To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

H8/38602

I²C and SSU Code Example for LCD Display

Introduction

The purpose of this application note is to provide information on the utilisation of two communication peripherals of the H8/38602 device. These peripherals are the Serial Synchronous Unit and the I²C peripheral which will be used to communicate with an LCD. The LCD will display a clock value provided by the real time clock peripheral of the H8/38602 device.

The demonstration uses an MCU board with H8/38602 device (MB-H838602) and some bespoke interfacing circuitry. The board is powered using a 3V3 power supply. All code was designed using Renesas' High Performance Embedded Workshop configured to use the Renesas C/C++ compiler version 6.0.2 and debugged using an E7.

Contents

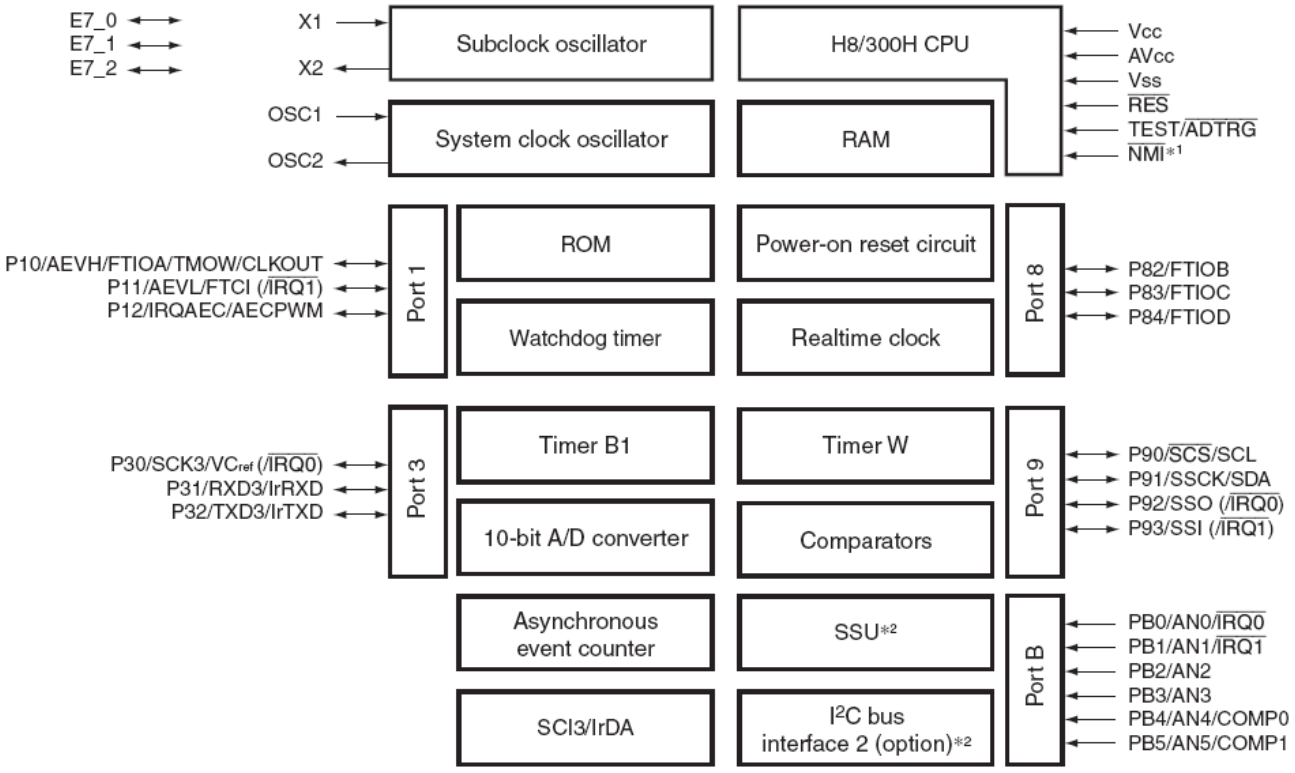
H8/38602 Overview and Features

High-speed H8/300H central processing unit with an internal 16-bit architecture

- Upward-compatible with H8/300 CPU on an object level
- Sixteen 16-bit general registers
- 62 basic instructions
- Various peripheral functions
- RTC (can be used as a free-running counter)
- Asynchronous event counter (AEC)
- Timer B1
- Timer W
- Watchdog timer
- SCI (asynchronous or clocked synchronous serial communication interface)
- SSU (synchronous serial communication unit)*
- I²C bus interface (conforms to the I²C bus format advocated by Philips Electronics)* (option)
- 10-bit A/D converter
- Comparators

Note: * SSU and I²C are shared.

Figure 1: H8/38602 Block Diagram



Notes: 1. The NMI pin is not available when the E7 or on-chip emulator debugger is used.
2. SSU and IIC2 are shared.

The I²C peripheral

- Selection of I²C format or clocked synchronous serial format
- Continuous transmission/reception
- Since the shift register, transmit data register, and receive data register are independent from each other, the continuous transmission/reception can be performed.
- Use of module standby mode enables this module to be placed in standby mode independently when not used.

A block diagram of the peripheral is shown in the following figure, figure 2.

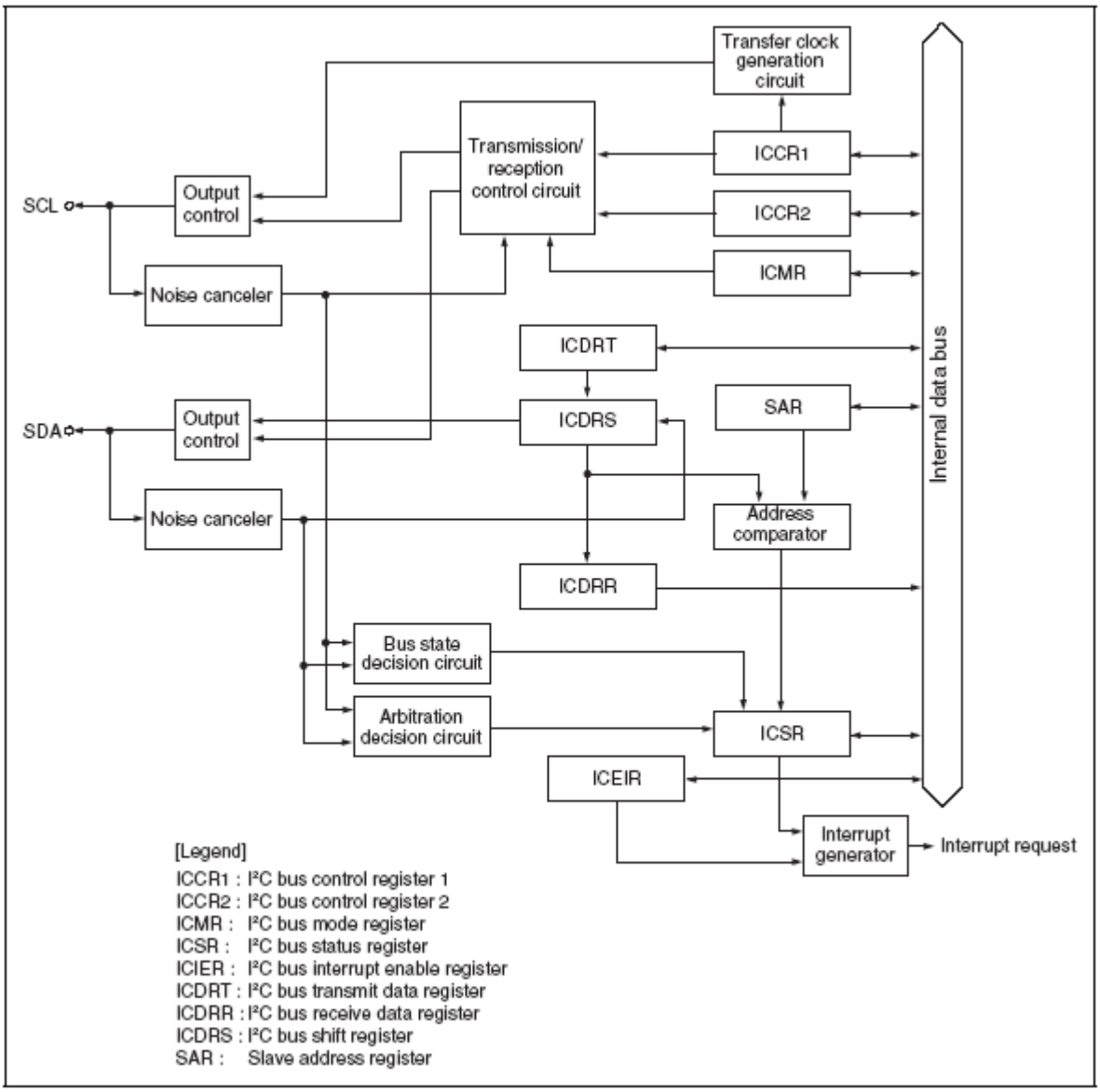


Figure 2: Block diagram of the I²C peripheral on H8/38602.

ICCR1: I²C Bus Control Register 1 H'F078

	7	6	5	4	3	2	1	0
	ICE	RCVD	MST	TRS	CKS3	CKS2	CKS1	CKS0
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	R/W

The two 8-bit I²C Control read/write registers configure the I²C behaviours. A description of each of the bits in register ICCR1 is given below:

Bit 7: **ICE – I²C Bus Interface Enable**

0: Halts the I²C module.

1: Enables the I²C module for transfer operations.

Bit 6: **RCVD – Reception Disable**

This bit enables or disables the next operation when TRS is 0 and ICDRR is read.

0: Enables next reception.

1: Disables next reception.

Bit 5: **MST – Master / Slave Select**

This bit is used in combination with the TRS bit below. When this bit is equal to 1 and the clocked synchronous serial format is selected the clock is output.

