltage to the LCD without using VCC voltage.
- The output of the 3V constant voltage supply circuit is adjustable.
- The LCD module may be placed in module standby mode

Figure 4 shows the block diagram of the LCD Controller

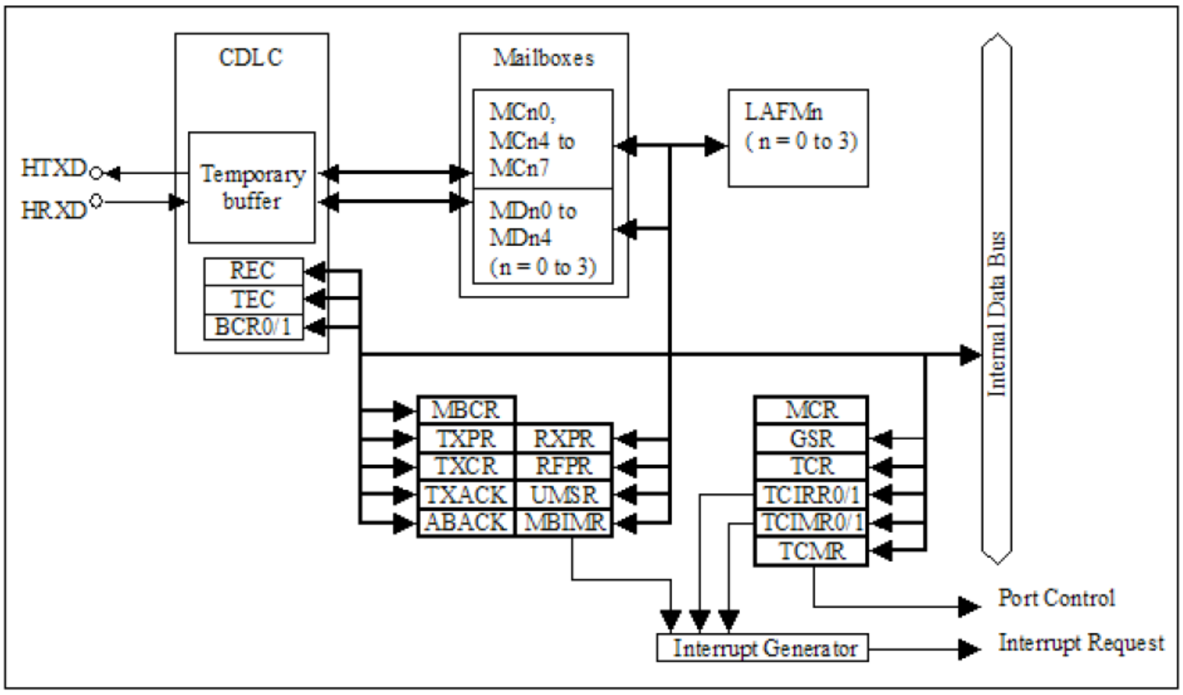


Figure 4: LCD Controller block diagram

LCD RAM

The LCD RAM is an area of memory within the device used to contain display information. The values written here will determine the value output on the corresponding SEG line. Figure 5 illustrates how the LCD memory is arranged when the LCD peripheral is set to ¼ duty.

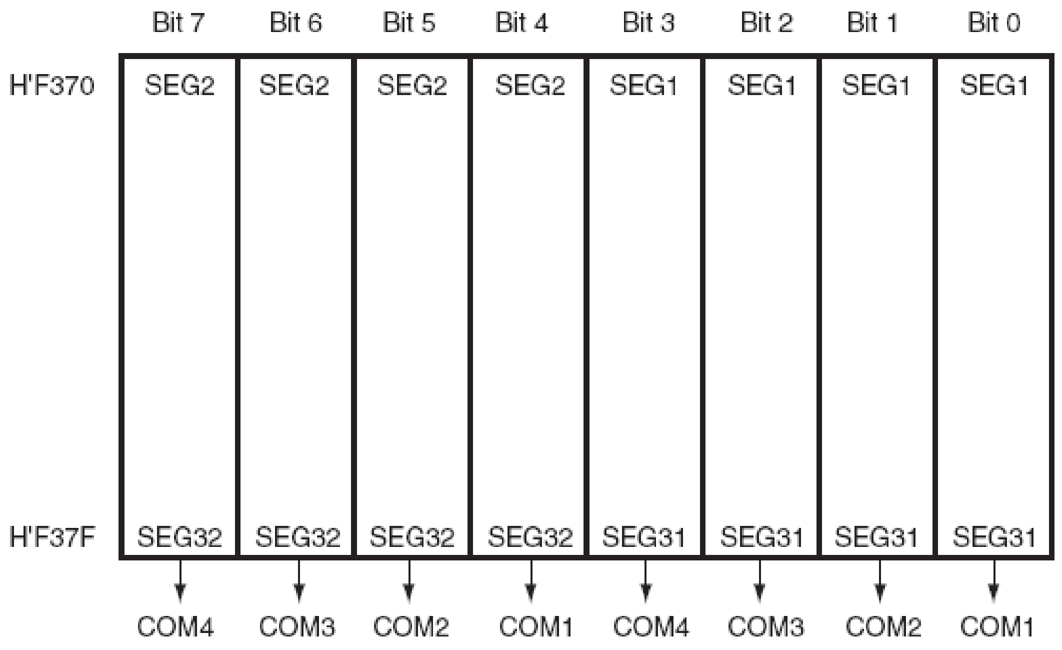


Figure 5: LCD RAM Memory arrangement with LCD controller set to ¼ duty.

With ¼ duty COM1 to COM4 is used. The SEG line values are arranged with even number segments values located in the upper nibble of the memory. Odd number segments are located in the lower nibble of the LCD RAM.

The arrangement of the LCD RAM when the duty is 1/3 is similar but the COM4 column is not used. The LCD RAM memory for ½ duty is shown in figure 6. The usable LCD RAM memory is halved, and only COM 1 and 2 are used. For static mode the memory is quartered and only COM1 is used.

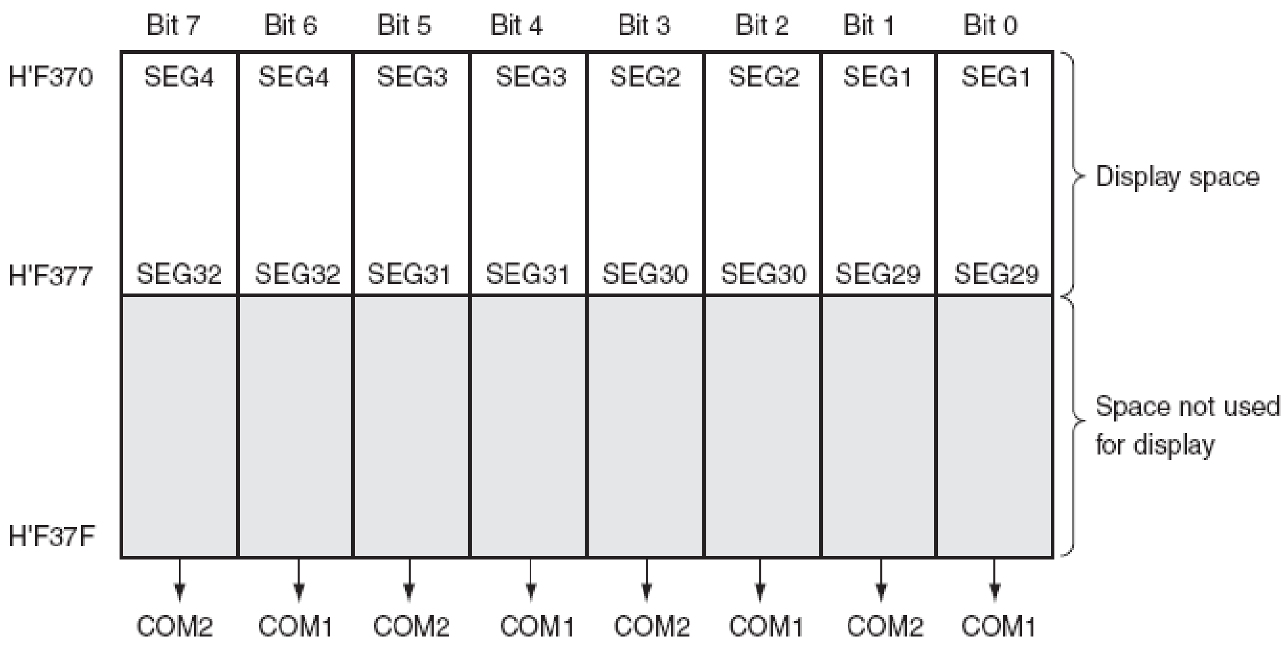


Figure 6: LCD RAM Memory arrangement when LCD controller duty is set to ½ duty.

8. LPCR – LCD Port Control Register H'FFA0

	7	6	5	4	3	2	1	0
	DTS1	DTS0	CMX	-	SGS3	SGS2	SGS1	SGS0
Initial Value:	0	0	0	-	0	0	0	0
Read/Write:	R/W	R/W	R/W	W	R/W	R/W	R/W	R/W

The LPCR is an 8 – bit R/W register used to select the duty cycle, the duty and the segment driver. A description of the use of each bit in the register is given below:

Bit [7:5]: **DTS[1:0] & CMX – Duty Cycle Select 1 and 0, Common Function Select**

These bits select the duty cycle and the common driver. See below for the DTS and CMX values that give the required duty cycle.

000	Static duty cycle using COM1 (Do not use COM4, COM3, COM2)
001	Static duty using COM4 to COM1, all of which output the same waveform.
010	½ duty using COM2 to COM1 (DO not use COM4 or COM3).
011	½ duty using COM4 to COM1, COM4 outputs same waveform as COM3 COM2 outputs same waveform as COM1
100	1/3 duty using COM3 to COM1 (Do not use COM4).
101	1/3 duty using COM4 to COM1 (Do not use COM4). This provides identical behaviour as the setting above.
1XX	¼ duty using COM 4 to COM1

Bit 4: **Reserved**

Bit [3:0]: **SGS[3:0] – Segment Driver Select 3:0**

These bits select which pins of the device are to be used to drive the LCD. Table 1 shows the relationship between the SGS values and the device IO pin used for LCD display.