Bit 4: **TRS – Transmit / Receive Select**

The TRS and MST bits are reset when arbitration is lost in the following condition: where the I²C peripheral operates in the I²C format (as opposed to the clocked synchronous serial format) and the peripheral is set to master mode. When the TRS and MST bits are reset, a transition to slave receive mode occurs. If an overrun error occurs when the device is set to master and clock synchronous mode, MST is cleared to 0. Slave receive mode is then entered.
A change of value of the TRS bit should only be performed between transfer frames.
TRS is automatically set to 1 when the I²C peripheral is set to slave mode and the first seven bits of a slave address sent on the I²C interface matches that in SAR with the eighth bit equal to 1.
The combination of TRS and MST bits which provide specific I²C configurations are given below:

00: Slave Receive Mode

01: Slave Transmit Mode

10: Master Receive Mode

11: Master Transmit Mode

CKS[3:0]: Transfer clock select 3:0

These bits set the clock transfer rate when the I²C peripheral is set to master mode. The relationship between the transfer rate and the setting of these bits is shown in the following table:

CKS bits				Clock	Transfer Rate		
Bit3	Bit 2	Bit 1	Bit 0	Clock	ϕ = 2MHz	ϕ = 5MHz	ϕ = 10MHz
0	0	0	0	ϕ/28	71.4 kHz	179 kHz	357 kHz
0	0	0	1	ϕ/40	50.0 kHz	125 kHz	250 kHz
0	0	1	0	ϕ/48	41.7 kHz	104 kHz	208 kHz
0	0	1	1	ϕ/64	31.3 kHz	78.1 kHz	156 kHz
0	1	0	0	ϕ/80	25.0 kHz	62.5 kHz	125 kHz
0	1	0	1	ϕ/100	20.0 kHz	50.0 kHz	100 kHz
0	1	1	0	ϕ/112	17.9 kHz	44.6 kHz	89.3 kHz
0	1	1	1	ϕ/128	15.6 kHz	39.1 kHz	78.1 kHz
1	0	0	0	ϕ/56	35.7 kHz	89.3 kHz	179 kHz
1	0	0	1	ϕ/80	25.0 kHz	62.5 kHz	125 kHz
1	0	1	0	ϕ/96	20.8 kHz	52.1 kHz	104 kHz
1	0	1	1	ϕ/128	15.6 kHz	39.1 kHz	78.1 kHz
1	1	0	0	ϕ/160	12.5 kHz	31.3 kHz	62.5 kHz
1	1	0	1	ϕ/200	10.0 kHz	25.0 kHz	50.0 kHz
1	1	1	0	ϕ/224	8.9 kHz	22.3 kHz	44.6 kHz
1	1	1	1	ϕ/256	7.8 kHz	19.5 kHz	39.1 kHz

Table 1: CKS[3:0] settings with respect to clock rate for serial synchronous transfers.

ICCR2: I²C Bus Control Register 2 H'F079

	7	6	5	4	3	2	1	0
	BBSY	SCP	SDA0	SDAOP	SCLO	-	IICRST	-
Initial Value:	0	1	1	1	1	1	0	1
Read/Write:	R/W	W	R/W	R/W	R	-	R/W	-

Bit 7: **BBSY – I²C Bus Interface Enable**

This bit confirms whether or not the I²C bus is busy or released. This bit is also used to issue start or stop conditions on the I²C bus when the I²C peripheral is set to master mode. See the description given below for more information. This bit has no meaning when it is set to the clocked synchronous serial format. The BBSY bit is set to 1 when SDA changes from High to Low whilst SCL is high, as it is assumed this is a start condition. The BBSY bit is cleared to 0 when SDA makes a transition from low to high whilst SL is high as it is assumed this is a stop condition.

Bit 6: **SCP – Start/Stop issue Condition Disable**

This bit controls the issue of start / stop conditions when the I²C peripheral is set to master mode. This bit is always read as 1. If a 1 is written then this data is not stored.

Start/Stop Condition note: To issue a start condition a 1 should be written to this bit and a 0 to SCP. This procedure should also be used when retransmitting a start condition. To issue a stop condition a 0 should be written to both BBSY and SCP using a MOV instruction.

Bit 5: **SDA0 – SDA Output Value Control**

This bit is used to modify the output level of SDA. It should be used in conjunction with the SDAOP bit as this protects the SDA0 bit from alteration. This bit should not be manipulated during transfer.

0: When reading, SDA pin outputs low. When writing, SDA pin is changed to output low.

1: When reading, SDA pin outputs high. When writing, SDA pin is changed to output Hi-Z

(outputs high by external pull-up resistance).

Bit 4: **SDAOP – SDAO Write Protect**

This bit allows the SDAO bit value above to be altered. To change the output level of the SDA pin, clear SDAO and SDAOP to 0 using the MOV instruction. This bit is always read as 1.

Bit 3: **SCLO – SCL output Value Control**

This bit monitors the output level of SCL. When SCL outputs a high, SCL0 is 1, when SCL outputs a low then SCLO is. When SCL is low SCLO is 0.

Bit 2: **Reserved**

This bit is reserved and is always read as 1.

Bit 1: **I²CRST – I²C Control Part Reset**

This bit resets the control part of the I²C peripheral except for the I²C registers. When a hang up occurs during I²C operation, this bit may be set to 1 resetting the I²C control part without setting ports and initialising registers.

Bit 0: **Reserved**

This bit is always read as 1 and cannot be modified.

ICMR: I²C Bus Mode Register 2 H'F07A

	7	6	5	4	3	2	1	0
	MLS	WAIT	-	-	BCWP	BC2	BC1	BC0
Initial Value:	0	0	1	1	1	0	0	0
Read/Write:	R/W	R/W	-	-	R/W	R/W	R/W	R/W

Bit 7: **MLS – First/LSB-First Select**

0: MSB First

1: LSB First

This bit should be set to 0 when the I²C format is used.

Bit 6: **Wait – Wait Insertion Bit**

When the I²C peripheral is set to master mode with the I²C format selected, this bit selects whether to insert a wait after the data transfer (not including the acknowledge bit). When WAIT is set to 1, after the fall of the clock for the final data bit, the low period is extended for two transfer clocks. If WAIT is cleared to 0, the data and acknowledge bits are transferred consecutively with no WAIT inserted.

Bit 5, 4: **Reserved**

Bit 3: **BCWP – BC Write Protect**

This bit protects the BC settings shown below. To modify the BC bit values, clear this bit to 0 using the MOV instruction. In clock synchronous serial mode, BC should not be modified.

0: Values of BC2 to BC0 are set.

1: Changes to the values of BC2 to BC0 are invalid.

Bit [2:0]: **BC[2:0] Bit Counter 2 to 0**

These bits specify the number of bits that are to be transmitted next. They hold the remaining number of bits that are to be transferred. When the I²C format is selected, the ninth bit holds the acknowledge bit. Changes to the value of the BC bits should be made between frame transfers. If the value entered to the BC bits is not equal to zero, they should be entered while the SCL is low.

At the end of a transfer the value in these bits is 000 (including the acknowledge bit). These bits should not be modified when the peripheral is set to synchronous serial format.

BC2	**BC1**	**BC0**	**Bits to be transferred in I²C Bus Format**	**Bits to be transferred in Clock Synchronous Format**
0	0	0	9 bits	8 bits
0	0	1	2 bits	1 bits
0	1	0	3 bits	2 bits
0	1	1	4 bits	3 bits
1	0	0	5 bits	4 bits
1	0	1	6 bits	5 bits
1	1	0	7 bits	6 bits
1	1	1	8 bits	7 bits

I²C Bus Format and Timing

Figure 3 below shows the I²C format for data transfer. The first format shows continuous transfer. In this instance the slave address is written onto the bus followed by a Read/Write value. The slave device is then expected to issue an acknowledge. After the time it takes for the slave device to issue an acknowledge, up to 8 bits of data may be placed on the bus either by the slave or master device depending on the R/W value. Once all the data is transmitted, a stop bit is issued. The second format shows how data is retransmitted.

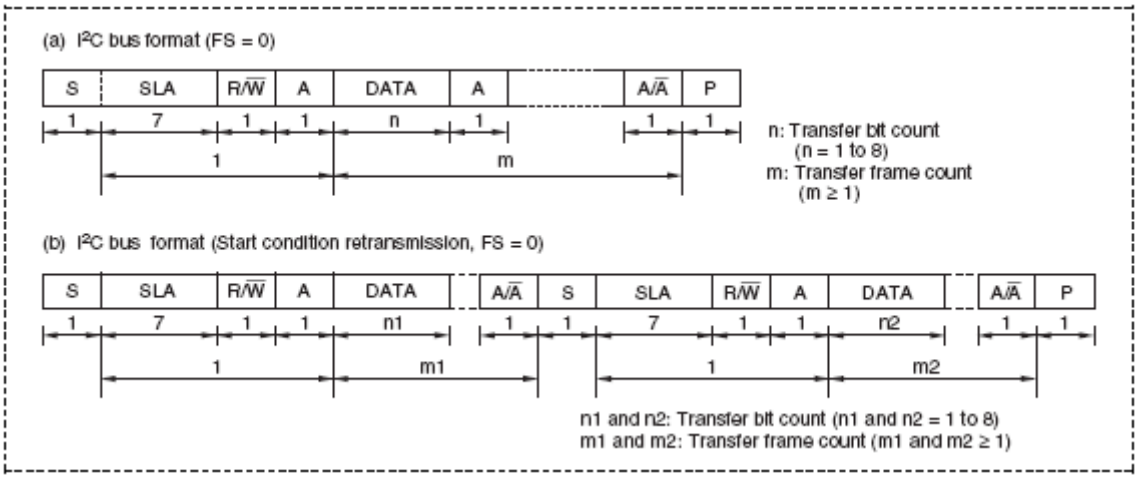


Figure 3: I²C Bus communication format

Figure 4 shows the timing of the I²C bus.