Bit 3: SGS3	Bit 2: SGS2	Bit 1: SGS1	Bit 0: SGS0	SEG32 to SEG29	SEG28 to SEG25	SEG24 to SEG21	SEG20 to SEG17	SEG16 to SEG13	SEG12 to SEG9	SEG8 to SEG5	SEG4 to SEG1
0	0	0	0	PORT	PORT	PORT	PORT	PORT	PORT	PORT	PORT
0	0	0	1	PORT	PORT	PORT	PORT	PORT	PORT	PORT	SEG
0	0	1	0	PORT	PORT	PORT	PORT	PORT	PORT	SEG	SEG
0	0	1	1	PORT	PORT	PORT	PORT	PORT	SEG	SEG	SEG
0	1	0	0	PORT	PORT	PORT	PORT	SEG	SEG	SEG	SEG
0	1	0	1	PORT	PORT	PORT	SEG	SEG	SEG	SEG	SEG
0	1	1	0	PORT	PORT	SEG	SEG	SEG	SEG	SEG	SEG
0	1	1	1	PORT	SEG	SEG	SEG	SEG	SEG	SEG	SEG
1	0	0	0	SEG	SEG	SEG	SEG	SEG	SEG	SEG	SEG
1	0	0	1	SEG	SEG	SEG	SEG	SEG	SEG	SEG	PORT
1	0	1	0	SEG	SEG	SEG	SEG	SEG	SEG	PORT	PORT
1	0	1	1	SEG	SEG	SEG	SEG	SEG	PORT	PORT	PORT
1	1	0	0	SEG	SEG	SEG	SEG	PORT	PORT	PORT	PORT
1	1	0	1	SEG	SEG	SEG	PORT	PORT	PORT	PORT	PORT
1	1	1	0	SEG	SEG	PORT	PORT	PORT	PORT	PORT	PORT
1	1	1	1	SEG	PORT	PORT	PORT	PORT	PORT	PORT	PORT

Table 1: Relationship between the SGS[3:0] bit values and the port pin use.

9. LCR – LCD Control Register H'FFA1

	7	6	5	4	3	2	1	0
	-	PSW	ACT	DISP	CKS3	CKS2	CKS1	CKS0
Initial Value:	1	0	0	0	0	0	0	0
Read/Write:	-	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 7: **Reserved**

Bit 6: **PSW – LCD Drive Power Supply Control**

This bit is used to turn off the LCD drive power supply when the LCD display is not required (either in a power down mode or when an external power supply is used).

0: LCD drive power supply is turned off

1: LCD drive power supply is turned on

Bit 5: **ACT – Display Function Activate**

This bit halts or starts operation of the LCD controller/driver. When the LCD driver is halted, the LCD drive power supply is also turned off, regardless of the setting of the PSW bit. The LCD register peripheral contents are retained.

0: LCD controller/driver halts

1: LCD controller/driver operates

Bit 5: **DISP – Display Data Control**

This bit specifies whether the LCD RAM contents or blank data is displayed on the LCD regardless of the LCD RAM contents.

0: Blank data is displayed

1: LCD RAM data is displayed

Bit [4:0]: **CKS[4:0] – Frame Frequency Select [4:0]**

These bits are used to select the operating clock and the frame frequency used by the LCD controller. Since only the subclock is available in some power modes, (subactive, subsleep and watch mode) one of the clocks ϕw, ϕw/2 or ϕw/4 should be selected if LCD display is required in all power modes.

					Frame Frequency	
CKS3	CKS2	CKS1	CKS0	Operating clock	ϕ = 2 MHz	ϕ = 250 kHz
0	X	0	0	ϕw	128 Hz	128Hz
0	X	0	1	ϕw/2	64 Hz	64 Hz
0	X	1	X	ϕw/4	32 Hz	32 Hz
1	0	0	0	ϕ/2	-	244 Hz
1	0	0	1	ϕ/4	977 Hz	122 Hz
1	0	1	0	ϕ/8	488 Hz	61 Hz
1	0	1	1	ϕ/16	244 Hz	30.5 Hz
1	1	0	0	ϕ/32	122 Hz	-
1	1	0	1	ϕ/64	61 Hz	-
1	1	1	0	ϕ/128	30.5 Hz	-
1	1	1	1	ϕ/256	-	-

10. LCR2 – LCD Control Register 2 H'FFA2

	7	6	5	4	3	2	1	0
	LCDAB	HCKS	CHG	SUPS	-	-	-	-
Initial Value:	0	0	0	0	-	-	-	-
Read/Write:	R/W	R/W	R/W	R/W	W	W	W	W

Bit 7: **LCDAB – Waveform A or B select**

When this bit is a 1, the free running counter interrupt has occurred. Write a 0 to this bit to clear the flag.

Bit 6: **HCKS – Step up clock selection for 3V constant voltage power supply circuit**

0: The clock selected by bits CKS3 to CKS0 in LCR is divided into 4

1: The clock selected by bits CKS3 to CKS0 in LCR is divided into 8.

Bit 5: **CHG – Connection control of LCD power supply split resistor**

Selects whether an LCD power supply split resistor is disconnected or connected from or to an LCD drive supply.
0: Disconnected
1: Connected

Bit 4: **SUPS – 3V constant voltage power supply control**

Used to turn off 3V constant voltage supply. The power supply is turned off when BGRSTPN in BGRMR is cleared to 0 even if SUPS is 1.

Bit [3:0] **Reserved**

11. LTRMR – LCD Trimming Register H'FFA3

	7	6	5	4	3	2	1	0
	TRM3	TRM2	TRM1	TRM0	-	CTRM2	CTRM1	CTRM0
Initial Value:	0	0	0	0	1	0	0	0
Read/Write:	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W

Bit [7:4] **TRM[3:0] Output voltage adjustment of 3V constant voltage power supply circuit**

0000: 3.00V
0001: 2.94V
0010: 2.88V
0011: 2.85V
0100: 2.79V
0101: 2.76V
0110: 2.70V
0111: 2.67V
1000: 3.48V
1001: 3.42V
1010: 3.36V
1011: 3.30V
1100: 3.24V
1101: 3.18V
1110: 3.12V
1111: 3.06V

Bit 3: **Reserved**

Bit [2:0]: CTRM[2:0] **Variable Voltage Adjustment of 3V constant voltage power supply**

3V power supply used for LCD drive power supply adjustable within range of 3V+/-10%. Set these bits to adjust panel functions for temperature.

000: 3.00V
001: 3.09V
010: 3.18V
011: 3.27V
100: 2.64V
101: 2.73V
110: 2.82V
111: 2.91V

12. BGRMR – BGR Control Register H'FFA4

	7	6	5	4	3	2	1	0
	BGRSTPN	-	-	-	-	BTRM2	BTRM1	BTRM0
Initial Value:	0	0	0	0	1	0	0	0
Read/Write:	R/W	R/W	R/W	R/W	-	R/W	R/W	R/W

Bit 7: **BGRSTPN – Band Gap Reference Circuit Control**

This controls whether the band gap reference circuit operates or halts.
0: Band gap reference circuit halts.
1: Band gap reference circuit operates.

Bit [6:3]: **Reserved**

Bit [2:0]: **BGR – Output Voltage Trimming**

Adjust approximately 1.2-V BGR output voltage. Selects whether an LCD power supply split resistor is disconnected or not from/to an LCD drive supply.
0: Disconnected
1: Connected

Power Modes and Power Mode Transitions

The H8/38076 device can operate in various power modes. This allows the power consumption of the device to be minimised in specific applications. Figure 7 shows how the transitions between various power modes are made. The conditions of the transitions are given in table 2 and 3.

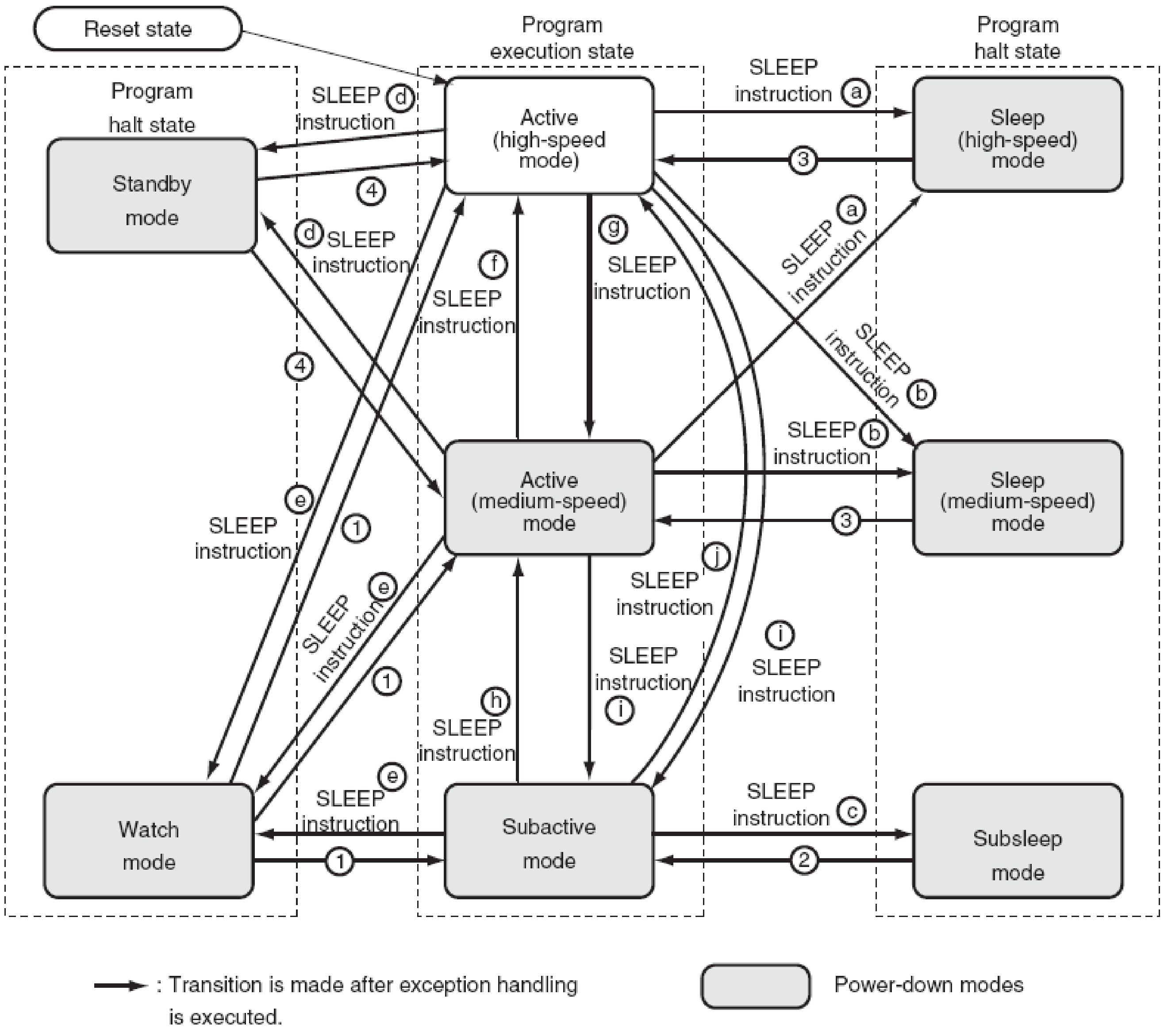


Figure 7: Power mode transitions in 38076 device.

The round edge grey boxes indicate the various power modes the device can enter. These determine whether the device runs from the subclock or the system clock and which functions of the device are enabled.

The thick black arrows indicate power mode transitions. The arrow head points to the power mode which the transition is going to. There are two types of transitions; those marked numerically which are interrupt driven and those marked alphabetically which indicate transitions initiated by a SLEEP instruction.

As the diagram shows, the device operates in active high speed mode when it first comes out of reset. The device is capable of making a direct transition to any power mode except subsleep mode. However a transition is possible from active high speed mode to subsleep mode via subactive mode. It is possible to go from any power mode to any other power mode either via direct transitions or by using other power modes.

The three power modes shown on the right hand side indicate "sleep"modes. In these modes the CPU operation will be halted but most peripherals still function. Any interrupt which occurs whilst the device is in sleep mode will cause a transition out of the sleep mode.

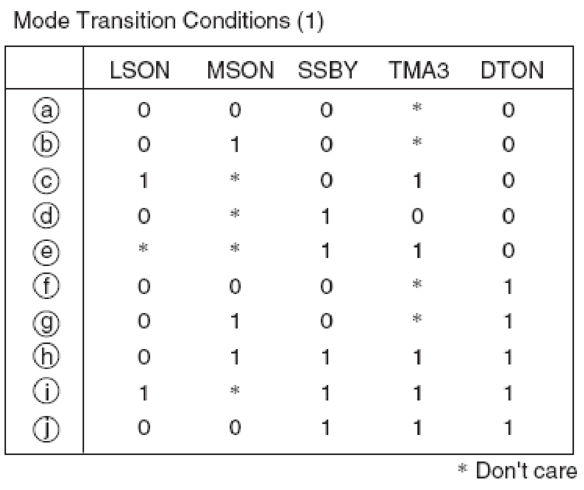
The two top-most power modes in the middle column are active modes. In these modes both clock oscillators, the CPU, and all on-chip peripheral modules function.

In subactive and subsleep mode the subclock is used as the clock source. The system clock is halted.

In Standby and Watch mode the system clock oscillator stops, the CPU and selected on-chip peripheral modules stop functioning. (All peripherals stop functioning when in Standby mode).

Tables 2 and 3 show the conditions that should be met for a particular mode transition

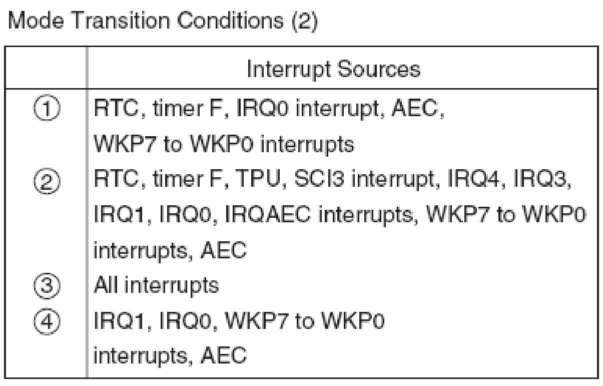
Table 2: SLEEP instruction activated transitions Table 3: Interrupt activated transitions



Mode Transition Conditions (1)

	LSON	MSON	SSBY	TMA3	DTON
ⓐ	0	0	0	*	0
ⓑ	0	1	0	*	0
ⓒ	1	*	0	1	0
ⓓ	0	*	1	0	0
ⓔ	*	*	1	1	0
ⓕ	0	0	0	*	1
ⓖ	0	1	0	*	1
ⓗ	0	1	1	1	1
ⓘ	1	*	1	1	1
ⓙ	0	0	1	1	1

* Don't care



Mode Transition Conditions (2)

	Interrupt Sources
①	RTC, timer F, IRQ0 interrupt, AEC, WKP7 to WKP0 interrupts
②	RTC, timer F, TPU, SCI3 interrupt, IRQ4, IRQ3, IRQ1, IRQ0, IRQAEC interrupts, WKP7 to WKP0 interrupts, AEC
③	All interrupts
④	IRQ1, IRQ0, WKP7 to WKP0 interrupts, AEC

There are two types of transitions. Those driven by interrupt (required for power modes that halt the CPU) and those instigated by the sleep instruction in combination with the status of the LSON, MSON, SSBY, TMA3 and DTON bits in SYSCR1 and SYSCR2.

Software Description for demonstration

13. Code Purpose

The software for this application is designed to demonstrate the RTC and LCD peripherals in operation in various power modes. The code prompts the user to enter an initial time shown on the LCD in 24 hour format. The time displayed on the LCD will then increment on the minute, every minute and the power mode the device operates in will also change.

14. External code operation

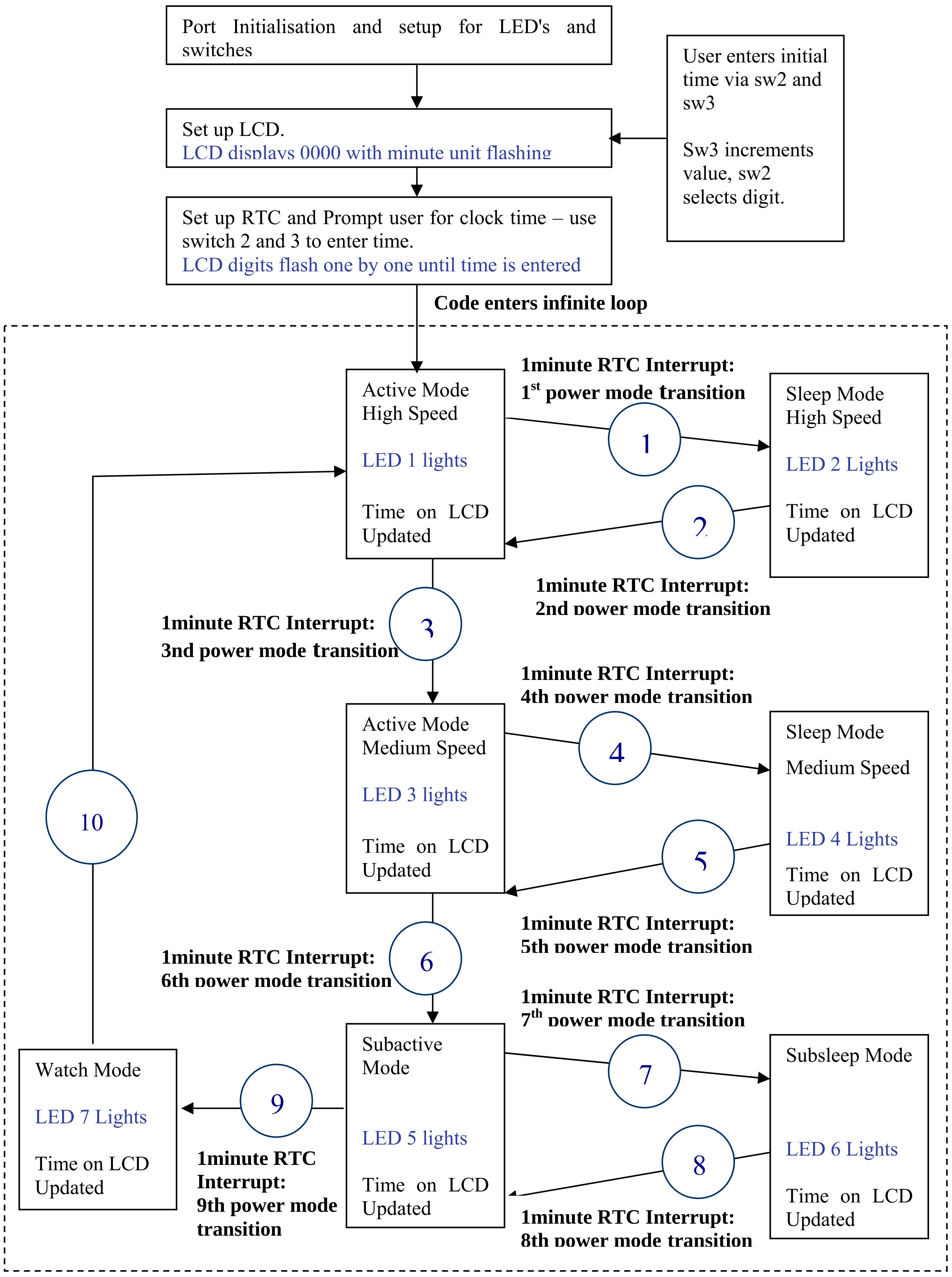
The software runs on 3DK38076 connected to the LCD base board. When the code starts to execute all the LED's on the 3DK will light. This notifies the user that the board is functional. The LCD located on the LCD baseboard will have “00.00” displayed on it, with the last “0” flashing. The user may now enter a time to start the real time clock peripheral. Switch 3 may be used to increment the value displayed, switch 2 to select the required digit for incrementation. If the user makes a mistake when entering the time or changes his/her mind the reset switch may be pressed and the time entered again after the reset.

Once a suitable time value is entered onto the LCD and set, the Clock value displayed on the LCD will start to increment every minute. The power mode will also change every minute. There are 8 LED's on the 3DK which are used to indicate what power mode the board is in at any particular time. See figure 8 for an explanation of the program flow and the power mode transitions.

15. Program Flow

The flow of the software program is shown in figure 8.