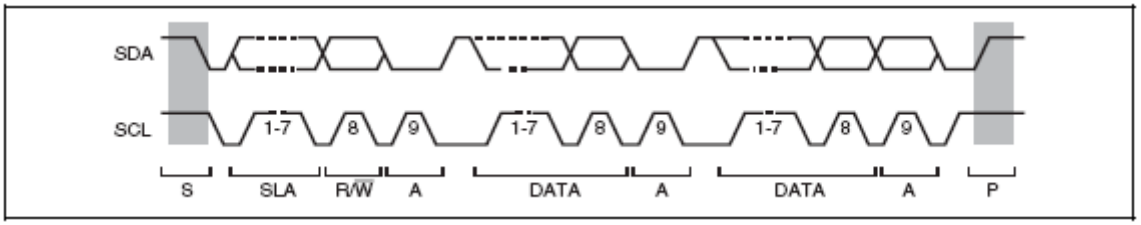
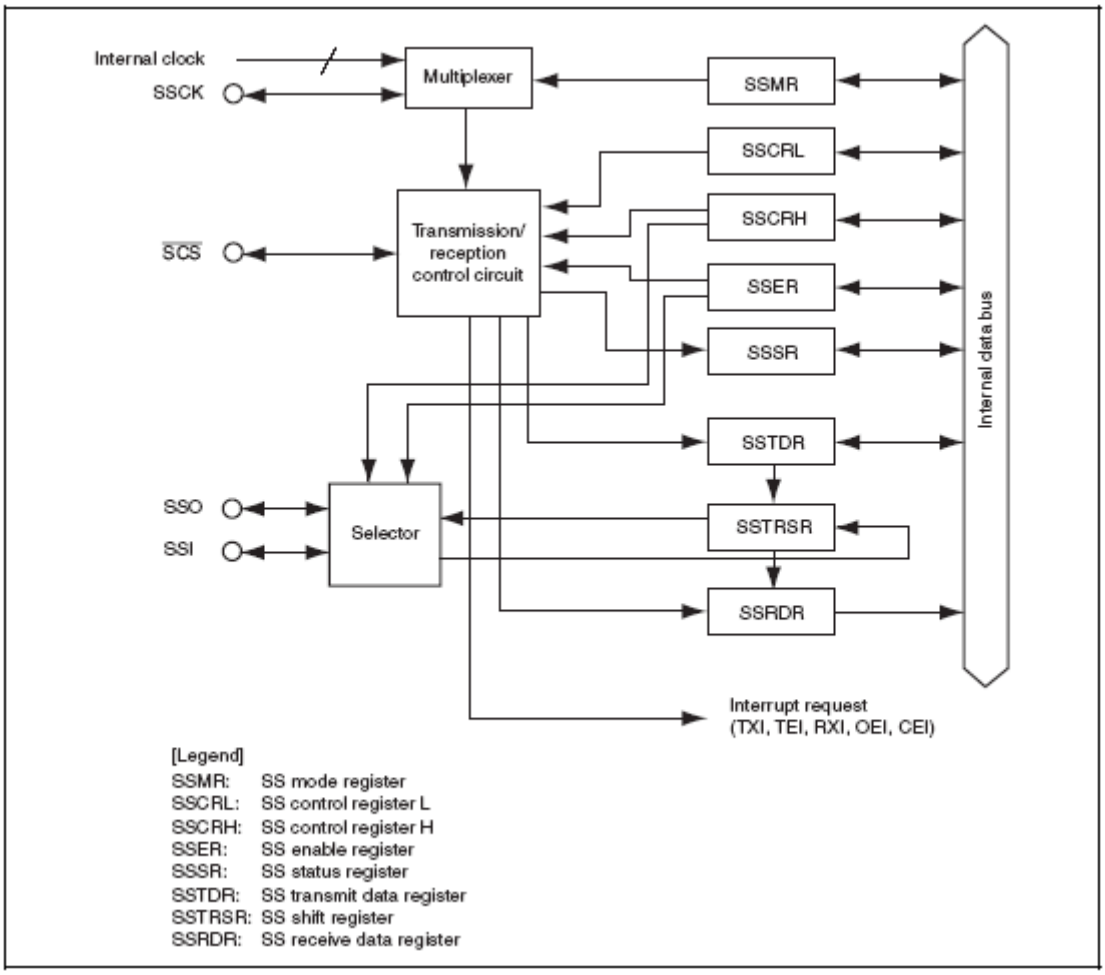


Figure 4: I²C Timing

S:	Start Condition
SLA:	Slave Address
R/Wn:	Direction of data transfer. For a read from the slave device, this will be 1. For a write to the slave address this will be 0.
A:	Acknowledge. The receiving device pulls SDA low during the 9th SCL clock pulse.
DATA:	Transfer Data
P:	Stop bit. This is issued by the master which pulls SDA high while SCL is also high.

The Synchronous Serial Unit Peripheral

- This is an SPI serial peripheral interface compatible peripheral which can be operated in clocked synchronous communication mode or four-line bus communication mode (including bi-directional communication mode)
- Can be operated as a master or a slave device
- Choice of eight internal clocks (ϕ/256, ϕ/128, ϕ/64, ϕ/32, ϕ/16, ϕ/8, ϕ/4, and ϕSUB/2) and an external clock as a clock source
- Clock polarity and phase of SSCK can be selected
- Choice of data transfer direction (MSB-first or LSB-first)
- Receive error detection: overrun error, Multi-master error detection: conflict error
- Five interrupt sources: transmit-end, transmit-data-empty, receive-data-full, overrun error, and conflict error
- Continuous transmission and reception of serial data are enabled since both transmitter and receiver have buffer structure
- Use of module standby mode enables this module to be placed in standby mode independently when not used.



SSCRH – SS Control Register H H'F0E0

	7	6	5	4	3	2	1	0
	MSS	BIDE	SOOS	SOL	SOLP	SCKS	CSS1	CSSO
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	R/W

Bit 7: **MSS – Master/Slave Device Select**

Selects whether this module is used as a master device or a slave device. When this module is used as a master device, transfer clock is output from the SSCK pin. When the CE bit in SSSR is set, this bit is automatically cleared.

0: Operates as a slave device

1: Operates as a master device

Bit 6: **BIDE – Bidirectional Mode Enable**

Selects whether the serial data input pin and the output pin are both used or only one pin is used. When the SSUMS bit in SSCRL is 0, this setting is invalid.

0: Normal mode. Communication is performed using two pins.

1: Bidirectional mode. Communication is performed using only one pin.

Bit 5: **SOOS - Serial Data Open-Drain Output Select**

Selects whether the serial data output pin is CMOS output or NMOS open-drain output. The serial data output pin is changed according to the register setting value.

0: CMOS output

1: NMOS open-drain output

Bit 4: **SOL - Serial Data Output Level Setting**

Although the value in the last bit of transmit data is retained in the serial data output after the end of transmission, the output level of serial data can be changed by manipulating this bit before or after transmission. When the output level is changed, the SOLP bit should be cleared to 0 and the MOV instruction should be used. If this bit is written during data transfer, erroneous operation may occur. Therefore this bit must not be manipulated during transmission.

0: Shows serial data output level to low in reading. Changes serial data output level to low in writing

1: Shows serial data output level to high in reading. Changes serial data output level to high in writing

Bit 3: **SOLP - SOL Write Protect**

When a change in the output level of the serial data lines is required, the MOV instruction should be used to clear this bit to 0 and then either set or clear the SOL bit.

0: In writing, output level can be changed according to the value of the SOL bit.

1: In reading, this bit is always read as 1. In writing, it cannot be modified output level.

Bit 2: **SCKS - SSCK Pin Select**

Selects whether the SSCK pin functions as a port or a serial clock pin.

0: Functions as a port

1: Functions as a serial clock pin

Bit [1:0]: **CSS[1:0] - SCSn Pin Select**

Selects whether the SCS pin functions as a port, an SCS input, or SCS output. When the SSUMS bit in SSCRL is 0, the SCS pin functions as a port regardless of the setting of this bit.

00: Functions as a port

01: Functions as an SCS input

1x: Functions as an SCS output (however, functions as an SCS input before starting transfer)

SSCRL – SS Control Register L H'F0E1

	7	6	5	4	3	2	1	0
	-	SSUMS	SRES	SCKOS	CSOS	-	-	-
Initial Value:	0	0	0	0	1	0	0	0
Read/Write:	-	R/W	R/W	R/W	R/W	-	-	-

Bit 7: **Reserved**

Bit 6: **SSUMS - SSU Mode Select**

Selects which combination of the serial data input pin and serial data output pin is used.

0: Clocked synchronous communication mode Data input:

SSI pin, Data output: SSO pin

1: Four-line bus communication mode

When MSS = 1 and BIDE = 0 in SSCRH:

Data input: SSI pin, Data output: SSO pin

When MSS = 1 and BIDE = 0 in SSCRH:

Data input: SSO pin, Data output: SSI pin

When BIDE = 1 in SSCRH:

Data input and output: SSO pin

Bit 5: **SRES – Software Reset**

When this bit is set to 1, the SSU internal sequencer is forcibly reset. Then this bit is automatically cleared. The register value in the SSU is retained.

Bit 4: **SCKOS - SSCK Pin Open-Drain Output Select**

Selects whether the SSCK pin functions as CMOS output or NMOS open-drain output.

0: CMOS output

1: NMOS open-drain output

Bit 3: **CSOS - SCS Pin Open-Drain Output Select**

Selects whether the SCS pin functions as CMOS output or NMOS open-drain output.

0: CMOS output

1: NMOS open-drain output

Bit[2:0]: **Reserved**

SSMR – SS Mode Register H'F0E2

	7	6	5	4	3	2	1	0
	MLS	CPOS	CPHS	-	-	-	-	-
Initial Value:	0	0	0	0	0	0	0	0
Read/Write:	R/W	R/W	R/W	-	-	-	-	-

Bit 7: **MLS - MSB-First/LSB-First Select**

Selects whether data transfer is performed in MSB-first or LSB-first.

0: LSB-first

1: MSB-first

Bit 6: **CPOS – Clock Polarity Select**

Selects the clock polarity of SSCK.

0: Idle state = high

1: Idle state = low

Bit 5: **CPHS - Clock Phase Select**

Selects the clock phase of SSCK.

0: Data change at first edge

1: Data latch at first edge

Bit [4:3]: **Reserved**

Bit [2:0]: **CKS[2:0] - Transfer Clock Rate Select**

Sets transfer clock rate (prescaler division ratio) when the internal clock is selected.
The system clock (φ) is halted in subactive mode or subsleep mode. Select φSUB/2 in these modes.