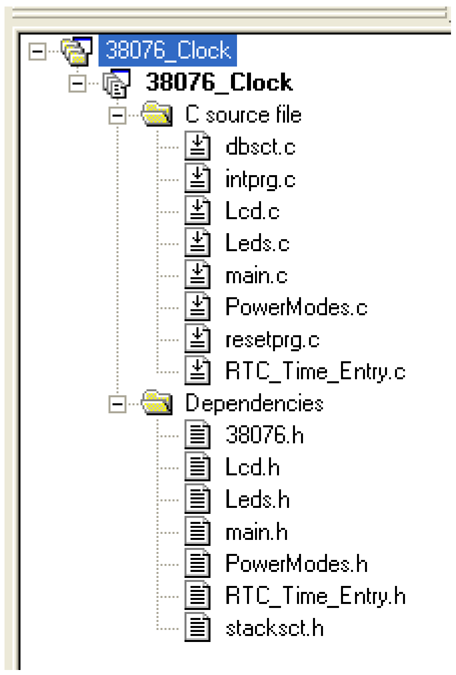
Figure 8: Program Flow diagram


Port Initialisation and setup for LED's and switches
Set up LCD.
LCD displays 0000 with minute unit flashing
User enters initial time via sw2 and sw3
Sw3 increments value, sw2 selects digit.
Set up RTC and Prompt user for clock time – use switch 2 and 3 to enter time.
LCD digits flash one by one until time is entered
Code enters infinite loop
Active Mode High Speed
LED 1 lights
Time on LCD Updated
1minute RTC Interrupt: 1st power mode transition
1
Sleep Mode High Speed
LED 2 Lights
Time on LCD Updated
2
1minute RTC Interrupt: 2nd power mode transition
1minute RTC Interrupt: 3nd power mode transition
3
1minute RTC Interrupt: 4th power mode transition
Active Mode Medium Speed
LED 3 lights
Time on LCD Updated
4
Sleep Mode
Medium Speed
LED 4 Lights
Time on LCD Updated
5
1minute RTC Interrupt: 5th power mode transition
10
1minute RTC Interrupt: 6th power mode transition
6
1minute RTC Interrupt: 7th power mode transition
Subactive Mode
LED 5 lights
Time on LCD Updated
7
Subsleep Mode
LED 6 Lights
Time on LCD Updated
8
1minute RTC Interrupt: 8th power mode transition
Watch Mode
LED 7 Lights
Time on LCD Updated
9
1minute RTC Interrupt: 9th power mode transition

This figure shows the first step of the code is to set up the LED ports so they may all light on a reset. The LCD peripheral is then configured (using the LCD peripheral registers described in the previous section "LCD Controller Peripheral") to display “00.00”. The last “0” will then flash, and the user will enter a time in 24 hour mode (see section 14 for details on entering a time value). The RTC is then initialised and the value obtained from the user placed in the appropriate time data registers (see section "Real Time Clock Peripheral"). The RTC minute interrupt is enabled and the code then enters an infinite loop. This loop repeatedly updates the time value on the LCD and controls the transition to the next power mode on receiving a minute interrupt from the RTC.

16. 38076_Clock Workspace

The workspace when opened will appear as figure 9.



```
38076_Clock
  38076_Clock
    C source file
      dbsct.c
      intprg.c
      Lcd.c
      Leds.c
      main.c
      PowerModes.c
      resetprg.c
      RTC_Time_Entry.c
    Dependencies
      38076.h
      Lcd.h
      Leds.h
      main.h
      PowerModes.h
      RTC_Time_Entry.h
      stacksct.h
```

Figure 9: Workspace view

File Names and Purposes

“main.c”: The main function is held in “main.c”. This holds some initialisation functions and the infinite loop discussed in figure 8. It also contains the interrupt service routines for the switches and the real time clock.

“PowerModes.c”: Holds a function for controlling the power mode transitions of the device and lighting an LED to indicate which power mode the device is in at any moment.

“RTC_Time_Entry.c”: Controls the user entry of the initial clock time to be displayed on the LCD. This value is then entered into the RTC time registers.

“**Lcd.c**”: Contains various LCD functions. Some of these functions flash characters, another displays specific characters at a given location on the LCD and another updates the RTC value stored on the LCD.

“Leds.c”: A short file with functions to light individual LED's.

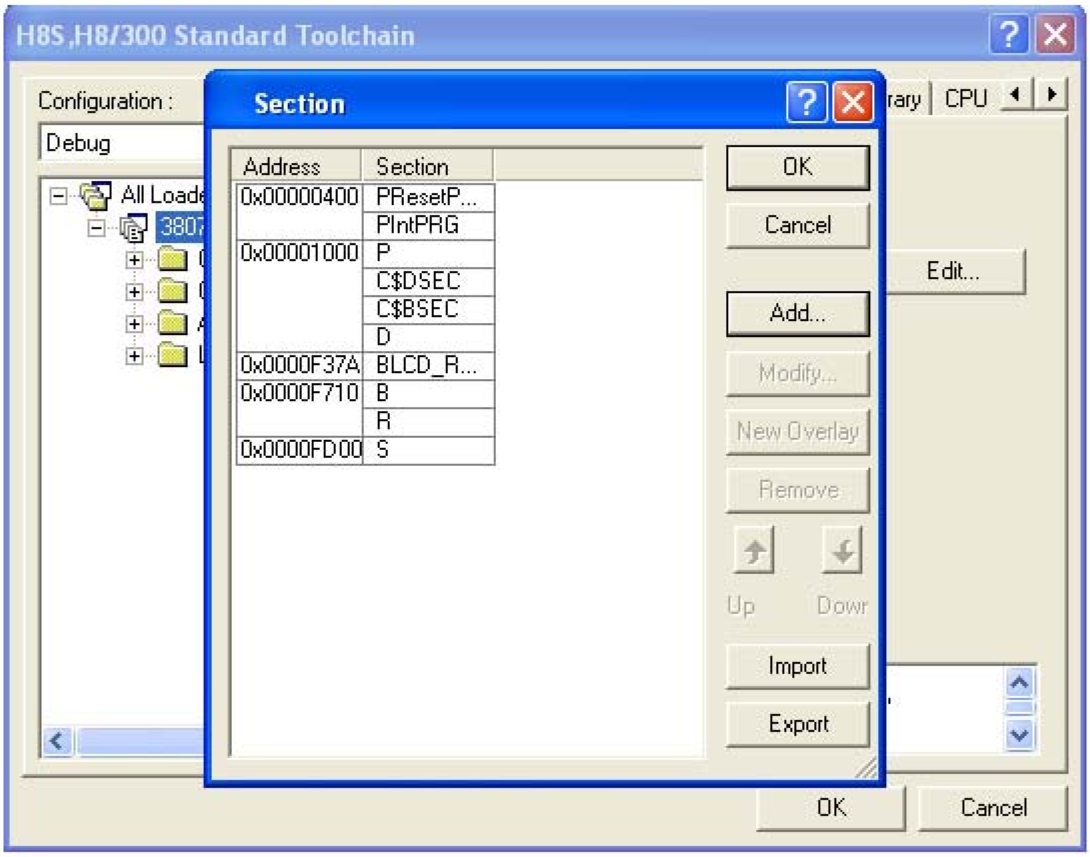
“intprg.c”: Stores the default execution code for all Interrupt Service Routines.

“Resetprg.c”: The code in this file determines what the device does immediately following a hardware reset.

“dbsct.c”: Initialises and sets up memory sections.

Memory and Code Sections

The memory sections are shown in figure 10.



Address	Section
0x00000400	PResetP...
	PIntPRG
0x00001000	P
	C$DSEC
	C$BSEC
	D
0x0000F37A	BLCD_R...
0x0000F710	B
	R
0x0000FD00	S

Figure 10: Section information

"PResetPRG" starts at 0x400 and contains the ResetPRG function. This tells the device what to do once a hardware reset has completed.

"PIntPRG" holds the default interrupt routines for all possible interrupts for the 38076.

"P" holds the program code.

"C$DSEC" is the address area for initialised data in ROM.

"C$BSEC" is the address area for Non-Initialised data in ROM.

"D" is for initialised data in ROM.

"B" holds Non-Initialised Data in RAM

"R" is a reserved area for Initialised Data in RAM.

"S" is an area of Ram reserved for the stack

"BLCD_RAM" is a user entered section that covers the entire internal LCD RAM memory in the device.

Session Files and Configurations with this workspace

Session files determine what tool is used (i.e. E7, simulator, HMON etc.), and the files available for download to a target. They are therefore target and tool specific files.

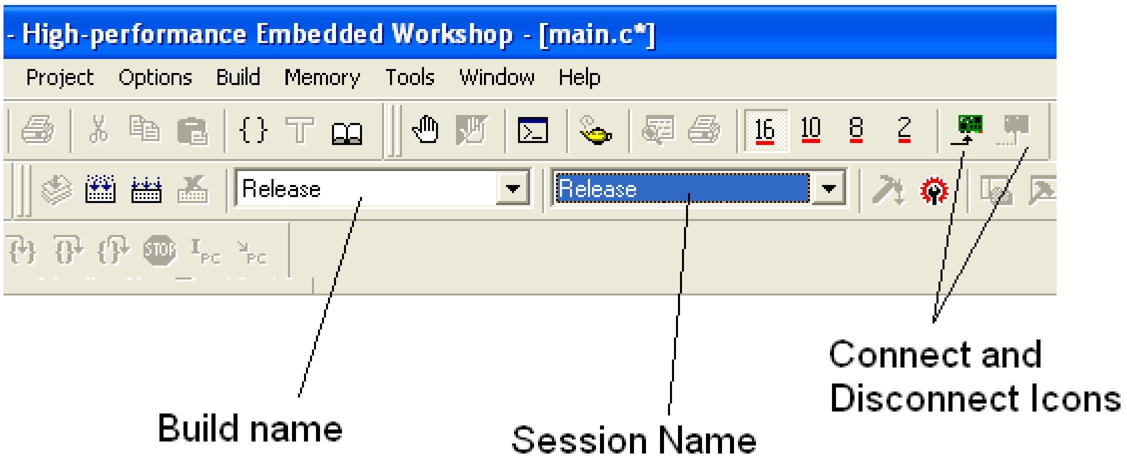
The build configuration determines what type of file is output by the toolchain, sections, linker/assembler/compiler settings etc.

In this workspace there are two build configurations and two session configurations. For debugging the code using the E7, the build should be set to Debug, and the session to E7.

The Debug build outputs debug information in the load file (this may be an elf / dwarf format file with the extension ".abs" for example). The debug session sets up the E7 as the tool, the 300H CPU as the target and the ".abs" as the download module.

The Release build has no debug information in the load file as it produces stand alone code for the target. When this code is programmed into the target, the target may be reset and the code runs. For programming the release version of the code into the target, set the both the build and the session to "release" (as in figure xx). This configures the tool as the E7, the target as a 300H CPU and the download file as the ".mot".

The debug session selects the ".abs" file to be downloaded to the target. The release build selects the ".mot" file.


- High-performance Embedded Workshop - [main.c*]
Project
Options
Build
Memory
Tools
Window
Help
Release
Release
Build name
Session Name
Connect and Disconnect Icons

17. Code Functions

Void ResetPRG(void)

When the H8/38076 comes out of reset comes out of reset the reset vector (address 0x000000) is read. Code execution begins at this address.

```
/***********************************************************************/
/*                                                                     */
/*  FILE        :resetprg.c                                            */
/*  DATE        :Mon, Mar 01, 2004                                     */
/*  DESCRIPTION :Reset Program                                         */
/*  CPU TYPE    :H8/Other                                              */
/*                                                                     */
/*  This file is generated by Renesas Project Generator (Ver.3.1).     */
/*                                                                     */
/***********************************************************************/

#include        <machine.h>
#include        <_h_c_lib.h>
#include        "stacksct.h"

extern void main(void);


#pragma section ResetPRG

__entry(vect=0) void PowerON_Reset(void)
{
         set_imask_ccr(1);
        _INITSCT();

        set_imask_ccr(0);

        main();

        sleep();
}
```

After the SP has been initialised, the function ResetPRG is executed. This disables the interrupt masking I bit in the CCR register temporarily while the section initialisation function is run.

The code then enters main.

void main(void)

```
/*****************************************************************
 main

 Purpose:   Sets up hardware, initialises ports and peripherals.
            Holds infinite loop for RTC update on LCD and power
            mode transition.

 Parameters Returned:   None

 Parameters Passed: None

*****************************************************************/
void main(void)
{
// Sets up direct transition interrupt and transitions from watch mode
    Hardware_Setup();
// Sets up LED port for output
    init_Port5();
// Sets up switch pins for interrupt inputs
    init_Port9();
// Configures LCD registers
    init_LCD();
// configures RTC registers and initialises / starts RTC
    init_RTC();
// Enter High Speed Mode
    Set_Power_Mode();
// Set upinfinite loop
    for(;;)
    {  // If RTC minute interrupt has gone off recently
        if(CHANGE == 1)
        {// Reset CHANGE to stop code going round in a loop
         CHANGE = 0;
         // Update the LCD with the new value
         UpDate_LCD();
          // Then change the power mode as it is safe to do so
         Set_Power_Mode();
        }
        else;  // User can enter own code here
    }
}
```

The first function to be called from main is Hardware_Setup(). This sets up some registers for system use. The second function configures port 5 as an output port (each of the pins of which is connected to an LED). The function which runs next sets up the switch inputs as interrupts. The init_LCD function configures the LCD registers for use with the LCD contained on the base board. The init_RTC function initialises and starts the RTC peripheral.

The main body of code is contained within an infinite loop. This detects whether or not an RTC interrupt has just occurred via the global variable CHANGE. If so, the LCD time value and the power mode are changed correspondingly using the functions UpDate_LCD() and Set_Power_Mode(). Whilst these updating functions are not running, the user is free to enter their own test code. It should be noted however, that in the sleep modes (high speed sleep, medium speed sleep and subsleep modes) the CPU is stopped and no code will run. (This is the reason the code shown here detects an RTC interrupt, and then updates the LCD *before* changing the power mode).

void Hardware_Setup(void)

This function performs some systems initialisations.

```
void Hardware_Setup(void)
{
    INT.IENR1.BIT.IENEC2 = 1;    // Enable IRQAEC interrupt
    INT.IENR2.BIT.IENDT = 1;     // Enable Direct Transition interrupts
    INT.IRR2.BIT.IRRDT = 0;      // Clear the interrupt flag
    SYSTEM.SYSCR1.BIT.STS2 = 1;  // setup standby timer select.
    SYSTEM.SYSCR1.BIT.STS1 = 1;
    SYSTEM.SYSCR1.BIT.STS0 = 1;
    SYSTEM.SYSCR2.BIT.SA1 = 1;   // subactive clock select.
    SYSTEM.SYSCR2.BIT.SA0 = 0;   // Slowest value chosen

    #ifdef Release
    SYSTEM.CKSTPR1.BIT.S4CKSTP = 0;
    SYSTEM.CKSTPR1.BIT.FROMCKSTP = 0;
    #endif

    #ifdef E7
    SYSTEM.CKSTPR1.BIT.S4CKSTP = 1;
    SYSTEM.CKSTPR1.BIT.FROMCKSTP = 1;
    #endif

    SYSTEM.CKSTPR1.BIT.S32CKSTP = 0;
    SYSTEM.CKSTPR1.BIT.ADCKSTP = 0;
    SYSTEM.CKSTPR1.BIT.TFCKSTP = 0;
    SYSTEM.CKSTPR2.BYTE = 0x00;
    SYSTEM.CKSTPR2.BIT.AECCKSTP = 1;  // enable Asynchronous Counter
    SYSTEM.CKSTPR2.BIT.LDCKSTP = 1;  // enable LCD

}
```

Firstly it enables the IRQAEC interrupt. This allows an IRQAEC interrupt to be generated when SW2 is pressed. It also enables a second interrupt; the direct transition interrupt. (The flag for this interrupt is cleared to prevent any erroneous direct transition interrupt handling).

Secondly it sets the standby timer select bits. The values in these bits determine the length of time the CPU and peripheral modules wait for a stable clock. To ensure these will not start before the clock is ready, the code has set this to the longest possible value. The timing for this application is not critical. Lastly the subactive clock value is selected. This is set to $\phi w/2$ to ensure that the LCD peripheral is functional in all of the used power modes. The code then switches off any modules that are not required. For release mode, modules used with the E7 may be switched off (such as the flash).

void Init_Port5(void)

```
void init_Port5(void)
{
      PORT.PCR5.BYTE = 0xFF;  // Configure all pins as outputs
      PORT.PMR5.BYTE = 0x00;  // Configure port as IO pins
}
```

This function sets up port 5 as an output port. This port is connected to 8 red LED's. Setting a 0 in a particular bit in the data register of this port lights the LED connected to the port pin associated with that bit.

void Init_Port9(void)

This function sets up pin 2 of port 9 as an interrupt pin (connected to Sw3). The 3rd bit of the port 9 mode register must therefore be configured as an IRQ4 input. The interrupt must then be enabled in the interrupt enable register. The interrupt edge register determines which edge of the interrupt is taken as the interrupt input.

```
void init_Port9(void)
{
 PORT.PMR9.BYTE = 0x04; // Enable IRQ4
 INT.IEGR.BIT.IEG4 = 0; // Set up interrupt edge select register
 INT.IENR1.BIT.IEN4 = 1; // Enable IRQ4 interrupt request
}
```

void Init_LCD(void)

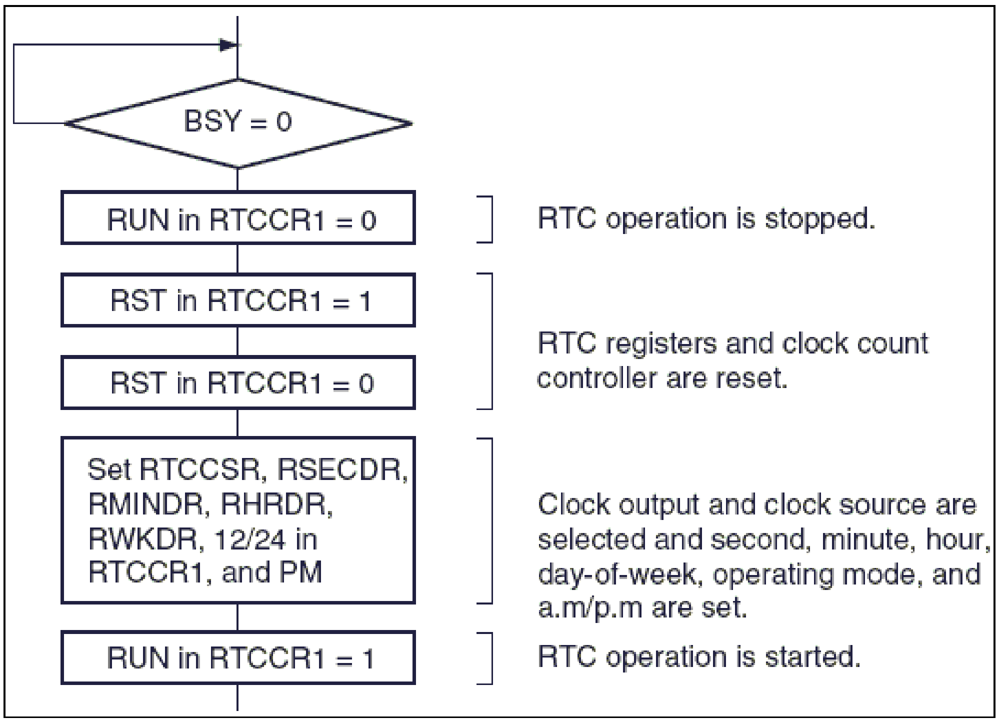
This function sets up the necessary LCD registers to drive the LCD on the baseboard. Firstly the clock halt register 1 is set to a value that ensures neither the LCD nor AEC modules are in standby mode. The LPCR register is then set for 1/3 duty and for the SEG lines SEG21 to 32 to be used to drive the LCD. The LCR register is set to turn on the LCD power supply, activate the display function and set the frame frequency to ϕw. The LCR2 register sets waveform B, the setup clock as ¼ of ϕw, connects the LCD power split resistor and turns of the 3V constant voltage supply. The LTRMR register sets the reference voltage adjustment for the LCD and the constant 3V power supply for the LCD. These are both set to 3V in the code below.

```
void init_LCD(void)
{

    SYSTEM.CKSTPR2.BYTE = 0x09;  /* Ensure LCD & AEC modules
                                    are not in standby */
    LCD.LPCR.BYTE    |= 0x8D; // Select duty cycle and segment driver
    LCD.LCR.BYTE     |= 0xE0; // LCD Power on, LCD RAM data not displayed
                              // LCD Driver operates, frame freq selected
    LCD.LCR2.BYTE    = 0xA0;  // waveform and power set up
    LCD.LTRMR.BYTE   |= 0x08; // Power trimming setup
    LCD.BGRMR.BYTE   |= 0xF8; // Band gap reference circuit setup
    LCD.LCR.BIT.DISP = 1;     // Display RAM data.

// Display 0000 on the LCD initially
 Display(1, 0);
 Display(2, 0);
 Display(3, 0);
 Display(4, 0);
}
```

void Init_RTC(void)

This function initialises the RTC peripheral. A flow diagram of the initialisation process is shown in figure 11.

Figure 11: RTC initialisation procedure


BSY = 0
RUN in RTCCR1 = 0
RTC operation is stopped.
RST in RTCCR1 = 1
RST in RTCCR1 = 0
RTC registers and clock count controller are reset.
Set RTCCSR, RSECDR, RMINDR, RHRDR, RWKDR, 12/24 in RTCCR1, and PM
Clock output and clock source are selected and second, minute, hour, day-of-week, operating mode, and a.m/p.m are set.
RUN in RTCCR1 = 1
RTC operation is started.