Bit 2	Bit 1	Bit 0	
0	0	0	ϕ/256
0	0	1	ϕ/128
0	1	0	ϕ/64
0	1	1	ϕ/32
1	0	0	ϕ/16
1	0	1	ϕ/8
1	1	0	ϕ/4
1	1	1	ϕSUB/2

Please note: There are other registers associated with the SSU peripheral. For details please see the Synchronous Serial Communication Unit (SSU) section in the H8/38602 hardware manual

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 32.768 KHz subclock.
- The RTC module may be placed in module standby mode

A block diagram of the Real time clock is shown below in figure 5

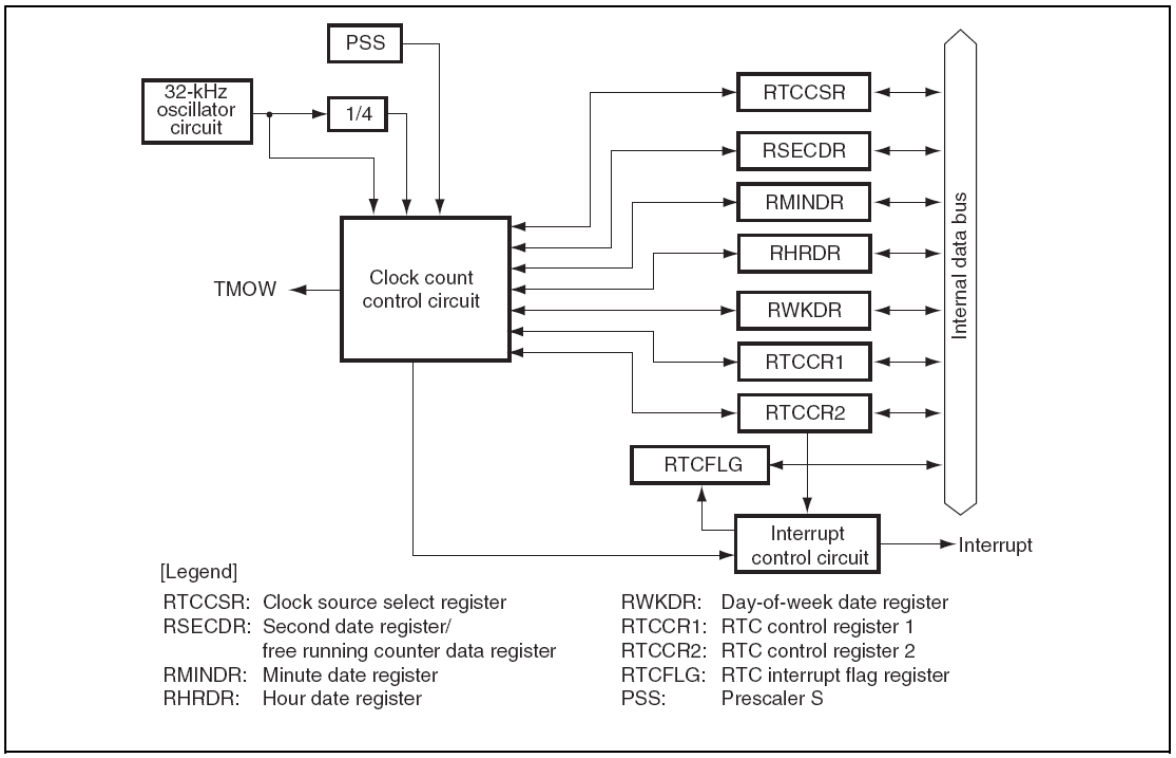


Figure 5: RTC Block Diagram

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.

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.

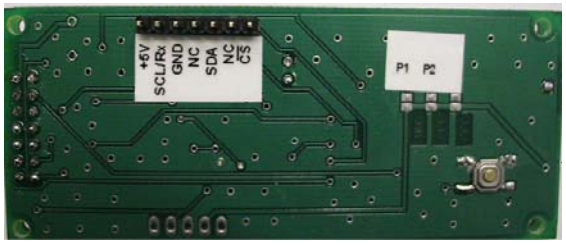
1: Enables a 0.25 second periodic interrupt. (This will occur once quarter of a second).

The LCD Module

The LCD used for this application was a LCD216S made by Mindsensors, a picture of the LCD is shown in figure 6. It has a 16 column by 2-line text display and a backlight which can be turned ON/OFF via software. It is possible to communicate with the LCD module via RS232, serial peripheral interface (SPI) or I²C.

Figure 6: LCD Pin Connection

The pins are as follows



Pin 1: +5V supply voltage for Module

Pin 2: Serial clock for I²C and SPI and Rx for RS232

Pin 3: GND connection for Module and reference for data communication

Pin 4: No connect

Pin 5: Serial Data in for I²C and SPI

Pin 6: No connect

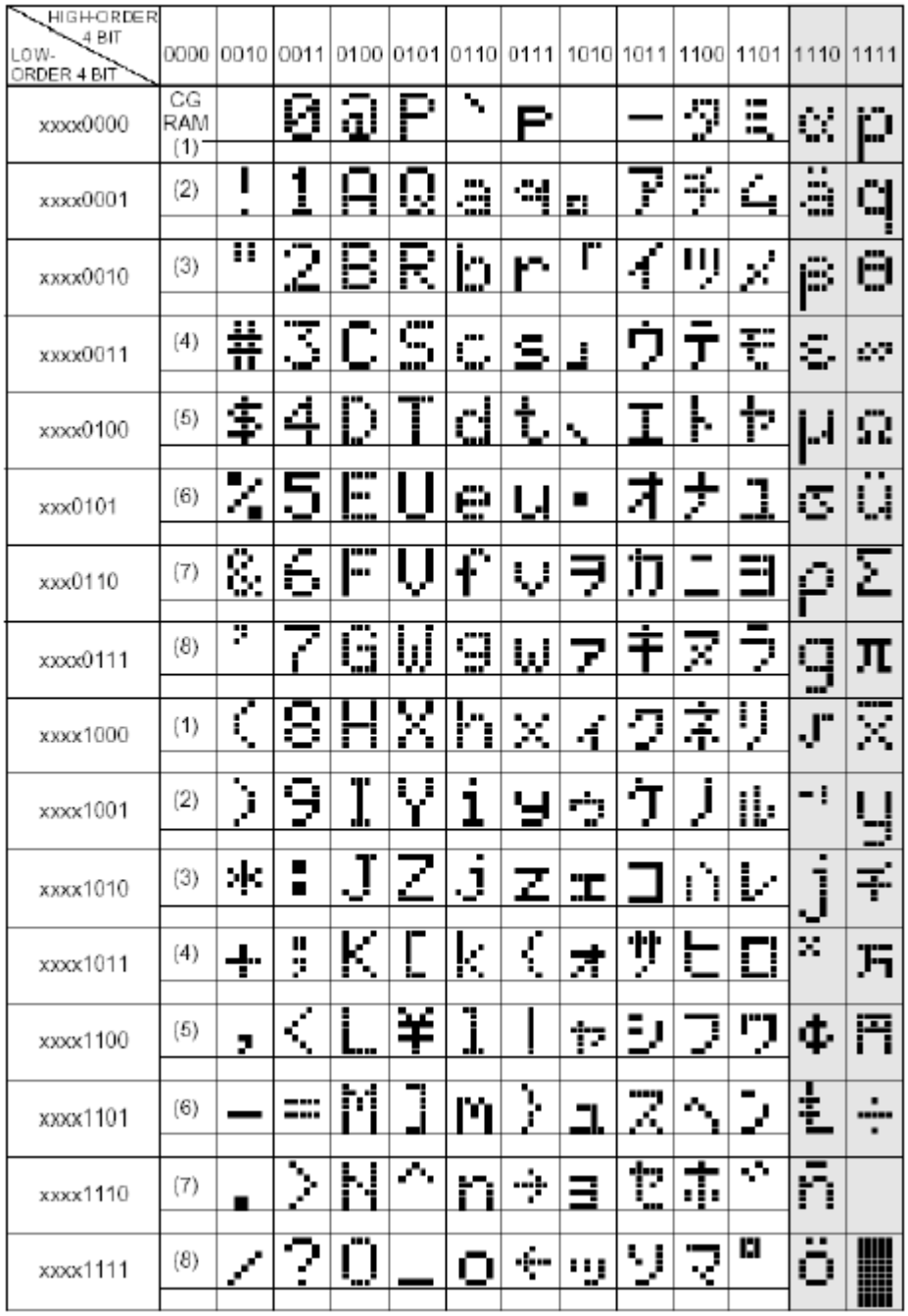
Pin 7: CS\ for SPI and RS232

When the LCD is first powered from a 5V power supply, it is configured for I²C communications with the slave address 0xE0. A setup switch is provided on the LCD module for selection and configuration of communication interface. To enter the setup interface on the LCD module, the setup switch should be pressed whilst the power is cycled until “Setup” appears on the display. When the switch is released, the LCD will display the current communication interface settings. The desired interface may then be selected by pressing the set up switch repeatedly if required. When the correct communication settings are displayed, the module should be powered down. The module will then power up in the new communication settings.

Once the LCD is appropriately set up for communication, characters may be displayed on the LCD by sending the characters' ASCII value to the LCD. Table 2 below shows the relationship of the value sent to the LCD and the character displayed on the LCD.

The LCD also has a number of commands which may be used to move the cursor, show the cursor, clear the screen, turn the backlight On/OFF, etc. All the commands start 0xFE (this is blank in the character table) which is then immediately followed by the command number.

Table 2: Relationship between the data value sent to LCD and the character displayed.



Hardware Setup Using I²C Communication

The H8/38602 device is operable from the 3V3 supply; however the LCD requires 5V CMOS levels at its I²C pins. Some hardware interface is therefore required to perform the voltage translation between the LCD and H8/38602 device. The following figure 7 shows this hardware interface.