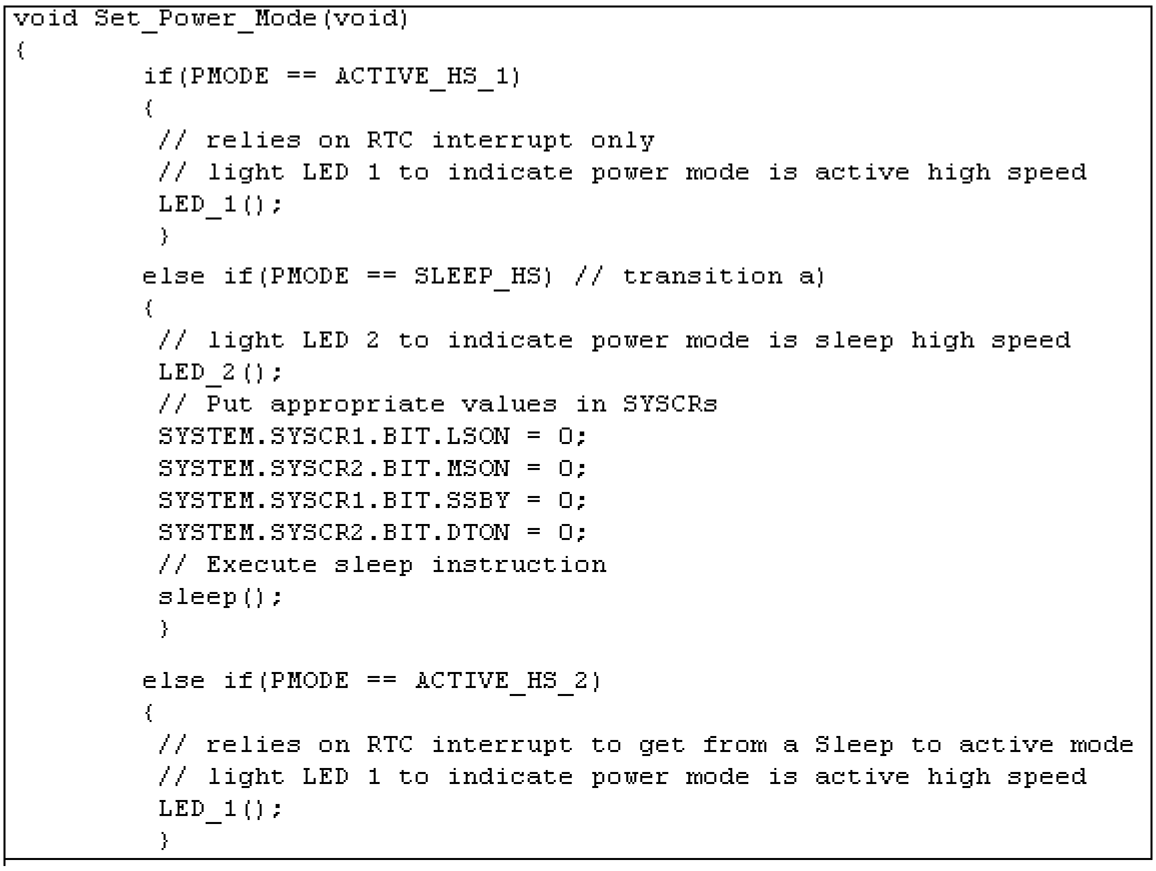
Firstly the BSY flag should be checked until it is 0. The RTC peripheral should then be stopped by putting the Run bit to 0. The RST bit should then be toggled to initiate an internal reset of the RTC registers and clock controller. In this program, the user provides the initial clock time. This value is then entered into the RTC registers. Finally the RUN bit is set to 1 again to start the peripheral. The code for implementing this initialisation is shown below:

```
void init_RTC (void)
{
// Create temporary variables to hold minute and hour values
     int temp_min_u = 0;
     int temp_min_t = 0;
     int temp_hr_u = 0;
     int temp_hr_t = 0;

    SYSTEM.CKSTPR1.BYTE = 0x8B;  /* Enable the AEC and LCD modules */

     /* set up RTC for a clock time base of 1 second */
     RTC.RTCCR1.BIT.RUN = 0;// STOP RTC
     RTC.RTCCR1.BIT.RST = 1;// RESET ON
     RTC.RTCCR1.BIT.RST = 0;// RESET OFF

// Collect user entered clock time
     temp_min_u = Get_Min_U_Value(); // get minute unit value
     temp_min_t = Get_Min_T_Value(); // get minute tens value
     temp_hr_u = Get_HR_U_Value();   // get hour units value
     temp_hr_t = Get_HR_T_Value(temp_hr_u); // get hour tens value

     RTC.RTCCR1.BIT.B12_24 = 1; // SET TO 24 HR MODE
     RTC.RTCCR2.BYTE |= 0x08; // ENABLE RTC INTERRUPT
     RTC.RTCCSR.BYTE |= 0x08; // SET TO REAL TIME CLOCK

     RTC.RSECDR.BIT.SCU = 0; // Set the second register value to 0
     RTC.RSECDR.BIT.SCT = 0; // Set the second regsiter value to 0

     RTC.RMINDR.BIT.MNU = temp_min_u; // Put user entered minute unit
     RTC.RMINDR.BIT.MNT = temp_min_t; // Put user entered minute tens
     RTC.RHRDR.BIT.HRU = temp_hr_u;   // Put user entered hour unit
     RTC.RHRDR.BIT.HRT = temp_hr_t;   // Put user entered hour tens

     RTC.RTCCR1.BIT.RUN = 1;// TURN RUN ON

     INT.IENR1.BYTE |= 0x80;  // int controller rtc iupt enable

     INT.IENR1.BIT.IENRTC = 1; // Enable the RTC interrupt
}
```

This function requires some temporary integer variables. These are used to store the results of the Get_X_Y_Value functions which obtain the user entered initial clock value. These functions are contained in the RTC_Time_Entry.c file. The system register CKSTPR1 is set so the RTC and AEC (required for the SW3 interrupt) modules are out of clock halt mode. Unused peripherals are halted so save power. As required by the RTC initialisation procedure (shown previously) the peripheral is halted via the Run bit and reset via the RTS bit. The user entered time value is then obtained, via the Get_X_Y_Value functions, and entered into the RTC time registers. The RUN bit is the put to 1 to start the RTC and the RTC interrupt is enabled.

Void Set_Power_Mode(void)

The code in main then enters an infinite loop which holds two functions, Set_Power_Mode and UpDate_LCD. Set_Power_Mode is contained within the file "PowerMode.c" and is responsible for looking at the value of the global variable "PMODE" (which numerically expresses what the current power mode should be / is) and putting the device into that power mode. It follows the power mode transitions shown in figure 7. The code for this function is shown below. The transitions are either driven by the RTC interrupt or initiated by the sleep instruction. For those which are interrupt driven, the power transition is automatic. All that is required of the Set_Power_Mode function is to light the appropriate LED. For those which are initiated by the sleep instruction, the bits are set in the SYSCR1 and SYSCR2 registers and the sleep instruction is executed.



```
void Set_Power_Mode(void)
{
        if(PMODE == ACTIVE_HS_1)
        {
         // relies on RTC interrupt only
         // light LED 1 to indicate power mode is active high speed
         LED_1();
         }
        else if(PMODE == SLEEP_HS) // transition a)
        {
         // light LED 2 to indicate power mode is sleep high speed
         LED_2();
         // Put appropriate values in SYSCRs
         SYSTEM.SYSCR1.BIT.LSON = 0;
         SYSTEM.SYSCR2.BIT.MSON = 0;
         SYSTEM.SYSCR1.BIT.SSBY = 0;
         SYSTEM.SYSCR2.BIT.DTON = 0;
         // Execute sleep instruction
         sleep();
         }

        else if(PMODE == ACTIVE_HS_2)
        {
         // relies on RTC interrupt to get from a Sleep to active mode
         // light LED 1 to indicate power mode is active high speed
         LED_1();
         }
```

```
else if(PMODE == ACTIVE_MS_1) // transition g)
{
 // light LED 3 to indicate power mode is active medium speed
 LED_3();
 // Put appropriate values in SYSCRs
 SYSTEM.SYSCR1.BIT.LSON = 0;
 SYSTEM.SYSCR2.BIT.MSON = 1;
 SYSTEM.SYSCR1.BIT.SSBY = 0;
 SYSTEM.SYSCR2.BIT.DTON = 1;
 // Execute sleep instruction
 sleep();
 }
else if(PMODE == SLEEP_MS) //transition b)
{
 // light LED 4 to indicate power mode is sleep medium speed
 LED_4();
 // Put appropriate values in SYSCRs
 SYSTEM.SYSCR1.BIT.LSON = 0;
 SYSTEM.SYSCR2.BIT.MSON = 1;
 SYSTEM.SYSCR1.BIT.SSBY = 0;
 SYSTEM.SYSCR2.BIT.DTON = 0;
 // Execute sleep instruction
 sleep();
 }
else if(PMODE == ACTIVE_MS_2)
{
 // relies on RTC interrupt only
 // light LED 3 to indicate power mode is active medium speed
 LED_3();
}
else if(PMODE == SUBACTIVE_1)
{
 // light LED 5 to indicate power mode is subactive
 LED_5();
 // Put appropriate values in SYSCRs
 SYSTEM.SYSCR1.BIT.LSON = 1;
 SYSTEM.SYSCR2.BIT.MSON = 1;
 SYSTEM.SYSCR1.BIT.SSBY = 1;
 SYSTEM.SYSCR1.BIT.TMA3 = 1;
 SYSTEM.SYSCR2.BIT.DTON = 1;
 // Execute sleep instruction
 sleep();
}
```

```
            else if(PMODE == SUBSLEEP)
            {
             // light LED 6 to indicate power mode is subsleep
             LED_6();
             // Put appropriate values in SYSCRs
             SYSTEM.SYSCR1.BIT.LSON = 1;
             SYSTEM.SYSCR2.BIT.MSON = 0;
             SYSTEM.SYSCR1.BIT.SSBY = 0;
             SYSTEM.SYSCR1.BIT.TMA3 = 1;
             SYSTEM.SYSCR2.BIT.DTON = 0;
             // Execute sleep instruction
             sleep();
            }

            else if(PMODE == SUBACTIVE_2)
            {
             // light LED 5 to indicate power mode is subactive
             LED_5();
            }

            else if(PMODE == WATCH)
            {
             // light LED 7 to indicate power mode is currently watch
             LED_7();
             // Put appropriate values in SYSCRs
             SYSTEM.SYSCR1.BIT.LSON = 0;
             SYSTEM.SYSCR2.BIT.MSON = 0;
             SYSTEM.SYSCR1.BIT.SSBY = 1;
             SYSTEM.SYSCR1.BIT.TMA3 = 1;
             SYSTEM.SYSCR2.BIT.DTON = 0;
             // Execute sleep instruction
             sleep();
            }
  }
```