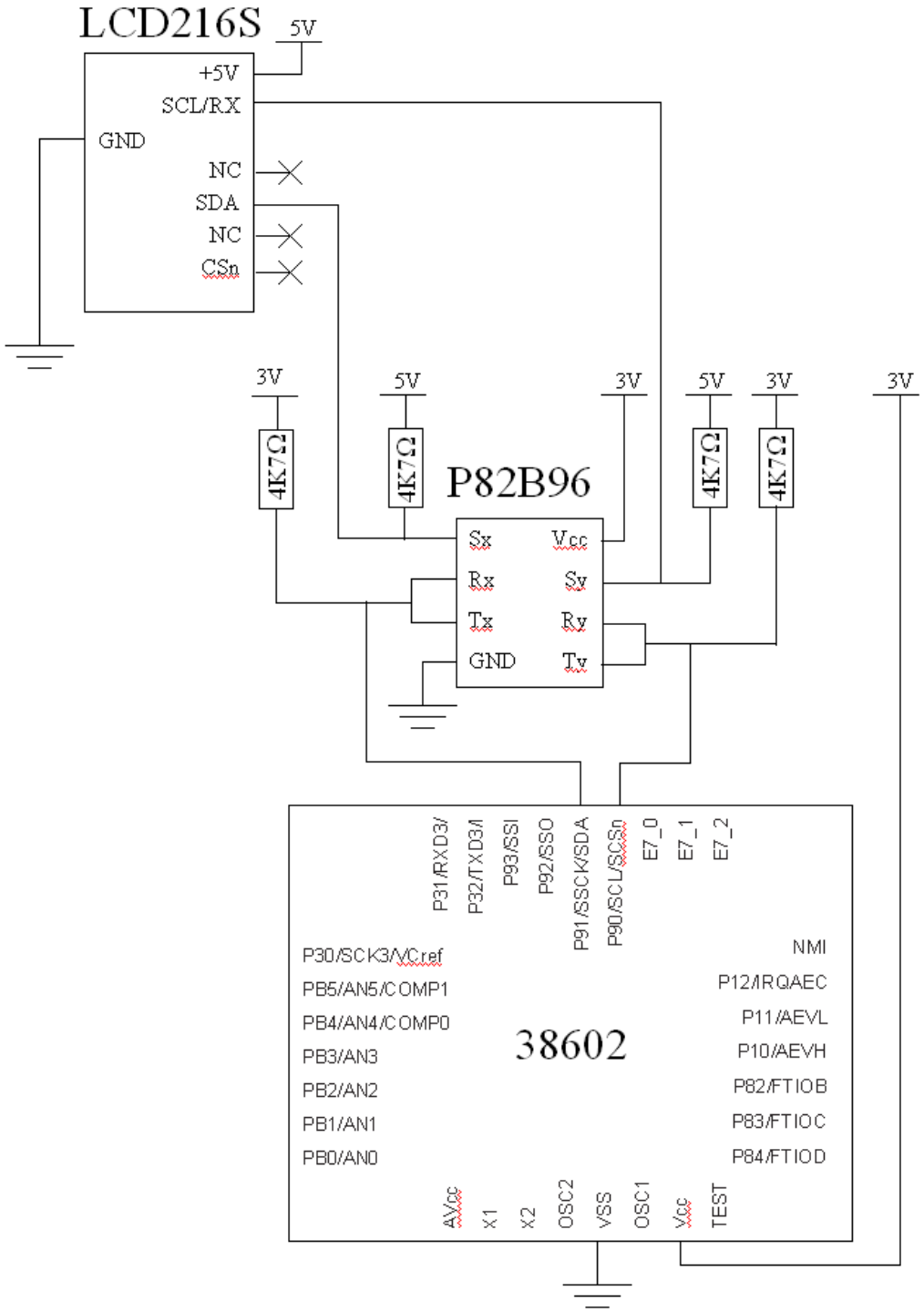
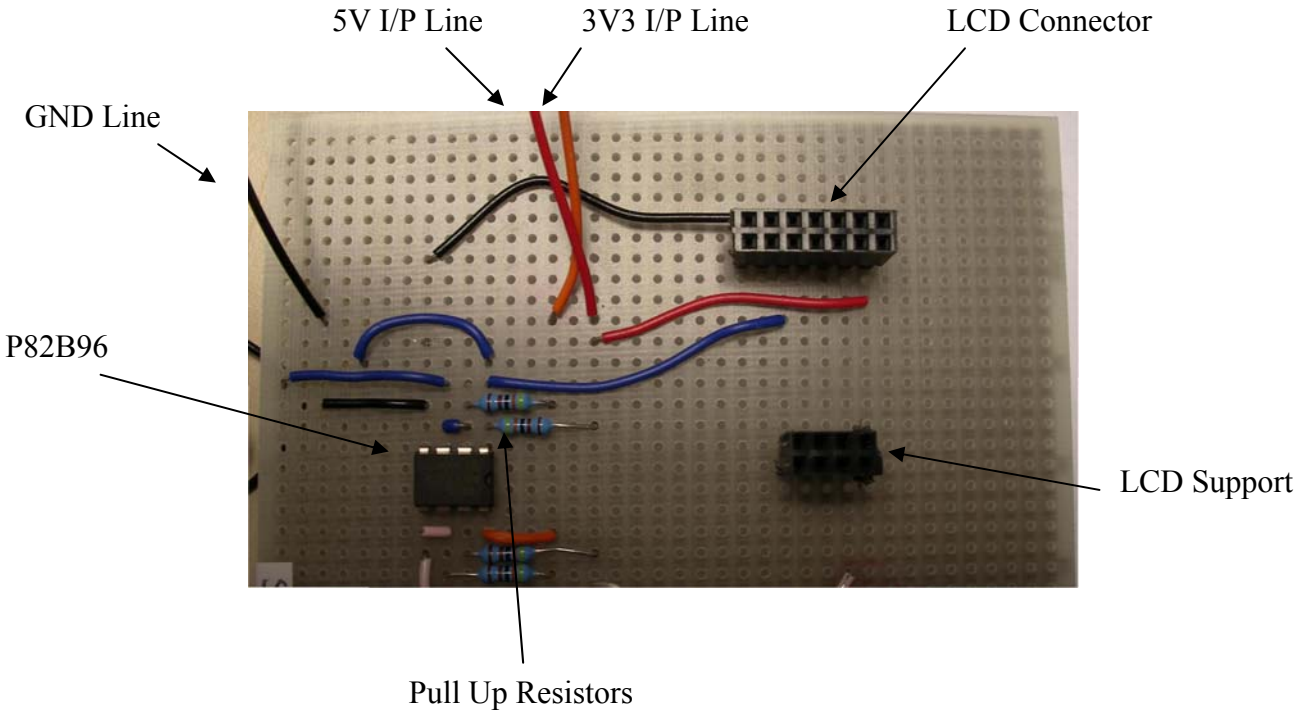


Figure 7: Hardware interface for I²C Communication between LCD and H8/38602 device.

The device used for performing the level shifting is a P82B96. This is designed for I²C communications up to 400 KHz. For this purpose it is possible to connect the Rx and Tx lines together and connect this to one of the I²C communication lines. Other I²C applications may require that they be kept separate. The P82B96 requires that one side of the I²C lines operates from 5V. The other side may operate from any voltage between 2 and 15V. For this application we require the LCD I²C communications to be at a 5V level and the H8/38602 I²C communications to be at a 3V3 level. Therefore the VCC level of the P82B96 is set to 3V3 volts.

The SDA and SCL lines on the microcontroller side are pulled to the 3V3 power supply via 4K7 resistors. This is a requirement of I²C communications, as the devices pulls the line low when required. Similarly, therefore, the SDA and SCL lines on the LCD side also have 4K7 resistors which pull these lines to 5V. Figure 8 is a picture of the board used as the I²C interface

Figure 8: I²C Interface Board



Hardware Setup Using SSU Communication

The H8/38602 is a 3V3 device, but the LCD requires 5V at its SPI pins. The hardware configuration used to perform the voltage translation between the LCD and the H8/38602 is shown in figure 9 below:

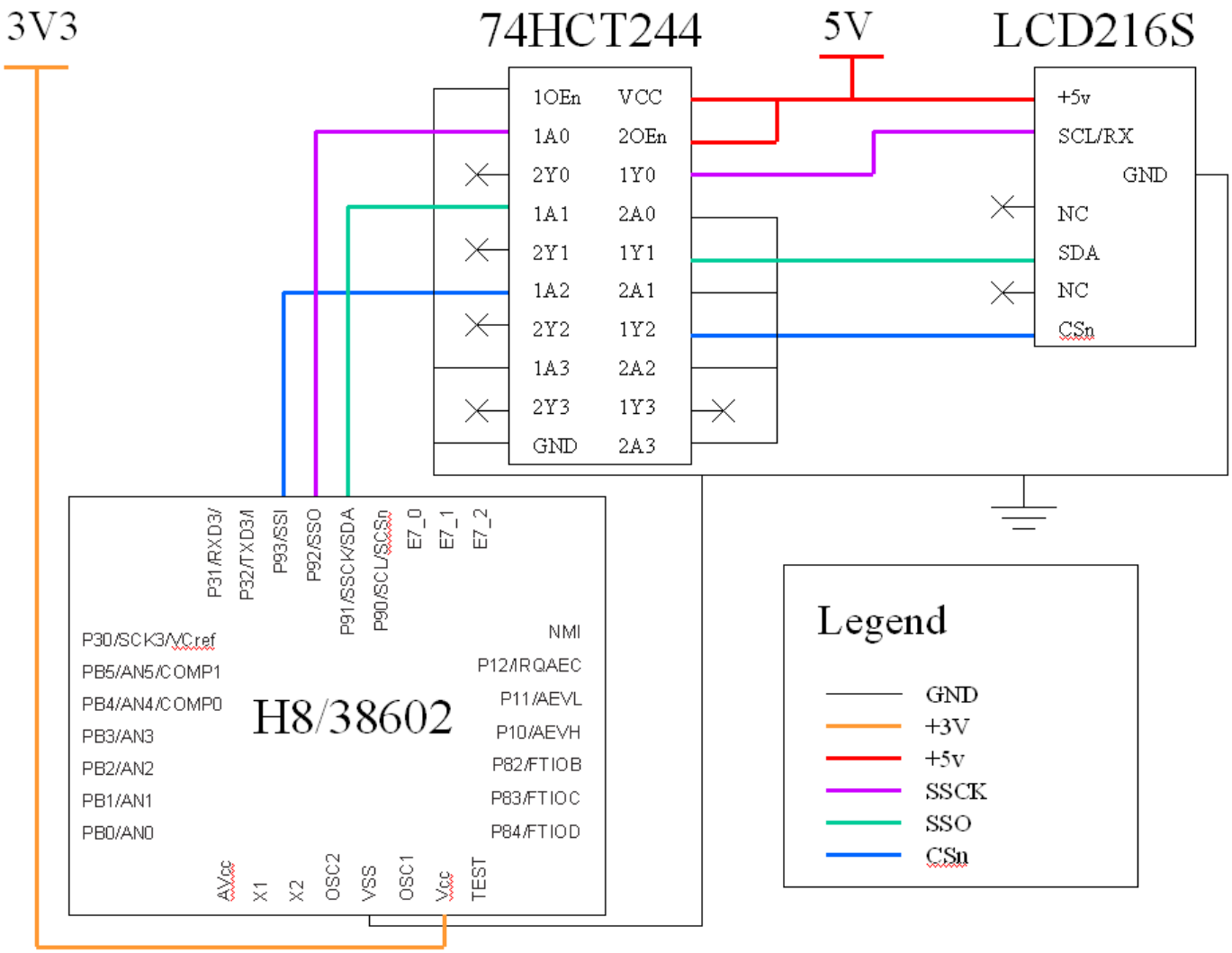


Figure 9: Hardware interface between LCD and H8/38602 for SSU communications

In this case a 74HCT244 device was used. This has two sets of four inputs and four outputs. Pulling the output enable line low enables the output. For this application only three lines are required therefore the second set of outputs are not enabled (2OEn is pulled high). All unused inputs are pulled low. Each input has been given the notation SAN where S is the set number and N is the number in that set. The A shows it is an input. The outputs are denoted SYN where S is the set and N is the number in that set. The Y shows this pin is an output. The output attributed to a particular input can be seen where the S and N numbers are equal. For example the input 1A0 has the associated output 1Y0.

In this application the serial communication is not bidirectional. This is not the case in all SPI applications. Here only three lines are required: SSCK for the serial clock, SDO as the output from the H8/38602 to the LCD providing the data line for communication and SCSn which notifies the LCD when the master wants to communicate with it.

Figure 10 shows a picture of the interface board used

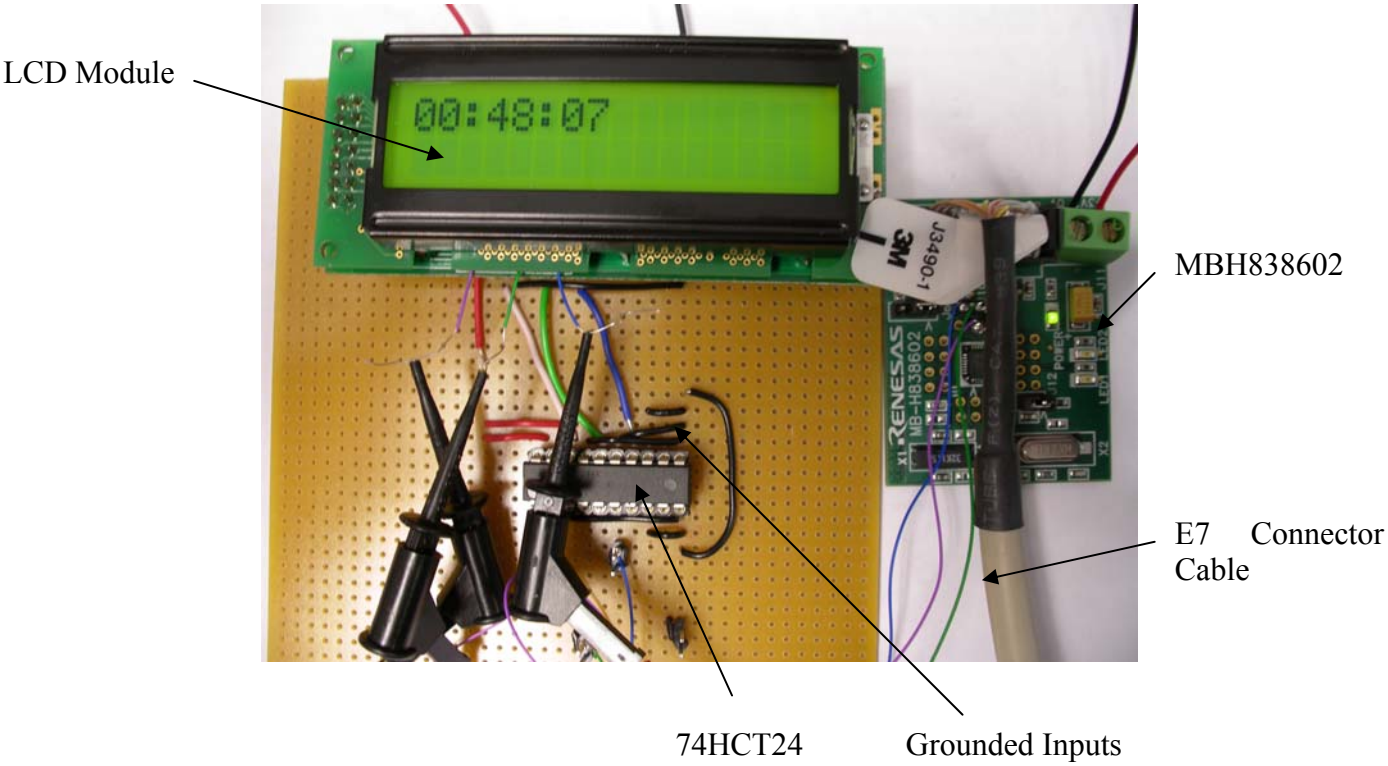


Figure 10: SSU interface board with MBH838602 and LCD devices.

Software for communication with LCD via I²C

Figure 11 shows the program flow diagram for I²C communication between the H8/38602 and the LCD.

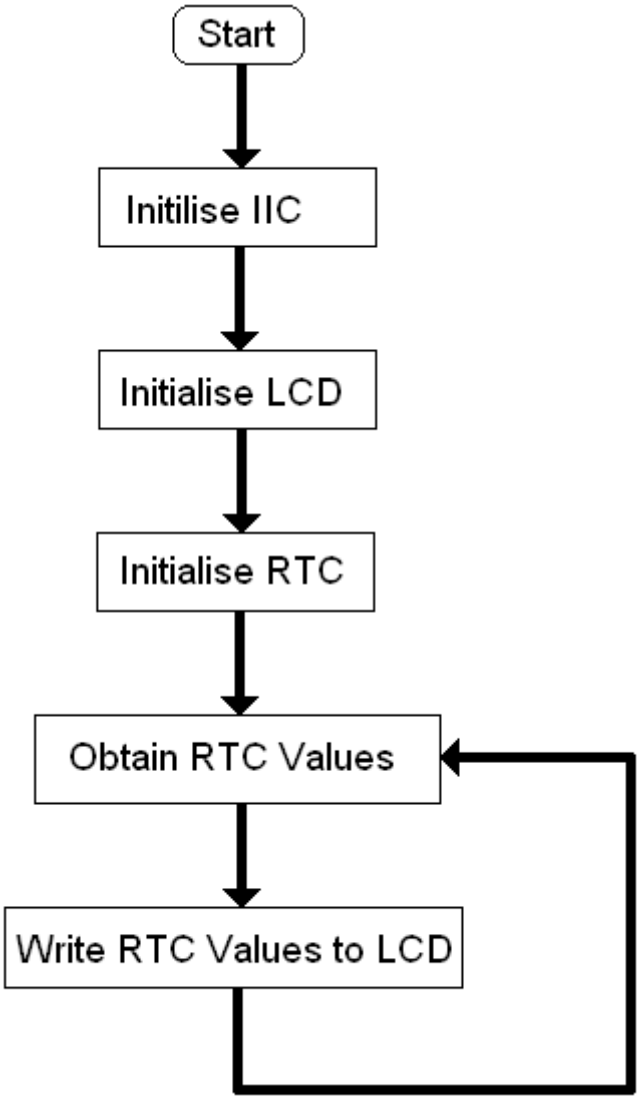


Figure 11: Initialisation Procedure for I²C interface

Firstly the I²C is initialised in order that this interface can be used to clear the LCD and transmit instruction / characters to the LCD. The LCD is then initialised so that it may be cleared ready for a time value to be written to it. Lastly, the RTC is initialised. This starts the RTC counting. It is now possible to grab the RTC values and write them to the LCD. An infinite loop is set up which takes the RTC register values and writes them to the LCD display. The following code shows the function Main.

```
void main(void)
{
 // Set up temporary variables to hold the current value of seconds
 // mins and hours.
    int temp_secs_units = 0x00;   // Initialise all of these to 0.
    int temp_secs_tens = 0x00;
    int temp_mins_units = 0x00;
    int temp_mins_tens = 0x00;
    int temp_hours_units = 0x00;
    int temp_hours_tens = 0x00;

    // Initialise the IIC peripheral of the 38602 device.
    Init_IIC();
    // Initialise the LCD so that a time may be written to it.
    Init_LCD();
    // Initialise the Real time clock module on the 38602 device.
    Init_RTC();

  for(;;)         // Initiate an infinite loop so that
                  // the time can be continously written to the LCD
   {
     Write_bytes(0xFE, 0x48);  // Move LCD cursor home
     Delay();  // allow time for instruction to complete

     // Wait until hour register is ready to be read
     while(RTC.RHRDR.BIT.BSY == 1);
        // store hour unit value in temporary variable
        temp_hours_units = RTC.RHRDR.BIT.HR0;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current hour units value
        temp_hours_units += 0x30;
        // store hour tens value in temporary variable
        temp_hours_tens = RTC.RHRDR.BIT.HR1;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current hour tens value
        temp_hours_tens += 0x30;

     // Wait until minute register is ready to be read
     while(RTC.RMINDR.BIT.BSY == 1);
        temp_mins_units = RTC.RMINDR.BIT.MN0;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current minute units value
        temp_mins_units += 0x30;
        // store minute tens value in temporary variable
        temp_mins_tens = RTC.RMINDR.BIT.MN1;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current minute tens value
        temp_mins_tens += 0x30;
```

```
    // Now it is possible to write ASCII time values to LCD
    // Write hours tens and units values to LCD as
    //  these should appear first
    Write_bytes(temp_hours_tens, temp_hours_units);
    Delay(); // Wait until LCD is ready to receive next characters

    // Write a ":" character to separate hours from mins
    // Write minute tens value to LCD
    Write_bytes(0x3A, temp_mins_tens);
    Delay(); // Wait until LCD is ready to receive next characters

    // Write minute units value to LCD
    // Write a ":" to separate minutes from seconds
    Write_bytes(temp_mins_units, 0x3A);
    Delay(); // wait until LCD is ready to receive next characters

    // Write second tens and units values to LCD
    Write_bytes(temp_secs_tens, temp_secs_units);
    Delay(); // wait until LCD is ready to receive next characters
   }
}
```

Firstly the function declares a set of temporary variables which are used to hold the current time value from each of the RTC registers. The I²C peripheral is then initialised, followed by the LCD device and finally the RTC module.

The code then enters an infinite loop as shown in figure 11. Here the code resets the LCD cursor back to the home position, ready to write a new time value over the old. The RTC registers are then read when appropriate, and their values stored in the temporary variables declared at the beginning of the function. These values may then be transmitted to the LCD in an ASCII format. Two ":"'s are used to separate the hours, minutes and seconds values.

The following code shows the Init_LCD function.

```
void Init_LCD(void)
{
     Write_bytes(0xFE, 0x54);  // Turn off blinking cursor
     Delay();
     Write_bytes(0xFE, 0x48);  // Move cursor to top left of the display.
     Delay();
     Write_bytes(0xFE, 0x58);  // Clear the display
     Delay();
}
```

This code turns off the blinking cursor on the LCD, sends the cursor to home so that the next character to be written to the display is displayed at the top left corner and clears the display of all text.

The code for the Init_RTC function is shown in the following text

```
void Init_RTC(void)
{
    // Create temporary variables to hold minute and hour values
    CKSTPR1.BIT.RTCCKSTP = 1;  /* Enable the AEC and LCD modules */

     // Wait until RTC registers are not being updated
     while(RTC.RSECDR.BIT.BSY == 1);

     /* 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


     RTC.RTCCR1.BIT.MD = 1; // Set to 24 hour mode
     RTC.RTCCR1.BIT.INT = 1;  // Enable RTC interrupt in RTC control register 1
     RTC.RTCCSR.BYTE |= 0x08; // Set to real time clock function

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

     RTC.RTCCR2.BIT._1SEIE = 1;  // int controller rtc iupt enable

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

This is the same function as shown in section "Software for Communication with LCD via SSU". For details of this code please refer to this section.

Three functions are used to communicate with the LCD via the I²C interface. These are Init_IIC which initialises the I²C peripheral of the H8/38602, the Write_bytes function which writes two bytes on the I²C bus to the LCD and the Transmit_data_End function which transmits the last byte of data to the LCD. The following code shows the Init_IIC function

```
void Init_IIC(void)
{
    CKSTPR2.BIT.IICCKSTP = 1;    // Switch on IIC peripheral of 38602
    IIC2.ICCR1.BYTE = 0x85;      /*ICE RCVD MST TRS CKS3 CKS2 CKS1 CKS0 */
                                 // 1   0    0   0   0    1    0    1
    IIC2.ICMR.BYTE = 0x38;       /* No Wait bit, clear BC write protect  */

    IIC2.ICIER.BYTE = 0x00;      // Disable all interrupts
    IIC2.ICSR.BYTE = 0x00;       // Clear all flags
    IIC2.ICCR2.BYTE = 0x3D;      // BBSY SCP SDAO SDAOP SCLO - ICCRST -
                                 //   0   0   1    1     1   1  0     1


    IIC2.ICCR2.BIT.BBSY = 0;  // Clear busy bit

    while (IIC2.ICCR2.BIT.BBSY == 1); // wait for free bus

}
```

This function turns on the I²C module by setting the IICKSTP bit in the clock stop register. The ICE bit in ICCR1 (the I²C Control Register 1) is then set to 1 enabling the I²C interface. The I²C Mode register is then configured for 9 bit LSB first transmission with no wait bit. The I²C interrupts are then all disabled and their flags cleared. The ICCR2 register is then set up to make the SDA output high. The code then waits for the bus to be free.

The Write_bytes function is shown in the code above

```
/*****************************************************
Function: Write_bytes();
Purpose: To transmit two bytes of data via IIC
Parameters Passed: wr_data1 (first byte of data)
                   wr_data2 (second byte of data)
Parameters Returned: None
*****************************************************/
void Write_bytes(unsigned char wr_data1, unsigned char wr_data2)
{
    // Set MST (Master) and TRS (Transmit) bit high
    IIC2.ICCR1.BYTE = 0xB5;
    // Set BSY bit and clear SCP to issue a start condition
    IIC2.ICCR2.BYTE = 0xBD;

    /* Wait until the transmit data is empty */
    while (IIC2.ICSR.BIT.TDRE == 0);

     /* Put Slave address into data register with Write bit (0) */
    IIC2.ICDRT = (unsigned char)(SLAVE_ADRS | 0x00);

    // Wait until this data has been transmitted
    while (IIC2.ICSR.BIT.TEND == 0);


    // Put first byte of data to be written into register
    IIC2.ICDRT = (unsigned char)(wr_data1 );

    // Wait for byte to be transmitted
    while (IIC2.ICSR.BIT.TDRE == 0);

        Write_data_End(wr_data2);

}
```

The status of the I²C bus was determined in Init_IIC so it is now possible to set the peripheral as a transmit master, and then issue a start condition. The start condition is issued by writing a '1' to the BSY bit and a '0' to SCP bit in the I²C control register 2 ICCR2. One a start condition is issued, the code waits to ensure the Transmit Data register is empty before putting data in the Transmit data register. The first byte to be transmitted should be the LCD I²C slave address followed by the Write value (0x00). Before transmitting the next byte of data, the device should wait until the transmit end flag is set. Again the device should wait until the transmit data register empty flag is set and that another acknowledge was received before transmitting another byte of data. The last byte of data is then passed to the Write_data_End function, the code for which follows.

```
void  Write_data_End( unsigned char wr_data )
{
    IIC2.ICDRT = wr_data;   // Put last byte in transmit register
    while (IIC2.ICSR.BIT.TDRE == 0); // wait until transmission ends
    while (IIC2.ICSR.BIT.TEND == 0);

    IIC2.ICSR.BIT.TEND = 0; // Clear the flag
    IIC2.ICCR2.BYTE &= 0x3f; // Issue a STOP condition.
    // BSY = 0 and SCP = 0.

    // wait until stop condition  is detected
    // Once this is 1, mode switches to slave receive
    while (IIC2.ICSR.BIT.STOP == 0);

    // Clear transmit data register empty flag to 0
    IIC2.ICSR.BIT.TDRE = 0;
}
```

The last byte of data may be placed in the data transmit register as the TDRE flag was checked before entering this function. The device should then wait until both the TDRE and TEND flags are set. The TEND flag is then cleared (ready for a byte of data to be transmitted in the future). A Stop condition is then given by clearing both the BBSY and SCP bit in IICR2. The device should then wait until the STOP flag is set. Now the TDRE flag can be cleared ready for future transmissions.

Software for communication with LCD via SSU

Figure 12 shows the software flow for the SSU interface

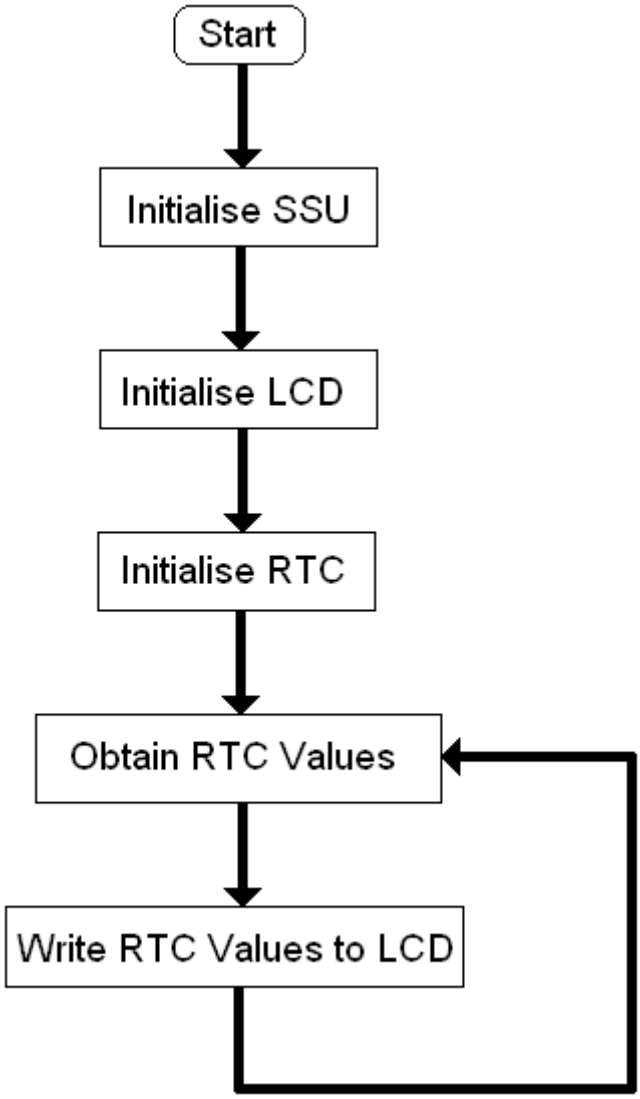


Figure 12: SSU interface software flow diagram

Firstly the SSU is initialised in order that this interface can be used to clear the LCD and transmit instruction / characters to the LCD. The LCD is then initialised so that it may be cleared ready for a time value to be written to it. Lastly, the RTC is initialised. This starts the RTC counting. It is now possible to grab the RTC values and write them to the LCD. An infinite loop is set up which takes the RTC register values and writes them to the LCD display. The following code shows the function Main.

```
void main(void)
{
 // Set up temporary variables to hold the current value of seconds
 // mins and hours.
    int temp_secs_units = 0x00;    // Initialise all of these to 0.
    int temp_secs_tens = 0x00;
    int temp_mins_units = 0x00;
    int temp_mins_tens = 0x00;
    int temp_hours_units = 0x00;
    int temp_hours_tens = 0x00;

    // Initialise the SSU peripheral of the 38602 device.
    Init_SSU();
    // Initialise the LCD so that a time may be written to it.
    Init_LCD();
    // Initialise the Real time clock module on the 38602 device.
    Init_RTC();

    for(;;)      // Initiate an infinite loop so that
                 // the time can be continously written to the LCD
    {
     Write_One_Char(0xFE);  // Write instuction to LCD
     Write_One_Char(0x48);  // to move the cursor home
     // Wait a period for instruction to be executed
     Delay();

     // Wait until hour register is ready to be read
     while(RTC.RHRDR.BIT.BSY == 1);
        // store hour unit value in temporary variable
        temp_hours_units = RTC.RHRDR.BIT.HR0;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current hour unit value
        temp_hours_units += 0x30;
        // store hour tens value in temporary variable
        temp_hours_tens = RTC.RHRDR.BIT.HR1;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current hour tens value
        temp_hours_tens += 0x30;

     // Wait until minute register is ready to be read
     while(RTC.RMINDR.BIT.BSY == 1);
        // store minute unit value in temporary variable
        temp_mins_units = RTC.RMINDR.BIT.MN0;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current minute unit value
        temp_mins_units += 0x30;
        // store minute tens value in temporary variable
        temp_mins_tens = RTC.RMINDR.BIT.MN1;
        // Add the ASCII offset value so variable now holds ASCII
        // value of current minute tens value
        temp_mins_tens += 0x30;
```

```
// Now it is possible to write ASCII time values to LCD
     // Write hours tens value to LCD as this value should appear first
     Write_One_Char(temp_hours_tens);
     // Write hours units value to LCD as this value should appear second
     Write_One_Char(temp_hours_units);
     // Write a ":" character to the LCD to separate hours from mins
     Write_One_Char(0x3A);
     // Write minute tens value to LCD as this should appear next
     Write_One_Char(temp_mins_tens);
     // Write minute unit value to LCD
     Write_One_Char(temp_mins_units);
     // Write a ":" character to separate minutes from seconds
     Write_One_Char(0x3A);
     // Write second tens value to LCD
     Write_One_Char(temp_secs_tens);
     // Write second units value to LCD
     Write_One_Char(temp_secs_units);
    }
 }
```

The variables declared at the top of the function are used to hold the temporary current time value from each part of the second, minute and hour RTC time registers. The Synchronous Serial Communication Unit is then initialised and configured to communicate with the LCD. The LCD may then be initialised. This clears the display and allows the next character transmitted to the LCD will be displayed in the top left corner of the LCD. The RTC is then started. Starting it at this stage prevents the first second from being missed in the display of the current time. The code then goes into an infinite loop, moving the LCD cursor home, grabbing the current time value from the RTC registers whilst these are not busy, and displaying the time value on the LCD.

The Init_SSU function is shown in the following code.

```
void Init_SSU(void)
{
 CKSTPR2.BIT.SSUCKSTP = 1; // Switch on the SSU peripheral
 PFCR.BIT.SSUS = 1;        // Change the pin assignments of the SSU
                           // SSI is P90, SSO is P91, SSCK is P92, SCS is P93
 SSU.SSER.BYTE &= 0x3F;    // Transmission is disabled (TE, bit 7 is cleared to 0)
                           // Reception is disabled (RE, bit 6 is cleared to 0)
 SSU.SSCRL.BIT.SSUMS = 1;  // Set the peripheral for four line comm mode
 SSU.SSMR.BYTE = 0x87;     // Set clock rate at thi/256 and set MSB first.

 // LCD communicates in mode 1, 1. Set polarity and phase bits to handle this
 SSU.SSMR.BIT.CPOS = 1;    // Set clock polarity to idle state low
 SSU.SSMR.BIT.CPHS = 0;    // Set data to change at first edge

 SSU.SSCRH.BYTE |= 0x86;   // Set bit 1 & 2 to enable serial clock pin and SCSn pins.
                           // set MSS bit to set as master
 SSU.SSCRH.BYTE &= 0x9F;   // Clear BIDE bit so that two pins are used for comms.
                           // SOOS bit cleared for CMOS output

 SSU.SSSR.BYTE = 0x00;     // Clear all the status flags
 SSU.SSER.BYTE = 0x80;     // Set the TE (Transmit enable) bit 7 bit
}
```

Firstly the SSU peripheral is switched on in the clock stop register. The following figure (figure 13) shows the flow diagram used to initialise the SSU peripheral.

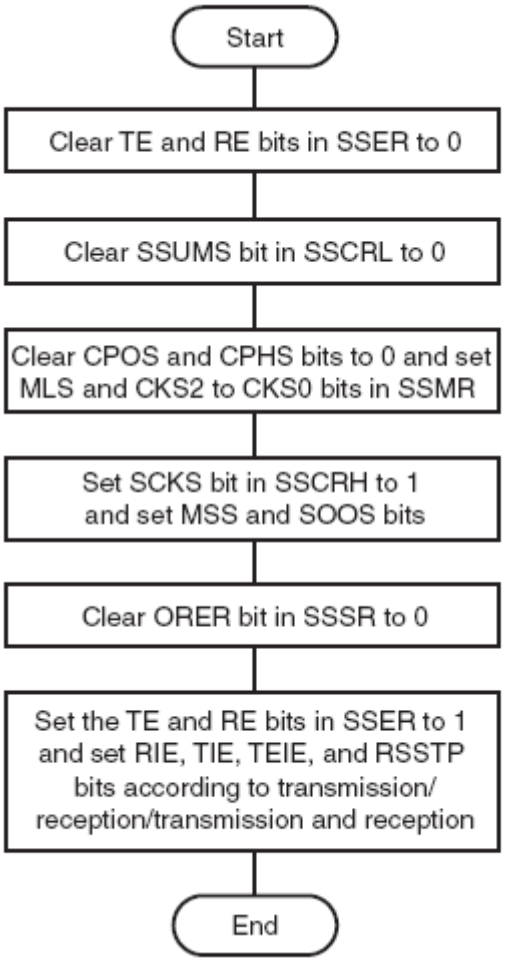


Figure 13: SSU Initialisation

The Init_SSU function then follows the flow diagram in figure 13. The TE and RE bits are cleared to 0. The SSUMS bit is then set to 1 (instead of 0), as this application requires four line communication mode. The LCD needs the SPI to communicate in mode 1, 1 and so the CPOS and CPHS bits are set to 1 and 0 respectively. MLS is set to MSB first and the clock bits are set to the slowest setting of ϕ/256. The SSCRH register is then set up to configure the peripheral as a master with serial clock pins and chip select pins enabled. The BIDE bit is also cleared for two pin communication and the SOOS bit cleared for CMOS output. The status flags may then all be cleared and the transmit enable bit set. The SSU is now ready to transmit to the LCD.

The LCD displays the character whose ASCII representation is sent to it. Instructions to the LCD itself start 0xFE; the byte following this indicates the instruction. The initialisation code for the LCD is shown below

```
Init_LCD(void)
{
     // Send instruction
     Write_One_Char(0xFE);
     // to clear the display
     Write_One_Char(0x58);

     Delay(); // wait for instruction to be executed

     // Send instruction
     Write_One_Char(0xFE);
     // to turn off blinking cursor
     Write_One_Char(0x54);

     // Send instruction
     Write_One_Char(0xFE);
     // to move cursor position to the top left of the display area.
     Write_One_Char(0x48);
     Delay(); // wait for instruction to be executed
}
```

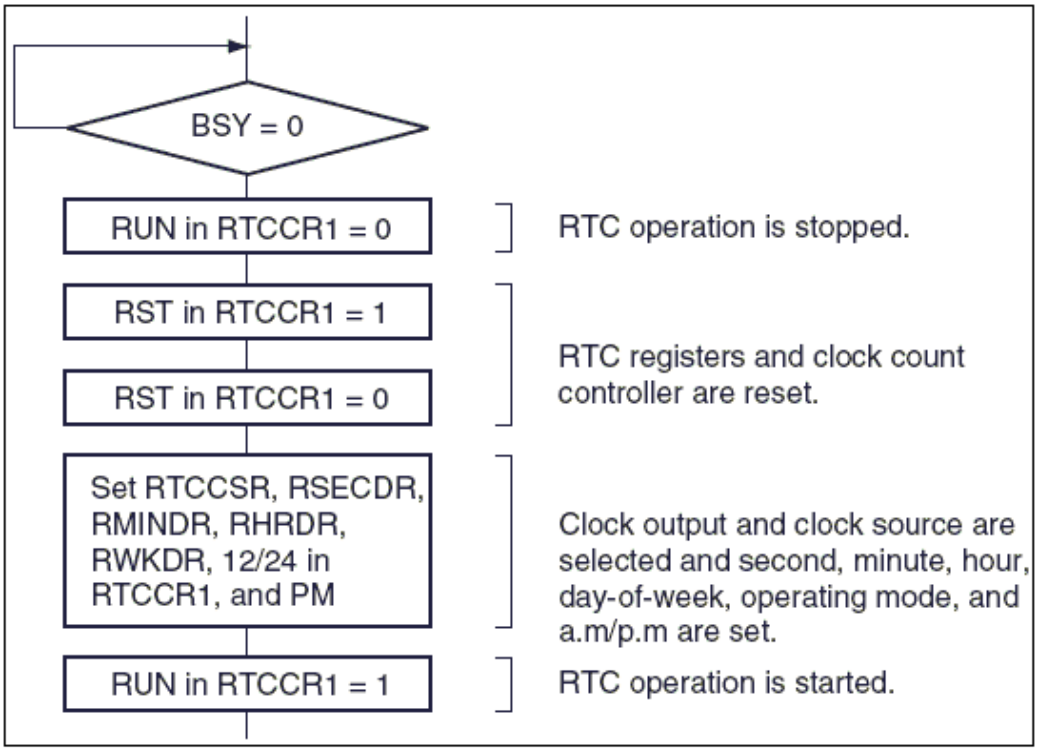
This function clears the LCD display, turns off the visibility of the blinking cursor, moving it to the top left of the LCD display. The LCD is now ready for the time to be displayed on it.

The last initialisation function to be called is Init_RTC. The code for this is shown in the following program code.

```
void Init_RTC(void)
{
// Create temporary variables to hold minute and hour values
    CKSTPR1.BIT.RTCCKSTP = 1;  /* Enable the AEC and LCD modules */
 // Make sure the time value is not at a transition and
 //  the RTC registers are not busy
     while(RTC.RSECDR.BIT.BSY == 1);

     /* 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
     RTC.RTCCR1.BIT.MD = 1; // Set to 24 hour mode
     RTC.RTCCR1.BIT.INT = 1;
     RTC.RTCCSR.BYTE |= 0x08; // set to real time clock function
     RTC.RSECDR.BIT.SC0 = 0; // Set the second register value to 0
     RTC.RSECDR.BIT.SC1 = 0; // Set the second reqsiter value to 0
     RTC.RTCCR1.BIT.RUN = 1;// TURN RUN ON
     RTC.RTCCR2.BIT._1SEIE = 1;  // int controller rtc iupt enable
     IENR1.BIT.IENRTC = 1; // Enable the RTC interrupt
}
```

Firstly, as with the SSU peripheral, the RTC peripheral is switched on via the clock stop register. Figure 14 shows the initialisation process required for the RTC peripheral



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. This value is then entered into the RTC registers. Finally the RUN bit is set to 1 again to start the peripheral. The Init_RTC code performs this initialisation with a couple of differences. The RTC interrupt is enabled in the RTC control register 1, and the RTC is set to 24 hour mode. Further the RTC second interrupt is enabled in the interrupt enable register and the RTC control register 2. It is not necessary to initialise the minute and hour RTC registers as these should be at 0 already.

Once all the peripherals have been initialised it is now possible to write data to the LCD. The Write_one_char function shown in the following coding writes one byte of data to the LCD via the SSU bus.

```
void Write_One_Char(char character1)
{
  SSU.SSER.BIT.TE = 1;     // Enable SSU transmission

 while(SSU.SSSR.BIT.TDRE == 0);  // wait until transmit data is empty

 SSU.SSTDR = character1;    // put data to be transmitted in SSTDR register

 while(SSU.SSSR.BIT.TEND == 0); // wait until transmission of data has ended

 SSU.SSSR.BIT.TEND = 0;    // clear the TEND flag as this should be finished
 SSU.SSER.BIT.TE = 0; // disable transmission
}
```

This function enables the transmit data so that the peripheral may transmit data on the bus. Next the device should wait to check that the transmit data register is empty before placing data in this register. Once this check is complete, a character of data (one byte) may be written into the transmit register. The device should then wait until this byte has been transmitted by checking the Transmit End flag. The TEND flag may then be cleared and transmission disabled to complete the transmission.

Conclusion

The MBH838602 board was successfully used with an LCD display and some interfacing circuitry. Communication to the LCD from the H8/38602 was achieved using both the SSU and I^2C peripheral of the device.

Website and Support

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

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

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.