Void UpDate_LCD(void)

This function is used to read the value of the RTC registers and display the register value on the LCD. The code for this function is contained in LCD.C.

```
 void UpDate_LCD(void)
 {
  // Temporary variables to store RTC register value
     unsigned int temp_u_1, temp_u_2, temp_t_1, temp_t_2;
  // Declare variable to determine successfull read
     unsigned char read_success = 0;

 // Read the minute value from the RTC min reg
  while(read_success != 1)
  {      // Wait until BSY flag is clear
         while (RTC.RMINDR.BIT.BSY == 1);
         // Check that the busy bit is 0
         if(RTC.RMINDR.BIT.BSY == 0)
         {   // Read the minute unit value once
              temp_u_1 = RTC.RMINDR.BIT.MNU;
             // Read the minute tens value once
              temp_t_1 = RTC.RMINDR.BIT.MNT;
         }
         // Check that the busy bit is 0
         if(RTC.RMINDR.BIT.BSY == 0)
         {    // Read minute units value for second time
              temp_u_2 = RTC.RMINDR.BIT.MNU;
              // Read minute tens value for second time
              temp_t_2 = RTC.RMINDR.BIT.MNT;
         }
         // Check the first and second reads are equal
         if((temp_u_1 == temp_u_2) && (temp_t_1 == temp_t_2))
         { read_success = 1; // If so - success!
           Display(1, temp_u_1); // Show values on LCD
           Display(2, temp_t_1);
         }
read_success = 0; // reset success variable
```

Firstly some variables are declared to hold temporarily, the minute units, tens value and the hours units, tens values. Another variable is declared to indicate that the RTC register read has been successful. For each register (the minute and the hour register), the code waits until the busy flag associated with that register is 0. When this flag is 0, the unit value and then the tens value is read once, and then again. The first and second values are then compared and if identical, are displayed on the LCD. The read_success variable is then reset for the next read. This code is an example of how to use each of the RTC register read methods stated in section 10.14.3, revision 3.00 of the 38076 manual.

```
// Read the hour value from the RTC min reg
  while(read_success != 1)
  {       // Wait until BSY flag is clear
          while(RTC.RHRDR.BIT.BSY == 1);
          // Check the flag is 0
          if (RTC.RHRDR.BIT.BSY == 0)
          {  // Read the hour unit value once
             temp_u_1 = RTC.RHRDR.BIT.HRU;
             // Read the hour tens value once
             temp_t_1 = RTC.RHRDR.BIT.HRT;
           }
          // Check the hour flag is 0
          if (RTC.RHRDR.BIT.BSY == 0)
          {  // Read the hour unit value for second time
             temp_u_2 = RTC.RHRDR.BIT.HRU;
             // Read the hour tens value for second time
             temp_t_2 = RTC.RHRDR.BIT.HRT;
           }
         // Check the first and second reads are equal
         if((temp_u_1 == temp_u_2) && (temp_t_1 == temp_t_2))
         { read_success = 1; // If so - success!
           Display(3, temp_u_1); // Show values on LCD
           Display(4, temp_t_1);
           }
  }
  read_success = 0; // reset variable
}
```

Void Display(x, y)

The LCD display used on the baseboard will appear as figure 11 when all segments are lit:

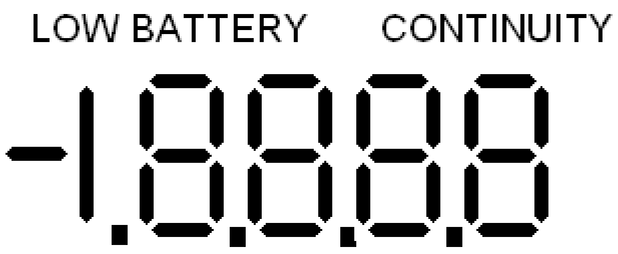


Figure 11: LCD Baseboard Display

The LCD has four characters, a MIN segment, a BC segment, LOW BATTERY text and CONTINUITY text. Each character has the structure and naming convention as shown in figure 12:

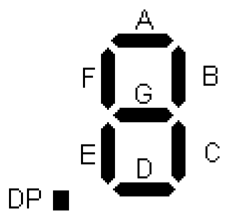
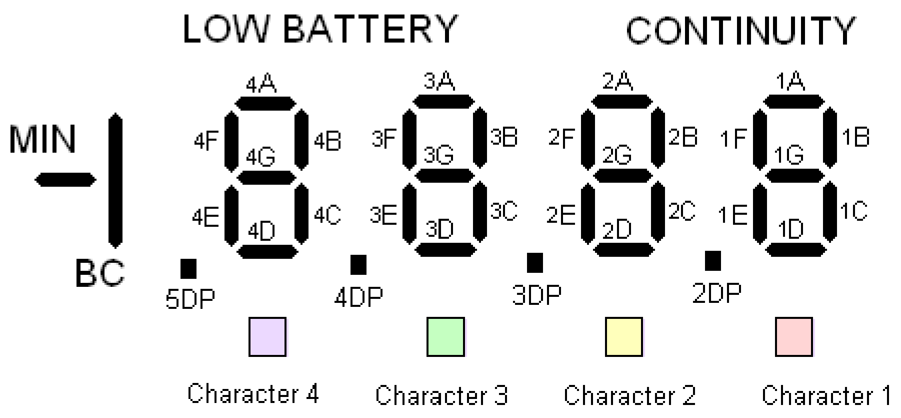


Figure 12: LCD character segment structure & naming convention

Any character can therefore produce any single digit number. Only three SEG lines are required to drive a single character. The values written through these SEG lines may be controlled via the LCD RAM. The relationship between the LCD RAM and the character segments is shown in figure 13.

LCD Segments



LCD Memory

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
H'F37A		SEG22 1D	SEG22 1G	SEG22 1A		SEG21 CONT	SEG21 1C	SEG21 1B
		SEG24 LOW	SEG24 2C	SEG24 2B		SEG23 2DP	SEG23 1E	SEG23 1F
		SEG25 3DP	SEG25 2E	SEG25 2F		SEG26 2D	SEG26 2G	SEG26 2A
		SEG27 MIN	SEG27 3C	SEG27 3B		SEG28 3D	SEG28 3G	SEG28 3A
		SEG29 4DP	SEG29 3E	SEG29 3F		SEG30 BC	SEG30 4B	SEG30 4C
H'F37F		SEG31 4D	SEG31 4G	SEG31 4A		SEG32 5DP	SEG32 4E	SEG32 4F
		COM3	COM2	COM1		COM3	COM2	COM1

Figure 13: LCD Memory and segment configuration

Note: The RAM address starts at F37A as we do not use any of the SEG lines before SEG21 in this application.

The function Display(x, y) is contained in lcd.c. Its function is to put the value y in the location on the LCD display given by x. A section of the code is shown below.

```
void Display(int character, int value)
{
// Structure is used to hold configurations of the SEG
// lines to output a specific value on the LCD
struct
    {
     unsigned char SEG_X_XC_XB:3;
     unsigned char SEG_XD_XG_XA:3;
     unsigned char SEG_X_E_F:3;
    } LCD_SEGS;

// The following is a list of SEGMENT configurations
// to obtain the value specified by "value"
    if(value == 0)  // If required "value" variable is 0,
    {
    /* Then we want to display:
          A
       *******          A, B, C, D, E & F are ON
       *     *
     F *     * B        G are OFF
       *     *
       *     *
     E *     * C
       *     *
       *******
          D
    */
      LCD_SEGS.SEG_X_XC_XB = 3;   // B and C are ON
      LCD_SEGS.SEG_XD_XG_XA = 5;  // A and D are ON (G is OFF)
      LCD_SEGS.SEG_X_E_F = 3;     // F and E are ON
     }

    else if(value == 1)
    {
    /* Then we want to display:

         A              B & F are ON
             *
     F       * B        A, C, D, E & G are OFF
         G   *
             *
     E       * C
         D   *

    */
      LCD_SEGS.SEG_X_XC_XB = 3;  // B and C are ON
      LCD_SEGS.SEG_XD_XG_XA = 0; // A, G and D are OFF
      LCD_SEGS.SEG_X_E_F = 0;    // E and F are OFF
     }
```

A temporary structure is declared to hold the required number to be displayed (given by the input parameter y). This only has to represent three SEG lines as this is all that is required to drive a single character on the LCD. Value x is used to determine which part of the LCD RAM exactly the number should be placed. The SEG lines used to generate each of the four LCD characters are X_XC_XB, (where X is continuity, BC, low battery or MIN), XD_XG_XA and X_XE_XF. In this application we will not use continuity, BC, low battery or MIN.

The section of code above shows the SEG definitions for 0 and 1. The code also contains similar definitions for 2 up to 9. Now that these have been completed, each of the characters on the LCD (1 to 4) may be attributed to one of these numerical definitions. As seen in the previous figure, SEG lines 21 to 23 hold character display information for character 1, SEG lines 24 to 26 hold information for character 2, SEGS 27 to 29 hold information for character 3, and finally SEGS 30 to 32 hold information for character 4. It is possible to reuse the temporary LCD_SEGS structure because we only write one character on the LCD every time Display_LCD(x, y) is called.

```
// Now we can put any numerical value between 0 and 9
// in our number storage structure we need to allocate
// this numerical structure to the LCD memory
    if(character == 1)
    {
     LCD_RAM.SEG_21 = LCD_SEGS.SEG_X_XC_XB;
     LCD_RAM.SEG_22 = LCD_SEGS.SEG_XD_XG_XA;
     LCD_RAM.SEG_23 = LCD_SEGS.SEG_X_E_F;
    }

    else if(character == 2)
    {
     LCD_RAM.SEG_24 = LCD_SEGS.SEG_X_XC_XB;
     LCD_RAM.SEG_25 = LCD_SEGS.SEG_XD_XG_XA;
     LCD_RAM.SEG_26 = (LCD_SEGS.SEG_X_E_F  | DOT);
     // This puts a dot to distinguish between mins and hours
    }

    else if(character == 3)
    {
     LCD_RAM.SEG_27 = LCD_SEGS.SEG_X_XC_XB;
     LCD_RAM.SEG_28 = LCD_SEGS.SEG_XD_XG_XA;
     LCD_RAM.SEG_29 = LCD_SEGS.SEG_X_E_F;
    }

    else if(character == 4)
    {
     LCD_RAM.SEG_30 = LCD_SEGS.SEG_X_XC_XB;
     LCD_RAM.SEG_31 = LCD_SEGS.SEG_XD_XG_XA;
     LCD_RAM.SEG_32 = LCD_SEGS.SEG_X_E_F;
    }

}
```

18. ISR's

All interrupt service routines for interrupts which are used in the code are held in the file "main.c".

RTC Minute Interrupt

The interrupt service routine for the minute RTC interrupt is shown below.

```
__interrupt (vect=22) void INT_RTC_MIN (void)
{
    // First clear the RTC minute flag
    RTC.RTCFLG.BIT.MNIFG    = 0;

    // Indicate to main that the RTC i/upt occurred
    CHANGE = 1;
    // If in watch mode, want to go back to active HS
    if(PMODE == WATCH)
     // so set power mode to high speed active
    PMODE = ACTIVE_HS_1;
    // if power mode is not watch, go to the next power mode
    else PMODE++;
}
```

The first part of the code clears the RTC minute flag. The code then sets the variable CHANGE to 1 so that the main function can detect that the RTC interrupt has recently gone off. The variable PMODE is then incremented to initiate a transition to the next power mode. The function Set_Power_Mode() discussed previously takes the variable PMODE to perform the transition and to light an LED accordingly. The definitions of PMODE are given in "powermodes.h".

IRQ4 (SW3)

IRQ4 is used to generate an interrupt when SW3 is pressed (SW3 is connected to the IRQ4 input pin). The interrupt service routine for IRQ4 is shown below:

```
// SW3 interrupt
__interrupt(vect=10) void INT_IRQ4(void)
{
  INT.IRR1.BIT.IRR4 = 0;
  SW3  = 1;
  SWITCH_PRESSED = 1;
}
```

Firstly the IRQ4 flag is cleared. The global variable SW3 is then set to 1 to indicate SW3 has been pressed. The global variable SWITCH_PRESSED is also set to 1 to indicate one of the two switches (SW2 and SW3) has been pressed.

IRQAEC (SW2)

IRQAEC is used to generate an interrupt when SW2 is pressed. SW2 is connected to the IRQAEC pin. The interrupt service routine for SW2 is shown below:

```
//SW2 interrupt
__interrupt(vect=8) void INT_IRQAEC(void)
{
  INT.IRR1.BIT.IRREC2 = 0;
  SW2 = 1;
  SWITCH_PRESSED = 1;
}
```

Firstly, as with all other interrupt service routines written here, the flag is cleared. The global variables SW2 and SWITCH_PRESSED are set to 1 to indicate that SW2 and one of the switches have been pressed respectively.

Current Consumption and Memory Resource Allocation

One of the aims of this application was to minimise the current consumption of the code/device, ensuring the current consumption was within the current consumption limits given in the manual.

19. Current Measurement Technique

Figure 14 shows the power circuitry for the 3DK38076. To measure the current consumed in each power mode, resistor R11 was removed and a current meter placed between terminals 1 and 2 of the jumper J11.

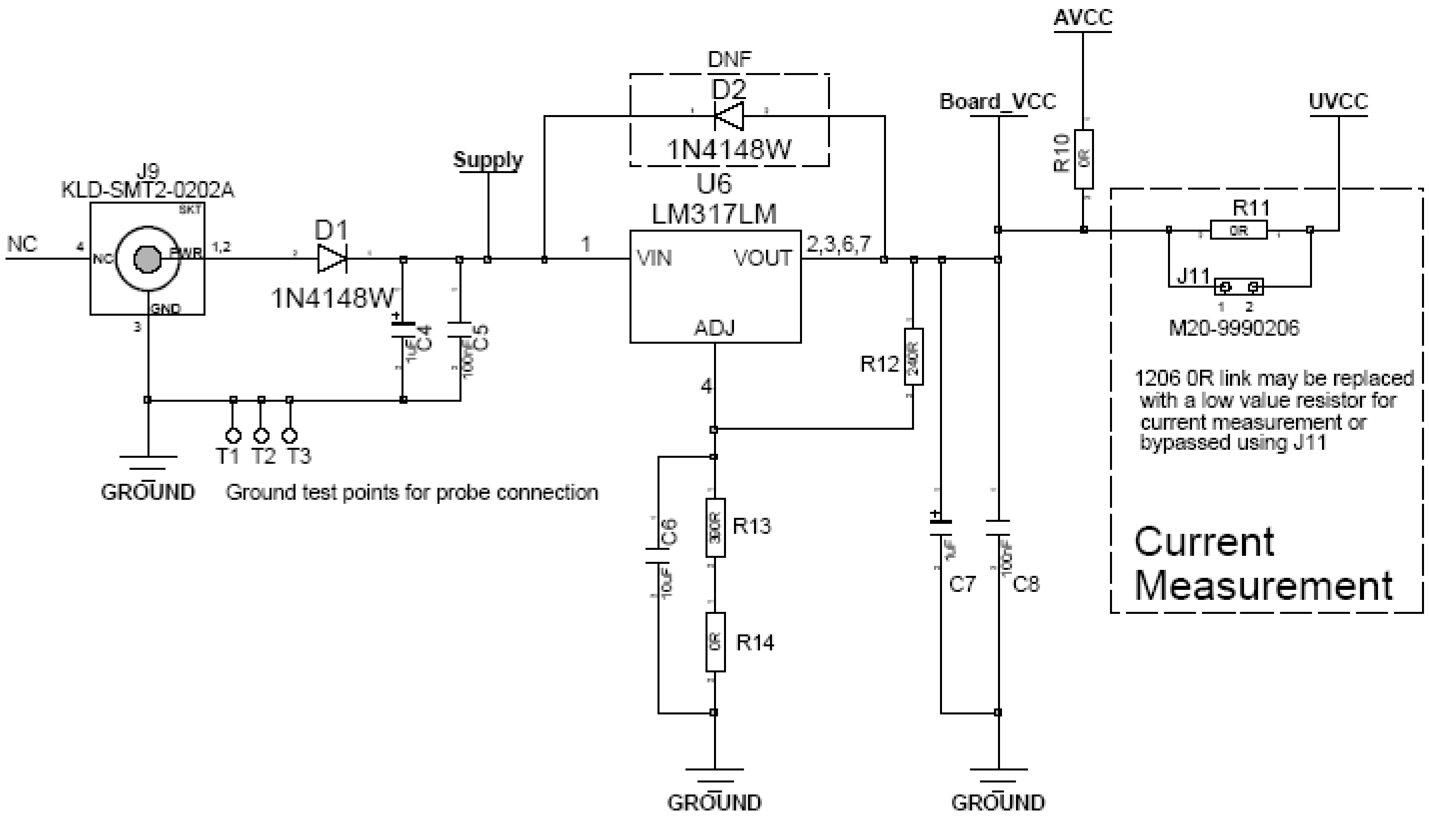


Figure 14: Power and current measurement circuit for 3DK38076

The current meter used was a Rapid Electronics 318DMM. The power supply used is a Thurlby PL320 which was set to 5V when running the code. The E7 is disconnected, and the device was programmed in release mode, under the release session.

20. Current Measurement Results

Table 4 below compares the expected from the user manual with the actual results obtained.

Power Mode	LED	Actual	Expected
Active High Speed	1	5.94mA	< 10mA
Sleep High Speed	2	3.89mA	< 5mA
Active Medium Speed	3	0.736mA	< 1.8 mA
Sleep Medium Speed	4	0.711mA	< 5mA
Subactive	5	32.5uA	< 50uA
Subsleep	6	15.3uA	< 16uA
Watch	7	13.8uA	> 6uA (as the LCD must be on)

As seen from the previous table, all current measurements are within the maximum limits given in the manual.

Please note that these values do not reflect the minimum power consumption values possible. This is due to the following reasons:

a) The 3DK board has unterminated IO pins.

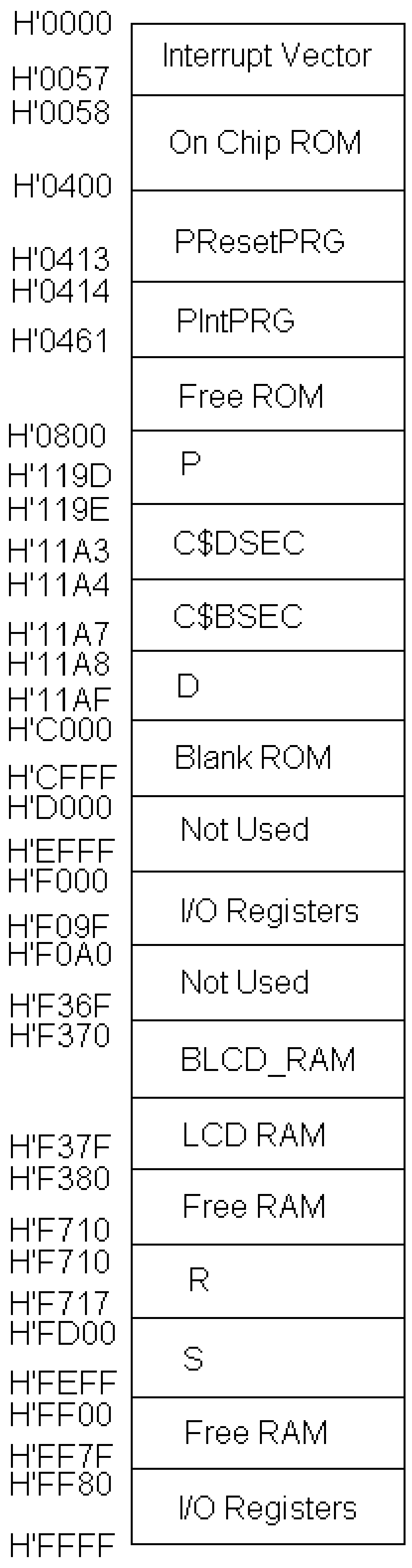
b) The RTC and LCD modules are enabled.

c) The subactive clock is set to the highest value

21. Memory Map

The memory map of the 38076 device with the demonstration code mappings included is shown in figure 15.

Figure 15: Memory map of 38076 with code sections included



The previous diagram shows that the code occupies 0xA12 bytes in ROM memory (not including the interrupt vector) and 20E bytes of RAM (the LCD Ram is not included). This leaves C596 bytes of ROM free and BFF – 20E = 9F1 bytes of free RAM.

Conclusion

The LCD peripheral on a H8/SLP device was successfully used to drive an LCD using the internal LCD RAM area. The RTC peripheral was also utilised to display the current time on the LCD. This time was updated every minute. This produced a clock which would operate within seven of the eight possible power down modes of the SLP device. The power consumption of the device was within the thresholds specified in the manual. The lowest power consumption recorded was 14.8uA.

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry