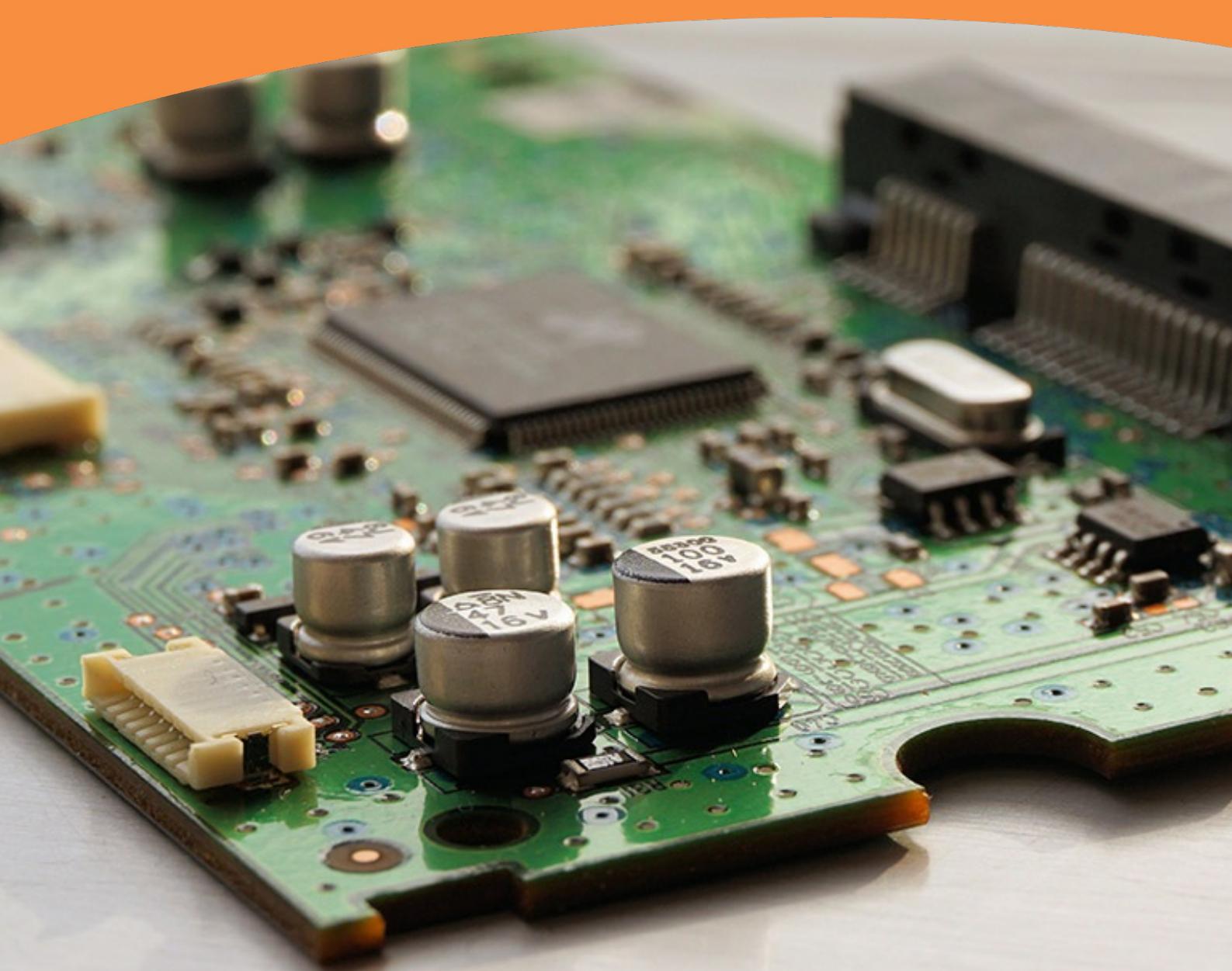


Paulos F040

A co-operative RTOS for the Silicon Labs C8051F040 µC

Paul P. Debono



PAUL DEBONO

PAULOS F040
A CO-OPERATIVE RTOS
FOR THE SILICON LABS
C8051F040 μ C

Paulos F040: A co-operative RTOS for the Silicon Labs C8051F040 µC

1st edition

© 2019 Paul Debono & bookboon.com

ISBN 978-87-403-2863-9

CONTENTS

Preface	8
Acknowledgements	11
1 C8051F040 Basics	13
1.1 Introduction	14
1.2 Memory Types	15
1.3 Program/Data Memory (Flash)	16
1.4 External Data Address Space (XRAM)	17
1.5 Special Function Register (SFR) Memory	20
2 PaulOS_F040 – a co-operative RTOS	25
2.1 Description of the RTOS Operation	25
2.2 PaulOS_F040.C System Commands	29
2.3 Descriptions of the commands	31
2.4 PaulOS_F040 parameters header file	46
2.5 Example using PaulOS_F040 RTOS	49

The image shows a woman with long blonde hair, wearing a blue tank top, smiling while wearing a black VR headset. To her right is a red rectangular advertisement for MT Højgaard. The ad features the company logo (a stylized 'M' and 'H') and the text 'MT Højgaard'. Below this, in large white capital letters, is 'BEDRE LØSNINGER'. Underneath, a block of Danish text reads: 'I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?' At the bottom of the ad, the website 'mth.dk/vorestilgang' is listed.

3	Master-Slave CAN bus RTOS	65
3.1	Multi-Controller RTOSs	65
3.2	CAN bus RTOS Example	65
4	Programming Tips and Pitfalls	133
4.1	RAM size	133
4.2	SFRs	134
4.3	Setup faults	135
4.4	Serial Port (UART0)	136
4.5	Interrupts	137
4.6	RTOSs pitfalls	137
4.7	C Tips	137
	Appendix A PaulOS_F040.C Source Code Listing	139
	Appendix B – Further Example	228
	Bibliography	307
	Index	309
	End Notes	310

LIST OF FIGURES

Figure 1-1 C8051F040TB Board	13
Figure 1-2 C8051F040 IC Block Diagram	14
Figure 2-1 RTOS Task states diagram	26
Figure 2-2 Blinking LED example	50
Figure 3-1 Screen shot with the CAN bus example	65
Figure 3-2 Screen shot showing the MASTER option	66
Figure 3-3 Screen showing the SLAVE option	67
Figure 4-1 Screenshot of the Keil Target setup	134

LIST OF TABLES

Table 1-1 C8051F040 on-chip memory map	16
Table 1-2 C8051F040 Internal Data Address Space	20
Table 1-3 C8051F040 Special Function Registers (SFRs)-DIRECT addressing ONLY	22

PREFACE

This text book is intended to be used as a reference book for those whose work or study requires familiarity with microcontrollers and real-time operating systems (RTOSs). It deals particularly with a modified version of the original PaulOS co-operative RTOS so as to be compatible with the Silicon Labs C8051F040 device, with its increased number of SFR pages, timers, Control Area Network (CAN bus) and interrupts. Ideally, in order for this book to make sense it should be used in conjunction with my previous books (Debono, 2013a) and (Debono, 2013b) where various simple RTOSs based on the basic 8051 microcontroller were fully explained. Another one of my books which would also be beneficial is (Debono, 2015) where a similar exercise was carried out for the C8051F020 device, also from Silicon Labs. Naturally one cannot do without reference to the C8051F040 data sheet/manual (Labs, 2004) in order to get the full benefits out of this powerful microprocessor and be able to better understand the examples found in this book.

It has basically the same format as the book for the C8051F020 device (Debono, 2015) but the software has obviously been modified to make full use of the architecture of the C8051F040. Namely the RTOS has the capability to make use of any one of the 5 timers available on the device as the source of its tick timer. Certain explanations for the essential SFRs are left out from this book and the reader is encouraged to read the book about the C8051F020 device to familiarise oneself with the SFRs, watchdog timer, system clock, crossbar registers etc. Moreover since this device has more than one SFR page, the RTOS had to be modified so as to switch over to the correct SFR page as and when needed. An explanation of the SFR paging requirement is given in this book, something which was not required in my earlier books simply because they do not have that capability/requirement. The example programs have also been expressly adapted to the new device. For example the Master-Slave RTOS which previously relied on the serial RS-232 transmissions and interrupts between the boards, has been replaced with one which uses the CAN bus to connect and synchronise the two boards.

As it is normally expected, it would be helpful if the reader has already got some familiarity with personal computers and has taken an introductory course in digital devices. Some experience in assembly language programming would also be fruitful and it is also assumed that the reader is familiar with binary and hexadecimal numbers.

Learning to write programs is like learning to ride a bicycle in that reading alone is not enough. Hands-on practical experience is essential. Therefore, to enhance the usefulness of this book as a learning tool, the reader is encouraged to test some of the example programs given throughout the text using easily available free software, such as the latest version of the ARM KEIL IDE (see <http://www.silabs.com> and <http://www.keil.com>).

This book is structured into 4 chapters and 2 appendices with full source code listings of the PaulOS RTOS and of the example programs. A brief outline of the contents of each chapter is given below:

Chapter 1:

This chapter describes the C8051F040 microcontroller and explains its internal organization and lists the internal special function registers (SFRs) used to set the mode of operation of the various peripherals which are present on this versatile mixed-signal (Analogue and Digital) microcontroller device. This device has so many peripherals that the need arose to have more than one SFR page, and hence SFR paging is used when programming this device so as to address the correct SFR.

Chapter 2:

The PaulOS co-operative RTOS is described in this chapter. This is the ‘flagship’ RTOS which we regularly use during the year with our students. It is heavily used also for their final year theses and it has been regularly refined to reflect the changes and upgrading requested by the students as they became more and more familiar with the performance and limitations of this co-operative RTOS. In this RTOS, each task is free to run for as long as it wishes. The task itself can control when to give up the processor time to allow other tasks to run. The modified version PaulOS_F040, running on the much faster C8051F040 at over 20MIPS, is ideally suited for small project applications and for getting the most out of the device.

Chapter 3:

We discuss a modification to the basic RTOS so that it can run on two separate boards connected together via the CAN bus. One board would be acting as a master and the other as a slave. The RTOS running on the slave would be synchronised to the one running on the master via messages using the CAN bus and its associated interrupts. A full listing of the modified CAN bus RTOS with an example program is also given in this chapter.

Chapter 4:

In this final chapter we discuss some programming tips and common pitfalls which should be avoided when programming such microcontrollers and when using the PaulOS RTOS. This chapter is very similar to the one found on (Debono, 2015) and it would be a good idea to read this chapter first before attempting to write the first program.

Appendix A:

Finally in the appendices we can find the full program source listings (C language format) of the files associated with the C8051F040 version of the PaulOS RTOS described in chapter 2.

Appendix B:

A further example is given here with full program source listings in C language format. It is a program which displays time by means of a single LED, flashing the time and date using the Morse code. Time can also be optionally synchronised to a GPS signal by connecting a GPS source to the serial port.

Whilst hoping that you will find this book useful, please feel free to contact me if you have any queries or suggestions. (email: pawlu.debono@yahoo.co.uk)

ACKNOWLEDGEMENTS

I would like to acknowledge the assistance given by my students who helped me test some of the examples and pointed out some mistakes and omissions. I would also like to take this opportunity to thank all my students for the wonderful productive time we had together during my time at the University of Malta and wish them all the best success in their future careers,

I am also very grateful for the contributions made by my son Luke who proof read the first draft, for his suggestions and constructive comments whilst formulating the structure of this book.

Finally I am deeply grateful to Prof. Ing. Victor Buttigieg who kindly reviewed the final version of the book and put forward valuable and much appreciated suggestions.

**To my wife Maria for being so supportive and patient
with me, my two sons Neil and Luke for their continuous
encouragement, and to my lovely grand daughter Mila.**

1 C8051F040 BASICS

This chapter briefly describes the C8051F040 microcontroller and explains its internal organization and the way its internal special function registers can be used to setup the various peripherals. Many web pages, books (see bibliography list) and tools are available for the 8051 developer, and many of them are free!. The use of the C8051F040 data sheet (Labs, 2004) and referring to (Debono, 2015), (Chew, et al., 2005), (Huang, 2009) and (Schultz, 2004) in conjunction with this book is highly recommended whilst following the code listed elsewhere throughout the book.

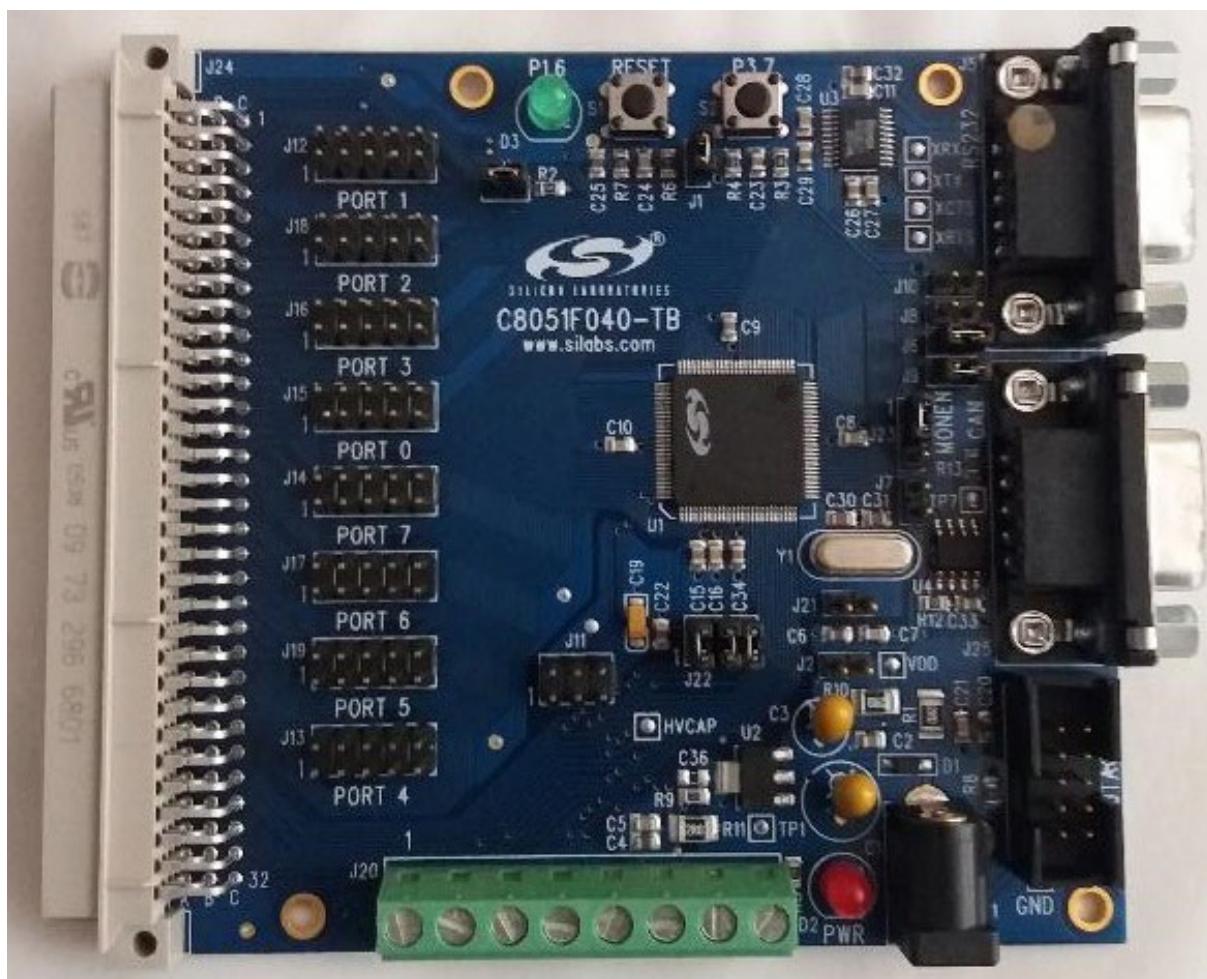


Figure 1-1 C8051F040TB Board

When using the KEIL IDE as stated in the Preface section, the Silicon Labs µV3/µV4/µV5 driver program SiC8051F_µVision.exe, which is freely available from the Silicon Labs site should be installed. This would enable the compiled program to be downloaded on the development board via the JTAG once the debug tab is pressed. New software packages from Silicon Labs (<http://www.silabs.com>) and arm Keil (<http://www.keil.com>) are also available.

At the University we use the Silicon Labs development boards shown in Figure 1-1. At the time of writing, it is one of a series of super-charged versions of the 8051 family from Silicon Labs.

1.1 INTRODUCTION

Despite its relatively old age, the 8051 (developed by Intel Corporation in the early 1980's) is one of the most popular microcontrollers in use today. Many derivative microcontrollers have since been developed that are based on and compatible with the 8051. Thus the ability to program an 8051 is an important skill for anyone who plans to develop products that will take advantage of most microcontrollers. My earlier books (Debono, 2013a, 2013b) did just that and introduced the reader to the RTOS, while (Debono, 2015) extended the RTOS on the the C8051F020 microprocessor.

The sections in these chapters are simply a further extension to the (Debono, 2015) book so that the RTOS is able to cater for any changes introduced with the C8051F040, namely the SFR paging requirement, the availability of the CAN bus and different interrupts. All attempts were made so as not to repeat any explanations and examples given in my previous books unless absolutely necessary, since the C8051F040 is very similar to the C8051F020 device. Throughout this book, it is therefore assumed that the reader has got some amount of programming knowledge in C and that he has a basic understanding of the hardware.

The C8051F040 is a 64 or 100-pin IC as shown in Figure 1-2.

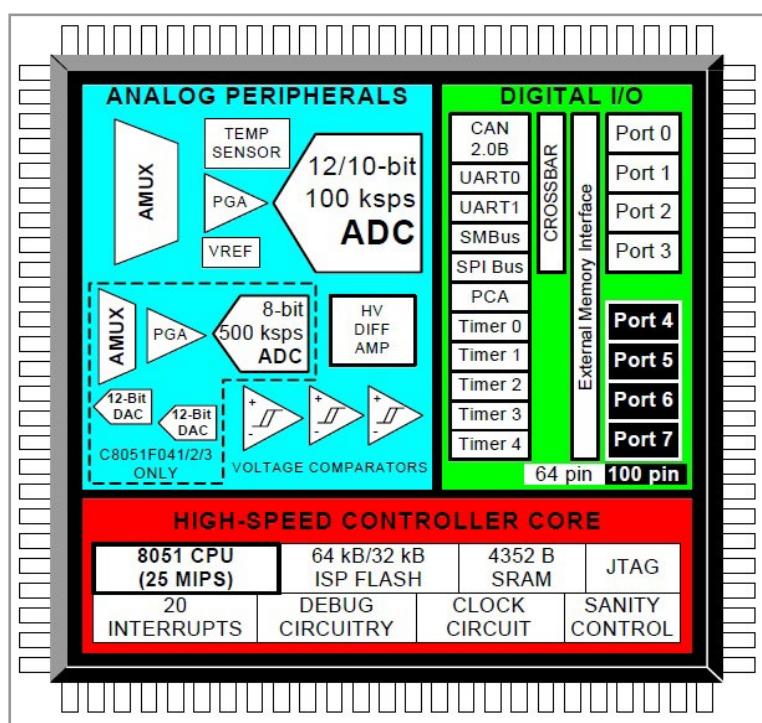


Figure 1-2 C8051F040 IC Block Diagram

Download free eBooks at bookboon.com

We shall now deal with the internal organisation of the C8051F040 microcontroller.

1.2 MEMORY TYPES

The C8051F040 has three types of memory and each type has to be addressed in a different way. To effectively program the device it is necessary to have a basic understanding of these memory types and how to address them, especially if and when programming directly in assembly language. The memory types found on the C8051F040 are illustrated in Table 11 and they are classified as the Data Memory (RAM) which is itself organised in two separate areas, namely the Internal Data Address Space which is identical with all the basic 8052/8032 devices, and External Data Address Space which has 4096 bytes present on-chip with the ability to have extra additional storage space added externally. Then there is the Program Code/Data Memory (Flash). Addresses throughout this book are shown suffixed either with a lower case h (i.e. 0Fh) or with a upper case H (i.e. 0FH) or pre-fixed with a '0x' (i.e. 0xFF) to signify that they are hexadecimal numbers.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



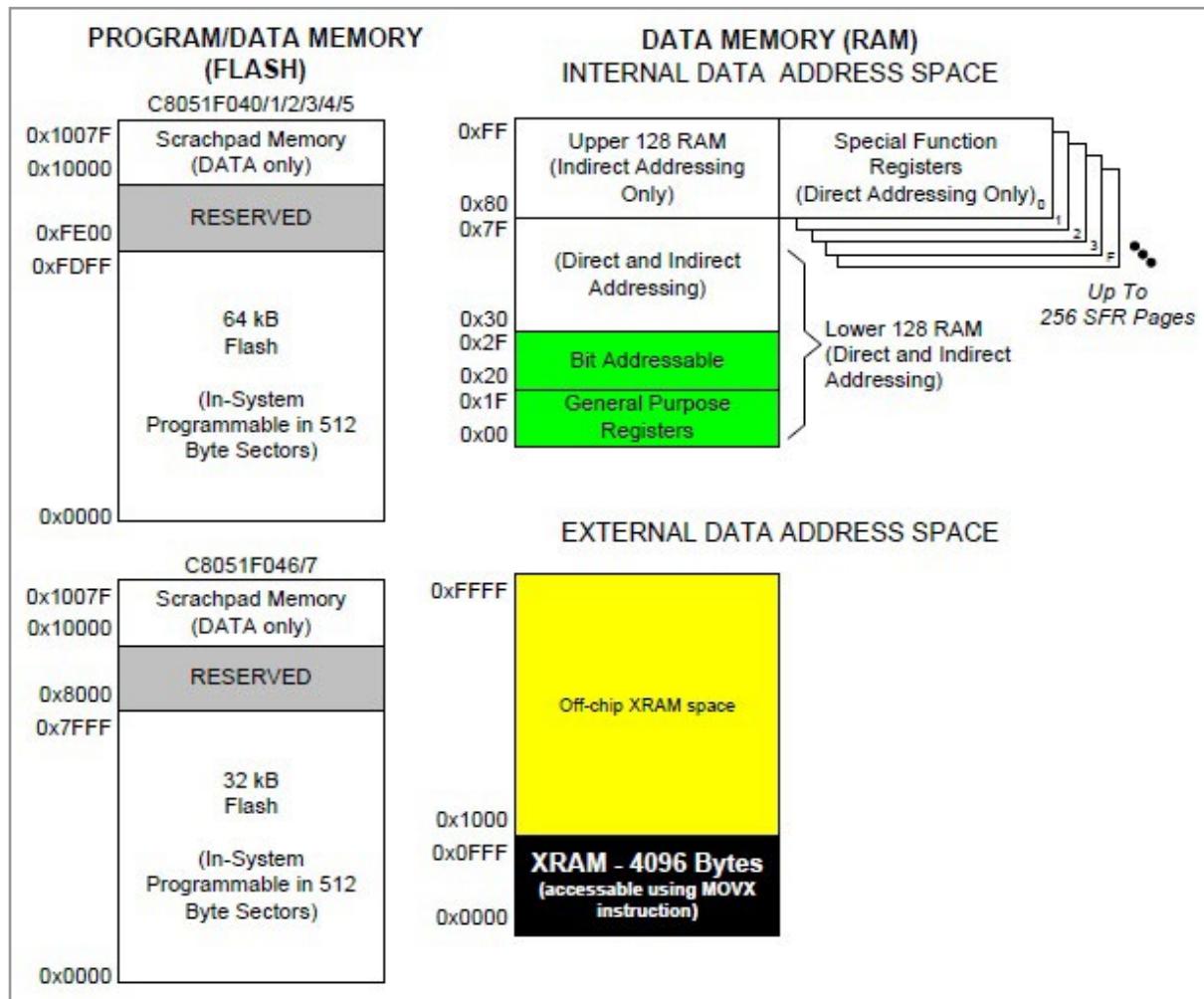


Table 1-1 C8051F040 on-chip memory map

1.3 PROGRAM/DATA MEMORY (FLASH)

The flash memory is the part of memory that holds the actual code or program that is to be executed. This memory is limited to 64KB but being a flash memory, the code can be re-written to it many times so as to change/update the program.

Also because it is a flash memory, it can also be used to store data which needs to be updated and stored for re-use. For example you might need to store some settings (variables), which although they can be varied whilst running the program, you would still need to store their value, so that when the program runs again (after having been switched off), it would start off again using those previously updated and stored values.

In the IDE, this memory would be denoted by the CODE keyword, and naturally you can (should) use this memory area also to store fixed constants.

1.4 EXTERNAL DATA ADDRESS SPACE (XRAM)

As an obvious opposite of Internal RAM, the C8051F040 also supports what is called External Data Address Space (XRAM). For the moment we shall make use of assembly language, we will use C language programming later. This is accessible using the MOVX assembly code instruction. For example, to increment an Internal RAM location (such as INC R1) by 1 requires only one instruction which is executed in one instruction cycle. To increment a 1-byte value stored in External RAM requires four assembly language instructions as shown here

MOV DPTR, #ADDRESS	(2 instruction cycles)
MOVX A, @DPTR	(2 instruction cycles)
INC A	(1 instruction cycle)
MOVX @DPTR, A	(2 instruction cycles)

which are executed in seven instruction cycles. In this case, external memory is seven times slower!

What External RAM loses in speed and flexibility it gains in quantity. While the Internal RAM is limited 256 bytes, the C8051F040 has an on-chip External RAM of to 4KB.

Modern devices now also have this so-called external RAM, physically residing on the same chip, but it is still referred to as external (or XDATA) and all the information listed in this book still holds.

1.4.1 INTERNAL DATA ADDRESS SPACE (RAM)

As mentioned at the beginning of this chapter, the 8051 includes a certain amount of on-chip memory. On-chip memory is really one of two types: Internal RAM usually used to store variables and Special Function Register (SFR) memory used to store the system registers which control the mode of operation of the built-in peripherals. The layout of the C8051F040's internal memory is presented in the memory map shown in Table 1-2 which is identical with the basic 8051 except of course that it now has more SFRs since this device is much more capable than the basic 8051, with a much larger list of peripherals. In fact this device has more than one page for storing the SFRs with the result that SFR paging is required. This is dealt with in section 1.5.1 where the procedure used for paging is discussed.

The C8051F040 has a bank of 256 bytes of Internal RAM. This Internal RAM is found on-chip on the device itself so it is the fastest RAM available, and it is also the most flexible in terms of reading, writing, and modifying its contents. Internal RAM is volatile, so that when the device is reset this memory is lost.

The 256 bytes of internal ram are subdivided as shown on the memory map in Table 1-2. The first eight bytes (00h - 07h) are referred to as register bank 0. By manipulating certain SFR bits (in the PSW special function register), a program may choose to use register banks 1, 2, or 3. These alternative register banks are located in internal RAM, occupying addresses 08h through 1Fh. Register banks and bit memory mentioned below were fully discussed in my earlier books.

Bit Memory is also another part of the internal RAM, which as the name implies is able to store and manipulate bit variables. We will say more about the bit memory area later (see section 1.6), but for now we just have to keep in mind that the bit memory actually resides in internal RAM, ranging from address 20h through address 2Fh.

A photograph of the Apollo Hotel 1 building at night. The hotel's name is visible on the sign above the entrance. In the foreground, there is a large red circular logo containing a white lightbulb icon. To the right of the logo, the text "CISO Conference" is written in white, with "Produced by Inspired" underneath. On the right side of the image, there is a white rectangular overlay containing the text "Apollo Hotel 1, Groenlandsekade Vinkeveen, Amsterdam, NL" and "Dec 5th 2019" in blue. At the bottom of the image, there is another white rectangular overlay containing the text "Listen, learn & build relationships with our Network of CISOs & Cyber Security Leaders" in black, and the "Inspired" logo with a lightbulb icon to its left.

The 80 bytes that remain in Internal RAM, from address 30h through address 7Fh, may be used to store any user variables that need to be accessed frequently or at high-speed during the execution of the program. This area is also utilised by the microcontroller as a storage area for the operating stack.

Hex Byte Address	Hex Bit Address								Notes
FFH	Indirectly Addressable General Purpose RAM								Used as a STACK Area and to store user variables
80H	Directly and Indirectly Addressable General Purpose RAM								Used as a STACK Area and to store user variables
30H									
2FH	7F	7E	7D	7C	7B	7A	79	78	Bit Addressable Section (Bit Addresses shown are in hex)
2EH	77	76	75	74	73	72	71	70	
2DH	6F	6E	6D	6C	6B	6A	69	68	
2CH	67	66	65	64	63	62	61	60	
2BH	5F	5E	5D	5C	5B	5A	59	58	
2AH	57	56	55	54	53	52	51	50	
29H	4F	4E	4D	4C	4B	4A	49	48	
28H	47	46	45	44	43	42	41	40	
27H	3F	3E	3D	3C	3B	3A	39	38	
26H	37	36	35	34	33	32	31	30	
25H	2F	2E	2D	2C	2B	2A	29	28	
24H	27	26	25	24	23	22	21	20	
23H	1F	1E	1D	1C	1B	1A	19	18	
22H	17	16	15	14	13	12	11	10	
21H	0F	0E	0D	0C	0B	0A	09	08	
20H	07	06	05	04	03	02	01	00	

Hex Byte Address	Hex Bit Address	Notes
1FH 18H	Register Bank 3 (R0 – R7)	
17H 10H	Register Bank 2 (R0 – R7)	
0FH 08H	Register Bank 1 (R0 – R7)	
07H 00H	Register Bank 0 (R0 – R7)	Bank is Selected Using RS0 and RS1 In the PSW Register. See SFRs.

Table 1-2 C8051F040 Internal Data Address Space

The stack is used to save return addresses when calling functions or subroutines. It is also used to store some values temporarily until they are retrieved again when needed. It should be noted, as illustrated in the memory map of Table 1-2, the area used for the stack is also shared with any user variables stored in ‘DATA’. If more stack space is required, the variables can be moved to ‘XDATA’ area either when declaring the variable or by setting the global default in the ‘Target Option’ tab of the IDE as shown in Figure 4-1.

1.5 SPECIAL FUNCTION REGISTER (SFR) MEMORY

Special Function Registers (SFRs) reside in areas of internal memory that control specific functionality of the C8051F040 chip, as shown in Table 1-3. For example, eight SFRs permit access to the 8 I/O port P0-P7. Other SFRs (SBUF0, SBUF1) allows a program to read from or write to its serial ports which are called UART0 and UART1 (Universal Asynchronous Receiver/ Transmitter). Other SFRs allow the user to set the serial baud rates, control and access timers, ADC, DAC, CAN bus etc. and also configure the 8051’s interrupt system. All the SFR names and bit names are already defined in the header file C8051F040.H found in appendix A.

1.5.1 SFR ADDRESSES AND PAGING

The C8051F040 is a flexible microcontroller with a relatively large number of modes of operation. In order to be able to make full use of these different modes or ways of using the built in peripherals of this microcontroller, our program may inspect and/or change the operating mode of the 8051 by manipulating the values of some specific SFRs.

They are accessed just as if they were normal Internal RAM. The only difference is that Internal RAM for the 8051 resides from address 00h through 7Fh whereas the SFR registers exist in the address range of 80h through FFh. Each SFR has an address (80h through FFh) and a name.

Table 1-3 provides tabular representation of the C8051F040's SFRs, their name, and their address in hexadecimal. Although the address range is from 80h through FFh, the device has 5 pages (0,1,2,3,4 and F) as shown in the on-chip memory map in Table 1-1 thus offering 128x5 possible addresses. Certain locations are not used and reading data from these empty addresses will in general return some meaningless random data while writing data to these addresses will have no effect at all. In fact the actual memory cell of these free locations might not be physically present on the chip. These free locations are reserved for future enhanced and upgraded versions of this family of microcontrollers, and certain versions have even more SFR pages. These devices which make use of more than one 128-byte page of SFRs have to switch pages in order to set the correct SFR as shown in Table 1-3.

It should be pointed out here once again that all the SFRs and control bits within the SFRs are already pre-defined in the C8051F040.H header file (see Appendix A.5).

A D D R E S S	0(8)	1(9)	2(A)	3(B)	4(C)	5(D)	6(E)	7(F)	SFR P A G E
F8	SPI0CN CAN0CN P7	PCA0L P7	PCA0H P7	PCA0CPL0 P7	PCA0CPH0 P7	PCA0CPL1 P7	PCA0CPH1 P7	WDTCN (ALL PAGES) P7	0 1 2 3 F
F0	B (ALL PAGES)						EIP1 (ALL PAGES)	EIP2 (ALL PAGES)	0 1 2 3 F
E8	ADC0CN ADC2CN P6	PCA0CPL2 P6	PCA0CPH2 P6	PCA0CPL3 P6	PCA0CPH3 P6	PCA0CPL4 P6	PCA0CPH4 P6	RSTSRC P6	0 1 2 3 F
E0	ACC (ALL PAGES)	PCA0CPL5 XBR0	PCA0CPH5 XBR1		XBR2 XBR3		EIE1 (ALL PAGES)	EIE2 (ALL PAGES)	0 1 2 3 F
D8	PCA0CN CAN0DATL P5	PCA0MD CAN0DATH P5	PCA0CPM0 CAN0ADR P5	PCA0CPM1 CAN0TST P5	PCA0CPM2 P5	PCA0CPM3 P5	PCA0CPM4 P5	PCA0CPM5 P5	0 1 2 3 F
D0	PSW (ALL PAGES)	REF0CN P5	DAC0L DAC1L P5	DAC0H DAC1H P5	DAC0CN DAC1CN P5		HVA0CN P5		0 1 2 3 F
C8	TMR2CN TMR3CN TMR4CN P4	TMR2CF TMR3CF TMR4CF P4	RCAP2L RCAP3L RCAP4L P4	RCAP2H RCAP3H RCAP4H P4	TMR2L TMR3L TMR4L P4	TMR2H TMR3H TMR4H P4		SMB0CR P4	0 1 2 3 F
C0	SMB0CN CAN0STA P4	SMB0STA P4	SMB0DAT P4	SMB0ADR P4	ADC0GTL ADC2GT P4	ADC0GTH P4	ADC0LTL ADC2LT P4	ADC0LTH P4	0 1 2 3 F
B8	IP (ALL PAGES)	SADENO P4	AMX0CF P4	AMX0SL P4	ADC0CF P4	AMX0PRT P4	ADC0L P4	ADC0H P4	0 1 2 3 F
	0(8)	1(9)	2(A)	3(B)	4(C)	5(D)	6(E)	7(F)	

A D D R E S S	0(8)	1(9)	2(A)	3(B)	4(C)	5(D)	6(E)	7(F)	SFR P A G E
B0	P3 (ALL PAGES)							FLSCL	0 1 2 3 F
A8	IE (ALL PAGES)	SADDR0				P1MDIN	P2MDIN	P3MDIN	0 1 2 3 F
A0	P2 (ALL PAGES)	EMIOTC	EMIOCN	EMI0CF		P0MDOUT	P1MDOUT	P2MDOUT	0 1 2 3 F
98	SCON0 SCON1	SBUF0 SBUF1	SPI0CFG	SPI0DAT		P4MDOUT	P5MDOUT	P6MDOUT	0 1 2 3 F
90	P1 (ALL PAGES)	SSTA0						SFRPGCN	0 1 2 3 F
88	TCON CPT0CN CPT1CN CPT2CN	TMOD CPT0MD CPT1MD CPT2MD	TL0 OSCION	TL1 OSCICL	TH0 OSCXCN	TH1	CKCON	PSCTL	0 1 2 3 F
80	P0 (ALL PAGES)	SP (ALL PAGES)	DPL (ALL PAGES)	DPH (ALL PAGES)	SFRPAGE (ALL PAGES)	SFRNEXT (ALL PAGES)	SFRLAST (ALL PAGES)	PCON (ALL PAGES)	0 1 2 3 F
	0(8)	1(9)	2(A)	3(B)	4(C)	5(D)	6(E)	7(F)	

Table 1-3 C8051F040 Special Function Registers (SFRs)-DIRECT addressing ONLY



Max's next Bookboon eBook
Your Boss: Sorted!
 By Patrick Forsyth - 55 pages

**Unlock your life.
Bookboon Premium is your key.**

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

Download free eBooks at bookboon.com



Click on the ad to read more

We should therefore stick to the rule that any user developed software should not write anything to these unimplemented locations, since they may be used in future products to invoke new features. All unimplemented addresses in the SFR range (80h through FFh) are considered invalid and writing to or reading from these non-existent register locations may produce undefined values or behaviour.

1.5.2 SFR TYPES

In this section we shall only mention some special SFRs found in C8051F040 version which are appreciably different from the basic 8051 SFR. The standard 8051 SFRs are still available and work in exactly the same way even on this device, and details about these ‘old’ SFRs can be found in an earlier book (Debono, 2013a).

In general, as mentioned in Table 1-3 itself, some SFRs are used to control the operation or the configuration of some aspect of the 8051. For example, TCON and TMOD control the timers while SCON0 controls serial port (UART0) operations.

As already mentioned in section 1.7.2 the basic 8051 SFRs were all fully described in a previous book (Debono, 2013a) and will not be covered again here. Instead we shall mainly deal here with some important new SFRs specific to the C8051F040. For a full description of all the SFRs, such as those dealing with the ADC, DAC, CAN bus etc it would be best to consult the manual/data sheet (Silicon Labs, 2003b).

The most important registers found in nearly all the enhanced 8051 devices are:
System Clock (CLKSEL, OSCICL, OSCICN,OSCXCN) registers
Watch Dog Timer (WDTCN) register
Crossbar registers (XBR0, XBR1, XBR2 and XBR3)
Entended Interrupt Enable and Priority registers (EIE1,EIE2,EIP1,EIP2)

These were explained in (Debono, 2015) and will not be coverered again here, although they are used in the examples given in other chapters and appendices.

Since this device features SFR paging, it allows the device to map many SFR’s into the 0x80 to 0xFF memory address space. Some of the new SFRs not available on the C8051F020 described in (Debono, 2015) are therefore those connected with SFR paging, the most important being SFRPGCN and SFRPAGE. The SFR memory space can have as many as 256 pages. In this way, each memory location from 0x80 to 0xFF can access up to 256 SFR’s. As stated earlier the C8051F04x family of devices utilizes five SFR pages referred to as SFR page 0, 1, 2, 3, and F. SFR pages are selected using the Special Function Register Page Selection register (SFRPAGE) SFR which is present on all pages.

The procedure for reading from and writing to an SFR is as follows:

1. Save the current SFR page (stored in SFRPAGE register) in a temporary location/variable.
2. Select the appropriate SFR page number using the SFRPAGE register.
3. Use direct accessing mode to read from or write to the required SFR (MOV instruction).
4. Restore the original SFR from the temporary variable by writing it back to the SFRPAGE register .

In C programming language, this would be written as:

```
char SFRPAGE_SAVE;                                // temporary variable to store current SFR
                                                // page
SFRPAGE_SAVE = SFRPAGE;                          // Save Current SFR page
SFRPAGE = 1;                                     // Switch to SFR page 1
....                                            // required program code
....                                            // required program code
....                                            // required program code
SFRPAGE = SFRPAGE_SAVE;                           // Restore SFR page when done
```

When an interrupt occurs, the SFR Page Register will automatically switch to the SFR page containing the flag bit that caused the interrupt. The automatic SFR Page switching function conveniently removes the burden of switching SFR pages from the interrupt service routine. Upon execution of the RETI instruction, the SFR page is automatically restored to the SFR Page in use prior to the interrupt. This is accomplished via a three-byte SFR Page Stack and there is therefore no need to store/restore the SFR in software when handling interrupts. This SFR paging technique is used throughout the PaulOS_F040 RTOS program

2 PAULOS_F040 – A CO-OPERATIVE RTOS

The PaulOS (Paul's Operating System) co-operative RTOS, modified for the C8051F040 device is described here. This is the ‘flagship’ RTOS which we regularly use during the year with our students. It is heavily used also for their final year theses and it has been regularly refined to reflect the changes and upgrading requested by the students as they became more and more familiar with the performance and limitations of this co-operative RTOS. In this RTOS, each task is free to run for as long as it wishes. The task itself can control when to give up the processor time to allow other tasks to run.

The original idea for this RTOS came from the book “C and the 8051 – Building Efficient Applications – Volume II” by Thomas W. Schultzⁱ. This RTOS is a direct adaptation of my PaulOS assembly language program, re-written in C so as to make it more versatile and more easily portable to other microcontrollers. The main task of translating it from assembly to C was undertaken years ago as a final year engineering degree thesis (Blaut 2004), then a student under my supervision. It was further developed and improved throughout the years by myself, thanks also to input and suggestions from other students taking my study-units during their degree program, into the version shown here. I consider this RTOS as providing a good basis to the study of a real-time operating system for the 8051.

2.1 DESCRIPTION OF THE RTOS OPERATION

The PaulOS RTOS is a co-operative RTOS and hence each task has to take the initiative to give up its own time so as to allow other tasks to run. It has to be kept in mind that this OS is running on an 8051-based microcontroller which can only run one program at a time and hence this task swapping RTOS only gives the impression of having tasks running simultaneously. In actual fact we can only have one task actually running, and at the time that the RTOS is doing its own checks, no tasks at all would be running. This time ideally should be kept as short as possible.

The operation of the RTOS is as follows:

Each task, when created, would have its own memory area in external memory where there would be stored all the registers (R’s, A, B, DPTR, PSW), stack area (including the return address of the task or function). Once a change of task is required, the RTOS would take care to swap the relevant registers and stack areas so that the microcontroller would have the correct data for the new task in its own internal RAM.

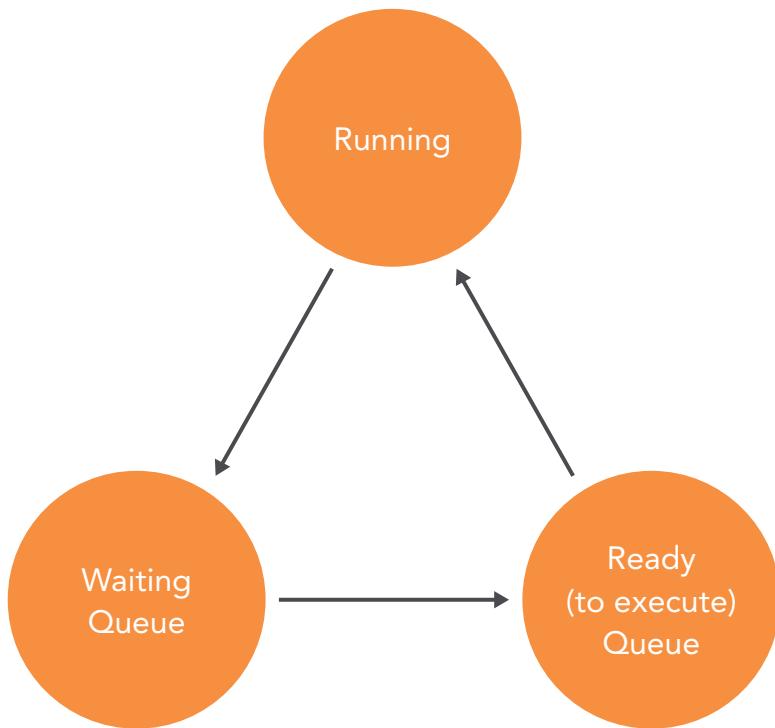


Figure 2-1 RTOS Task states diagram



The RTOS tick-timer can be chosen by the user who can select from the different timers available on the controller. Once set, at every timer overflow, an interrupt call is made to the main RTOS tick timer interrupt service routine. This is the most important routine in the program since at every interrupt the RTOS has to check the status of all the tasks so as to be able to decide whether a task can be moved from the Waiting queue to the Ready to execute queue (see Figure 2-1) or a task swap if the main() was running is required. The RTOS achieves this by counting down the parameter variables holding the individual waiting time required for those tasks in the waiting queue. When anyone of these timeout parameters reaches 0, it means that the time to move on has arrived. Once again, being a co-operative RTOS, the scheduler cannot swap tasks on its own accord. Only the main() code can be forced to give up its time, so that if at any time whilst the main() code is running, there is a task which moves into the Ready to execute queue (or simply the Ready queue), then that task takes over.

On the other hand, when one of the OS commands which forces a task change is encountered in a task, then it is only at that instance that a task swap is implemented. The currently running task is marked as being in the Waiting queue and the first task in the Ready queue takes over, with the stack and registers being conveniently swapped.

The idea behind the PaulOS RTOS is that any task (a function or a routine in a program) can be in any ONE of three states, Running, Waiting (for some event or time delay) or Ready (to execute) state.

RUNNING

A task can be RUNNING, (obviously in the single 8051 environment, there can only be one task which is actually in the running state). If there are no tasks which are ready to execute, then the RTOS will set the main() as the running task. This will be interrupted at any time, as soon as a task becomes ready to run.

WAITING

A task can be in the WAITING (sometimes also referred to as SLEEPING) queue. Here a task could be waiting for any one of the following time delays or events to occur:

- a specified amount of time delay, selected by the user with OS_WAITT command. OS_DEFER command is actually an OS_WAITT(2) – wait for 2 ticks.
- a specified amount of time delay, selected by the user with OS_PERIODIC command. The actual task is placed in the waiting queue when the OS_WAITP (wait for periodic interval) is encountered.
- a specified interrupt to occur within a specified time, selected by the user with the OS_WAITI command.
- a signal from some other task within a specified timeout, selected by the user with the OS_WAITS(timeout) command.
- a signal from some other task indefinitely, selected by the user with the OS_WAITS(0) command.
- a never-ending waiting period. A task could be waiting here indefinitely, effectively behaving as if the task did not exist. This is specified by the OS_KILL_IT command.

READY

It can also be in the READY QUEUE, waiting for its turn to execute. This can be visualised in Figure 2-1 which shows how the tasks can move from one state to another. The RTOS, when permitted to do so, will select the top task from this queue to execute instead of the currently running task, which would then be placed in the waiting queue.

The RTOS itself always resides in the background, and comes into play:

- At every RTOS TIMER interrupt (usually when Timer 2 or Timer 0 overflows, say every one millisecond) so as to update the waiting time left for any tasks.
- At any other interrupt from other timers or external inputs so as to check whether it needs to move to the ready queue any tasks which were waiting for such events or interrupts.
- Whenever an RTOS system command is issued by the main program or tasks, to perform that system command.

The RTOS which is effectively supervising and scheduling all the other tasks, then has to make a decision whether it has to pause the current task and resume a new one or whether it can let the current task run on. There could be various reasons for changing tasks, as explained further on, but in order to do this task swap smoothly, the RTOS has to save all the environment of the presently running task and substitute it with the environment of the next task which is about to run. This is accomplished by saving all the BANK 0

registers, the ACC, B, PSW, and DPTR registers. The STACK too has to be saved since the task might have pushed some data on the stack (apart from the address at the point that the task was interrupted, where it has to return to after the interrupt). This is the crux of the PaulOS RTOS.

2.2 PAULOS_F040.C SYSTEM COMMANDS

We now list and explain all the PaulOS RTOS system commands. These are first listed or grouped according to whether or not they take any parameters. The list is then repeated, this time sorted according to whether the command causes a task swap or not.

The following RTOS system calls do not receive any parameters :

- OS_DEFER (void); // Stops current task and passes control to next task in queue
- OS_KILL_IT (void); // Kills a task - sets it waiting forever
- OS_RUNNING_TASK_ID(void); // Returns the task number of the currently executing task



Ses vi til DSE-Aalborg?
Kom forbi vores stand den
9. og 10. oktober 2019.
Vi giver en is og fortæller
om jobmulighederne hos
os.



- OS_SCHECK (void); // Checks if running task's signal bit is set, returns a bit value // of 1 if signal is already present.
- OS_WAITP (void); // Waits for end of task's periodic interval, set by // the OS_PERIODIC command.

The following RTOS system calls do receive parameters :

- OS_CREATE_TASK (uchar tasknum, uint taskadd); // Creates a task
- OS_INIT_RTOS (uchar iemask); // Initialises all RTOS variables
- OS_PERIODIC (uint ticks); // Tasks run periodically every number of ticks
- OS_RESUME_TASK (uchar tasknum); // Resumes a task which was previously KILLED
- OS_RTOS_GO (uchar prior); // Starts the RTOS with priorities if required
- OS_SIGNAL_TASK (uchar tasknum); // Signals a task
- OS_WAITI (uchar intnum); // Waits for an event (interrupt) to occur
- OS_WAITS (uint ticks); // Waits for a signal within a number of ticks
- OS_WAITT (uint ticks); // Waits for a timeout defined by number of ticks

The list of commands can also be grouped as those which cause a change of task, might cause a change of task and those which do not cause a task swap.

The following RTOS system calls force a task change after executing this command:

- OS_DEFER (void); // Stops current task and passes control to next task in queue
- OS_KILL_IT (void); // Kills a task - sets it waiting forever
- OS_WAITI (uchar intnum); // Waits for an event (interrupt) to occur
- OS_WAITT (uint ticks); // Waits for a timeout defined by number of ticks
- OS_WAITP (void); // Waits for the end of the task's periodic interval

The following RTOS system calls might force a task change after executing this command:

- OS_WAITS (uint ticks); // Waits for a signal within a number of ticks

If the signal is already present when the command is issued, then no task swap is made, otherwise a task change is performed.

The following RTOS system calls do not force a task change, and the task using any of these commands would continue to run after executing the command:

- OS_CREATE_TASK (uchar tasknum, uint taskadd); // Creates a task
- OS_INIT_RTOS (uchar iemask); // Initialises all RTOS variables
- OS_PERIODIC (uint ticks); // run periodically every number of ticks
- OS_RESUME_TASK (uchar tasknum); // Resumes a task which was previously KILLED
- OS_RTOS_GO (uchar prior); // Starts the RTOS with priorities if required
- OS_RUNNING_TASK_ID(void); // Returns the task number of the currently running task
- OS_SCHECK (void); // Checks if running task's signal bit is set
- OS_SIGNAL_TASK (uchar tasknum); // Signals a task

2.3 DESCRIPTIONS OF THE COMMANDS

The detailed description of the commands now follows, which would completely describe the RTOS. The complete files for PaulOS_F040 RTOS source program can be found in Appendix A and examples are given at the end of this chapter which should make it easier to understand.

2.3.1 OS_INIT_RTOS(FAKE PARAMETER)

This system command must be the **first** command to be issued in the main program in order to initialise the RTOS variables. It is called from the main program and takes a one byte parameter which is actually not used in this RTOS. It is simply left here to make it compatible with previous versions of the RTOS. This implies that in order to change the tick timer (and also the interrupt number for the RTOS) we have to change the TICK_TIMER parameter in the PaulOS_F040_Parameters.h file.

This system command performs the following:

- Clears the external memory area which is going to be used to store the stack of each task.
- Sets up the IE register (location A8H in the SFR area).
- Selects edge triggering on the external interrupts. This can be amended if a different triggering is required by changing directly the default initialisation in the RTOS source code listing found in Appendix D or by re-setting the correct triggering mode after having initialised the RTOS so as to override the default value. This is done by setting the correct bit value for IT0 and IT1 residing in the TCON SFR.

The advertisement features a night photograph of the Apollo Hotel 1 building. On the left, there's a red circular logo with a white lightbulb icon. To its right, the text "CISO Conference" is written in large white letters, with "Produced by Inspired" in smaller text below it. On the right side of the image, there's a white rectangular overlay containing the text "Apollo Hotel 1, Groenlandsekade Vinkeveen, Amsterdam, NL" and "Dec 5th 2019". At the bottom of the advertisement, there's a white box with the text "Listen, learn & build relationships with our Network of CISOs & Cyber Security Leaders". To the right of this text is the "Inspired" logo, which includes a blue lightbulb icon and the word "Inspired" in a stylized font. The entire advertisement is framed by a thin black border.

- Loads the Ready Queue with the main idle task number, so that initially only the main task will execute.
- Initialises all tasks as being not waiting for a timeout.
- Sets up the Stack Pointer (SP) variable of each task to point to the correct location in the stack area of the particular task. The stack pointer, initially, is made to point to an offset of 14 bytes above the base of the stack [(MAIN_STACK - 1) + NOOFPUSHES + 2] since NOOFPUSHES in this case is 13. The first 13 locations would initially all contain a zero. This is done so as to ensure that when the first RET instruction is executed after transferring the stack from external RAM on to the 8032 RAM, the SP would be pointing correctly to the address of the task to be started. This is seen in the QSHFT routine, where before the last RET instruction, there is the Pop_Bank0_Reg macro which effectively pops 13 registers. The RET instruction would then read the correct address to jump to from the next 2 locations.

2.3.2 OS_CREATE_TASK(TASK NO:, TASK NAME)

This system command is used in the main program for each task to be created. It takes two parameters, namely the task number (the first task is normally numbered as task 0), and the task address, which in the C environment, would simply be the name of the procedure or function. An example of the syntax used for this command is:

```
OS_CREATE_TASK(0, MotorOn);
```

This would create a task, numbered 0 which would refer to the MorotOn() procedure or function.

This system command performs the following:

- Places the task number in the next available location in Ready Queue, meaning that this task is ready to execute. The location pointer in Ready Queue is referred to as READYQTOP in the program, and is incremented every time this command is issued.
- Loads the address of the start of the task at the bottom of the stack area in external ram allocated to this task. The SP for this task would have been already saved, by the INIT_RTOS command, pointing to an offset 13 bytes above this, to compensate for the pops.

2.3.3 OS_RTOS_GO(PRIORITY)

This system command is used only ONCE in the main program, when the RTOS would be required to start supervising the processes. It takes one Priority bit parameter.

The Priority bit parameter (0 or 1) if set to 1, implies that those tasks placed in the Ready Queue (meaning ready to execute), would be sorted in descending order before the RTOS selects the next task to run. A task number of 0 is taken to mean by the RTOS as the **highest** priority task, and would obviously be given preference during the sorting. The main() task or function is automatically given the highest task number (thus meaning the lowest priority) by the RTOS, so as all the other tasks would be able to interrupt it.

An example of the syntax used for this command is:

```
OS_RTOS_GO(1);
```

This would start the RTOS ticking with priority enabled. The tick time interval is determined by the parameter TICKTIME set in the parameters header file (say 1ms, 5ms or 10ms). This value would then become the basic reference unit for other system commands which use any timeout parameter.

The RTOS would also be required to execute “ready-tasks” sorting prior to any task change, since the priority parameter was set to 1.

Assuming Timer 2 is being used to generate the ticktime this system command performs the following :

- Loads the variable DELAY (LO and HI bytes), with the number of BASIC_TICKS required to obtain the required ticktime delay.
- Sets the PRIORITY bit according to the priority parameter supplied.
- Loads RCAP2H and RCAP2L, the Timer 2 registers, with the required count in order to obtain the required delay between Timer 2 overflow interrupts.

The value used depends on the crystal frequency used on the board. The clock registers count up at one twelfth the clock frequency, and using a clock frequency of 11.0592 MHz, each count would involve a time delay of $12/11.0592 \mu\text{s}$ or $1.08507 \mu\text{s}$. Therefore to get a delay of 1ms ($1000 \mu\text{s}$), $1000/1.08507$ or 921.6 counts would be needed. We would use integer 921 to get this delay, hence the reload registers (RCAP2H,RCAP2L) would be loaded with $65536 - 921$ since the timers count up till they overflow.

- Stores the reference time signal parameter in GOPARAM and TICKCOUNT.
- Starts Timer 2 in 16-bit auto-reload mode.
- Enables interrupts.
- Sets TF2, which is the Timer 2 overflow interrupt flag, thus causing the 1st interrupt immediately.

2.3.4 OS_RUNNING_TASK_ID()

This system command is used by a task to get the number of the task itself. It returns an unsigned character (1 byte) value and the same task continues to run after executing this system command.

An example of the syntax used for this command is:

```
X = OS_RUNNING_TASK_ID(); /* where X would be an unsigned character */
```

The advertisement features a photograph of a man with dark hair and a beard, wearing a white striped shirt, sitting at a desk and holding his head in his hands in a gesture of stress or despair. In the background, there's a window with white horizontal blinds and a laptop on the desk. On the left side of the ad, there's a small thumbnail image of an eBook cover titled "Your Boss: Sorted!" by Patrick Forsyth, described as having 55 pages. The main text on the right side reads: "Max's next Bookboon eBook Your Boss: Sorted! By Patrick Forsyth - 55 pages". Below this, a large blue banner contains the text "Unlock your life. Bookboon Premium is your key." in white and orange. At the bottom of the banner, it says "2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business." and provides the website "bookboon.com".

Download free eBooks at bookboon.com

2.3.5 OS_SCHEK()

This system command is used by a task to test whether there was any signal sent to it by some other task.

- It returns a bit value of:
 - 0 if Signal is not present
 - 1 if Signal is present
- If the signal was present, the signal flag (bit) is also cleared before returning to the calling task. The same task continues to run, irrespective of the returned value.

An example of the syntax used for this command is:

```
X = OS_SCHEK(); /* where X would be a bit-type variable */
```

or one may use it as in the following example to test the presence of the signal bit:

```
if (OS_SCHEK() == 1)
{
/* do these instructions if a signal was present */
}
```

2.3.6 OS_SIGNAL_TASK(TASK NO:)

This system command is used by a task to send a signal to another task. If the other task was already waiting for a signal, then the other task is placed in the Ready Queue and its waiting for signal flag is cleared. The task issuing the OS_SIGNAL_TASK command continues to run, irrespective of whether the called task was waiting or not waiting for the signal. If we need to halt the task after the OS_SIGNAL_TASK command to give way to other tasks, we must use the OS_DEFER() system command after the OS_SIGNAL_TASK command.

This system command performs the following:

- It first checks whether the called task was already waiting for a signal.
- If the called (signaled) task was not waiting, it sets its waiting for signal (SIGW) flag and exits to continue the same task.

- If the signaled task was already waiting, it places the called task in the Ready Queue and it clears both the waiting for signal (SIGW) and the signal present (SIGS) flags.
- It also sets a flag (TINQFLAG) to indicate that a new task has been placed in the Ready Queue. This flag is used by the RTOS_TIMER_INT routine (every half a millisecond) in order to be able to decide whether there has to be a task change. It then exits the routine to continue the same task.

An example of the syntax used for this command is:

```
OS_SIGNAL_TASK(1);          // send a signal to task number 1
OS_DEFER();                 // give cpu time to other tasks, if necessary
```

2.3.7 OS_PERIODIC (UINT TICKS)

This command initialises the task to repeat periodically, every certain number of ticks given as a parameter in the command. It is used at the beginning of a task, OUTSIDE of the endless loop, as shown in the next sub-section 2.3.8. An example of its usage is also given in that same sub-section.

We now deal with the commands that do perform a voluntary (co-operative) change of task:

2.3.8 OS_WAITP (VOID)

This command sets the task waiting for the preset periodic interval (set previously by the OS_PERIODIC(ticks) command. The task goes into a waiting state and the next ready task takes over.

If the interval has already passed when this command is executed, then the task would continue to execute. This is not normally the case, and only happens when there is a programming logic or algorithm mistake, since it would generally mean that the task is actually taking longer to execute than the requested periodic interval between executions.

It performs the following:

- Saves task environment in preparation for the expected task swap.
- If the periodic interval has not yet passed, as is generally the case, it sets the periodic interval flag to indicate that it is waiting for the periodic interval and issues a voluntary task change.
- If however the periodic interval has already elapsed (this is usually due to bad programming, in cases where the code of the task itself takes a longer time to execute than the required periodic interval), then it clears the periodic interval flag and exits.

Such a command is used in a task, in conjunction with the OS_PERIODIC() command and an example of its usage is shown below:

The image shows a woman with long blonde hair, wearing a blue tank top, smiling while wearing a black VR headset. To her right is a red advertisement for MT Højgaard. The ad features the company logo (a stylized 'M' and 'H') and the text 'MT Højgaard'. Below this, in large white capital letters, is 'BEDRE LØSNINGER'. Underneath, there is a block of Danish text: 'I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?' At the bottom of the ad is the website 'mth.dk/vorestilgang'. The background of the ad is orange, and the overall composition is a composite image where the woman's head and shoulders are superimposed onto the advertisement.

```
OS_PERIODIC(50);           // declare task as wishing to execute every
                           // 50 ticks
while(1)                  // repeat forever
{
....                      // code to be executed every 50 ticks
....                      // which should not take longer than
....                      // 50 ticks to execute.
OS_WAITP();               // wait for the periodic interval to pass
}
```

2.3.9 OS_WAITI(Interrupt No:)

This system command is called by a task to sleep and wait for an interrupt to occur. Another task, next in line in the Ready Queue would then take over. If the interrupt never occurs, then the task will effectively sleep for ever. This is one way of writing Interrupt Service Routines under PaulOS RTOS control. ISRs can also be written in such a way as to run independently, as described in section 2.3.15.

If required, this command can be modified to allow another timeout parameter to be passed, so that if the interrupt does not arrive within the specified timeout, the task would resume. A timeout of 0 would still leave the task waiting for the interrupt forever. The modification required to the RTOS source listing would be similar to the OS_WAITS command, and the operation would then be as explained further down in sub-section 2.3.10.

This system command performs the following:

- It sets the bit which corresponds to the interrupt number passed on as a parameter.
- It then calls the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

```
OS_WAITI(0);           // wait for an interrupt from external int 0
```

The task would then go into the sleep or waiting mode and a new task would take over.

2.3.10 OS_WAITS(TIMEOUT)

This system command is called by a task to sleep and wait for a signal to arrive from some other task. If the signal is already present (previously set or signaled by some other task), then the signal is simply cleared and the task continues on. If the signal does not arrive within the specified timeout period, the task resumes just the same. However, a timeout number of 0 would imply that the task has to keep on waiting for a signal indefinitely. If the signal does not arrive, then the task never resumes to run and effectively the task is killed.

This system command performs the following:

- It first checks whether the signal is already present.
- If the signal is present, then it clears the signal flag, exits and continues running.
- If the signal is not present, then:
 - It sets its own waiting for signal (SIGW) flag.
 - It also sets the waiting for timeout variable according to the supplied parameter.
 - It then jumps to the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

```
OS_WAITS(50);
// wait for a signal within 50 units or ticks, the value of the unit depends on
// the TICKTIME parameter used.
```

If for example, the TICKTIME was set to 10 milliseconds in the header file, an OS_WAITS(50) would then imply waiting for a signal to arrive within 500 milliseconds.

or you can use:

```
OS_WAITS(0); // this would wait for a signal for ever
```

In both examples, if the signal is not already present, the task would then go into the sleep or waiting mode and a new task would take over.

2.3.11 OS_WAITT(TIMEOUT)

This system command is called by a task to sleep and wait for a specified timeout period. The timeout period is in units whose value depends on the TICKTIME parameter used. Valid values for the timeout period are in the range 1 to 65535. A value of 0 is reserved for the OS_KILL_IT command, meaning permanent sleep, and therefore is not allowed for this command. The OS_WAITT system command therefore performs the required check on the parameter before accepting the value. If by mistake a value of 0 is given as a timeout parameter, then it is automatically changed to a 1. Once the timeout period passes, the task which had issued this command, would be moved from the waiting to the ready queue.

This system command performs the following:

- If the parameter is 0, then set it to 1, to avoid permanent sleep.
- Save the correct parameter in its correct place in the TTS table.
- Jump to the QSHFT routine in order to start the task next in line.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



An example of the syntax used for this command is:

```
OS_WAITT(60);
// wait for a signal for 60 units, the value of the unit depends on
// the TICKTIME parameter used.
```

If for example, the command TICKTIME was set to 10, the reference unit would be 10 milliseconds, and OS_WAITT(60) would then imply waiting or sleeping for 600 milliseconds. The task would then go into the sleep or waiting mode for 600ms and a new task would take over. After 600ms it would move to the ready queue.

2.3.12 OS_KILL_IT()

This system command is used by a task in order to stop or terminate the task. As explained earlier in OS_WAITT, this is simply the command OS_WAITT with an allowed timeout of 0. The task is then placed permanently waiting and never resumes execution.

This system command performs the following:

- First it clears any waiting for signal or waiting for interrupt flags, so that that task would definitely never restart.
- Then it sets its timeout period in the TTS table to 0, which is the magic number the RTOS uses to define any non-timing task.
- Then it sets the INTVLRLD and INTVLCNT to 0, again implying that it is not a periodic task.
- Finally it jumps to the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

```
OS_KILL_IT();
/* the task simply stops to execute and a new task would take over.*/
```

2.3.13 OS_DEFER()

This system command is used by a task in order to hand over processor time to another task. The task is simply placed at the end of the Ready Queue, while a new task resumes execution.

This system command performs the following:

- It sets its timeout period in the TTS table to 0, which is the magic number the RTOS uses to describe any non-timing task.
- It places the task in the Ready Queue, by simply placing the task number in the next available location in Ready Queue area.
- It then flows on to the QSHFT routine in order to start the task next in line.

An example of the syntax used for this command is:

```
OS_DEFER();
/* the task simply stops execution and is placed in Ready Queue.*/
/* A new task would then take over. */
```

2.3.14 ENHANCED EVENT-WAITING AND OTHER ADD-ON MACROS

These macros (#define statements) perform the same functions of the OS_WAITT, OS_WAITS and OS_PERIODIC calls but rather than ticks they accept absolute time values as parameters in terms of minutes, seconds and millisecs. This difference is denoted by the _A suffix (the A standing for Absolute) - eg. OS_WAITT_A(0,0,300) would cause a task to wait for 300ms and is the absolute-time version of OS_WAITT(x), where x would have to be calculated to give the required number of ticks equivalent to a 300ms delay.

Range of values (65535 TICKTIMES) accepted is listed below:

Using a minimum TICKTIME of 1ms :

Range from 1ms to 1m, 5s, 535ms in steps of 1ms.

Using a recommended TICKTIME of 10ms :

Range from 10ms to 10m, 55s, 350ms in steps of 10ms.

Using a maximum TICKTIME of 50 ms :

Range from 50ms - 54m, 36s, 750ms in steps of 50ms

If the conversion from absolute time to ticks results in 0 (all parameters being 0 or overflow) this result is only accepted by OS_WAITT() by virtue of how the OS_WAITT(), OS_WAITS() and OS_PERIODIC() calls were written. In the case of the OS_WAITT() and OS_PERIODIC() calls the tick count would automatically be changed to 1 meaning an interval of 1 ticktime.

OS_WAITT_A(M,S,ms)	// Absolute OS_WAITT for minutes, seconds and milliseconds
OS_WAITS_A(M,S,ms)	// Absolute OS_WAITS for minutes, seconds and milliseconds
OS_PERIODIC_A(M,S,ms)	// Absolute OS_PERIODIC for minutes, seconds and milliseconds
OS_PAUSE_RTOS()	// Disable the RTOS ticktimer interrupts, used in a stand-alone ISR
OS_RESUME_RTOS()	// Re-enable the RTOS ticktimer interrupts, used in a stand-alone ISR



**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

 **Inspired**

```
OS_CPU_IDLE()           // Sets the µC in idle mode in PCON SFR (section
                        // 1.8.10).
                        // This is usually used in the main program endless loop
                        // after
                        // initialising and starting the RTOS.
OS_CPU_DOWN()          // Sets the µC in power-down mode in PCON SFR
                        // (section 1.8.10).
```

2.3.15 STAND-ALONE INTERRUPT SERVICE ROUTINES

In the C version of the RTOS, a simple method of having one or more stand-alone interrupt service routine (ISR) which would run whenever some interrupt is generated has been included.

All we have to do is to set to '1' the correspond interrupt in the PaulOS_F040_Parameters.H file 2.4. For example if we intend to have an ISR running under the EXT 0 interrupt (and not under RTOS control), then we have to make sure to set to one the corresponding #define statement in PaulOS.H file. In some examples or listings shown in this book these STAND_ALONE_ISR parameters were moved to the parameters.h header file so as to have all parameters which can be changed by the user in one file, but the effect is the same.

```
#define STAND_ALONE_ISR_00 1 // EXT0 - set to 1 if using this interrupt as a stand alone ISR
```

Then in the ISR itself we should also include the commands OS_PAUSE_RTOS() when starting the ISR and then OS_RESUME_RTOS() in order to resume the RTOS before exiting the ISR. This would ensure that the RTOS does not interfere with the stand-alone ISR.

It is best to use register banks 2 or 3 for these ISRs.

Example of a stand-alone ISR, interrupting the RTOS and executing immediately when the interrupt occurs.

```
void ISR_EXT0 (void) interrupt 0 using 2
{
    OS_PAUSE_RTOS()           // Disable the RTOS, used in a stand-alone ISR
    /* Our service routine code goes in here */
    /* Our service routine code goes in here */
    /* Our service routine code goes in here */
    OS_RESUME_RTOS()          // Re-enable the RTOS, before exiting the stand-
                                // alone ISR
}
```

2.4 PAULOS_F040 PARAMETERS HEADER FILE

This is the RTOS parameters header file. We would mainly be needing to set the TICK_TIMER, TICKTIME and NOOFTAKS parameters to reflect our particular application program.



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

```
/*
*****RTOS KERNEL HEADER FILE
*****
* PaulOS_F040_Parameters.H
*
* For use with PaulOS_F040.C - Co-Operative
* RTOS written in C based on PaulOS
* by Ing. Paul P. Debono
* for use with the C8051F040 family of microcontrollers
*
* File      : PaulOS_F040_Parameters.H
* Revision  : 10
* Date      : Revised for C8051F040 February 2015
* By        : Paul P. Debono
*
*          B. Eng. (Hons.) Elec.
*          University Of Malta
*
***** */
#endif __PARAMETERS_H__



/*
*****DATA TYPE DEFINITIONS
*****
*/
/*
***** */
*/
```

```
#define TICK_TIMER 4

#define TICKTIME      1      // Length of RTOS basic tick in msec
                           // refer to the RTOS timing definitions
                           // suitable values are: 1, 2,
                           4, 5, 8, 10, 20, 25

#define NOOFTASKS     2      // Number of tasks used in application

#define STACKSIZE 0x0F // Number of bytes to allocate for the stack
                     // Usually there is no need to
                     change this parameter

/*
*****
*/
#define STAND_ALONE_ISR_00 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_01 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_02 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_03 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_04 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_05 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_06 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_07 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_08 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_09 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_10 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_11 0 // set to 1 if using
```

```
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_12 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_14 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_15 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_16 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_17 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_18 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_19 0 // set to 1 if using
this interrupt as a stand alone ISR
#define STAND_ALONE_ISR_20 0 // set to 1 if using
this interrupt as a stand alone ISR

/*
*****
*/
#endif
```

2.5 EXAMPLE USING PAULOS_F040 RTOS

This is an example using the PaulOS_F040 RTOS. It displays a clock via UART0 and it also blinks the LED every second.

The files associated with this project are shown highlighted in the red rectangle in Figure 22. The files PaulOS_F040.h, PaulOS_F040.c although part of the clock project, are not being included here since they are listed in Appendix A.

```

8 // 
9 // This program flashes the green LED on the C8051F040 target board
10 // One task switches on the LED
11 // Another task displays the clock using UART0
12 // Target: C8051F04x
13 //
14 //
15 //
16 //-----
17 // Includes
18 //-----
19 #include "C8051F040.h"      /* special function registers 8052 */
20 #include "PaulOS_F040.h"    /* PaulOS_F040 system calls definitions */
21 #include "UART0_T1.h"
22 #include "F04x_Osc_Init.h"
23 #include <stdio.h>
24 #include <stdlib.h>
25 //
26 //-----
27 // Global CONSTANTS
28 //-----
29

```

Build Output

```

compiling UART0_T1.c...
UART0_T1.c(48): message C320: "Using Timer 1 as the baud rate generator for UART0"
compiling PaulOS_F040.C...
PaulOS_F040.C(142): message C320: "Using Timer 4 as the Tick Timer"
assembling PaulOS_F040_Startup.A51...
compiling F04x_Osc_Init.c...

```

Figure 2-2 Blinking LED example

MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang

2.5.1 BLINKY2.C

```
-----  
// Blinky2.c  
-----  
// Copyright (C) 2005  
//  
// AUTH: PD  
// DATE: 21 FEB 05  
//  
// This program flashes the green LED on the C8051F040 target board  
// One task switches on LED P1.6 on the C8051F040-TB  
// Another task displays the clock using the serial port UART0  
// Target: C8051F04x  
//  
//-----  
// Includes  
//-----  
#include "C8051F040.h"      /* special function registers 8052 */  
#include "Paulos_F040.h"    /* Paulos_F040 system calls definitions */  
#include "UART0_T1.h"  
#include "F04x_Osc_Init.h"  
#include <stdio.h>  
#include <stdlib.h>  
//-----  
// Global CONSTANTS  
//-----  
sbit LED = P1^6;           // green LED: '1' = ON; '0' = OFF  
  
struct time {                /* structure of the time record */  
    unsigned char hour;       /* hour */  
    unsigned char min;        /* minute */  
    unsigned char sec;        /* second */  
};  
  
struct time ctime = { 17, 45, 0 }; /* storage for clock time  
values */  
//-----
```

```
// Function PROTOTYPES
-----
void PORT_Init (void);
void DISABLE_Watchdog (void);

-----
// DISABLE_Watchdog
// -----
//
// Disables the watchdog timer
//
void DISABLE_Watchdog (void)
{
    char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = CONFIG_PAGE;          // Switch to
                                    configuration page
    EA = 0;
    WDTCN = 0xDE;
    WDTCN = 0xAD;
    EA = 1;
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

-----
// PORT_Init
// -----
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = CONFIG_PAGE;          // Switch to configuration page
    XBR0 = 0x04;                     // Enable UART 0
    XBR1 = 0x00;
    XBR2 = 0x40;                     // Enable crossbar and
                                    weak pull-ups
```

```

P0MDOUT |= 0x01;                      // enable TX0 as a push-pull output
P1MDOUT |= 0x40;                      // enable P1.6 (LED) as
                                         push-pull output
SFRPAGE = SFRPAGE_SAVE;                // Restore SFR page
}

/*
*****
*****
*/
/*          Task 0 'Blink'          */
void BlinkTask (void)
{
    OS_PERIODIC_A(0,0,500);           /* Repeat every 500 ms */
while(1)
{
    LED = ~LED;
    OS_WAITP();
}
}

/*
*****          Task 1 'clock'          */
void clock (void)
{
    static char SFRPAGE_SAVE;
    OS_PERIODIC_A(0,1,0);           /* Repeat every 1 second */

while (1) {                                /* clock is an endless loop */
    if (++ctime.sec == 60) {                 /* calculate the second */
        ctime.sec = 0;
        if (++ctime.min == 60) {            /* calculate the minute */
            ctime.min = 0;
            if (++ctime.hour == 24) {      /* calculate the hour */
                ctime.hour = 0;
            }
        }
    }
}
}

```

```
}

SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
SFRPAGE = UART0_PAGE;
/* display time */
printf ("Clock Time: %02bu:%02bu:%02bu\
r", ctime.hour, ctime.min, ctime.sec);
SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
OS_WAITP();                      /* wait for the remaining of the
1 sec periodic time */

}

//-----
// MAIN Routine
//-----
void main (void) {

    DISABLE_Watchdog ();
    SYSCLK_Crystal_Init ();
    PORT_Init ();

    UART0_Init (115200);
    OS_INIT_RTOS(TICK_TIMER);      /* initialise RTOS (Timer
                                    2 interrupt), */
                                    /* variables and stack */

    OS_CREATE_TASK(0, BlinkTask);   /* CREATE tasks
    OS_CREATE_TASK(1, clock);

    OS_RTOS_GO(0);                /* Start multitasking */

while (1)
{
    OS_CPU_IDLE();               /* Go to idle mode if doing nothing,
                                    to conserve energy */
}

/*
***** *****
*/
```

2.5.2 F04X_OSC_INIT.H

```
//-----  
// Function PROTOTYPES  
// F04x_Osc_Init.h  
//-----  
  
#ifndef      SYSCLK_H  
#define      _SYSCLK_H  
  
void SYSCLK_IntOsc_Init (void);  
void SYSCLK_Crystal_Init (void);  
void SYSCLK_C_RC_Init (void);  
void SYSCLK_CMOS_Init (void);  
  
#endif
```

2.5.3 F04X_OSC_INIT.C

```
//-----  
// F04x_Osc_Init.c  
//-----  
// Copyright 2003 Cygnal Integrated Products, Inc.  
//  
// AUTH: FB  
// DATE: 27 DEC 02  
//  
// This program shows an example of configuring the internal  
// and external oscillators.  
//  
// Target: C8051F04x  
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51  
//  
//-----  
// Includes  
//-----  
#include "c8051f040.h" // SFR declarations  
#include "F04x_Osc_Init.h"  
  
void SYSCLK_IntOsc_Init (void)  
{  
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page  
    SFRPAGE = CONFIG_PAGE; // Set SFR Page  
    OSCICN = 0x83; // Set internal oscillator to  
    // maximum frequency  
    CLKSEL = 0x00; // Select internal oscillator as  
    // SYSCLK source  
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page  
}  
  
//-----  
// SYSCLK_Crystal_Init  
//-----  
//
```

```
// This routine initializes the system clock
// to use an 22.1184MHz crystal
// as its clock source. Assumes a 22.1184 MHz
// crystal and associated loading
// capacitors are connected at XTAL1 and XTAL2.
// void SYSCLK_Crystal_Init (void)
{
    int i; // delay counter
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    SFRPAGE = CONFIG_PAGE; // Set SFR Page
    OSCXCN = 0x67; // start external oscillator with
    // 22.1184MHz crystal (XFCN = 7)
    //OSCXCN = 0x77; // start external oscillator with
    // 22.1184MHz crystal (XFCN = 7) divided by 2 = 11.592MHz
    for (i=0; i < 256; i++) ; // XTLVLD blanking interval (>1ms)
    while (!(OSCXCN & 0x80)) ; // Wait for crystal osc. to settle
    SFRPAGE = LEGACY_PAGE;
    RSTSRC = 0x04; // enable missing clock detector
    SFRPAGE = CONFIG_PAGE;
    CLKSEL = 0x01; // select external oscillator as SYSCLK source
    OSCICN = 0x00; // disable internal oscillator
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}
-----
// SYSCLK_C_RC_Init
-----
//
// This routine initializes the system clock
// to use an external RC network
// or a single capacitor as its clock source.
// Assumes an RC network is
// connected to XTAL1 or a single capacitor is connected to XTAL1 and
// XTAL2.
//
void SYSCLK_C_RC_Init (void)
{
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    SFRPAGE = CONFIG_PAGE; // Set SFR Page
```

```
OSCXCN = 0x47; // start external oscillator
in C/RC mode with XFCN = 7
SFRPAGE = LEGACY_PAGE;
RSTSRC = 0x04; // enable missing clock detector
SFRPAGE = CONFIG_PAGE;
CLKSEL = 0x01; // select external oscillator as SYSCLK source
OSCICN = 0x00; // disable internal oscillator
SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}
//-----
// SYSCLK_CMOS_Init
//-----
//
// This routine initializes the system clock
// to the external oscillator in
// CMOS clock mode. Assumes a CMOS clock
// generator is connected to XTAL1.
//
void SYSCLK_CMOS_Init (void)
{
char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
SFRPAGE = CONFIG_PAGE; // Set SFR Page
OSCXCN = 0x20; // start external oscillator in CMOS clock mode.
SFRPAGE = LEGACY_PAGE;
RSTSRC = 0x04; // enable missing clock detector
SFRPAGE = CONFIG_PAGE;
CLKSEL = 0x01; // select external oscillator as SYSCLK source
OSCICN = 0x00; // disable internal oscillator
SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}
```

```

//-----
// This routine initializes the UART for
// operation using Timer1 overflows as
// its UART bit clock. The baud rate is specified
// by the passed parameter baudrate
//
void UART0_Init (unsigned long baudrate)
{
    #message "Using Timer 1 as the baud rate generator for UART0"
    char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    //UART0 on SFR page 0
    SFRPAGE = UART0_PAGE;
    SCON0 = 0x50; // UART mode 1, 8-bit UART, enable RX
    SSTA0 = 0x10; // Enable baudrate doubler,
    // Timer1 RX and TX baudrate source

    TI0 = 1;

    //Timer1 SFR page is the same
    SFRPAGE = TIMER01_PAGE;
    TMOD = 0x20; // TMOD: timer 1, mode 2, 8-bit reload
    TH1 = -(SYSCLK/baudrate/16); // Set Timer1 reload value for baudrate
    TL1 = TH1; // Set Timer1 initial value
    CKCON |= 0x10; // Timer1 uses SYSCLK as time base
    TR1 = 1; // Start Timer1

    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

/*
-----*
The following _getkey function replaces the one in the library.
-----*/
char _getkey (void)
{
    static char c;
    static char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page

    SFRPAGE = UART0_PAGE;
}

```

```
while (!RI0); // Wait for a character
c = SBUFO;
RI0 = 0;

SFRPAGE = SFRPAGE_SAVE; // Restore SFR page

return (c);
}

/*
The following putchar function replaces the
one in the standard library.
*/
char putchar (char c)
{
    static char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page

    SFRPAGE = UART0_PAGE;
    while (!TI0); // Wait for the Tx to be ready
    TI0 = 0;
    SBUFO = c; // Start transmission

    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page

    return (c);
}

//End code-----
```

2.5.4 UART0_T1.C

```
-----  
//-----  
//  UART0_T1.c  
//-----  
// Copyright 2002 Cygnal Integrated Products, Inc.  
//  
// DATE: AUG 02  
//  
// UART0 Demonstration  
//  
// Target: C8051F04x  
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51  
//  
// Note the system clock frequency and baud rate used.  
  
// Also note that in order to use the printf function,  
// you must ensure you are on the correct SFR page.  
  
-----  
// Includes  
-----  
  
#include "C8051F040.h"// SFR declarations  
#include "UART0_T1.h"  
#include <stdio.h>  
  
-----  
// Global CONSTANTS  
-----  
  
-----  
// Global VARIABLES  
-----  
  
-----  
// Initialization Subroutines  
-----  
  
-----  
// UART_Init
```

```

//-----
// This routine initializes the UART for
// operation using Timer1 overflows as
// its UART bit clock. The baud rate is specified
// by the passed parameter baudrate
//
void UART0_Init (unsigned long baudrate)
{
    #message "Using Timer 1 as the baud rate generator for UART0"
    char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    //UART0 on SFR page 0
    SFRPAGE = UART0_PAGE;
    SCON0 = 0x50; // UART mode 1, 8-bit UART, enable RX
    SSTA0 = 0x10; // Enable baudrate doubler,
    // Timer1 RX and TX baudrate source

    TI0 = 1;

    //Timer1 SFR page is the same
    SFRPAGE = TIMER01_PAGE;
    TMOD = 0x20;           // TMOD: timer 1, mode 2, 8-bit reload
    TH1 = -(SYSCLK/baudrate/16); // Set Timer1 reload value for baudrate
    TL1 = TH1;             // Set Timer1 initial value
    CKCON |= 0x10;         // Timer1 uses SYSCLK as time base
    TR1 = 1;               // Start Timer1

    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

/*
-----*
The following _getkey function replaces the one in the library.
-----*/
char _getkey (void)
{
    static char c;
    static char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page

    SFRPAGE = UART0_PAGE;
}

```

```
while (!RI0); // Wait for a character
c = SBUF0;
RI0 = 0;

SFRPAGE = SFRPAGE_SAVE; // Restore SFR page

return (c);
}

/*
The following putchar function replaces the
one in the standard library.
*/
char putchar (char c)
{
    static char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page

    SFRPAGE = UART0_PAGE;
    while (!TI0); // Wait for the Tx to be ready
    TI0 = 0;
    SBUF0 = c; // Start transmission

    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page

    return (c);
}

//End code-----
```

2.5.5 UART0_T1.H

```
/*-----  
  
UART0_T1.h  
  
-----*/  
  
#ifndef      UART0_H  
#define      UART0_H  
  
#define SYSCLK (22118400UL)  
  
char putchar (char c);  
char getkey (void);  
  
void UART0_Init (unsigned long baudrate);  
  
#endif  
=====
```

3 MASTER-SLAVE CAN BUS RTOS

3.1 MULTI-CONTROLLER RTOSS

We now present a modification to the RTOS so that it can function in a CAN bus networked environment. The idea here is to have two C8051F040 boards (one acting as the master the other as the slave), each running its own RTOS. The master RTOS uses Timer 2 as the source of the RTOS ticks whilst the slave RTOS tick is triggered by an interrupt from the CAN bus whenever a message is received. At each timer 2 tick, the master sends a message via the CAN bus to the slave so as the trigger the slave RTOS (which is depends on a CAN bus interrupt). This simple method synchronises the two boards.

3.2 CAN BUS RTOS EXAMPLE

This example implements the master-slave RTOS described above. Each board flashes its led and transmits the time HH:MM:SS via the serial port. In the project, two targets are set up one for the master board and the other for the slave, as highlighted in red rectangle in Figure 3-1.

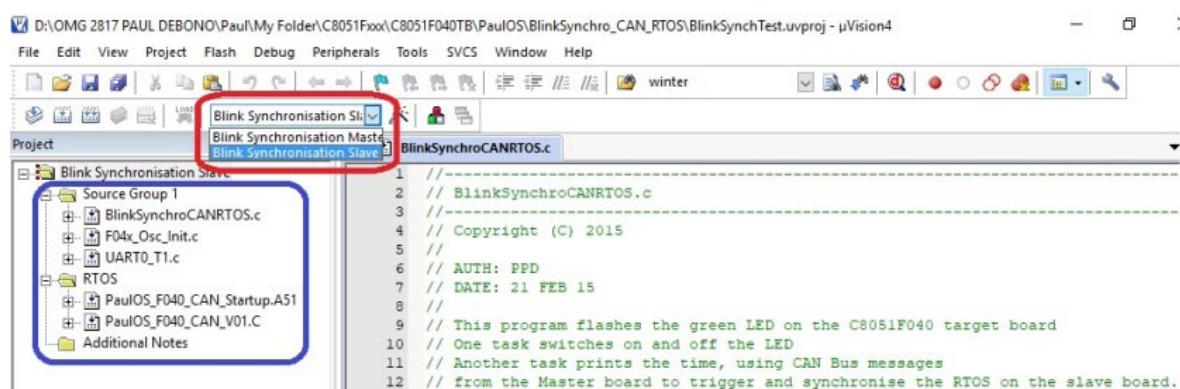


Figure 3-1 Screen shot with the CAN bus example

The programs associated with this project are highlighted in the blue rectangle in Figure 3-0-1 and only the new programs associated with the CAN bus RTOS are being listed in chapter. The others F04x_Osc_Init.c and UART0_T1.c were already used and listed in previous chapters. When compiling the program for the master board, the option MASTER (see Figure 3-2) should be defined in the C51 tab of the target option and the option SLAVE (see Figure 3-3) should be defined for the slave target options.

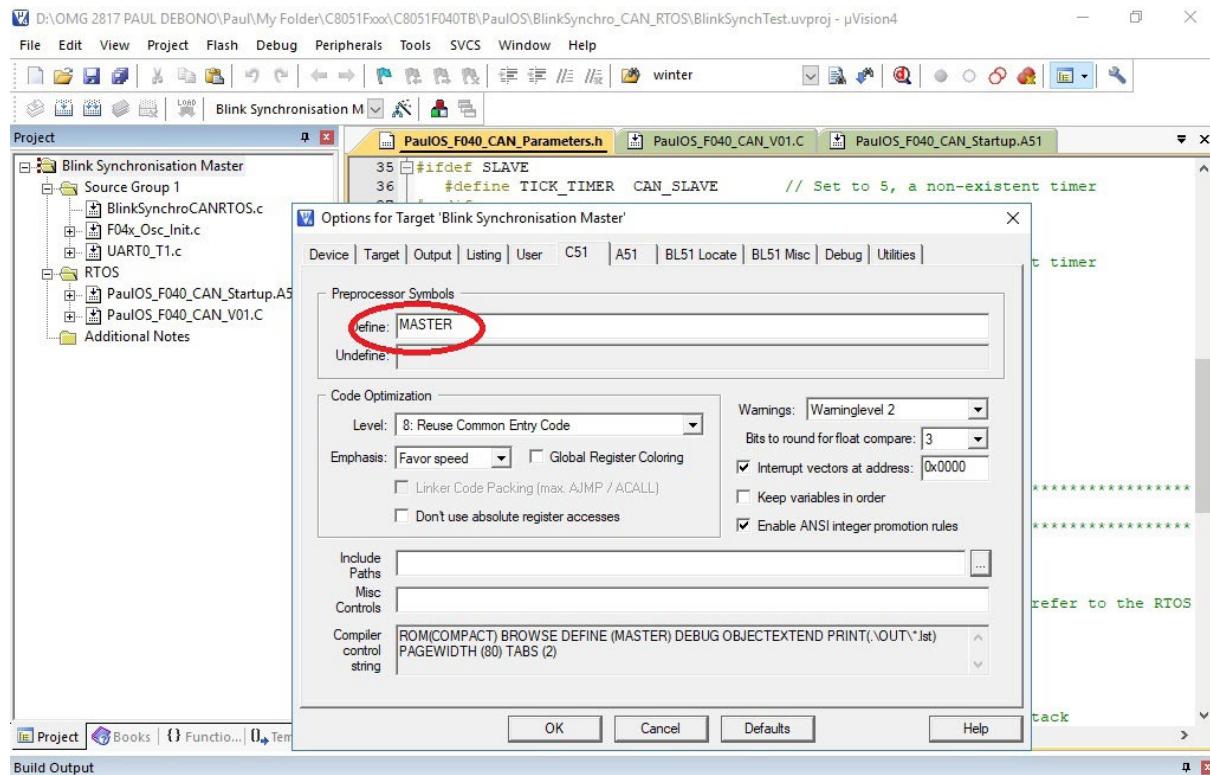


Figure 3-2 Screen shot showing the MASTER option



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



This will ensure that when compiling for the master board or for the slave, the correct program will be compiled since it uses some conditional compilation, which is a section of program which is only compiled if a certain definition is present, such as:

```
#ifdef MASTER
    #define TICK_TIMER CAN_MASTER           // Set to 6, a non-
                                              existent timer
#endif
...
.....
#elif (TICK_TIMER == CAN_MASTER)
    #message "Using Timer 2 as the Tick Timer for the Master"
    #message "Using the CAN bus interrupt as
the Tick Timer on the Slave board"
    SFRPAGE = TMR2_PAGE;
    IE &= 0x7F;
```

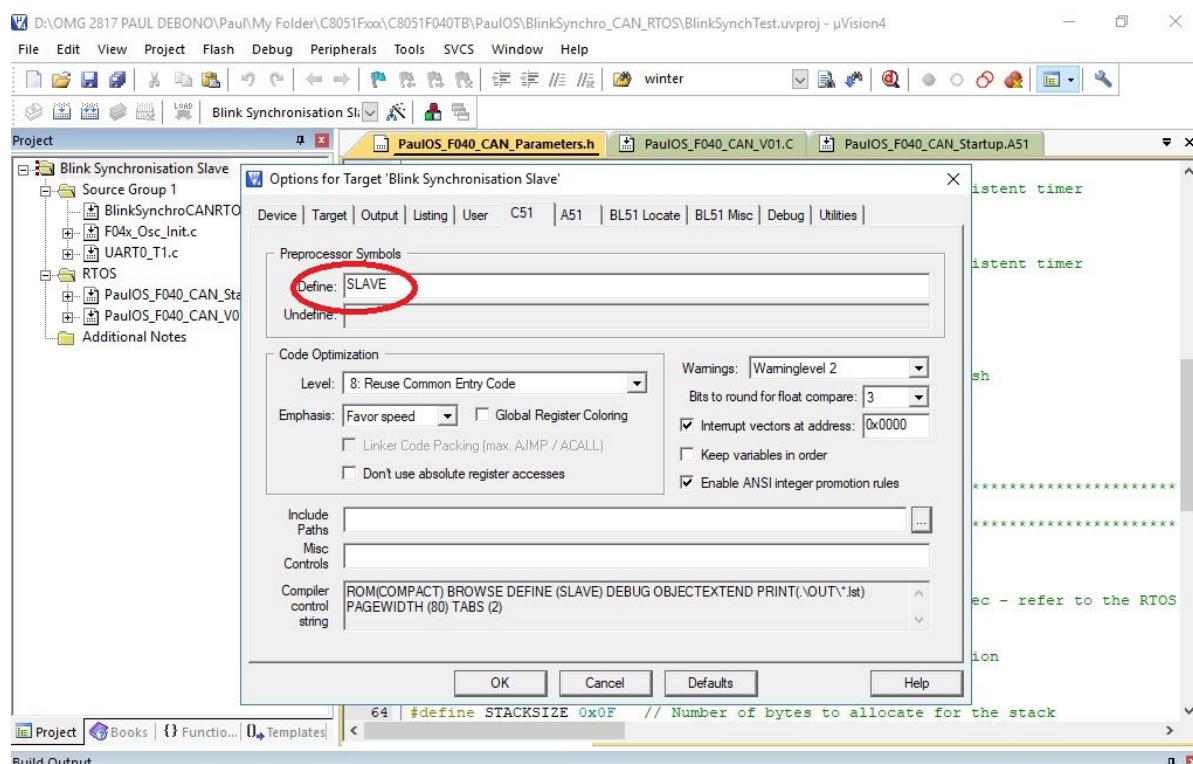


Figure 3-3 Screen showing the SLAVE option

3.2.1 BLINKSYCHROCANRTOS.C

```
-----  
// BlinkSynchroCANRTOS.c  
-----  
  
// Copyright (C) 2015  
//  
// AUTH: PPD  
// DATE: 21 FEB 15  
//  
// This program flashes the green LED on the C8051F040 target board  
// One task switches on and off the LED  
// Another task prints the time, using CAN bus messages  
// from the Master board to trigger and synchronise  
// the RTOS on the slave board.  
// Target: C8051F040  
// In order to synchronise the 2 boards:  
//     First switch on the Slave board (or reset  
//     it) with the Master board OFF  
//     then switch on the Master board.  
//  
//     or  
//  
//     Press and hold both resets (on the 2  
//     boards), then first release the Slave  
//     reset button and then release the Master reset button.  
//  
  
-----  
// Includes  
-----  
  
#include "C8051F040.h"      /* special function registers F040 */  
#include "UART0_T1.h"  
#include "F04x_Osc_Init.h"  
#include "PaulOS_F040_CAN_V01.h" /* PaulOS system calls definitions */  
#include <stdio.h>  
#include <stdlib.h>  
  
-----  
// Global CONSTANTS  
-----
```

```
sbit LED = P1^6;                                // green LED: '1' = ON; '0' = OFF

struct time {                                     /* structure of the time record      */
    unsigned char hour;                          /* hour                               */
    unsigned char min;                           /* minute                            */
    unsigned char sec;                           /* second                            */
};

struct time ctime = { 12, 0, 0 }; /* hh,mm,ss
storage for clock time values */

enum Tasks { Task0, Task1, Task2, Task3 };

//-----
// Function PROTOTYPES
//-----
void PORT_Init (void);
void DISABLE_Watchdog (void);

//-----
// DISABLE_Watchdog
//-----
//
// Disables the watchdog timer
//
void DISABLE_Watchdog (void)
{
    EA = 0;
    WDTCN = 0xDE;
    WDTCN = 0xAD;
    EA = 1;
}

//-----
// PORT_Init
//-----
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
```

```

{
    static char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = CONFIG_PAGE;          // set SFR page
    XBR0 = 0x04;                   // Enable UART 0
    XBR1 = 0x00;
    XBR2 = 0x40;                   // Enable crossbar and
                                   weak pull-ups
    XBR3 = 0x80;                   // Configure CAN TX pin (CTX) as
                                   push-pull digital output
    P0MDOUT |= 0x01;              // enable TX0 as a push-pull output
    P1MDOUT |= 0x40;              // enable P1.6 (LED) as
                                   push-pull output
    SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
}
/*
*****
*/
/* Task 0 'Blink ON/OFF' */
void BlinkTask (void)
{
    OS_PERIODIC_A(0,0,500); /* Repeat every 500 ms */
    while(1)
    {
        LED = ~LED;
        OS_WAITP();
    }
}

/*
*/
/* Task 1 'clock' - Display clock via serial port */
void ClockTask (void)
{
    static char SFRPAGE_SAVE;

    OS_PERIODIC_A(0,1,0);          /* Repeat every 1 second */
}

```

```
while (1) {                                /* clock is an endless loop      */
    if (++ctime.sec == 60) {                  /* calculate the second        */
        ctime.sec = 0;
        if (++ctime.min == 60) {              /* calculate the minute       */
            ctime.min = 0;
            if (++ctime.hour == 24) {        /* calculate the hour         */
                ctime.hour = 0;
            }
        }
    }
}

SFRPAGE_SAVE = SFRPAGE;                    // Save Current SFR page
SFRPAGE = UART0_PAGE;                     // printf uses the UART

#if (TICK_TIMER == CAN_MASTER)
    printf ("Master Clock Time: %02bu:%02bu:%02bu
            ctime.hour, ctime.min, ctime.sec); \r", /* display
                                                       time */
#elif (TICK_TIMER == CAN_SLAVE)
    printf ("Slave Clock Time: %02bu:%02bu:%02bu
            ctime.hour, ctime.min, ctime.sec); \r", /* display
                                                       time */
#endif

SFRPAGE = SFRPAGE_SAVE;                   // Restore SFR page

OS_WAITP();                                /* wait for 1 second */
}

//-----
// MAIN Routine
//-----
void main (void) {

    DISABLE_Watchdog ();
    SYSCLK_Crystal_Init ();

    UART0_Init (115200);
    PORT_Init ();
}
```

```
OS_INIT_RTOS(TICK_TIMER);           /* initialise RTOS */  
/* variables and stack */  
  
OS_CREATE_TASK(Task0, BlinkTask);    /* CREATE task */  
OS_CREATE_TASK(Task1, ClockTask);   /* CREATE task */  
  
LED = 0;  
OS_RTOS_GO(0);                     /* Start multitasking */  
  
while (1)  
{  
    OS_CPU_IDLE(); /* Go to idle mode if  
                      doing nothing, to conserve energy */  
}  
  
}
```



Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders

 **Inspired**

3.2.2 PAULOS_F040_CAN_STARTUP.A51

```
; PaulOS_F040_CAN_Startup.A51
$NOMOD51
;-----
; This file is part of the C51 Compiler package
; Copyright (c) 1988-2002 Keil Elektronik
; GmbH and Keil Software, Inc.
;-----
; STARTUP.A51: This code is executed after processor reset.
;
; To translate this file use A51 with the following invocation:
;
;     A51 STARTUP.A51
;
; To link the modified STARTUP.OBJ file to your
; application use the following
; BL51 invocation:
;
;     BL51 <your object file list>, STARTUP.OBJ <controls>
;
;-----
;
; User-defined Power-On Initialization of Memory
;
; With the following EQU statements the initialization of memory
; at processor reset can be defined:
;
;           ; the absolute start-address of
;           ; IDATA memory is always 0
IDATALEN      EQU 100H      ;the
length of IDATA memory in bytes.
;
XDATASTART    EQU 0H        ;the absolute start-
                           ; address of XDATA memory
XDATALEN      EQU 4096     ;the length of XDATA
                           ;memory in bytes.
;
PDATASTART    EQU 0H        ;the absolute start-
                           ; address of PDATA memory
PDATALEN      EQU 0H        ;the length of PDATA
                           ;memory in bytes.
;
```

```
; Notes: The IDATA space overlaps physically
;         the DATA and BIT areas of the
;         8051 CPU. At minimum the memory space occupied from the C51
;         run-time routines must be set to zero.
;-----
;
; Reentrant Stack Initialization
;
; The following EQU statements define the stack pointer for reentrant
; functions and initialized it:
;
; Stack Space for reentrant functions in the SMALL model.
IBPSTACK      EQU 0          ; set to 1 if small reentrant is used.
IBPSTACKTOP   EQU OFFH+1    ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the LARGE model.
XBPSTACK      EQU 0          ; set to 1 if large reentrant is used.
XBPSTACKTOP   EQU OFFFFH+1  ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the COMPACT model.
PBPSTACK      EQU 0          ; set to 1 if compact reentrant is used.
PBPSTACKTOP   EQU 0FFFFH+1  ; set top of stack to
                           ; highest location+1.
;
;-----
;
; Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
; The following EQU statements define the xdata page used for pdata
; variables. The EQU PPAGE must conform with the PPAGE control used
; in the linker invocation.
;
PPAGEENABLE   EQU 0          ; set to 1 if pdata object are used.
;
PPAGE         EQU 0          ; define PPAGE number.
;
PPAGE_SFR     DATA 0A0H    ; SFR that supplies
                           ; uppermost address byte
;                           (most 8051 variants use P2 as uppermost address byte)
;
```

```
; -----  
; Standard SFR Symbols  
ACC      DATA    0E0H  
B        DATA    0F0H  
SP       DATA    81H  
DPL      DATA    82H  
DPH      DATA    83H  
  
NAME  ?C_STARTUP  
  
?C_C51STARTUP      SEGMENT   CODE  
?STACK              SEGMENT   IDATA  
  
#include "Paulos_F040_CAN_Parameters.h"  
  
MAINSTACK:          RSEG      ?STACK  
                    DS        STACKSIZE  
  
EXTRN CODE (?C_START)  
PUBLIC ?C_STARTUP  
PUBLIC MAINSTACK  
  
?C_STARTUP:         CSEG      AT      0  
                    LJMP     STARTUP1  
  
RSEG      ?C_C51STARTUP  
  
STARTUP1:  
IF IDATALEN <> 0  
    MOV      R0, #IDATALEN - 1  
    CLR      A  
IDATALOOP:          MOV      @R0,A  
                    DJNZ    R0, IDATALOOP  
  
ENDIF  
IF XDATALEN <> 0  
    MOV      DPTR, #XDATASTART  
    MOV      R7, #LOW (XDATALEN)  
    IF (LOW (XDATALEN)) <> 0  
        MOV      R6, #(HIGH (XDATALEN)) +1  
    ELSE  
        MOV      R6, #HIGH (XDATALEN)
```

```
ENDIF

        CLR      A
XDATALOOP:   MOVX    @DPTR, A
              INC     DPTR
              DJNZ    R7, XDATALOOP
              DJNZ    R6, XDATALOOP

ENDIF

IF PPAGEENABLE <> 0
    MOV     PPAGE_SFR, #PPAGE

ENDIF

IF PDATALEN <> 0
    MOV     R0, #LOW (PDATASTART)
    MOV     R7, #LOW (PDATALEN)
    CLR     A
PDATALOOP:   MOVX    @R0, A
              INC     R0
              DJNZ    R7, PDATALOOP

ENDIF
IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)
    MOV     ?C_IBP, #LOW IBPSTACKTOP

ENDIF
IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)
    MOV     ?C_XBP, #HIGH XBPSTACKTOP
    MOV     ?C_XBP+1, #LOW XBPSTACKTOP

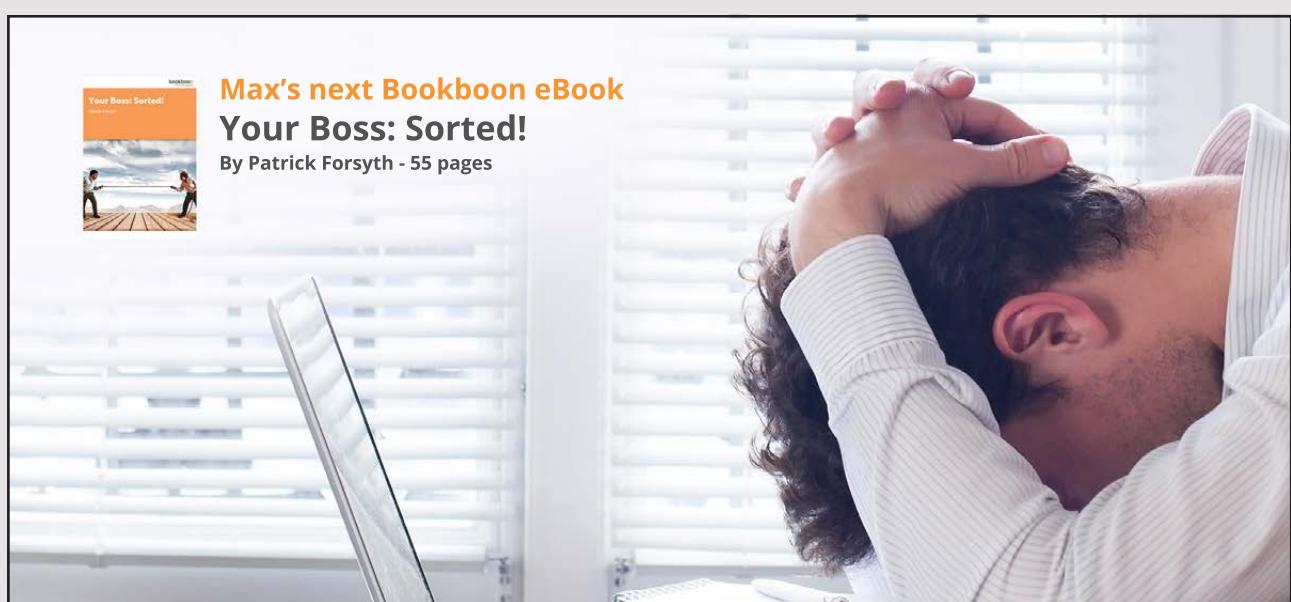
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
    MOV     ?C_PBP, #LOW PBPSTACKTOP
ENDIF

    MOV SP, #?STACK-1
; This code is required if you use L51_BANK.A51 with Banking Mode 4
```

```
; EXTRN CODE (?B_SWITCH0)
;           CALL      ?B_SWITCH0 ; init bank mechanism
;                           to code bank 0
; LJMP  ?C_START

END
```



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

**Unlock your life.
Bookboon Premium is your key.**

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

Download free eBooks at bookboon.com

3.2.3 PAULOS_F040_CAN_V01.C

```
/*
*****
*          PaulOS_F040_CAN_V01.c           RTOS KERNEL SOURCE CODE
*
* Co-Operative RTOS written in C by Ing. Paul P. Debono :
* -----
*
* For use with the C8051F040 family of microcontrollers
*
* Notes:
*
* Timer to use for the RTOS ticks is user
* selectable, Timer 0, 1, 2, or 3
*
* Assign the correct values to 'TICK_TIMER', 'CPU', 'MAINSTACK'
* and 'NOOFTASKS' in PaulOS_F040_CAN_V01.H
*
* If it is noticed that timing parameters are not
* being met well - the system's TICKTIME
* can be modified by changing the value
* 'TICKTIME' in PaulOS_F040_CAN_V01.H
* - please adhere to the conditions mentioned in PaulOS_F040_CAN_V01.H
*
* File      : PaulOS_F040_CAN_V01.C
* Revision  : 10
* Date      : FEBRUARY 2015
* By        : Paul P. Debono
*
*                      B. Eng. (Hons.) Elec. Course
*                      University Of Malta
*
*****
*/
/* **** INCLUDES **** */

```

```
#include "C8051F040.h" /* special function registers 8051F040 */  
#include "Paulos_F040_CAN_V01.h"  
/* RTOS system calls definitions (IN PROJECT DIRECTORY) */  
  
/*  
*****  
*/  
  
/*  
*****  
STRUCTURE DEFINITIONS  
*****  
*/  
struct task_param {  
    uchar stackptr;  
    uchar flags;  
    uchar intnum;  
    uint timeout;  
    uint interval_count;  
    uint interval_reload;  
    char stack[STACKSIZE];  
};
```



The image shows a woman with long blonde hair, wearing a blue tank top, smiling while wearing a black VR headset. To her right is an orange rectangular advertisement for MT Højgaard. The ad features the company logo (a stylized 'M' and 'H') and the text 'MT Højgaard'. Below this, in large white capital letters, is 'BEDRE LØSNINGER'. Underneath, there is a block of Danish text: 'I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?' At the bottom of the ad, the website 'mth.dk/vorestilgang' is listed.

```
struct task_param xdata task[NOOFTASKS + 1];

/*
*****GLOBAL VARIABLES*****
*/
uchar data * data ReadyQTop;      // Address of last ready task
uchar data Running;              // Number of current task
bit bdata IntFlag;               // Flag indicating a task waiting
                                 // for an interrupt was found
bit bdata TinQFlag;              // Flag indicating that
                                 // a task timed out
bit bdata Priority;              // Flag indicating whether priority
                                 // is enabled or disabled
uchar data ReadyQ[NOOFTASKS + 2]; // Queue stack for tasks
                                 // ready to run
```

```
/*
*****
*****
*/
/*          FUNCTION DEFINITIONS
*****
*/
/* Function name : OS_INIT_RRTOS
*
* Function type : Initialisation System call
*
* Description   : This system call initialises the RTOS variables,
*                  task SPs and * enables any required interrupts
*
* Arguments     : ticktimer Represents the timer which is used
*                  to generate the RTOS ticks.
*
*
* Returns : None
*
*****
*/
void OS_INIT_RRTOS(uchar TickTimer)
{
    uchar xdata i,j;
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page

    TickTimer = TickTimer;             /* parameter not used here, just
                                         to keep compatibility */

    #if (TICK_TIMER == 0)
    #message "Using Timer 0 as the Tick Timer"
    SFRPAGE = TIMER01_PAGE;
```

```

IE &= 0x7F;
IE |= 0x02;           /* Set up 8051 IE register,
                      using timer 0 */
IP = 0x02;           /* Assign scheduler interrupt
                      high priority */

#if (TICK_TIMER == 1)
#message "Using Timer 1 as the Tick Timer"
SFRPAGE = TIMER01_PAGE;
IE &= 0x7F;
IE |= 0x08;           /* Set up 8051 IE register,
                      using timer 1 */
IP = 0x08;           /* Assign scheduler interrupt
                      high priority */

#elif (TICK_TIMER == 2)
#message "Using Timer 2 as the Tick Timer"
SFRPAGE = TMR2_PAGE;
IE &= 0x7F;
IE |= 0x20;           /* Set up 8051 IE register,
                      using timer 2 */
IP = 0x20;           /* Assign scheduler interrupt
                      high priority */

#elif (TICK_TIMER == CAN_MASTER)
#message "Using Timer 2 as the Tick
Timer for the Master"
#message "Using the CAN bus interrupt as
the Tick Timer on the Slave board"
SFRPAGE = TMR2_PAGE;
IE &= 0x7F;
IE |= 0x20;           /* Set up 8051 IE register,
                      using timer 2 */
IP = 0x20;           /* Assign scheduler interrupt
                      high priority */

#elif (TICK_TIMER == 3)
#message "Using Timer 3 as the Tick Timer"
SFRPAGE = TMR3_PAGE;
EIE2 |= ET3;           /* Set up 8051 IE register,
                      using timer 3 */
EIP2 |= PT3;           /* Assign scheduler interrupt
                      high priority */

#elif (TICK_TIMER == 4)

```

```

#message "Using Timer 4 as the Tick Timer"
SFRPAGE = TMR4_PAGE;
    EIE2 |= ET4;                                /* Set up 8051 IE register,
                                                using timer 4 */
EIP2 |= PT4;                                /* Assign scheduler interrupt
                                                high priority */
    #elif (TICK_TIMER == CAN_SLAVE)
#message "Using Timer 2 as the Tick Timer on the Master board"
#message "Using the CAN bus interrupt as
the Tick Timer for the Slave"
    enable_CAN_interrupts();                  /* */

#endif

Running = IDLE_TASK;      /* Set idle task as the running task */

for (i = 0; i < NOOFTASKS; i++)
{
    task[i].timeout = NOT_TIMING; /* Initialise task
time outs, */
    task[i].interval_count = NOT_TIMING;
/* periodic interval count */
    task[i].interval_reload = NOT_TIMING;
/* and reload variables. */
    task[i].intnum = NO_INTERRUPT; /* not
waiting for any interrupt. */
    ReadyQ[i] = IDLE_TASK; /* Fill the READY queue with */
}                                              /* with the idle task */

ReadyQ[NOOFTASKS] = IDLE_TASK;
ReadyQ[NOOFTASKS + 1] = IDLE_TASK;

ReadyQTop = ReadyQ; /* Pointer to last task made
to point to */
                    /* base of the queue. */

for (i = 0; i < NOOFTASKS + 1; i++)
{
    task[i].stackptr = MAINSTACK + 2; /* Initialise task
SP values */
}

```

```
        task[i].flags = 0; /* Initialise task status bytes */
for(j=0;j<STACKSIZE;j++) task[i].stack[j] =
    0; /* clear all ext. stack area */
}
SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
}

/*
*****
*****
*
* Function name : OS_CREATE_TASK
*
* Function type : Initialisation System call
*
* Description   : This system call is used in the main program for
*                  each task to be created for use in the application.
*
*
* Arguments : task          Represents the task number
*              (1st task is numbered as 0).
*
*             taskadd       Represents the task's start
*                           address, which in the C
*                           environment, would
*                           simply be the name of the procedure
*
* Returns : None
*
*****
*/
void OS_CREATE_TASK(uchar tasknum, uint taskadd)
{
    ReadyQTop++;           /* Task is added to next available */
```

```
*ReadyQTop = tasknum; /* position in the READY queue.      */
task[tasknum].stack[0] = taskadd % 256;
task[tasknum].stack[1] = taskadd / 256;
}

/*
*****
***** Function name : OS_RTOS_GO
*
* Function type : Initialisation System call
*
* Description   : This system calls is used to start
*                  the RTOS going such that it
*                  supervises the application processes.
*
*
* Arguments : prior           Determines whether tasks
*              ready to be executed
*              are sorted prior to processing
*              or not. If prior = 0
*              a FIFO queue function is
*              implied, if prior = 1 the
*              queue is sorted by task number
*              in ascending order,
*              as a higher priority is associated
*              with smaller task number
*              (task 0 would have the highest
*              priority), such that the first
*              task in the queue, which would
*              eventually run, would be the
*              one with the smallest task number
*              having highest priority.
*
* Returns : None
*
*****/
```

```

void OS_RTOS_GO(uchar prior)
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    if (prior == 1)                      /* Checks if tasks priorities */
        Priority = 1;                  /* are to be enabled */
    else
        Priority = 0;

    #if (TICK_TIMER == 0)

/* Configure Timer 0 in 16-bit timer mode for the 8051 */
    SFRPAGE = TIMER01_PAGE;
    TMOD &= 0xF0;          /* Clear T0 mode
control, leaving T1 untouched */
    TMOD |= 0x01;          /* Set T0 mode control */
    CKCON &= 0xF0;          /* Use sysclk/12 (TOM = 0) */
    TR0 = 1;                /* Start timer 0 */
    TF0 = 1;                /* Cause first interrupt
immediately */

#elif (TICK_TIMER == 1)

/* Configure Timer 1 in 16-bit timer mode for the 8051 */
    SFRPAGE = TIMER01_PAGE;
    TMOD &= 0x0F;          /* Clear T1 mode control,
leaving T0 untouched */
    TMOD |= 0x10;          /* Set T1 mode control */
    CKCON &= 0xE8;          /* Use sysclk/12 (T1M = 0) */
    TR1 = 1;                /* Start timer 1 */
    TF1 = 1;                /* Cause first interrupt
immediately */

#elif (TICK_TIMER == 2)
    SFRPAGE = TMR2_PAGE;
    RCAP2 = BASIC_TICK; /* Configures Timer 2 in 16-bit      */
                         /* auto-reload mode */
    TMR2CF = 0x00;          /* Use sysclk/12 */
    TMR2CN = 0x84;          /* TR2 = TF2 = 1, causes first
interrupt immediately */

```

```
#elif (TICK_TIMER == 3)
SFRPAGE = TMR3_PAGE;
    RCAP3 = BASIC_TICK; /* Configures Timer 3 in 16-bit
                           auto-reload mode */
TMR3CF = 0x00; /* Use sysclk/12 */
    TMR3CN = 0x84; /* TR3 = TF3 = 1, causes first
                      interrupt immediately */

#elif (TICK_TIMER == 4)
SFRPAGE = TMR4_PAGE;
    RCAP4 = BASIC_TICK; /* Configures Timer 4 in 16-
                           bit auto-reload mode */
TMR4CF = 0x00; /* Use sysclk/12 */
    TMR4CN = 0x84; /* TR4 = TF4 = 1, causes first
                      interrupt immediately */

#elif (TICK_TIMER == CAN_SLAVE)
clear_msg_objects(); // set up the CAN bus
init_msg_object_RX_Slave(0x01);
init_msg_object_TX_Slave(0x02);
start_CAN();

#elif (TICK_TIMER == CAN_MASTER)
clear_msg_objects(); // set up the CAN bus
init_msg_object_RX_Slave(0x01);
init_msg_object_TX_Slave(0x02);
start_CAN();
SFRPAGE = TMR2_PAGE;
    RCAP2 = BASIC_TICK; /* Configures Timer 2 in 16-bit */
                           /* auto-reload mode for the Master RTOS */
TMR2CF = 0x00; /* Use sysclk/12 */
    TMR2CN = 0x84; /* TR2 = TF2 = 1, causes first
                      interrupt immediately */

#endif

SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
TinQFlag = 1; /* Signals scheduler that tasks have been */
```

```
        /* added to the queue. */
EA = 1;      /* Interrupts are enabled, starting the RTOS */
}

/*
*****
*****
*/
/* Function name : OS_RUNNING_TASK_ID
*
* Function type : Inter-task Communication System call
*
* Description   : This system call is used to check
*                  to get the number of the
*                  current task.
*
* Arguments     : None
*
* Returns       : Number of currently running task
*                  from which it must be called
*
*****
*/
uchar OS_RUNNING_TASK_ID(void)
{
    return (Running);
}

/*
*****
*****
*/
/* Function name : OS_SCHECK
*
* Function type : Inter-task Communication System call
*
* Description   : This system call is used to check if the
*                  current task has its signal set.
```

```
*           It tests whether there was any signal
*           sent to it by some other task.
*
* Arguments      : None
*
* Returns        : 1 if its signal bit is set, 0 if not set
*
***** */
*/
bit OS_SCHECK(void)
{
    EA = 0;
    if (task[Running].flags & SIGS_Flag) /* If a signal is
                                         already */
    {
        /* present it's cleared           */
        task[Running].flags &= ~SIGS_Flag; /* and a 1 is
                                         returned. */
        EA = 1;
        return 1;
    }
    else
        /* If a signal is not present,
           0 is returned */
    {
        EA = 1;
        return 0;
    }
}

/*
*****
*/
/*
*****
* Function name : OS_SIGNAL_TASK
*
* Function type : Inter-task Communication System call
*/
```

```

* Description   : This system call is used to send a
                  signal to another task.
*
* Arguments     : task      Represents the task to which a
                  signal is required to be sent.
*
* Returns       : None
*
***** ****
*/
void OS_SIGNAL_TASK(uchar tasknum)
{
    EA = 0;

    if (task[tasknum].flags & SIGW_Flag)
    {
        task[tasknum].flags &= ~SIGS_Flag; /* If a task has been
                                         already waiting */
        task[tasknum].flags &= ~SIGW_Flag; /* for a
                                         signal, the task no */
        task[tasknum].timeout = NOT_TIMING; /* longer has
                                         to wait and is */
        ReadyQTop++; /* added to the READY queue. */
        *ReadyQTop = tasknum;
        TinQFlag = 1;
        EA = 1;
    }
    else
        /* If it was not
           waiting, its */
        task[tasknum].flags |= SIGS_Flag; /* signal sent
                                         bit is set*/
    EA = 1;
}\

/*
*****
*/

```

```
/*
*****
* Function name : OS_WAITS
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to
*                  wait for a signal to arrive
*                  within a given number of RTOS ticks.
*                  If the signal is already present,
*                  the task continues to execute.
*
* Arguments      : ticks      Represents the number of ticks
*                  for which the task will wait
*                  for a signal to arrive. Valid
*                  range for this argument is
*                  0 to 4294967295. A value of 0
*                  means waiting forever for a
*                  signal to arrive.
*
* Returns        : None
*
*****
*/
void OS_WAITS (uint ticks)
{
    EA = 0;

    if (task[Running].flags & SIGS_Flag) /* If signal already */
                                         /* preswent
                                         clears the */
    {
        task[Running].flags &= ~SIGS_Flag; /* signal and the
                                             task */
                                         /* continues to
                                         run. */
    }
    else
    {
        /* If signal is
           not present */
    }
}
```

```
task[Running].flags |= SIGW_Flag; /* the task is sent
                                   in the */
task[Running].timeout = ticks;   /* waiting state,
                                   by causing */
QShift();                      /* a task switch. */

}

/*
*****
***** Function name : OS_WAITT
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to
                  go in the waiting state for
                  a timeout period given by a defined
                  number of RTOS ticks.
*
* Arguments     : ticks      Represents the number of ticks for which
*                  the task will wait. Valid range
*                  for this parameter is
*                  1 to 4294967295. A zero waiting
*                  time parameter is set to 1
*                  by the RTOS itself, since a zero
*                  effectively kills the task,
*                  by making it wait forever.
*
* Returns       : None
*
*****
*/
void OS_WAITT (uint ticks)
{
```

```

EA = 0;
if (ticks == 0)
    ticks = 1;                      /* Task's time out variable
                                         is updated           */
task[Running].timeout = ticks; /* and the task then
                                enters the      */
QShift();                     /* waiting state.      */
}

/*
*****
*****
*/
/*
*****
*****
*
* Function name : OS_WAITP
*
* Function type : Event-Waiting System call
*
* Description   : This system call is used by a task
                  to wait for the end of its
                  periodic interval. If the interval
                  has already passed, the task
                  continues to execute.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
void OS_WAITP(void)
{
    EA = 0;
    if (task[Running].flags & SIGV_Flag) /* If the periodic      */

```

```

        {
            /* interval time has */
            task[Running].flags &= ~SIGV_Flag;           /* has
                                                       elapsed, the */
            EA = 1;                                     /* task continues to */
            /* execute. */
        }
        else
        {
            /* Else the task */
            task[Running].flags |= SIGV_Flag;    /* enters the waiting */
            QShift();                           /* state. */
        }
    }

/*
*****
* Function name : OS_PERIODIC
*
* Function type : Event-Waiting System call
*
*Description   : This system call causes a task to
                 repeat its function every
                 given number of RTOS ticks.
** Arguments    : ticks     Represents the length of the
                 periodic interval in
                 terms of RTOS ticks, after
                 which the task repeats
                 itself. Valid range for this
                 parameter is 1 to
                 4294967295.
*
* Returns       : None
*
*****
*/
void OS_PERIODIC (uint ticks)
{

```

```
EA = 0;
if (ticks == 0)
    ticks = 1;
task[Running].interval_reload = ticks; /* Task's periodic
                                         interval count */
task[Running].interval_count = ticks; /* and reload variables
                                         are */
                                         /* initialised. */
EA = 1;
}

/*
*****
*****
*/
/* 
*****
*****
*
* Function name : OS_WAITI
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task
                  to wait for a given event
                  (interrupt). It identifies for which
                  interrupt the task has to
                  wait. Once identified - the task's appropriate
                  flag is set and the task is set in the waiting
                  state by causing a task swap - the task
                  would wait indefinitely for the
                  interrupt as its timeout variable
                  would be set to 0 (NOT_TIMING) either
                  during initialisation of the
                  RTOS or after expiry of its timeout
                  period due to other prior
                  invocations of wait-inducing system calls.
*
* Arguments      : intnum     Represents the interrupt number
                  associated with the given
                  interrupt for which the calling
                  task intends to wait

```

```

*
* Returns      : None
*
***** ****
void OS_WAITI(uchar intnum)
{
    EA = 0;

    switch (intnum)
    {
#if (!STAND_ALONE_ISR_00)
        case 0:                                /* Interrupt
                                                    number 0 */
            task[Running].intnum = IE0_INT;      /* Task ade to
                                                    wait for */
            QShift();                          /* external
                                                    interrupt
                                                    0 */
            break;
#endif

#if ((TICK_TIMER != 0) && (!STAND_ALONE_ISR_01))
        case 1:                                /* Interrupt
                                                    number 1 */
            task[Running].intnum = TF0_INT;      /* Task made to
                                                    wait for */
            QShift();                          /* timer 0
                                                    interrupt */
            break;
#endif

#if (!STAND_ALONE_ISR_02)
        case 2:                                /* Interrupt
                                                    number 2 */
            task[Running].intnum = IE1_INT;      /* Task made to
                                                    wait for */
            QShift();                          /* external
                                                    interrupt
                                                    1 */
#endif
    }
}

```

```
        break;
#endif

#if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )
    case 3:                                /* Interrupt
                                                number 3 */
        task[Running].intnum = TF1_INT;      /* Task made
                                                to wait
                                                for */
        QShift();                          /* timer 1
                                                interrupt
                                                */
        break;
#endif

#if (!STAND_ALONE_ISR_04)
    case 4:                                /* Interrupt number 4 */

        task[Running].intnum = UART0_INT;   /* Task made to
                                                wait for */
        QShift();                          /* serial
                                                port
                                                interrupt
                                                */
        break;
#endif

#if ( (TICK_TIMER != 2) && (!STAND_ALONE_ISR_05) )
    case 5:                                /* Interrupt
                                                number 5 */
        task[Running].intnum = F2_INT;      /* Task made to
                                                wait for */
        QShift();                          /* timer 1 interrupt */
        break;
#endif m

#if (!STAND_ALONE_ISR_06)
    case 6:                                /* Interrupt number 6 */

```

```
task[Running].intnum = SPIF_INT; /* Task made to
                                wait for */
QShift();                      /* serial peripheral
                                interface */

break;
#endif

#if (!STAND_ALONE_ISR_07)
    case 7:                      /* Interrupt
                                number 7 */

task[Running].intnum = SI_INT; /* Task made to
                                wait for */
QShift();                      /* SMBus interface */

break;
#endif

#if (!STAND_ALONE_ISR_08)
    case 8:                      /* Interrupt
                                number 8 */

task[Running].intnum = AD0WIN_INT; /* Task made to
                                wait for */
QShift();                      /* ADC0 Window
                                comparator */

break;
#endif

#if (!STAND_ALONE_ISR_09)
    case 9:                      /* Interrupt
                                number 9 */

task[Running].intnum = CF_INT;      /* Task made to
                                wait for */
QShift();                      /* Programmable
                                Counter Array */

break;
#endif

#if (!STAND_ALONE_ISR_10)
```

```
        case 10:                                /* Interrupt
                                                    number 10 */

            task[Running].intnum = CP0FIF_INT;      /* Task made to
                                                    wait for */
            QShift();                               /* comparator
                                                    0 Falling
                                                    Edge */

            break;
#endif

#if (!STAND_ALONE_ISR_11)
    case 11:                                /* Interrupt
                                                    number 11 */

        task[Running].intnum = CP1FIF_INT;      /* Task made to
                                                    wait for */
        QShift();                               /* comparator
                                                    0 Rising
                                                    Edge */

        break;
#endif

#if (!STAND_ALONE_ISR_12)
    case 12:                                /* Interrupt
                                                    number 12 */

        task[Running].intnum = CP2FIF_INT;      /* Task made to
                                                    wait for */
        QShift();                               /* comparator 1
                                                    Falling Edge */

        break;
#endif

#if (TICK_TIMER != 3) && (!STAND_ALONE_ISR_14)
    case 14:                                /* Interrupt
                                                    number 14 */

```

```
task[Running].intnum = TF3_INT;      /* Task made to
                                         wait for */
QShift();                           /* timer 3
                                         interrupt */

break;
#endif

#if (!STAND_ALONE_ISR_15)
    case 15:                         /* Interrupt
                                         number 15 */

        task[Running].intnum = AD0INT_INT; /* Task made to
                                         wait for */
        QShift();                      /* ADC0 end of
                                         conversion */

        break;
#endif

#if ( (TICK_TIMER != 4) && (!STAND_ALONE_ISR_16) )
    case 16:                         /* Interrupt
                                         number 16 */

        task[Running].intnum = TF4_INT; /* Task made to
                                         wait for */
        QShift();                      /* timer 4
                                         interrupt */

        break;
#endif

#if (!STAND_ALONE_ISR_17)
    case 17:                         /* Interrupt
                                         number 17 */

        task[Running].intnum = AD2WINT_INT; /* Task made
                                         to wait
                                         for */
        QShift();                      /* ADC1 end of
                                         concersion
                                         */

        break;

```

```
#endif

#if (!STAND_ALONE_ISR_18)
    case 18: /* Interrupt
               number 18 */

        task[Running].intnum = ADC2INT_
        INT; /* Task made to wait for */
        QShift(); /* external
                   interrupt 18
                   */
        break;
#endif

#if ( (TICK_TIMER != CAN_SLAVE) && (!STAND_ALONE_ISR_19) )
    case 19: /* Interrupt
               number 19 */

        task[Running].intnum = CAN_INT; /* Task made to
                                         wait for */
        QShift(); /* external
                   interrupt 19 */
        break;
#endif

#if (!STAND_ALONE_ISR_20)
    case 20: /* Interrupt
               number 20 */

        task[Running].intnum = UART1_INT; /* Task made to
                                         wait for */
        QShift(); /* UART1
                   interrupt */
        break;
#endif

default:
EA = 1;
break;
```

```

        }

}

/*
*****
* Function name : OS_DEFER
*
* Function type : Task Suspension System call
*
* Description   : This system call is used to stop
*                  the current task in order for
*                  the next task in the queue to
*                  execute. In the meantime the
*                  current task is placed at the end of the queue.
** Arguments     : None
*
* Returns       : None
*
*****
*/
void OS_DEFER(void)
{
    EA = 0;

    task[Running].timeout = 2;          /* Task added to the waiting */
                                       /* queue, for 2 tick times, prior to */
    QShift();                         /* causing a task switch.      */
}

/*
*****
*/

```

```
*  
* Function name : OS_KILL_IT  
*  
* Function type : Task Suspension System call  
*  
* Description   : This system call kills the current  
                  task, by putting it permanently  
                  waiting, such that it never executes  
                  again. It also clears any set  
                  waiting signals which the task might have.  
*  
* Arguments      : None  
*  
* Returns        : None  
*  
*****  
*/  
  
void OS_KILL_IT(void)  
{  
    EA = 0;  
    task[Running].flags = 0;           /* Task is killed by  
                                      clearing its flags */  
    task[Running].intnum = NO_INTERRUPT;  
    task[Running].timeout = NOT_TIMING; /* setting it to wait  
                                         forever */  
    task[Running].interval_count = 0; /* Task's periodic interval  
                                      count is set to zero */  
    QShift();                         /* and then cause a  
                                      task switch. */  
}  
  
/*  
*****  
  
/*  
* Function name : OS_RESUME_TASK  
*
```

```
* Function type : Inter-task Communication System call
*
* Description   : This system call is used to resume
*                  another KILLED task.
*
* Arguments      : task          Represents the task to which
*                  is to be restarted.
*
* Returns        : None
*
***** */
*/
```

```
void OS_RESUME_TASK (uchar tasknum)
{
    EA = 0;

    if (task[tasknum].interval_reload != 0)
        /* if task was a KILLED periodic task */
        task[tasknum].interval_count = 1; /* resume periodic
                                         task otherwise */
    else
        task[tasknum].timeout = 1; /* resume normal waiting
                                     task after 1 tick */

    task[Running].timeout = 2; /* Place the current task
                               waiting for the */
    QShift(); /* next 2 ticks in the
               waiting state, thus */
               /* giving up its time for other tasks. */
}

/*
***** */
*/
```

```
/*
***** */
*
* Function name : QShift
*
```

```

* Function type : Context Switcher (Internal function)
*
* Description   : This function is used to perform a
                  context switch i.e. task swapping
*
* Arguments     : None
*
* Returns       : None
*
*****,

void QShift (void) using 1
{
    uchar data i, temp;
    uchar idata * idata internal;
    uchar data * idata qtask;
    uchar data * idata qptr;

    TinQFlag = 0;
    task[Running].stackptr = temp = SP; /* Current task's
                                         SP is saved */
    internal = MAINSTACK;
    /* Current task's USED stack area is saved */
    i = 0;
    do {
        task[Running].stack[i++] = *(internal++);
        } while (internal<=temp);
    qtask = ReadyQ;                      /* READY queue is
                                         shifted down */
    qptr = ReadyQ + 1;
    while (qtask <= ReadyQTop)           /* by one position */
    {
        *qtask++ = *qptr++;
    }
    ReadyQTop--; /* Pointer to last task in queue is decremented */
    if (ReadyQTop < ReadyQ)             /* Ensure that this
                                         pointer is never */
        ReadyQTop = ReadyQ;            /* below the start of the
                                         READY queue */

```

```

if (Priority == 1)          /* If task priorities
                            are enabled */
{
    /* the queue is sorted
       such that */

    qptr = ReadyQTop;      /* the highest priority task */
    while (qptr > ReadyQ) /* becomes the running
                            task, i.e. */
    {
        /* the one having
           the smallest */

        /* task number. */
        /* Just one scan through
           the list */

        qptr--;
        if (*qptr > *(qptr + 1))
        {
            temp = *qptr;
            *qptr = *(qptr + 1);
            *(qptr + 1) = temp;
        }
    }
}

/* The first task in the
   READY queue */
Running = ReadyQ[0];        /* becomes the new running
                            task */
                            /* The new running task's stack */
                            /* area is copied to internal RAM */

temp = task[Running].stackptr;
internal = MAINSTACK;
/* The new running task's USED stack area
   is copied to internal RAM */
i=0;
do {
    *(internal++) = task[Running].stack[i++];
} while (internal<=temp);

SP = task[Running].stackptr; /* The new running task's
                            SP is restored */
                            /* such that the new task
                               will execute. */

EA = 1;

```

```
}

/*
*****
* Function name : Xtra_Int_0
*
* Function type : Interrupt Service Routine
*
* Description   : This is the EXT0 (\INT0) ISR whose
*                  associated interrupt number is 0.
*
* Arguments      : None
*
* Returns        : None
*
*****
****

*/
void Xtra_Int_0 (void) interrupt 0 using 1
{
    EA = 0;
    Xtra_Int(IE0_INT);           /* Passes EXT0W for
                                identification purposes */
}

/*
*****
* Function name : RTOS_Timer_Int
*
* Function type : Scheduler Interrupt Service Routine
*
* Description   : This is the RTOS scheduler ISR. It
*                  generates system ticks and

```

```

*
      calculates any remaining waiting and
      periodic interval time for
*
      each task.

*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#endif

#if (TICK_TIMER == 0) /* If Timer 0 is used for
                     the scheduler */
void RTOS_Timer_Int (void) interrupt 1 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    uchar data k;                         /* Timer 0 is used */
    uchar data * idata q;                /* for scheduling. */
    bit data On_Q;

    SFRPAGE = TIMER01_PAGE;
    TH0 = BASIC_TICK / 256;   /* Timer registers reloaded */
    TL0 = BASIC_TICK % 256;

#elif (TICK_TIMER == 1) /* If Timer 1 is used for
                     the scheduler */
void RTOS_Timer_Int (void) interrupt 3 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    uchar data k;                         /* Timer 1 is used */
    uchar data * idata q;                /* for scheduling. */
    bit data On_Q;

    SFRPAGE = TIMER01_PAGE;
    TH1 = BASIC_TICK / 256;   /* Timer registers reloaded */
    TL1 = BASIC_TICK % 256;

#elif (TICK_TIMER == 2) /* If Timer 2 is
                     used for the scheduler */
void RTOS_Timer_Int (void) interrupt 5 using 1

```

```
{  
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page  
    uchar data k;                         /* For the 8032, Timer  
                                              2 is used */  
    uchar data * idata q;                 /* for  
                                              scheduling. */  
    bit data On_Q;  
  
    SFRPAGE = TMR2_PAGE;  
    TF2 = 0;                             /* Timer 2 interrupt flag is  
                                              cleared by software */  
  
#elif (TICK_TIMER == 3) /* If Timer 3 is used for  
                           the scheduler */  
void RTOS_Timer_Int (void) interrupt 14 using 1  
{  
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page  
    uchar data k;                         /* For the 8032, Timer 3  
                                              is used */  
    uchar data * idata q;                /* for scheduling. */  
    bit data On_Q;  
  
    SFRPAGE = TMR3_PAGE;  
    TF3 = 0;                             /* Timer 3 interrupt flag is  
                                              cleared by software */  
  
#elif (TICK_TIMER == 4) /* If Timer 4 is used for  
                           the scheduler */  
void RTOS_Timer_Int (void) interrupt 16 using 1  
{  
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
```

```

uchar data k;                                /* For the , 8032 Timer
                                             4 is used */
uchar data * idata q;                      /* for scheduling. */
bit data On_Q;

SFRPAGE = TMR4_PAGE;
TF4 = 0;                                     /* Timer 4 interrupt flag is
                                             cleared by software */
#elsif (TICK_TIMER == CAN_SLAVE) /* If CAN interrupt is used
                                 as the tick timer */
void RTOS_Timer_Int (void) interrupt 19 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    uchar data k;                          /* Timer 2 is used */
    uchar data * idata q;                /* for scheduling. */
    bit data On_Q;
    char status;

    SFRPAGE = CAN0_PAGE;
    status = CAN0STA;

    if ((status & 0x10) != 0)           // RxOk is set, interrupt
                                         caused by reception
    {
        CAN0STA = (CAN0STA & 0xEF) | 0x07; // Reset RxOk, set
                                             LEC to NoChange
        k = receive_data(0x01);          // Up to now, we have
                                         only one RX message
    }
    if ((status & 0x08) != 0)           // TxOk is set, interrupt
                                         caused by transmision
    {
        CAN0STA = (CAN0STA & 0xF7) | 0x07; // Reset TxOk, set
                                             LEC to NoChange
    }
    if (((status & 0x07) != 0) && ((status & 0x07 != 7)))
    {                                    // Error interrupt, LEC changed

```

```

/* error handling !? */
CAN0STA = CAN0STA|0x07;           // Set LEC to NoChange
}

#elif (TICK_TIMER == CAN_MASTER) /* If this is the CAN Master,
                                then use timer 2 for tick */
void RTOS_Timer_Int (void) interrupt 5 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    uchar data k;                         /* Timer 2 is used */
    uchar data * idata q;                 /* for scheduling. */
    bit data On_Q;
    uint i;

    TF2 = 0;                             /* Timer 2 interrupt flag
                                             is cleared */
    transmit_data(0x01,0x11);
    for(i=0;i<500;i++);

// delay for synchronisation purposes, to c
ompensate for CAN transmission delay

    SFRPAGE = TMR2_PAGE;

#endif

SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
for (k = 0; k < NOOFTASKS; k++)
{
    if (task[k].interval_count != NOT_TIMING) /* Updates the
                                                tasks' */
                                                /* periodic intervals. */
    {
        task[k].interval_count--;
        if (task[k].interval_count == NOT_TIMING)
        {
            task[k].interval_count =
            task[k].interval_reload;
            if (task[k].flags & SIGV_Flag)
            {

/* If periodic interval */
```

```

/* has elapsed and the          */
/* task has been waiting      */
/* for this to occur, the      */
/* task is placed in the       */
/* READY queue, if it is       */
/* verified that the task     */
/* does not already reside   */
/* in the queue, as now       */
/* the task no longer         */
/* requires to wait.          */

/* If however the task        */
/* was not waiting for        */
/* this event, the task        */
/* is not place in the        */
/* the ready queue.           */

else
    task[k].flags |= SIGV_Flag;
}

}

if (task[k].timeout != NOT_TIMING)
{
    /* Updates the tasks' */
    task[k].timeout--;           /* timeout
variables. */
}

if (task[k].timeout == NOT_TIMING)
{
    ReadyQTop++; /* If a waiting task's */
    *ReadyQTop = k; /* timeout elapses */
    TinQFlag = 1; /* the task is placed */
}

/* task[k].flags &= ~SIGV_Flag;
q = ReadyQ;
On_Q = 0;
while (q <= ReadyQTop)
{
    if (k == *q)
        On_Q = 1;
    break;
}
q++;
}

if (On_Q == 0)
{
    ReadyQTop++;
    *ReadyQTop = k;
    TinQFlag = 1;
}
}
}

```

```
        task[k].flags &= ~SIGW_Flag; /* in theready
                                     queue. */
    }
}

/* If the idle task is running,
   when tasks are */
/* known to reside in the queue,
   a task switch */
/* is purposely induced so these
   tasks can run. */
}

if ((TinQFlag == 1) && (Running == IDLE_TASK))
    QShift();
}

/*
*****
*/
/*
*****
*/
/*
*****
*
* Function name : Xtra_Int_1
*
* Function type : Interrupt Service Routine
*
* Description    : This is the Timer 0 ISR whose
                  associated interrupt number is 1.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/

```

```
#if ( (TICK_TIMER != 0) && (!STAND_ALONE_ISR_01) )
/* Timer 0 interrupt used for RTOS */

void Xtra_Int_1 (void) interrupt 1 using 1
{
    EA = 0;
    Xtra_Int(TFO_INT); /* Passes TIM0W
for identification purposes */
}

#endif

/*
*****
***** Function name : Xtra_Int_2
*****
* Function type : Interrupt Service Routine
*
* Description : This is the EXT1 (\INT1) ISR
whose associated interrupt number is 2.
*
* Arguments : None
*
* Returns : None
*
*****
*/
#endif (!STAND_ALONE_ISR_02)
void Xtra_Int_2 (void) interrupt 2 using 1
{
    EA = 0;
    Xtra_Int(IE1_INT); /* Passes IE1_INT for
identification purposes */
}
```

```
/*
*****
* Function name : Xtra_Int_3
*
* Function type : Interrupt Service Routine
*
* Description   : This is the Timer 1 ISR whose
*                  associated interrupt number is 3.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#endif ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )

void Xtra_Int_3 (void) interrupt 3 using 1
{
    EA = 0;
    Xtra_Int(TF1_INT); /* Passes TF1_INT for
                        identification purposes */
}

#endif
/*
*****
* Function name : Xtra_Int_4
*
* Function type : Interrupt Service Routine
*
*****
```

```
* Description    : This is the serial port ISR whose associated
                  interrupt number is 4.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#ifndef (!STAND_ALONE_ISR_04)
void Xtra_Int_4 (void) interrupt 4 using 1
{
    EA = 0;
    Xtra_Int(UART0_INT);           /* Passes UART0_INT for
                                    identification purposes */
}
#endif

/*
*****
*/
/* Function name : Xtra_Int_5
*
* Function type  : Interrupt Service Routine
*
* Description    : This is the Timer 2 ISR whose
                  associated interrupt number is 5.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#ifndef ((TICK_TIMER != 2) && (TICK_TIMER != CAN_
MASTER) && (!STAND_ALONE_ISR_05) )
```

```
void Xtra_Int_5 (void) interrupt 5 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = TMR2_PAGE;
    EA = 0;
    TF2 = 0;                /* may be cleared in the task itself */
    SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
    Xtra_Int(TF2_INT); /* Passes TF2_INT for
                        identification purposes */

}

#endif

/*
*****
* Function name : Xtra_Int_6
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 6.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#ifndef STAND_ALONE_ISR_06
void Xtra_Int_6 (void) interrupt 6 using 1
{
    EA = 0;
    Xtra_Int(SPIF_INT); /* Passes SPIF_INT for
                          identification purposes */
}

#endif

/*
*****
* Function name : Xtra_Int_7

```

```
/*
 * Function type : Interrupt Service Routine
 *
 * Description    : This is the ISR whose associated
 *                   interrupt number is 7.
 *
 * Arguments      : None
 *
 * Returns        : None
 *
 ****
 */
#ifndef !STAND_ALONE_ISR_07
void Xtra_Int_7 (void) interrupt 7 using 1
{
    EA = 0;
    Xtra_Int(SI_INT); /* Passes SI_INT for identification
                        purposes */
}
#endif

/*
 ****
 *
 * Function name : Xtra_Int_8
 *
 * Function type : Interrupt Service Routine
 *
 * Description    : This is the ISR whose
 *                   associated interrupt number is 8.
 *
 * Arguments      : None
 *
 * Returns        : None
 *
 ****
 */
#ifndef !STAND_ALONE_ISR_08
void Xtra_Int_8 (void) interrupt 8 using 1
{
    EA = 0;
```

```
Xtra_Int(AD0WIN_INT); /* Passes AD0WIN_
INT for identification purposes */
}

#endif

/*
*****
* Function name : Xtra_Int_9
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose
associated interrupt number is 9.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#if (!STAND_ALONE_ISR_09)
void Xtra_Int_9 (void) interrupt 9 using 1
{
    EA = 0;
    Xtra_Int(CF_INT); /* Passes CF_INT for identification
purposes */
}
#endif

/*
*****
* Function name : Xtra_Int_10
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
interrupt number is 10.
*
* Arguments      : None
*
```

```
* Returns      : None
*
***** ****
#endif (!STAND_ALONE_ISR_10)
void Xtra_Int_10 (void) interrupt 10 using 1
{
    EA = 0;
    Xtra_Int(CP0FIF_INT); /* Passes CP0FIF_INT for
                           identification purposes */
}
#endif

/*
***** ****
*
* Function name : Xtra_Int_11
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 11.
*
* Arguments     : None
*
* Returns       : None
*
***** ****
*/
#endif (!STAND_ALONE_ISR_11)
void Xtra_Int_11 (void) interrupt 11 using 1
{
    EA = 0;
    Xtra_Int(CP1FIF_INT); /* Passes CP1FIF_INT for
                           identification purposes */
}
#endif

/*
***** ****
*
```

```
* Function name : Xtra_Int_12
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
                  interrupt number is 12.
*
* Arguments      : None
*
* Returns        : None
*
***** */
#endif (!STAND_ALONE_ISR_12)
void Xtra_Int_12 (void) interrupt 12 using 1
{
    EA = 0;
    Xtra_Int(CP2FIF_INT); /* Passes CP2FIF_
                           INT for identification purposes */
}
#endif

/*
***** */
* Function name : Xtra_Int_14
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
                  interrupt number is 14.
*
* Arguments      : None
*
* Returns        : None
*
***** */
#endif ((TICK_TIMER != 3) && (!STAND_ALONE_ISR_14) )
void Xtra_Int_14 (void) interrupt 14 using 1
{
```

```

char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
SFRPAGE = TMR3_PAGE;
EA = 0;
TMR3CN &= 0x7F;          /* may be cleared in the task itself */
SFRPAGE = SFRPAGE_SAVE;
Xtra_Int(TF3_INT);      /* Passes TF3_INT for
                           identification purposes */
}

#endif

/*
*****
* Function name : Xtra_Int_15
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 15.
*
* Arguments     : None
*
* Returns       : None
*
*****
#endif
#if (!STAND_ALONE_ISR_15)
void Xtra_Int_15 (void) interrupt 15 using 1
{
    EA = 0;
    Xtra_Int(AD0INT_INT); /* Passes AD0INT_INT for
                           identification purposes */
}
#endif

/*
*****
* Function name : Xtra_Int_16
*
* Function type : Interrupt Service Routine
*

```

```
* Description : This is the ISR whose associated
               interrupt number is 16.
*
* Arguments    : None
*
* Returns      : None
*
*****
*/
#ifndef (TICK_TIMER != 4) && (!STAND_ALONE_ISR_16)
void Xtra_Int_16 (void) interrupt 16 using 1
{
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    SFRPAGE = TMR4_PAGE;
    EA = 0;
    TMR4CN &= 0x7F; /* may be cleared in the task itself */
    SFRPAGE = SFRPAGE_SAVE;
    Xtra_Int(TF4_INT); /* Passes TF4_INT for
                        identification purposes */
}
#endif

/*
*****
*
* Function name : Xtra_Int_17
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated interrupt number
is 17.
*
* Arguments     : None
*
* Returns       : None
*
*****
*/
#ifndef (!STAND_ALONE_ISR_17)
void Xtra_Int_17 (void) interrupt 17 using 1
{
```

```
EA = 0;
Xtra_Int(AD2WINT_INT); /* Passes AD2WINT_INT for
                           identification purposes */
}

#endif

/*
*****
*
* Function name : Xtra_Int_18
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
                  interrupt number is 18.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#ifndef STAND_ALONE_ISR_18
void Xtra_Int_18 (void) interrupt 18 using 1
{
    EA = 0;
    Xtra_Int(ADC2INT_INT); /* Passes ADC2INT_INT for
                           identification purposes */
}
#endif

/*
*****
*
* Function name : Xtra_Int_19
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
                  interrupt number is 19.
```

```
*  
* Arguments      : None  
*  
* Returns       : None  
*  
*****  
*/  
#if ( (TICK_TIMER != CAN_SLAVE) && (!STAND_ALONE_ISR_19) )  
void Xtra_Int_19 (void) interrupt 19 using 1  
{  
    EA = 0;  
    Xtra_Int(CAN_INT);           /* Passes CAN_INT for  
                                identification purposes */  
}  
#endif  
  
/*  
*****  
*  
* Function name : Xtra_Int_20  
*  
* Function type : Interrupt Service Routine  
*  
* Description   : This is the ISR whose associated  
                  interrupt number is 20.  
*  
* Arguments      : None  
*  
* Returns       : None  
*  
*****  
*/  
#if (!STAND_ALONE_ISR_20)  
void Xtra_Int_20 (void) interrupt 20 using 1  
{  
    EA = 0;  
    Xtra_Int(UART1_INT);        /* Passes UART1_INT for  
                                identification purposes */  
}  
#endif
```

```
/*
*****
* Function name : Xtra_Int
*
* Function type : Interrupt Handling (Internal function)
*
* Description   : This function performs the operations
*                  required by the previous ISRs.
*
* Arguments      : task_intflag      Represents the flag mask for a given
*                  interrupt against which
*                  the byte storing the
*                  flags of each task will be
*                  compared in order to
*                  determine whether any task
*                  has been waiting for
*                  the interrupt in question.
*
* Returns        : None
*
*****
*/
void Xtra_Int (uchar current_intnum) using 1
{
    uchar data k;

    IntFlag = 0;

    for (k = 0; k < NOOFTASKS; k++)
    {
        if (task[k].intnum == current_intnum)
        {
            task[k].intnum = NO_INTERRUPT;
            IntFlag = 1;
```

```

        task[k].timeout = NOT_TIMING; /* If it found
                                       that a task */
        ReadyQTop++;                /* has been waiting
                                       for the */
        *ReadyQTop = k;              /* given interrupt, it
                                       no           */
        }                           /* longer requires
                                       to wait */
    }                           /* and is therefore
                               placed   */
                                /* on the READY queue.      */
if ((IntFlag == 1) && (Running == IDLE_TASK))
{
    TinQFlag = 1;               /* If tasks are known to
                                   now reside in the */
    QShift();                  /* READY queue while the
                                   idle task is */
    }                           /* running, a task switch
                                   is purposely */
                                /* induced, such that these tasks can run. */
else if ((IntFlag == 1) && (Running != IDLE_TASK))
{
    /* Otherwise, the ISR exits after */
    TinQFlag = 1;               /* interrupts are re-enabled, */
                                /* since RTOS cannot pre-empt task */
    EA = 1;
}

else EA = 1;                  /* Otherwise exit normally */
}

/*
*****CAN BUS ADD-ONS FOR RTOS USE*****
*/
//Clear Message Objects
void clear_msg_objects (void)
{

```

```

uchar i;
char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
SFRPAGE = CAN0_PAGE;
CAN0ADR = IF1CMDMSK;      // Point to Command Mask Register 1
CAN0DATL = 0xFF;          // Set direction to WRITE all
                           // IF registers to Msg Obj
for (i=1;i<33;i++)
{
    CAN0ADR = IF1CMDRQST; // Write blank (reset) IF
                           // registers to each msg obj
    CAN0DATL = i;
}
SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
}

//Initialize Message Object for RX
void init_msg_object_RX_Slave (char MsgNum)
{
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    SFRPAGE = CAN0_PAGE;
    CAN0ADR = IF1CMDMSK;      // Point to Command Mask 1
    CAN0DAT = 0x00B8;         // Set to WRITE, and alter all
                           // Msg Obj except ID MASK
                           // and data bits
    CAN0ADR = IF1ARB1;        // Point to arbitration1 register
    CAN0DAT = 0x0000;         // Set arbitration1 ID to "0"
    CAN0DAT = 0x8004;         // Arb2 high byte:Set MsgVal
                           // bit, no extended ID,
                           // Dir = RECEIVE
    CAN0DAT = 0x0480;         // Msg Cntrl: set RXIE, remote
                           // frame function disabled
    CAN0ADR = IF1CMDRQST; // Point to Command Request reg.
    CAN0DATL = MsgNum;       // Select Msg Obj passed into
                           // function parameter list
                           // --initiates write to Msg Obj
// 3-6 CAN clock cycles to move IF register
// contents to the Msg Obj in CAN RAM
    SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
}
//Initialize Message Object for TX
void init_msg_object_TX_Slave (char MsgNum)

```

```
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = CAN0_PAGE;
    CAN0ADR = IF1CMDDMSK;     // Point to Command Mask 1
    CAN0DAT = 0x00B2;          // Set to WRITE, & alter all Msg
                               Obj except ID MASK bits
    CAN0ADR = IF1ARB1;         // Point to arbitration1 register
    CAN0DAT = 0x0000;          // Set arbitration1 ID to highest priority
    CAN0DAT = 0xA000;          // Autoincrement to Arb2 high byte:
                               // Set MsgVal bit, no extended ID, Dir = WRITE
    CAN0DAT = 0x0081;          // Msg Cntrl: DLC = 1, remote
                               frame function not enabled
    CAN0ADR = IF1CMDRQST; // Point to Command Request reg.
    CAN0DAT = MsgNum;          // Select Msg Obj passed into
                               function parameter list
                               // --initiates write to Msg Obj
// 3-6 CAN clock cycles to move IF reg contents
// to the Msg Obj in CAN RAM.
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

//Start CAN
void start_CAN (void)
{
/* Calculation of the CAN bit timing :

System clock          f_sys = 22.1184 MHz/2 = 11.0592 MHz.
System clock period   t_sys = 1/f_sys = 90.422454 ns.
CAN time quantum      tq = t_sys (at BRP = 0)

Desired bit rate is 1 MBit/s, desired bit time is 1000 ns.
Actual bit time = 11 tq = 996.65ns ~ 1000 ns
Actual bit rate is 1.005381818 MBit/s = Desired bit rate + 0.5381%

CAN bus length = 10 m, with 5 ns/m signal delay time.
Propagation delay time : 2*(transceiver loop
delay + bus line delay) = 400 ns
(maximum loop delay between CAN nodes)

Prop_Seg = 5 tq = 452 ns ( >= 400 ns).
Sync_Seg = 1 tq
}
```

```

Phase_seg1 + Phase_Seg2 = (11-6) tq = 5 tq
Phase_seg1 <= Phase_Seg2, => Phase_seg1 = 2 tq and Phase_Seg2 = 3 tq
SJW = (min(Phase_Seg1, 4) tq = 2 tq
TSEG1 = (Prop_Seg + Phase_Seg1 - 1) = 6
TSEG2 = (Phase_Seg2 - 1) = 2
SJW_p = (SJW - 1) = 1

Bit Timing Register = BRP + SJW_p*0x0040 =
TSEG1*0x0100 + TSEG2*0x1000 = 2640
Clock tolerance df :

A: df < min(Phase_Seg1, Phase_Seg2) / (2
* (13*bit_time - Phase_Seg2))
B: df < SJW / (20 * bit_time)

A: df < 2/(2*(13*11-3)) = 1/(141-3) = 1/138 = 0.7246%
B: df < 2/(20*11) = 1/110 = 0.9091%

Actual clock tolerance is 0.7246% - 0.5381%
= 0.1865% (no problem for quartz)
*/

```

```

char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page

SFRPAGE = CAN0_PAGE;
CAN0CN |= 0x41; // Configuration Change Enable CCE and INIT
CAN0ADR = BITREG ; // Point to Bit Timing register
// CAN0DAT = 0x2640; // see above
CAN0DAT = 0x5EC0; // see page 232 on C8051F04x pdf manual

CAN0ADR = IF1CMDMSK; // Point to Command Mask 1
CAN0DAT = 0x0087; // Config for TX : WRITE to CAN
RAM, write data bytes,
// set TXrqst/NewDat, clr IntPnd

// RX-IF2 operation may interrupt TX-IF1 operation
CAN0ADR = IF2CMDMSK; // Point to Command Mask 2
CAN0DATL = 0x1F; // Config for RX : READ CAN
RAM, read data bytes,
// clr NewDat and IntPnd

```

```

CAN0CN |= 0x06;           // Global Int. Enable IE and SIE
CAN0CN &= ~0x41;          // Clear CCE and INIT bits,
                           starts CAN state machine
SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

// Enable CAN interrupts
void enable_CAN_interrupts(void)
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = CAN0_PAGE;
    EIE2 |= ECANO;                      // Enable CAN interrupt
    EIP2 |= PX7;                        // CAN interrupt
                                         high priority
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

// transmit data to another node in system
void transmit_data(char MsgNum, char dat_tx)
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = CAN0_PAGE;
    CAN0ADR = IF1CMDMSK;     // Point to Command Mask 1
    CAN0DAT = 0x0087;         // Config for TX : WRITE to CAN
                           RAM, write data bytes,
                           // set TXrqst/NewDat, clr IntPnd

    // RX-IF2 operation may interrupt TX-IF1 operation
    CAN0ADR = IF1DATA1; // Point to Command Mask 2
    CAN0DATL = dat_tx;      // data to be transmitted
    CAN0ADR = IF1CMDRQST; // point to Command 1 Request Register
    CAN0DATL = MsgNum;       // move new data for transmission to Msg Obj
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}
// receive data from another node in system
char receive_data(char MsgNum)
{
    char rx_data;
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
}

```

```
SFRPAGE = CAN0_PAGE;
CAN0ADR = IF2CMRDQST; // Point to Command Request Reg.
CAN0DATL = MsgNum;      // move new data for receiving Msg obj
CAN0ADR = IF2DATA1;     // point to Command 1 Request Register
rx_data = CAN0DATL;     // move new data for transmission to Msg Obj
SFRPAGE = SFRPAGE_SAVE;           // Restore SFR page
return (rx_data);      // move new data for transmission to Msg Obj
}

/*
*****
```

4 PROGRAMMING TIPS AND PITFALLS

In this final chapter we discuss some programming tips and common pitfalls which should be avoided when programming such microcontrollers.

4.1 RAM SIZE

The C8051F040 development target board has 64KB of flash memory (On-chip ROM) for code and constants and a 4K of RAM (On-chip XRAM). Thus the KEIL IDE should be setup as shown in Figure 4-1 to make use of this on board memory.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



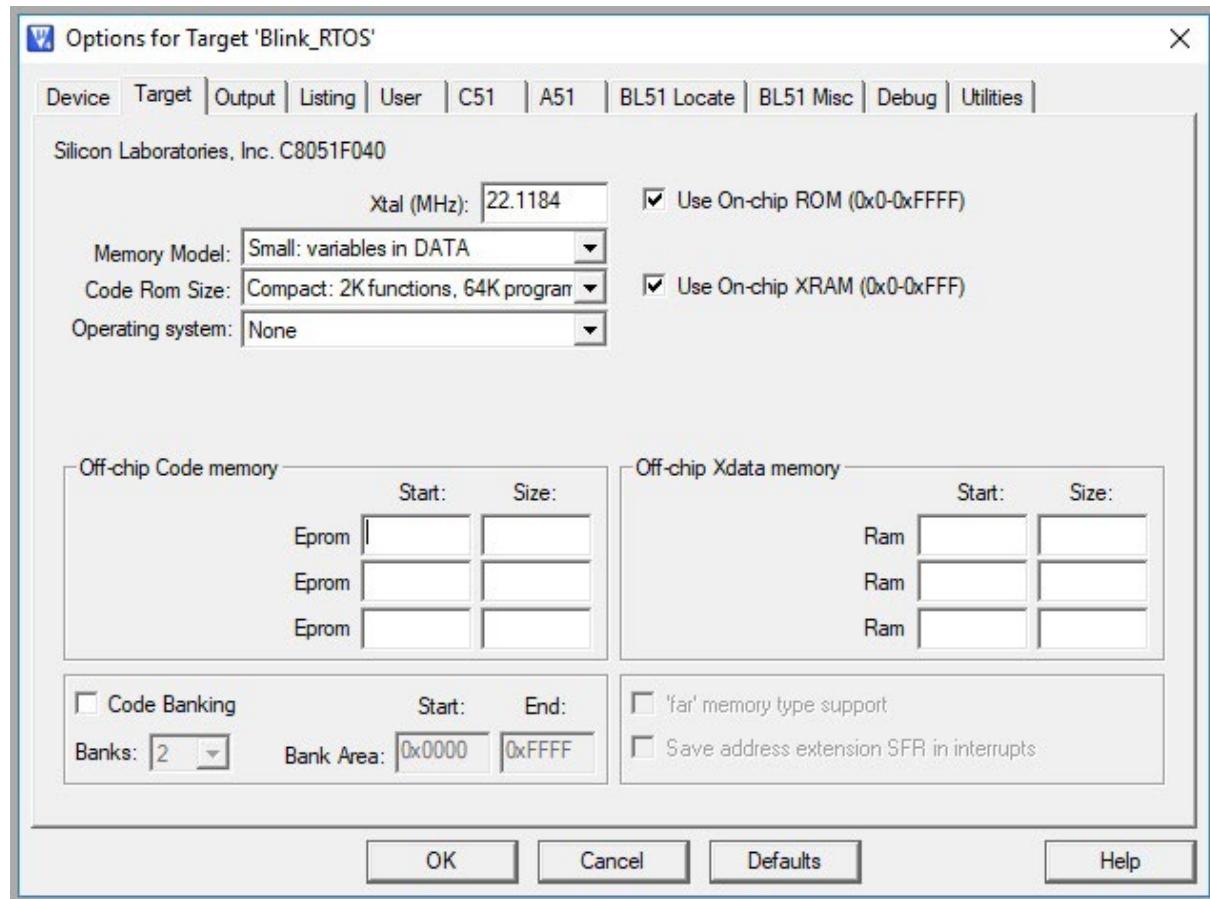


Figure 4-1 Screenshot of the Keil Target setup

4.2 SFRS

SFRs are used to control the way the 8051 peripherals functions. Not all the addresses above 80h are assigned to SFRs. However, this area may not be used as additional RAM memory even if a given address has not been assigned to an SFR. Free locations are reserved for future versions of the microcontroller and if we use that area, then our program would not be compatible with future versions of the microcontroller, since those same locations might be used for special additional SFRs in the upgraded version. Moreover, certain unused locations may actually be non-existent, in the sense that the actual cells for that memory would not form part of the memory mask when being manufactured, and hence even if we do write the code to use these locations, no actual data would be stored!

It is therefore recommended that we do not read from or write to any SFR addresses that have not been actually assigned to an SFR. Doing so may provoke undefined behaviour and may cause our program to be incompatible with other 8051 derivatives that use those free addresses to store the additional SFRs for some new timer or peripheral included in the new derivative.

If we write a program that utilizes the new SFRs that are specific to a given derivative chip (and which therefore were not included in the standard basic 8051 SFR list), our program will not run properly on a standard 8051 where those SFRs simply did not exist. Thus, it is best to use non-standard SFRs only if we are sure that our program will only have to run on that specific micro-controller. If we happen to write code that uses non-standard SFRs and subsequently share it with a third-party, we must make sure to let that party know that our code is using non-standard SFRs and can only be used with that particular device. Good remarks, notes and warnings within the program source listing would help.

For the C8051F040 in particular, since we have more than one page of SFRs, it is important to make sure that we are in the correct page when reading or writing to the SFRs. See Table 1-3 and section 1.5.1 for exact details about the SFR paging technique.

4.3 SETUP FAULTS

The setup during the initialisation is very critical and basically we would need to initialise the system clock, watchdog timer, crossbar registers, any input/output ports and whether we need to use them for digital or for analogue signals. And then of course, any timers, serial ports, ADC, DAC, SPIs etc would need to be initialised if they are going to be required in our application program. We now list some common faults which are easily made during this setup process.

4.3.1 SYSTEM CLOCK SETUP

The System clock should be setup and initialised at the start of your program. Forgetting to set it up or not setting it up correctly is a common initial fault for new comers. Checking for the clock stabilisation during a simulation run can cause problems in cases where the simulation of the clock is not well implemented as mentioned in section 1.8.1.

4.3.2 WATCHDOG TIMER SETUP

Forgetting to disable the watchdog timer or disabling it late is a common fault with beginners to this device. The effect would be for the micro-controller to keep on resetting itself while executing the few initial commands in the main program.

4.3.3 CROSSBAR SETUP

Another very common fault with newcomers to this device is setting the wrong configuration of the crossbar SFRs: XBR0, XBR1 and XBR2. Consulting the manual and reviewing the examples would help a lot to enable the user to become familiar with the initialisations required, and at which pins to expect the input or output signal to be available. (See Table 1-4, Figure 1-10, Figure 1-11 and Figure 1-12)

4.4 SERIAL PORT (UART0)

To use the ‘printf’ command, the on-board serial port or UART0 must be correctly setup at the required baud rate. It is generally necessary to initialise at least the owing four SFRs: SCON0, PCON, SCON0, and TMOD. See section 2.5.4 for a complete listing and remarks of the UART0_T1.c program and also section 4.4 in (Debono, 2015).

The advertisement features a night photograph of the Apollo Hotel building. On the left, there's a large red circular logo containing a white lightbulb icon. To its right, the text "CISO Conference" is written in white, with "Produced by Inspired" in smaller text below it. On the right side of the image, a white rectangular box contains the text "Apollo Hotel 1, Groenlandsekade Vinkeveen, Amsterdam, NL Dec 5th 2019". At the bottom, a white box contains the text "Listen, learn & build relationships with our Network of CISOs & Cyber Security Leaders". To the right of this text is the "Inspired" logo, which includes a blue lightbulb icon.

4.5 INTERRUPTS

Some common problems encountered with interrupts when using assembly language are addressed here:

Forgetting to re-start a timer: We might turn off a timer to re-load the timer register values or to read the counter in an interrupt service routine (ISR) and then forget to turn it on again before exiting from the ISR. In this case, the ISR would only execute once.

Forgetting to clear the Timer 2 interrupt flag: When using Timer 2 interrupts, the Timer 2 overflow flag TF2 is not cleared automatically when the ISR is serviced. We have to clear it in the ISR software (using CLR TF2). The same problem occurs if we forget to clear the RI or the TI flags when using the Serial Interrupt. In this case, the ISR keeps on being called repeatedly.

4.6 RTOSS PITFALLS

The PaulOS F040 co-operative RTOS is a robust and secure RTOSSs which we have used extensively throughout the years with our students. This is mainly due to the fact that being a co-operative RTOS, the task changes occur when we want them since there cannot be any forced pre-emptive task changes. However there can still be hidden problems. We should take special care when handling global variables which are accessible to all the tasks. We have to make sure that these variables are allowed to be manipulated only when we want them to. Otherwise it might happen that a task starts with one value of a global variable, then it goes on to a wait state, and when it later on resumes to run, it might end up using the wrong value of the same variable simply because it was modified in the meantime by another task.

The same problem exists in the RTOS with register banks and tasks which use the same functions which are non re-entrant.

4.7 C TIPS

- We should always try to keep functions (or tasks) as simple as possible.
- Use the correct required types for the variables; do not use **int** type if we really need **byte** or **bit** type. . Naturally, the corresponding conversion character (%c, %bu, %d etc) should then be used with ‘printf’ or ‘scanf’ commands.
- Use signed or unsigned types correctly.
- Use specified locations for storing pointers. That is use declarations such as

```
char data * xdata str;  
/* pointer stored in xdata, pointing to char stored in data */  
  
int xdata * data numtab;  
/* pointer stored in data, pointing to int stored in to xdata */  
  
long code * idata powtab;  
/* pointer stored in idata, pointing to long stored in code */
```

- In order to improve the performance during code execution or to reduce the memory size requirement for our code, we should analyse the generated list files and assembly code so as to determine which routines can be improved in speed or reduced in size.
- We should always try to minimize the variable usage by scoping.
- Set the NUMBER_OF_TASKS, TICKTIME and TICK_TIMER definitions in the PaulOS_F040_Parameter.h header file to correspond to your application program. This is often a common mistake to make.
- Ensure that if you are using interrupts, make sure that they are enabled.
- Remember that the timer used for the RTOS tick timer cannot be used also for say the baud rate generation of a UART.
- Remember to use the correct ISR parameter in PaulOS_F040_Parameter.h header file when you are using a stand-alone ISR.

APPENDIX A PAULOS_F040.C SOURCE CODE LISTING

This is the program source listing for the C version of PaulOS_F040 RTOS. It consists of:

- The header file PaulOS_F040_Parameters.h
- The header file PaulOS_F040.h
- The startup file Startup_PaulOS_F040.A51
- The main RTOS source program PaulOS_F040.C

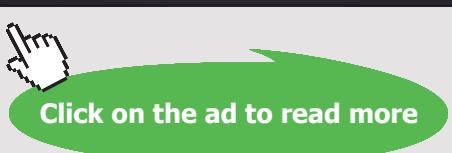
Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

Download free eBooks at bookboon.com



A.1 PaulOS_F040_Parameters.h

```
#ifndef _PaulOS_F040_Parameters_H_
#define _PaulOS_F040_Parameters_H_

/*
***** RTOS KERNEL HEADER FILE *****
*
*                               PaulOS_F040_Parameters.H
*
* For use with PaulOS_F040.C - Co-Operative
* RTOS written in C based on
* PaulOS by Ing. Paul P. Debono for use with
* the 8051 family of microcontrollers
*
* File      : PaulOS_F040_Parameters.H
* Revision  : 10
* Date      : Revised for C8051F040 February 2015
* By        : Paul P. Debono
*
*                         B. Eng. (Hons.) Elec.
*                         University Of Malta
*
***** */
#endif

#ifndef __PARAMETERS_H__
#define __PARAMETERS_H__

/*
***** DATA TYPE DEFINITIONS *****
*/
#endif

/*
***** */

```

```
#define TICK_TIMER          0      // can be 0,1,2,3 or 4

#define TICKTIME             1
// Length of RTOS basic tick in msec - refer
// to the RTOS timing definitions
// suitable values are: 1, 2, 4, 5, 8, 10, 20, 25

#define NOOFTASKS            5      // Number of tasks used in application
#define STACKSIZE             0x0F    // Number of bytes to allocate
// for the stack
// Usually no need to change this parameter

/*
*****
*/
#define STAND_ALONE_ISR_00 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_01 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_02 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_03 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_04 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_05 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_06 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_07 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_08 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_09 0 // set to 1 if using this interrupt
// as a stand alone ISR
#define STAND_ALONE_ISR_10 0 // set to 1 if using this interrupt
// as a stand alone ISR
```

```
#define STAND_ALONE_ISR_11 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_12 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_14 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_15 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_16 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_17 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_18 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_19 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_20 0 // set to 1 if using this interrupt
                           as a stand alone ISR

/*
*****
*/
#endif
```

A.2 PaulOS_F040.h

```
/*
*****RTOS KERNEL HEADER FILE*****
*
*                               PaulOS_F040.H
*
* For use with PaulOS_F040.C - Co-Operative RTOS written in C
* based on PaulOS by Ing. Paul P. Debono
* for use with the 8051 F040 family of microcontrollers
*
* File      : PaulOS_F040.H
* Revision  : 10
* Date      : Revised for C8051F040 February 2015
* By        : Paul P. Debono
*
*           B. Eng. (Hons.) Elec.
*           University Of Malta
*
*****DATA TYPE DEFINITIONS*****
*/
#ifndef __PAULOS_F040_H__
#define __PAULOS_F040_H__


/*
*****DATA TYPE DEFINITIONS*****
*
*                               DATA TYPE DEFINITIONS
*****DATA TYPE DEFINITIONS*****
*/
typedef unsigned char uchar;
typedef unsigned int uint;
typedef unsigned long ulong;
#include "Paulos_F040_Parameters.h"
#include "C8051F040.h"          /* special function registers
                                C8051F040 */

/* Stack variable points to the start
pointer in hardware stack and */
/* should be defined in Paulos_F040_Startup.A51 */
```

```

extern idata unsigned char MAINSTACK[1];

/*
***** RTOS FUNCTION PROTOTYPES *****
*
* The following RTOS system calls do not receive any parameters :
* -----
*/
void OS_DEFER (void); // Stops current task and passes
                      control to next task in queue
void OS_KILL_IT (void); // Kills a task - sets it
                        waiting forever
bit OS_SCHECK (void); // Checks if running task's signal bit is set
void OS_WAITP (void); // Waits for end of task's periodic interval
uchar OS_RUNNING_TASK_ID(void); // Returns the number of the
                                 currently executing task

/* The following commands are simply defines as MACROS below
   OS_CPU_IDLE() Set the microprocessor into a sleep
                  mode (awakes every interrupt)
   OS_CPU_DOWN() Switch off microprocessor, activated
                  again only by a hardware reset
   OS_PAUSE_RTOS() Disable RTOS, used in a stand-alone ISR
   OS_RESUME_RTOS() Re-enable RTOS, used in a stand-alone ISR
*/
/* The following RTOS system calls do receive parameters :
* -----
*/
void OS_INIT_RTOS (uchar ticktimer); // Initialises all
                                      RTOS variables
void OS_RTOS_GO (uchar prior); // Starts the RTOS running with
                               priorities if required

```

```

void OS_SIGNAL_TASK (uchar task);           // Signals a task
void OS_WAITI (uchar intnum);             // Waits for an event
                                         (interrupt) to occur
void OS_WAITT (uint ticks);   // Waits for a time out period given
                                by a defined number of ticks
void OS_WAITS (uint ticks);
// Waits for a signal to arrive within a given
number of ticks (0=wait forever)
void OS_PERIODIC (uint ticks); // Sets task to behave periodically
                               every given number of ticks
void OS_CREATE_TASK (uchar task, uint taskadd); // Creates a task
void OS_RESUME_TASK (uchar task); // Resumes a task which was
                                 previously killed

/* The following commands are simply defines as MACROS below
   OS_WAITT_A(M,s,ms) Absolute OS_WAITT() for minutes,
                       seconds and milliseconds
   OS_WAITS_A(M,s,ms) Absolute OS_WAITS() for minutes,
                       seconds and milliseconds
   OS_PERIODIC_A(M,s,ms) Absolute OS_PERIODIC() for minutes,
                         seconds and milliseconds
*/
/*
*****
*          RTOS USER DEFINITIONS
*****
*/
#define SYSCLOCK 22118400UL // 22.1184 MHz crystal
#define CPU      5140        // set to 8051F040 (denoted by 5140)

/*
*****
*          RTOS MACROS
*****
*/

```

```

#define OS_CPU_IDLE();           { PCON |= IDLE; PCON=PCON; }
// Sets the microprocessor in idle mode
#define OS_CPU_DOWN()          PCON |= STOP // Sets the
                                         // microprocessor in
                                         // power-down mode

#if (TICK_TIMER == 0)
#define OS_PAUSE_RTOS()        ET0 = 0
#define OS_RESUME_RTOS()       ET0 = 1

#elif (TICK_TIMER == 1)
#define OS_PAUSE_RTOS()        ET1 = 0
#define OS_RESUME_RTOS()       ET1 = 1

#elif (TICK_TIMER == 2)
#define OS_PAUSE_RTOS()        ET2 = 0
#define OS_RESUME_RTOS()       ET2 = 1

#elif (TICK_TIMER == 3)
#define OS_PAUSE_RTOS()        EIE2 &= ~ET3
#define OS_RESUME_RTOS()       EIE2 |= ET3

#elif (TICK_TIMER == 4)
#define OS_PAUSE_RTOS()        EIE2 &= ~ET4
#define OS_RESUME_RTOS()       EIE2 |= ET4

#endif
/*
*****
*/
/*          RTOS TIMING DEFINITIONS
*****
*/
/* Timers used for this RTOS use the system clock divided by 12 */
/* For a system clock of 22.1184 MHz
   they count up once every 0.542535 micro seconds */

```

```

/* For 1 msec, count = SYSCLOCK/12/1000 = 1843.2 */
// #define MSEC (SYSCLOCK/12UL/100UL) // 10msec = 1834.2 * 10 =
                                         18432 at 22.1184MHz

#define TWEAK 22UL // to compensate for error due to temp. etc
// This was verified by testing the date/time clock over a
// long period of days. Clock was going slow.

#define MSEC (18432UL - TWEAK) // In theory 1843.2 counts
                             represent 1 msec assuming an
// 22.1184 MHz crystal
// See F04x_Osc_Init.c
// 10msec is used to avoid floating point multiplication

#define CLOCK          ((TICKTIME * MSEC)/10UL)
// i.e. approx. 35ms max TICKTIME value -
    However respecting the condition
// above, max. acceptable TICKTIME = 25 msecs. Hence all
// suitable values are: 1, 2, 4, 5, 8, 10, 20, 25

#define BASIC_TICK      (65535 - CLOCK + 1)
//#define BASIC_TICK 63697 // count for 1 ms ticktime delay
                         at 22.1184 MHz clock

#define NOT_TIMING      0
// An indefinite period of waiting time in the
    RTOS is given by a value of 0
#define NO_INTERRUPT 0xFF
/*
*****
*/
/*
*****
*               COMPILE-TIME ERROR TRAPPING
*****
*/
#endif (CPU != 5140)
#error Invalid CPU Setting
#endif

```

```
#if (NOOFTASKS > 254)
#error Number of tasks is out of range. The
ReadyQ can store up to 254 tasks
#endif

#if 0
#if (CPU == 5140) /* C8051F040 SiLabs processor */
#if ((MAINSTACK + STACKSIZE) > 0x100)
#error Internal RAM Space exceeded. Please recheck
the MAINSTACK and STACKSIZE definitions
#endif
#elif (CPU == 8051)
#if ((MAINSTACK + STACKSIZE) > 0x80)
#error Internal RAM Space exceeded. Please recheck
the MAINSTACK and STACKSIZE definitions
#endif
#endif
#endif
#endif

#if ((TICKTIME * SYSCLOCK / 12000) > 65535)
#error Tick time value exceeds valid range
of the timer counter setting
#endif

#if ((TICKTIME * SYSCLOCK / 12000) < 65535)
&& ((1000 % TICKTIME) != 0)
#error Undesirable TICKTIME setting. Please
choose from 1, 2, 4, 8, 10, 20, 25 ms
#endif

#if (CLOCK > 65535)
#error Timer setting exceeded valid range. Please
recheck the TICKTIME and MSEC definitions
#endif

/*
*****
```

```
/*
***** TASK-RELATED DEFINITIONS *****
*/
#define IE0_INT      0
#define TF0_INT      1
#define IE1_INT      2
#define TF1_INT      3
#define UART0_INT    4
#define TF2_INT      5
#define SPIF_INT     6
#define SI_INT       7
#define AD0WIN_INT   8
#define CF_INT       9
#define CP0FIF_INT  10
#define CP1FIF_INT  11
#define CP2FIF_INT  12

#define TF3_INT      14
#define AD0INT_INT   15
#define TF4_INT      16
#define AD2WINT_INT  17
#define ADC2INT_INT  18
#define CAN_INT      19
#define UART1_INT    20

#define SIGS_Flag    0x80
#define SIGW_Flag    0x40
#define SIGV_Flag    0x20

#define IDLE_TASK NOOFTASKS
// Main endless loop in application given a
// task number equal to NOOFTASKS

/*
*****
```

```
/*
*****
* ENHANCED EVENT-WAITING ADD-ON MACROS
*****
*
* These macros perform the same functions of
* the WAITT, WAITS and PERIODIC calls
* but rather than ticks
* they accept absolute time values as parameters
* in terms of days, hours, minutes,
* seconds and millisecs
* This difference is denoted by the _A suffix -
* eg. WAITT_A() is the absolute-time version of WAITT()
*
* Range of values accepted:
*
* Using a minimum TICKTIME of 1 msec : 1
* msec - 1 min, 5 secs, 535 millisecs
* Using a recommended TICKTIME of 10 msec : 10
* millisecs - 10 mins, 55 secs, 350 millisecs
* Using a maximum TICKTIME of 50 msec : 50
* millisecs - 54 mins, 36 secs, 750 millisecs
*
* If the conversion from absolute time to ticks results in 0
* (all parameters being 0 or overflow) this
* result is only accepted by WAITS() by virtue
* of how the WAITT(), WAITS()
* and PERIODIC() calls were written. In the case
* of the WAITT() and PERIODIC() calls
* the tick count would automatically be
* changed to 1 meaning an interval of eg. 50
* millisecs in case the TICKTIME is defined
* to be 50 millisecs
*
* Liberal use of parentheses is made in the
* following macros in case the arguments
* might be expressions
*
*****
*/

```

```

#define OS_WAITT_A(M,S,ms) OS_WAITT((uint)((60000*(##M)
+ 1000*(##S) + (##ms))/TICKTIME))

#define OS_WAITS_A(M,S,ms) OS_WAITS((uint)((60000*(##M)
+ 1000*(##S) + (##ms))/TICKTIME))

#define OS_PERIODIC_A(M,S,ms) OS_PERIODIC((uint)
((60000*(##M) + 1000*(##S)+(##ms)) /TICKTIME))
/*
*****
****

*/
/* 
* Other functions used internally by the RTOS :
* -----
*/
void QShift (void);                                // Task swapping function
void RTOS_Timer_Int (void);                      // RTOS Scheduler ISR
void Xtra_Int (uchar task_intflag); // Function used by ISRs other
                                         // than the RTOS Scheduler

#if (!STAND_ALONE_ISR_00)
void Xtra_Int_0 (void);                         // External Interrupt 0 ISR
#endif

#if ( (TICK_TIMER != 0) && (!STAND_ALONE_ISR_01) )
void Xtra_Int_1 (void);                         // Timer 0 ISR
#endif

#if (!STAND_ALONE_ISR_02)
void Xtra_Int_2 (void);                         // External Interrupt 1 ISR
#endif

#if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )
void Xtra_Int_3 (void);                         // Timer 1 ISR
#endif

#if (!STAND_ALONE_ISR_04)

```

```
void Xtra_Int_4 (void);           // Serial Port ISR
#endif

#if ( (TICK_TIMER != 2 ) && (!STAND_ALONE_ISR_05) )
void Xtra_Int_5 (void);           // Timer 2 ISR
#endif

#if (!STAND_ALONE_ISR_06)
void Xtra_Int_6 (void);
#endif

#if (!STAND_ALONE_ISR_07)
void Xtra_Int_7 (void);
#endif

#if (!STAND_ALONE_ISR_08)
void Xtra_Int_8 (void);
#endif

#if (!STAND_ALONE_ISR_09)
void Xtra_Int_9 (void);
#endif

#if (!STAND_ALONE_ISR_10)
void Xtra_Int_10 (void);
#endif

#if (!STAND_ALONE_ISR_11)
void Xtra_Int_11 (void);
#endif

#if (!STAND_ALONE_ISR_12)
void Xtra_Int_12 (void);
#endif

#if ((TICK_TIMER != 3 ) && (!STAND_ALONE_ISR_14) )
void Xtra_Int_14 (void);          // Timer 3 isr
#endif
```

```
#if (!STAND_ALONE_ISR_15)
void Xtra_Int_15 (void);
#endif

#if ((TICK_TIMER != 4) && (!STAND_ALONE_ISR_16) )
void Xtra_Int_16 (void); // Timer 4 isr
#endif

#if (!STAND_ALONE_ISR_17)
void Xtra_Int_17 (void);
#endif

#if (!STAND_ALONE_ISR_18)
void Xtra_Int_18 (void);
#endif

#if (!STAND_ALONE_ISR_19)
void Xtra_Int_19 (void);
#endif

#if (!STAND_ALONE_ISR_20)
void Xtra_Int_20 (void);
#endif

/*
*****
*/
#endif // __PAULOS_F040_H__
```

A.3 Startup_PaulOS_F040.A51

```
=====
$NOMOD51
; Startup_PaulOS_F040.A51
;-----
; This file is part of the C51 Compiler package
; Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
;-----
; STARTUP.A51: This code is executed after processor reset.
;
; To translate this file use A51 with the following invocation:
;
;     A51 STARTUP.A51
;
; To link the modified STARTUP.OBJ file to your
; application use the following
; BL51 invocation:
;
;     BL51 <your object file list>, STARTUP.OBJ <controls>
;
;-----
;
; User-defined Power-On Initialization of Memory
;
; With the following EQU statements the initialization of memory
; at processor reset can be defined:
;
;           ; the absolute start-address of IDATA memory is always 0
IDATALEN      EQU    100H      ;the length of IDATA memory in bytes.
;
XDATASTART    EQU    0H        ; the absolute
start-address of XDATA memory
XDATALEN      EQU    4096     ; the length of XDATA memory in bytes.
;
PDATASTART    EQU    0H        ; the absolute
start-address of PDATA memory
PDATALEN      EQU    0H        ; the length of PDATA memory in bytes.
;
; Notes: The IDATA space overlaps physically
;         the DATA and BIT areas of the
```

```
;      8051 CPU. At minimum the memory space occupied from the C51
;      run-time routines must be set to zero.
;-----
;
; Reentrant Stack Initialization
;
; The following EQU statements define the stack pointer for reentrant
; functions and initialized it:
;
; Stack Space for reentrant functions in the SMALL model.
IBPSTACK      EQU      0          ; set to 1 if small reentrant is used.
IBPSTACKTOP   EQU      0FFH+1    ; set top of stack to
                                ; highest location+1.
;
; Stack Space for reentrant functions in the LARGE model.
XBPSTACK      EQU      0          ; set to 1 if large reentrant is used.
XBPSTACKTOP   EQU      0FFFFH+1; set top of stack to
                                ; highest location+1.
;
; Stack Space for reentrant functions in the COMPACT model.
PBPSTACK      EQU      0          ; set to 1 if compact
                                ; reentrant is used.
PBPSTACKTOP   EQU      0FFFFH+1; set top of stack to
                                ; highest location+1.
;
;-----
;
; Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
; The following EQU statements define the xdata page used for pdata
; variables. The EQU PPAGE must conform with the PPAGE control used
; in the linker invocation.
;
PPAGEENABLE   EQU      0          ; set to 1 if pdata object are used.
;
PPAGE         EQU      0          ; define PPAGE number.
;
;PPAGE_SFR DATA 0A0H ; SFR that supplies uppermost address byte
; (most 8051 variants use P2 as uppermost address byte)
;
;
```

```
; Standard SFR Symbols
ACC      DATA      0E0H
B        DATA      0F0H
SP       DATA      81H
DPL      DATA      82H
DPH      DATA      83H

NAME ?C_STARTUP

?C_C51STARTUP SEGMENT CODE
?STACK         SEGMENT IDATA

#include "Paulos_F040_Parameters.h"

MAINSTACK:    RSEG      ?STACK
                DS        STACKSIZE

                EXTRN CODE (?C_START)
                PUBLIC C_STARTUP
                PUBLIC MAINSTACK

?C_STARTUP:   CSEG      AT      0
                LJMP     STARTUP1

                RSEG      ?C_C51STARTUP

STARTUP1:
IF IDATALEN <> 0
                MOV      R0, #IDATALEN - 1
                CLR      A
IDATALOOP:    MOV      @R0, A
                DJNZ    R0, IDATALOOP

ENDIF

IF XDATALEN <> 0
                MOV      DPTR, #XDATASTART
                MOV      R7, #LOW (XDATALEN)
IF (LOW (XDATALEN)) <> 0
```

```

        MOV      R6, #(HIGH (XDATALEN)) +1
ELSE
        MOV      R6, #HIGH (XDATALEN)
ENDIF
        CLR      A
XDATALOOP: MOVX    @DPTR, A
        INC      DPTR
        DJNZ    R7, XDATALOOP
        DJNZ    R6, XDATALOOP
ENDIF
IF PPAGEENABLE <> 0
        MOV      PPAGE_SFR, #PPAGE
ENDIF
IF PDATALEN <> 0
        MOV      R0, #LOW (PDATASTART)
        MOV      R7, #LOW (PDATALEN)
        CLR      A
PDATALOOP: MOVX    @R0, A
        INC      R0
        DJNZ    R7, PDATALOOP
ENDIF

IF IBPSTACK <> 0
EXTRN DATA (?C_IBP)
        MOV      ?C_IBP, #LOW IBPSTACKTOP
ENDIF
IF XBPSTACK <> 0
EXTRN DATA (?C_XBP)
        MOV      ?C_XBP, #HIGH XBPSTACKTOP
        MOV      ?C_XBP+1, #LOW XBPSTACKTOP
ENDIF

IF PBPSTACK <> 0
EXTRN DATA (?C_PBP)
        MOV      ?C_PBP, #LOW PBPSTACKTOP
ENDIF

        MOV      SP, #?STACK-1
; This code is required if you use L51_BANK.A51 with Banking Mode 4
; EXTRN CODE (?B_SWITCH0)

```

```
;           CALL      ?B_SWITCH0      ; init bank
mechanism to code bank 0

LJMP  ?C_START

END
```

The image shows a woman with long blonde hair, wearing a blue tank top, smiling while wearing a black VR headset. To her right is an orange rectangular advertisement for MT Højgaard. The ad features the company logo (a stylized 'M' and 'H') and the text 'MT Højgaard'. Below this, in large white capital letters, is the slogan 'BEDRE LØSNINGER'. Underneath the slogan, there is a block of Danish text: 'I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?' At the bottom of the ad, the website 'mth.dk/vorestilgang' is listed.

A.4 PaulOS_F040.c

```
/*
*****
*          PaulOS_F040.c           RTOS KERNEL SOURCE CODE
*
* Co-Operative RTOS written in C by Ing. Paul P. Debono :
* -----
*
* For use with the 8051 family of microcontrollers
*
* Notes:
*
* Timer to use for the RTOS ticks is user
* selectable, Timer 0, 1, 2, 3 or 4
*
* Assign the correct values to 'TICK_TIMER', 'CPU', 'MAINSTACK'
* and 'NOOFTASKS' in PaulOS_F040.H
*
* If it is noticed that timing parameters are not
* being met well - the system's TICKTIME
* can be modified or tweaked by changing the
* value 'TWEAK' in PaulOS_F040.H
*
*
* File      : PaulOS_F040.C
* Revision  : 10
* Date      : FEBRUARY 2015
* By        : Paul P. Debono
*
*                      B. Eng. (Hons.) Elec. Course
*                      University Of Malta
*
*****
*/
/*          INCLUDES
*****
*/

```

```
#include "C8051F040.h"           /* special function registers 8052 */
#include "Paulos_F040.h"          /* RTOS system calls definitions
                                         (IN PROJECT DIRECTORY) */

/*
*****STRUCTURE DEFINITIONS*****
*/
struct task_param
{
    uchar stackptr;
    uchar flags;
    uchar intnum;
    uint timeout;
    uint interval_count;
    uint interval_reload;
    char stack[STACKSIZE];
};

struct task_param xdata task[NOOFTASKS + 1];

/*
*****GLOBAL VARIABLES*****
*/
uchar data * data ReadyQTop;      // Address of last ready task
uchar data Running;               // Number of current task
bit bdata IntFlag;                // Flag indicating a task
                                   // waiting for an
                                   // interrupt was found
bit bdata TinQFlag;               // Flag indicating that a task timed out
bit bdata Priority;                // Flag indicating whether
                                   // priority is enabled
                                   // or disabled
```

```

uchar data ReadyQ[NOOFTASKS + 2]; // Queue s tack for tasks
                                    ready to run

/*
*****
***** FUNCTION DEFINITIONS
*****
*/
/* Function name : OS_INIT_RTOS
 * Function type : Initialisation System call
 * Description   : This system call initialises the RTOS variables,
 *                  task SPs and enables any required
 *                  interrupts
 * Arguments     : ticktimer fake parameter left here for compatibility
 * Returns       : None
 */
void OS_INIT_RTOS(uchar TickTimer)
{
    uchar xdata i,j;
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page

    TickTimer = TickTimer; /* parameter not used here, just
                           to keep compatibility */

#if (TICK_TIMER == 0)
#message "Using Timer 0 as the Tick Timer"

```

```

SFRPAGE = TIMER01_PAGE;
IE &= 0x7F;                                // leave interrupt settings as they were
ET0 = 1;                                     /* Set up 8051 IE register,
                                                using timer 0 int */
PT0 = 1;                                     /* Assign scheduler interrupt
                                                high priority */

#elif (TICK_TIMER == 1)
#message "Using Timer 1 as the Tick Timer"
SFRPAGE = TIMER01_PAGE;
IE &= 0x7F;                                /* Set up 8051 IE register, using timer 1 */
ET1 = 1;                                     /* Assign scheduler interrupt
                                                high priority */

#elif (TICK_TIMER == 2)
#message "Using Timer 2 as the Tick Timer"
SFRPAGE = TMR2_PAGE;
IE &= 0x7F;                                /* Set up 8051 IE register, using timer 2 */
ET2 = 1;                                     /* Assign scheduler interrupt high priority */

#elif (TICK_TIMER == 3)
#message "Using Timer 3 as the Tick Timer"
SFRPAGE = TMR3_PAGE;
EIE2 |= ET3;                                 /* Set up 8051 IE register,
                                                using timer 3 */
EIP2 |= PT3;                                 /* Assign scheduler interrupt
                                                high priority */

#elif (TICK_TIMER == 4)
#message "Using Timer 4 as the Tick Timer"
SFRPAGE = TMR4_PAGE;
EIE2 |= ET4;                                 /* Set up 8051 IE register,
                                                using timer 4 */
EIP2 |= PT4;                                 /* Assign scheduler interrupt
                                                high priority */

#endif

Running = IDLE_TASK;    /* Set idle task as the running task */

```

```

for (i = 0; i < NOOFTASKS; i++)
{
    task[i].timeout = NOT_TIMING; /* Initialise task timeouts, */
    task[i].interval_count = NOT_TIMING; /* periodic interval
                                         count */
    task[i].interval_reload = NOT_TIMING; /* and reload
                                         variables. */
    task[i].intnum = NO_INTERRUPT; /* not waiting for any
                                    interrupt. */
    ReadyQ[i] = IDLE_TASK; /* Fill the READY queue with */
} /* with the idle task */

ReadyQ[NOOFTASKS] = IDLE_TASK;
ReadyQ[NOOFTASKS + 1] = IDLE_TASK;

ReadyQTop = ReadyQ; /* Pointer to last task
                      made to point to */
/* base of the queue. */

for (i = 0; i < NOOFTASKS + 1; i++)
{
    task[i].stackptr = MAINSTACK + 2; /* Initialise task
                                         SP values */
    task[i].flags = 0; /* Initialise task status bytes */
    for(j=0; j<STACKSIZE; j++) task[i].stack[j]
        = 0; /* clear all ext. stack area */
}
SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

/*
*****
*/
/*
*****
*
* Function name : OS_CREATE_TASK
*
* Function type : Initialisation System call
*

```

```

* Description   : This system call is used in the main
                  program for each task to be created *
                  * for use in the application.
*
* Arguments     : task          Represents the task number (1st
                                task is numbered as 0).
*
*                 taskadd       Represents the task's start
                                address, which in the C
                                environment, would simply be
                                the name of the procedure
*
* Returns       : None
*
***** ****
*/

```

```

void OS_CREATE_TASK(uchar tasknum, uint taskadd)
{
    ReadyQTop++;           /* Task is added to next available */
    *ReadyQTop = tasknum;   /* position in the READY queue.      */

    task[tasknum].stack[0] = taskadd % 256;
    task[tasknum].stack[1] = taskadd / 256;

}

/*
***** ****
*/

```

```

/*
***** ****
* Function name : OS_RTOS_GO
*
* Function type : Initialisation System call
*
* Description   : This system calls is used to start the
                  RTOS going such that it supervises
                  the application processes.
*/

```

```
*  
* Arguments      : prior    Determines whether tasks ready  
*                   to be executed are sorted  
*  
*                   prior to processing or not. If  
*                   prior = 0 a FIFO queue function  
*                   is implied, if prior = 1 the queue  
*                   is sorted by task number in  
*                   ascending order, as a higher  
*                   priority is associated with  
*                   smaller task number (task 0 would  
*                   have the highest priority),  
*                   such that the first task in the  
*                   queue, which would eventually  
*                   run, would be the one with the  
*                   smallest task number having  
*                   highest priority.  
*  
* Returns       : None  
*  
*****  
*/  
  
void OS_RTOS_GO(uchar prior)  
{  
    char SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page  
  
    if (prior == 1)          /* Checks if tasks priorities */  
        Priority = 1;        /* are to be enabled */  
    else  
        Priority = 0;  
#if (TICK_TIMER == 0)  
    /* Configure Timer 0 in 16-bit timer mode 1 for the 8051 */  
    SFRPAGE = TIMER01_PAGE;  
    TMOD &= 0xF0;           /* Clear T0 mode control,  
                           leaving T1 untouched */  
    TMOD |= 0x01;           /* Set T0 mode control to mode 1 */  
    CKCON &= 0xF0;           /* Use sysclk/12 (T0M = 0) */  
    TR0 = 1;                /* Start timer 0 */  
    TF0 = 1;                /* Cause first interrupt immediately */
```

```
#elif (TICK_TIMER == 1)
    /* Configure Timer 1 in 16-bit timer mode 1 for the 8051      */
    SFRPAGE = TIMER01_PAGE;
    TMOD &= 0x0F;           /* Clear T1 mode control,
                                leaving T0 untouched */
    TMOD |= 0x10;          /* Set T1 mode control */
    CKCON &= 0xE8;          /* Use sysclk/12 (T1M = 0) */
    TR1 = 1;                /* Start timer 1 */
    TF1 = 1;                /* Cause first interrupt immediately */

#elif (TICK_TIMER == 2)
    SFRPAGE = TMR2_PAGE;
    RCAP2 = BASIC_TICK;      /* Configures Timer 2 in 16-bit      */
                            /* auto-reload mode */
    TMR2CF = 0x00;          /* Use sysclk/12 */
    TMR2CN = 0x84;          /* TR2 = TF2 = 1, causes first
                                interrupt immediately */

#elif (TICK_TIMER == 3)
    SFRPAGE = TMR3_PAGE;
    RCAP3 = BASIC_TICK;      /* Configures Timer 3 in 16-bit
                                auto-reload mode */
    TMR3CF = 0x00;          /* Use sysclk/12 */
    TMR3CN = 0x84;          /* TR3 = TF3 = 1, causes first
                                interrupt immediately */

#elif (TICK_TIMER == 4)
    SFRPAGE = TMR4_PAGE;
    RCAP4 = BASIC_TICK;      /* Configures Timer 4 in 16-bit
                                auto-reload mode */
    TMR4CF = 0x00;          /* Use sysclk/12 */
    TMR4CN = 0x84;          /* TR4 = TF4 = 1, causes first
                                interrupt immediately */

#endif

    SFRPAGE = SFRPAGE_SAVE;        // Restore SFR page
    TinQFlag = 1; /* Signals scheduler that tasks have been */
                   /* added to the queue. */
```

```
EA = 1;          /* Interrupts are enabled, starting the
                   RTOS                                         */
}

/*
***** *****
* Function name : OS_RUNNING_TASK_ID
*
* Function type : Inter-task Communication System call
*
* Description   : This system call is used to check
                  to get the number of the
                  current task.
*
* Arguments     : None
*
* Returns       : Number of currently running task
                  from which it must be called
*
***** *****
*/
uchar OS_RUNNING_TASK_ID(void)
{
    return (Running);
}

/*
***** *****
* Function name : OS_SCHECK
*
* Function type : Inter-task Communication System call
*
* Description   : This system call is used to check if the
                  current task has its signal set. It tests
```

```
*                      whether there was any signal sent
*                      to it by some other task.
*
* Arguments      : None
*
* Returns       : 1 if its signal bit is set, 0 if not set
*
***** */
bit OS_SCHECK(void)
{
    EA = 0;
    if (task[Running].flags & SIGS_Flag) /* If a signal is already */
    {
        /* present it's cleared */
        task[Running].flags &= ~SIGS_Flag; /* and a 1 is returned. */
        EA = 1;
        return 1;
    }
    else /* If a signal is not present,
           0 is returned */
    {
        EA = 1;
        return 0;
    }
}

/*
*****
*/
/*
*****
* Function name : OS_SIGNAL_TASK
*
* Function type : Inter-task Communication System call
*
```

```

* Description    : This system call is used to send
                  a signal to another task.
*
* Arguments      : task    Represents the task to which a
                  signal is required to be sent.
*
* Returns        : None
*
*****
*/
void OS_SIGNAL_TASK(uchar tasknum)
{
    EA = 0;
    if (task[tasknum].flags & SIGW_Flag)
    {
        task[tasknum].flags &= ~SIGS_Flag;           /* If a task has
                                                       been already
                                                       waiting */
        task[tasknum].flags &= ~SIGW_Flag;           /* for a signal, the
                                                       task no */
        task[tasknum].timeout = NOT_TIMING;          /* longer has to
                                                       wait and is */
        ReadyQTop++;                                /* added to the READY
                                                       queue. */
        *ReadyQTop = tasknum;
        TinQFlag = 1;
        EA = 1;
    }
    else                                         /* If it was not waiting, its */
        task[tasknum].flags |= SIGS_Flag;          /* signal sent bit is
                                                       set */
    EA = 1;
}

/*
*****
*/
/*
*****

```

```

/*
 * Function name : OS_WAITS
 *
 * Function type : Event-Waiting System call
 *
 * Description    : This system call causes a task to wait for a
 *                  signal to arrive within a given number of
 *                  RTOS ticks. If the signal is already
 *                  present, the task continues to execute.
 *
 * Arguments       : ticks      Represents the number of ticks
 *                   for which the task will
 *                   wait for a signal to arrive.
 *                   Valid range for this
 *                   argument is 0 to 4294967295.
 *                   A value of 0 means
 *                   waiting forever for a
 *                   signal to arrive.
 *
 * Returns        : None
 *
***** */
void OS_WAITS (uint ticks)
{
    EA = 0;

    if (task[Running].flags & SIGS_Flag)      /* If signal already */
    {
        /* preswent clears the */
        task[Running].flags &= ~SIGS_Flag;      /*
        signal and the task */
        EA = 1;                                /* continues to run. */
    }
    else
    {
        /* If signal is not
           present */
    }
}

```

```
task[Running].flags |= SIGW_Flag; /* the task is sent
                                   in the */
task[Running].timeout = ticks; /* waiting state,
                                by causing */
QShift();                      /* a task switch. */
}

/*
*****
*****
* Function name : OS_WAITT
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to go in
                  the waiting state for a timeout
                  period given by a defined number of RTOS ticks.
*
* Arguments      : ticks    Represents the number of ticks
                  for which the task will wait.
*                   Valid range for this parameter
                  is 1 to 4294967295.
*
*                   A zero waiting time parameter is
                  set to 1 by the RTOS itself,
                  since a zero effectively kills
                  the task, making it wait
                  forever.
*
* Returns        : None
*****
*/
void OS_WAITT (uint ticks)
{
```

```

EA = 0;

if (ticks == 0)          /* set ticks to wait at
                           least for 1 tick, */
/* if 0 is entered by mistake */
    ticks = 1;           /* Task's timeout variable is updated */
task[Running].timeout = ticks; /* and the task then enters
                           the */
                           /* waiting state.      */
QShift();                /* waiting state.      */
}

/*
*****
***** Function name : OS_WAITP
*
* Function type : Event-Waiting System call
*
* Description   : This system call is used by a task to
                  wait for the end of its periodic
                  interval.
*                  If the interval has already passed,
                  the task continues to execute.
*
* Arguments     : None
*
* Returns       : None
*
*****
*/
void OS_WAITP(void)
{
    EA = 0;
    if (task[Running].flags & SIGV_Flag) /* If the periodic */
    {

```

```
/* interval time has */
task[Running].flags &= ~SIGV_Flag;      /* has elapsed, the */
EA = 1;                                /* task continues to */
                                         /* execute. */

}

else
{

    /* Else the task */
    task[Running].flags |= SIGV_Flag;     /* enters the waiting */
QShift();                                /* state. */

}

/*
*****
***** Function name : OS_PERIODIC
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to repeat
*                  its function every given number of
*                  RTOS ticks.
*
* Arguments      : ticks             Represents the length of the
*                  periodic interval in terms
*                  of RTOS ticks, after which
*                  the task repeats itself.
*
*                  Valid range for this parameter
*                  is 1 to 4294967295.
*
* Returns        : None
*
*****
*/
```

```
void OS_PERIODIC (uint ticks)
{
    EA = 0;
    if (ticks == 0)
        ticks = 1;
    task[Running].interval_reload = ticks; /* Task's periodic
                                               interval count      */
    task[Running].interval_count = ticks; /* and reload variables
                                               are          */
    EA = 1;                                /* initialised. */
}

/*
*****
*****
*/
/* Function name : OS_WAITI
*
* Function type : Event-Waiting System call
*
* Description   : This system call causes a task to wait
                  for a given event (interrupt).
*                  It identifies for which interrupt the
                  task has to wait. Once identified
                  the task's appropriate flag is set and the
                  task is set in the waiting state
                  by causing a task swap - the task would
                  wait indefinitely for the interrupt
                  as its timeout variable would be set
                  to 0 (NOT_TIMING) either during
                  initialisation of the RTOS or after
                  expiry of its timeout period due to
                  other prior invocations of wait-
                  inducing system calls.
*
* Arguments     : intnum  Represents the interrupt number
                  associated with the given
```

```
*                      interrupt for which the calling
*                      task intends to wait
*
* Returns : None
*
*****
*/
void OS_WAITI(uchar intnum)
{
    EA = 0;

    switch (intnum)
    {
#if (!STAND_ALONE_ISR_00)
        case 0:                  /* Interrupt number 0 */
            task[Running].intnum = IE0_INT; /* Task made to wait for */
            QShift();                /* external
                                         interrupt 0 */
            break;
#endif

#if ( (TICK_TIMER != 0) && (!STAND_ALONE_ISR_01) )
        case 1:                  /* Interrupt number 1 */
            task[Running].intnum = TF0_INT; /* Task made to wait for */
            QShift();                /* timer 0 interrupt */
            break;
#endif

#if (!STAND_ALONE_ISR_02)
        case 2:                  /* Interrupt number 2 */
            task[Running].intnum = IE1_INT; /* Task made to wait for */
            QShift();                /* external
                                         interrupt 1 */
            break;
#endif

#if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )
        case 3:                  /* Interrupt number 3 */
            task[Running].intnum = TF1_INT; /* Task made to wait for */
            QShift();                /* timer 1 interrupt */
            break;
#endif
    }
}
```

```

        task[Running].intnum = TF1_INT;           /* Task made to
                                                wait for */
        QShift();                                /* timer 1
                                                interrupt */
        break;
#endif

#if (!STAND_ALONE_ISR_04)
    case 4:                               /* Interrupt number 4 */
        task[Running].intnum = UART0_INT;     /* Task made to
                                                wait for */
        QShift();                            /* serial port interrupt */
        break;
#endif

#if ( (TICK_TIMER != 2) && (!STAND_ALONE_ISR_05) )
    case 5:                               /* Interrupt number 5 */
        task[Running].intnum = TF2_INT;      /* Task made to
                                                wait for */
        QShift();                            /* timer 1 interrupt */
        break;
#endif

#if (!STAND_ALONE_ISR_06)
    case 6:                               /* Interrupt number 6 */
        task[Running].intnum = SPIF_INT;    /* Task made to
                                                wait for */
        QShift();                            /* serial peripheral
                                                interface */
        break;
#endif

#if (!STAND_ALONE_ISR_07)
    case 7:                               /* Interrupt number 7 */
        task[Running].intnum = SI_INT;     /* Task made to wait for */
        QShift();                            /* SMBus interface */
        break;
#endif

#if (!STAND_ALONE_ISR_08)
    case 8:                               /* Interrupt number 8 */

```

```
task[Running].intnum = AD0WIN_INT;           /* Task made to
                                             wait for */
QShift();                                     /* ADC0 Window
                                             comnparator
                                             */
break;
#endif

#if (!STAND_ALONE_ISR_09)
    case 9:                                /* Interrupt number 9 */
        task[Running].intnum = CF_INT;      /* Task made to wait for */
        QShift();                         /* Programmable
                                             Counter Array */
break;
#endif

#if (!STAND_ALONE_ISR_10)
    case 10:                               /* Interrupt number 10 */
        task[Running].intnum = CP0FIF_INT; /* Task made to
                                             wait for */
        QShift();                         /* comparator 0
                                             Falling Edge */
break;
#endif

#if (!STAND_ALONE_ISR_11)
    case 11:                               /* Interrupt number 11 */
        task[Running].intnum = CP1FIF_INT; /* Task made to
                                             wait for */
        QShift();                         /* comparator 0
                                             Rising Edge */
break;
#endif

#if (!STAND_ALONE_ISR_12)
    case 12:                               /* Interrupt number 12 */
        task[Running].intnum = CP2FIF_INT; /* Task made to
                                             wait for */
        QShift();                         /* comparator 1
                                             Falling Edge */
break;
#endif
```

```
        break;  
#endif  
  
#if ( (TICK_TIMER != 3) && (!STAND_ALONE_ISR_14) )  
    case 14:                      /* Interrupt number 14 */  
        task[Running].intnum = TF3_INT;      /* Task made to  
                                                wait for */  
        QShift();                         /* timer 3 interrupt */  
        break;  
#endif  
  
#if (!STAND_ALONE_ISR_15)  
    case 15:                      /* Interrupt number 15 */  
        task[Running].intnum = AD0INT_INT;    /* Task made to  
                                                wait for */  
        QShift();                         /* ADC0 end of  
                                                conversion */  
        break;  
#endif  
  
#if ( (TICK_TIMER != 4) && (!STAND_ALONE_ISR_16) )  
    case 16:                      /* Interrupt number 16 */  
        task[Running].intnum = TF4_INT;      /* Task made to  
                                                wait for */  
        QShift();                         /* timer 4 interrupt */  
        break;  
#endif  
  
#if (!STAND_ALONE_ISR_17)  
    case 17:                      /* Interrupt number 17 */  
        task[Running].intnum = AD2WINT_INT;  /* Task made to  
                                                wait for */  
        QShift();                         /* ADC1 end of  
                                                concersion */  
        break;  
#endif  
  
#if (!STAND_ALONE_ISR_18)  
    case 18:                      /* Interrupt number 18 */
```

```

        task[Running].intnum = ADC2INT_INT;           /* Task made to
                                                   wait for */
        QShift();                                     /* external
                                                   interrupt 18 */
        break;
#endif

#if (!STAND_ALONE_ISR_19)
    case 19:                                     /* Interrupt number 19 */
        task[Running].intnum = CAN_INT;            /* Task made to
                                                   wait for */
        QShift();                                     /* external
                                                   interrupt 19 */
        break;
#endif

#if (!STAND_ALONE_ISR_20)
    case 20:                                     /* Interrupt number 20 */
        task[Running].intnum = UART1_INT;          /* Task made to
                                                   wait for */
        QShift();                                     /* UART1 interrupt */
        break;
#endif

default:
    EA = 1;
    break;
}
}

/*
*****
***** Function name : OS_DEFER
*
* Function type : Task Suspension System call
*/

```

```
*  
* Description : This system call is used to stop the  
* current task in order for the next  
* task in the queue to execute. In the  
* meantime the current task is placed  
* at the end of the queue.  
*  
* Arguments : None  
*  
* Returns : None  
*  
*****  
*/  
  
void OS_DEFER(void)  
{  
    EA = 0;  
  
    task[Running].timeout = 2; /* Task added to the waiting */  
    /* queue, for 2 tick  
     * times, prior to */  
    QShift(); /* causing a task switch. */  
}  
  
/*  
*****  
*/  
  
/*  
*****  
* Function name : OS_KILL_IT  
*  
* Function type : Task Suspension System call  
*  
* Description : This system call kills the current task,  
* by putting it permanently waiting,  
* such that it never executes again. It  
* also clears any set waiting signals  
* which the task might have.  
*/
```

```
/*
 * Arguments      : None
 *
 * Returns       : None
 *
 ****
 */
void OS_KILL_IT(void)
{
    EA = 0;
    task[Running].flags = 0;           /* Task is killed by
                                         clearing its flags */
    task[Running].intnum = NO_INTERRUPT;
    task[Running].timeout = NOT_TIMING; /* setting it to wait
                                         forever */
    task[Running].interval_count = 0; /* Task's periodic interval
                                         count is set to zero */
    QShift();                         /* and then cause a task switch. */
}

/*
 ****
 */
/* ****
 *
 * Function name : OS_RESUME_TASK
 *
 * Function type : Inter-task Communication System call
 *
 * Description   : This system call is used to resume
 *                  another KILLED task.
 *
 * Arguments     : task   Represents the task to which
 *                  is to be restarted.
 *
 * Returns       : None
 *
 ****
*/
```

```
/*
void OS_RESUME_TASK (uchar tasknum)
{
    EA = 0;

    if (task[tasknum].interval_reload != 0)
        /* if task was a KILLED periodic task */
        task[tasknum].interval_count = 1; /* resume periodic task
                                         otherwise */

    else
        task[tasknum].timeout = 1;      /* resume normal waiting
                                         task after 1 tick */

    task[Running].timeout = 2;          /* Place the current task
                                         waiting for the */

    QShift();                         /* next 2 ticks in the waiting
                                         state, thus */
                                         /* giving up its time for other tasks. */

}

/*
*****
*/
/* Function name : QShift
*
* Function type : Context Switcher (Internal function)
*
* Description   : This function is used to perform a
                  context switch i.e. task swapping
*
* Arguments     : None
*
* Returns       : None
*
*****
*/

```

```
void QShift (void) using 1
{
    uchar data i, temp;
    uchar idata * idata internal;
    uchar data * idata qtask;
    uchar data * idata qptr;

    TinQFlag = 0;
    task[Running].stackptr = temp = SP; /* Current task's SP
                                             is saved */
    internal = MAINSTACK;
    /* Current task's USED stack area is saved */
    i = 0;
    do {
        task[Running].stack[i++] = *(internal++);
    }
    while (internal<=temp);

    qtask = ReadyQ;                      /* READY queue is
                                             shifted down */
    qptr = ReadyQ + 1;
    while (qtask <= ReadyQTop)           /* by one position */
    {
        *qtask++ = *qptr++;
    }
    ReadyQTop--;                         /* Pointer to last task in
                                             queue is decremented */

    if (ReadyQTop < ReadyQ) /* Ensure that this pointer is never */
        ReadyQTop = ReadyQ; /* below the start of the READY queue */

    if (Priority == 1)                  /* If task priorities are enabled */
    {
        /* the queue is sorted such that */
        qptr = ReadyQTop; /* the highest priority task */
        while (qptr > ReadyQ) /* becomes the running task, i.e. */
        {
            /* the one having the smallest */
            /* task number. */
            /* Just one scan through the list */
    }
}
```

```
    qptr--;
    if (*qptr > *(qptr + 1))
    {
        temp = *qptr;
        *qptr = *(qptr + 1);
        *(qptr + 1) = temp;
    }
}
/* The first task in the READY queue */
Running = ReadyQ[0];           /* becomes the new running task */
                                /* The new running task's stack */
                                /* area is copied to internal RAM */
temp = task[Running].stackptr;
internal = MAINSTACK;
/* The new running task's USED stack area
   is copied to internal RAM */
i=0;
do {
    *(internal++) = task[Running].stack[i++];
}
while (internal<=temp);

SP = task[Running].stackptr;      /* The new running task's
                                SP is restored */
                                /* such that the new task
                                will execute. */
EA = 1;
}

/*
*****
*/
/*
*****
*
```

```
* Function name : Xtra_Int_0
*
* Function type : Interrupt Service Routine
*
* Description    : This is the EXT0 (\INT0) ISR whose
                  associated interrupt number is 0.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
void Xtra_Int_0 (void) interrupt 0 using 1
{
    EA = 0;
    Xtra_Int(IE0_INT);      /* Passes EXT0W for identification
                           purposes */
}

/*
*****
*
* Function name : RTOS_Timer_Int
*
* Function type : Scheduler Interrupt Service Routine
*
* Description    : This is the RTOS scheduler ISR. It
                  generates system ticks and calculates
                  any remaining waiting and periodic
                  interval time for each task.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/

```

```
#if (TICK_TIMER == 0) /* If Timer 0 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 1 using 1
{
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    uchar data k; /* Timer 0 is used */ 
    uchar data * idata q; /* for scheduling. */ 
    bit data On_Q;

    SFRPAGE = TIMER01_PAGE;
    TH0 = BASIC_TICK / 256; /* Timer registers reloaded */
    TL0 = BASIC_TICK % 256;

#elif (TICK_TIMER == 1) /* If Timer 1 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 3 using 1
{
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    uchar data k; /* Timer 1 is used */ 
    uchar data * idata q; /* for scheduling. */ 
    bit data On_Q;

    SFRPAGE = TIMER01_PAGE;
    TH1 = BASIC_TICK / 256; /* Timer registers reloaded */
    TL1 = BASIC_TICK % 256;

#elif (TICK_TIMER == 2) /* If Timer 2 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 5 using 1
{
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
    uchar data k; /* Timer 2 is used */ 
    uchar data * idata q; /* for scheduling. */ 
    bit data On_Q;

    SFRPAGE = TMR2_PAGE;
    TF2 = 0; /* Timer 2 interrupt flag is
               cleared by software */
}
```

```
#elif (TICK_TIMER == 3) /* If Timer 3 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 14 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page
    uchar data k;                  /* Timer 3 is used */
    uchar data * idata q;          /* for scheduling.           */
    bit data On_Q;

    SFRPAGE = TMR3_PAGE;
    TF3 = 0;                      /* Timer 3 interrupt flag is
                                    cleared by software */

#elif (TICK_TIMER == 4) /* If Timer 4 is used for the scheduler */
void RTOS_Timer_Int (void) interrupt 16 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;      // Save Current SFR page
    uchar data k;                  /* Timer 4 is used */
    uchar data * idata q;          /* for scheduling.           */
    bit data On_Q;

    SFRPAGE = TMR4_PAGE;
    TF4 = 0;                      /* Timer 4 interrupt flag is
                                    cleared by software */

#endif

    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page

    for (k = 0; k < NOOFTASKS; k++)
    {
        if (task[k].interval_count != NOT_
TIMING) /* Updates the tasks' */
        {
            /* periodic intervals. */
            task[k].interval_count--;
            if (task[k].interval_count == NOT_TIMING)
            {

                task[k].interval_count = task[k].interval_reload;
                if (task[k].flags & SIGV_Flag)
                {

```

```
        /* If periodic
           interval */
    task[k].flags &= ~SIGV_Flag; /* has elapsed
                                   and the */
    q = ReadyQ;                /* task has been
                                   waiting */
    On_Q = 0;                  /* for this to
                                   occur, the */
    while (q <= ReadyQTop)    /* task is placed
                                   in the */
                               /* READY queue,
                                   if it is */

    {

        if (k == *q) /* verified that the task */
                       /* does not already reside */
        {
            On_Q = 1; /* in the queue, as now */
            break;   /* the task no longer */
                       /* requires to wait. */
        }
        q++;
    }
    if (On_Q == 0)
    {
        ReadyQTop++;
        *ReadyQTop = k;
        TinQFlag = 1;
    }
}
/* If however the task */
/* was not waiting for */
/* this event, the task */
/* is not place in the */
/* the ready queue. */

else
    task[k].flags |= SIGV_Flag;
}
```

```

if (task[k].timeout != NOT_TIMING)
{
    /* Updates the tasks' */
    task[k].timeout--;           /* timeout variables. */
    if (task[k].timeout == NOT_TIMING)
    {
        ReadyQTop++;          /* If a waiting task's */
        *ReadyQTop = k;         /* timeout elapses */
        TinQFlag = 1;           /* the task is placed */
        task[k].flags &= ~SIGW_Flag; /* in the ready queue. */
    }
}
/* If the idle task is running, when tasks are */
/* known to reside in the queue, a task switch */
/* is purposely induced so these tasks can run. */
}

if ((TinQFlag == 1) && (Running == IDLE_TASK))
    QShift();
}

/*
*****
*****
*/
/* */

/*
*****
*****
*/
/* */

/*
*****
*****
*/
*
* Function name : Xtra_Int_1
*
* Function type : Interrupt Service Routine
*
* Description    : This is the Timer 0 ISR whose
*                   associated interrupt number is 1
*
* Arguments      : None
*

```

```
* Returns      : None
*
*****
*/
#endif
/* Timer 0 interrupt used for RTOS on 8051 */

void Xtra_Int_1 (void) interrupt 1 using 1
{
    EA = 0;
    Xtra_Int(TF0_INT); /* Passes TIM0W for identification purposes */
}
#endif
/*
*****
*/
/*
* Function name : Xtra_Int_2
*
* Function type : Interrupt Service Routine
*
* Description   : This is the EXT1 (\INT1) ISR whose
*                  associated interrupt number is 2.
*
* Arguments     : None
*
* Returns       : None
*
*****
*/
#endif
#if (!STAND_ALONE_ISR_02)
void Xtra_Int_2 (void) interrupt 2 using 1
{
    EA = 0;
    Xtra_Int(IE1_INT); /* Passes IE1_INT for identification purposes */
}
#endif
```

```
/*
*****
*/
/* Function name : Xtra_Int_3
* Function type : Interrupt Service Routine
* Description    : This is the Timer 1 ISR whose
                  associated interrupt number is 3.
* Arguments      : None
* Returns        : None
*****
*/
#if ( (TICK_TIMER != 1) && (!STAND_ALONE_ISR_03) )
void Xtra_Int_3 (void) interrupt 3 using 1
{
    EA = 0;
    Xtra_Int(TF1_INT); /* Passes TF1_INT for identification purposes */
}
#endif
/*
*****
*/
/* Function name : Xtra_Int_4
* Function type : Interrupt Service Routine
* Description    : This is the serial port ISR whose
                  associated interrupt number is 4.
```

```

/*
 * Arguments      : None
 *
 * Returns       : None
 *
 ****
 */#if (!STAND_ALONE_ISR_04)
void Xtra_Int_4 (void) interrupt 4 using 1
{
    EA = 0;
    Xtra_Int(UART0_INT); /* Passes UART0_INT for
                           identification purposes */
}
#endif
/*
 ****
 */
/*
 *
 * Function name : Xtra_Int_5
 *
 * Function type : Interrupt Service Routine
 *
 * Description    : This is the Timer 2 ISR whose
                   associated interrupt number is 5.
 *
 * Arguments      : None
 *
 * Returns       : None
 *
 ****
 */
#if ( (TICK_TIMER != 2) && (!STAND_ALONE_ISR_05) )

void Xtra_Int_5 (void) interrupt 5 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page

```

```
SFRPAGE = TMR2_PAGE;
EA = 0;
TF2 = 0; /* may be cleared in the task itself */
SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
Xtra_Int(TF2_INT); /* Passes TF2_INT for
identification purposes */
}

#endif

/*
*****
*
* Function name : Xtra_Int_6
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
interrupt number is 6.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#ifndef STAND_ALONE_ISR_06
void Xtra_Int_6 (void) interrupt 6 using 1
{
    EA = 0;
    Xtra_Int(SPIF_INT); /* Passes SPIF_INT for
identification purposes */
}
#endif
/*
*****
*
* Function name : Xtra_Int_7
*
* Function type : Interrupt Service Routine

```

```
/*
 * Description      : This is the ISR whose associated
 *                      interrupt number is 7.
 *
 * Arguments       : None
 *
 * Returns        : None
 *
 ****
 */
#endif (!STAND_ALONE_ISR_07)
void Xtra_Int_7 (void) interrupt 7 using 1
{
    EA = 0;
    Xtra_Int(SI_INT); /* Passes SI_INT for identification purposes */
}
#endif
/*
 ****
 *
 * Function name   : Xtra_Int_8
 *
 * Function type   : Interrupt Service Routine
 *
 * Description      : This is the ISR whose associated
 *                      interrupt number is 8.
 *
 * Arguments       : None
 *
 * Returns        : None
 *
 ****
 */
#endif (!STAND_ALONE_ISR_08)
void Xtra_Int_8 (void) interrupt 8 using 1
{
    EA = 0;
    Xtra_Int(AD0WIN_INT); /* Passes AD0WIN_
    INT for identification purposes */
}
#endif
```

```
/*
*****
*
* Function name : Xtra_Int_9
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
*                   interrupt number is 9.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#endif (!STAND_ALONE_ISR_09)
void Xtra_Int_9 (void) interrupt 9 using 1
{
    EA = 0;
    Xtra_Int(CF_INT); /* Passes CF_INT for identification
                        purposes */
}
#endif
/*
*****
*
* Function name : Xtra_Int_10
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
*                   interrupt number is 10.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#endif (!STAND_ALONE_ISR_10)
```

```
void Xtra_Int_10 (void) interrupt 10 using 1
{
    EA = 0;
    Xtra_Int(CP0FIF_INT); /* Passes CP0FIF_INT for
                           identification purposes */
}

#endif
/*
*****
*
* Function name : Xtra_Int_11
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 11.
*
* Arguments     : None
*
* Returns       : None
*
*****
*/
#ifndef STAND_ALONE_ISR_11
void Xtra_Int_11 (void) interrupt 11 using 1
{
    EA = 0;
    Xtra_Int(CP1FIF_INT); /* Passes CP1FIF_INT for
                           identification purposes */
}
#endif
/*
*****
*
* Function name : Xtra_Int_12
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 12.
```

```
/*
 * Arguments      : None
 *
 * Returns       : None
 *
 ****
 */
#ifndef (!STAND_ALONE_ISR_12)
void Xtra_Int_12 (void) interrupt 12 using 1
{
    EA = 0;
    Xtra_Int(CP2FIF_INT); /* Passes CP2FIF_INT for
                           identification purposes */
}
#endif
/*
 ****
 *
 * Function name : Xtra_Int_14
 *
 * Function type : Interrupt Service Routine
 *
 * Description   : This is the ISR whose associated
                  interrupt number is 14.
 *
 * Arguments      : None
 *
 * Returns       : None
 *
 ****
 */
#ifndef ( (TICK_TIMER != 3) && (!STAND_ALONE_ISR_14) )
void Xtra_Int_14 (void) interrupt 14 using 1
{
    char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page

    SFRPAGE = TMR3_PAGE;
    EA = 0;
    TMR3CN &= 0x7F; /* may be cleared in the task itself */
    SFRPAGE = SFRPAGE_SAVE;
}
```

```
Xtra_Int(TF3_INT);      /* Passes TF3_INT for
                           identification purposes */
}

#endif

/*
*****
*
* Function name : Xtra_Int_15
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 15.
*
* Arguments     : None
*
* Returns       : None
*
*****
*/
#endif (!STAND_ALONE_ISR_15)
void Xtra_Int_15 (void) interrupt 15 using 1
{
    EA = 0;
    Xtra_Int(AD0INT_INT); /* Passes AD0INT_INT for
                           identification purposes */
}

#endif

/*
*****
*
* Function name : Xtra_Int_16
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 16.
*
```

```
* Arguments      : None
*
* Returns       : None
*
*****
*/
#ifndef (TICK_TIMER != 4) && (!STAND_ALONE_ISR_16)
void Xtra_Int_16 (void) interrupt 16 using 1
{
    char SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page
    SFRPAGE = TMR4_PAGE;
    EA = 0;
    TMR4CN &= 0x7F; /* may be cleared in the task itself */
    SFRPAGE = SFRPAGE_SAVE;
    Xtra_Int(TF4_INT); /* Passes TF4_INT for
                           identification purposes */
}
#endif

/*
*****
*
* Function name : Xtra_Int_17
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
                  interrupt number is 17.
*
* Arguments      : None
*
* Returns       : None
*
*****
*/
#ifndef (!STAND_ALONE_ISR_17)
void Xtra_Int_17 (void) interrupt 17 using 1
{
    EA = 0;
    Xtra_Int(AD2WINT_INT); /* Passes AD2WINT_INT for
                           identification purposes */
}
```

```
}

#endif

/*
*****
* Function name : Xtra_Int_18
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
*                   interrupt number is 18.
*
* Arguments      : None
*
* Returns        : None
*
*****
*/
#if (!STAND_ALONE_ISR_18)
void Xtra_Int_18 (void) interrupt 18 using 1
{
    EA = 0;
    Xtra_Int(ADC2INT_INT);           /* Passes ADC2INT_INT for
                                         identification purposes */
}
#endif

/*
*****
* Function name : Xtra_Int_19
*
* Function type : Interrupt Service Routine
*
* Description    : This is the ISR whose associated
*                   interrupt number is 19.
*
* Arguments      : None
*
```

```
* Returns      : None
*
*****
*/
#ifndef (!STAND_ALONE_ISR_19)
void Xtra_Int_19 (void) interrupt 19 using 1
{
    EA = 0;
    Xtra_Int(CAN_INT); /* Passes CAN_INT for identification purposes */
}
#endif

/*
*****
*
* Function name : Xtra_Int_20
*
* Function type : Interrupt Service Routine
*
* Description   : This is the ISR whose associated
*                  interrupt number is 20.
*
* Arguments     : None
*
* Returns       : None
*
*****
*/
#ifndef (!STAND_ALONE_ISR_20)
void Xtra_Int_20 (void) interrupt 20 using 1
{
    EA = 0;
    Xtra_Int(UART1_INT); /* Passes UART1_INT for
                           identification purposes */
}
#endif
/*
*****
*/
```

```
/*
*****
*
* Function name : Xtra_Int
*
* Function type : Interrupt Handling (Internal function)
*
* Description    : This function performs the operations
*                   required by the previous ISRs.
*
* Arguments       : task_intflag      Represents the flag mask
*                   for a given interrupt
*                   against which the byte
*                   storing the flags of each
*                   task will be compared in
*                   order to determine
*                   whether any task has been
*                   waiting for the
*                   interrupt in question.
*
* Returns        : None
*
*****
*/
void Xtra_Int (uchar current_intnum) using 1
{
    uchar data k;

    IntFlag = 0;

    for (k = 0; k < NOOFTASKS; k++)
    {
        if (task[k].intnum == current_intnum)
        {
            task[k].intnum = NO_INTERRUPT;
            IntFlag = 1;
            task[k].timeout = NOT_TIMING; /* If it found that a task */
                                         /* has been waiting for the */
                                         /* given interrupt, it no */
                                         /* longer requires to wait */
        }
    }
}
```

```
        /* and is therefore placed */  
/* on the READY queue. */  
  
if ((IntFlag == 1) && (Running == IDLE_TASK))  
{  
    TinQFlag = 1;      /* If tasks are known to now reside in the */  
    QShift();          /* READY queue while the idle task is */  
}                      /* running, a task switch is purposely */  
                      /* induced, such that these tasks can run. */  
  
else if ((IntFlag == 1) && (Running != IDLE_TASK))  
  
{  
    /* Otherwise, the ISR exits after */  
    TinQFlag = 1;      /* interrupts are re-enabled, */  
                      /* since RTOS cannot pre-empt task */  
    EA = 1;  
}  
else EA = 1;           /* else exit normally */  
}  
  
/*  
*****  
*/
```

A.5 C8051F040.H

```
/*-----  
; Copyright (C) 2002 CYGNAL INTEGRATED PRODUCTS, INC.  
; All rights reserved.  
;  
;  
; FILE NAME : C8051F040.H  
; TARGET MCUS : C8051F040, 'F041, 'F042, 'F043  
; DESCRIPTION : Register/bit definitions for the  
; C8051F04x product family.  
;  
; REVISION 1.3  
; Added some more remarks and bit  
; definitions - P. Debono 2015  
;  
-----*/  
  
#ifndef __C8051F040_H__  
#define __C8051F040_H__  
  
/* BYTE Registers */  
sfr P0 = 0x80; /* PORT 0 */  
sfr SP = 0x81; /* STACK POINTER */  
sfr DPL = 0x82; /* DATA POINTER - LOW BYTE */  
sfr DPH = 0x83; /* DATA POINTER - HIGH BYTE */  
sfr SFRPAGE = 0x84; /* SFR PAGE SELECT */  
sfr SFRNEXT = 0x85; /* SFR STACK NEXT PAGE */  
sfr SFRLAST = 0x86; /* SFR STACK LAST PAGE */  
sfr PCON = 0x87; /* POWER CONTROL */  
sfr TCON = 0x88; /* TIMER CONTROL */  
sfr CPT0CN = 0x88; /* COMPARATOR 0 CONTROL */  
sfr CPT1CN = 0x88; /* COMPARATOR 1 CONTROL */  
sfr CPT2CN = 0x88; /* COMPARATOR 2 CONTROL */  
sfr TMOD = 0x89; /* TIMER MODE */  
sfr CPT0MD = 0x89; /* COMPARATOR 0 MODE */  
sfr CPT1MD = 0x89; /* COMPARATOR 1 MODE */  
sfr CPT2MD = 0x89; /* COMPARATOR 2 MODE */  
sfr TL0 = 0x8A; /* TIMER 0 - LOW BYTE */  
sfr OSCICN = 0x8A; /* INTERNAL OSCILLATOR CONTROL */  
sfr TL1 = 0x8B; /* TIMER 1 - LOW BYTE */  
sfr OSCICL = 0x8B; /* INTERNAL OSCILLATOR CALIBRATION */
```

```

sfr TH0          = 0x8C; /* TIMER 0 - HIGH BYTE           */
sfr OSCXCN      = 0x8C; /* EXTERNAL OSCILLATOR CONTROL        */
sfr TH1          = 0x8D; /* TIMER 1 - HIGH BYTE           */
sfr CKCON        = 0x8E; /* TIMER 0/1 CLOCK CONTROL           */
sfr PSCTL        = 0x8F; /* FLASH WRITE/ERASE CONTROL         */
sfr P1           = 0x90; /* PORT 1                         */
sfr SSTA0        = 0x91; /* UART 0 STATUS                   */
sfr SFRPGCN     = 0x96; /* SFR PAGE CONTROL                */
sfr CLKSEL        = 0x97; /* SYSTEM CLOCK SELECT             */
sfr SCON0        = 0x98; /* UART 0 CONTROL                 */
sfr SCON1        = 0x98; /* UART 1 CONTROL                 */
sfr SBUFO        = 0x99; /* UART 0 BUFFER                  */
sfr SBUF1        = 0x99; /* UART 1 BUFFER                  */
sfr SPI0CFG      = 0x9A; /* SPI 0 CONFIGURATION            */
sfr SPI0DAT      = 0x9B; /* SPI 0 DATA                     */
sfr P4MDOUT      = 0x9C; /* PORT 4 OUTPUT MODE             */
sfr SPI0CKR      = 0x9D; /* SPI 0 CLOCK RATE CONTROL       */
sfr P5MDOUT      = 0x9D; /* PORT 5 OUTPUT MODE             */
sfr P6MDOUT      = 0x9E; /* PORT 6 OUTPUT MODE             */
sfr P7MDOUT      = 0x9F; /* PORT 7 OUTPUT MODE             */
sfr P2           = 0xA0; /* PORT 2                         */
sfr EMI0TC        = 0xA1; /* EMIF TIMING CONTROL            */
sfr EMI0CN        = 0xA2; /* EMIF CONTROL                   */
sfr EMI0CF        = 0xA3; /* EMIF CONFIGURATION             */
sfr P0MDOUT      = 0xA4; /* PORT 0 OUTPUT MODE             */
sfr P1MDOUT      = 0xA5; /* PORT 1 OUTPUT MODE             */
sfr P2MDOUT      = 0xA6; /* PORT 2 OUTPUT MODE CONFIGURATION */
sfr P3MDOUT      = 0xA7; /* PORT 3 OUTPUT MODE CONFIGURATION */
sfr IE            = 0xA8; /* INTERRUPT ENABLE               */
sfr SADDR0        = 0xA9; /* UART 0 SLAVE ADDRESS           */
sfr SADDR1        = 0xA9; /* UART 1 SLAVE ADDRESS           */
sfr P1MDIN        = 0xAD; /* PORT 1 INPUT MODE              */
sfr P2MDIN        = 0xAE; /* PORT 2 INPUT MODE              */
sfr P3MDIN        = 0xAF; /* PORT 3 INPUT MODE              */
sfr P3           = 0xB0; /* PORT 3                         */
sfr FLSCL         = 0xB7; /* FLASH TIMING PRESCALAR        */
sfr FLACL         = 0xB7; /* FLASH ACCESS LIMIT             */
sfr IP            = 0xB8; /* INTERRUPT PRIORITY             */
sfr SADENO        = 0xB9; /* UART 0 SLAVE ADDRESS MASK     */
sfr AMX2CF        = 0xBA; /* ADC 2 MUX CONFIGURATION        */
sfr AMX0PRT       = 0xBD; /* ADC 0 MUX PORT PIN SELECT REGISTER */

```

```

sfr AMX0CF      = 0xBA; /* ADC 0 CONFIGURATION REGISTER          */
sfr AMX0SL      = 0xBB; /* ADC 0 AND ADC 1 MODE SELECTION           */
sfr AMX2SL      = 0xBB; /* ADC 2 MUX CHANNEL SELECTION             */
sfr ADC0CF      = 0xBC; /* ADC 0 CONFIGURATION                      */
sfr ADC2CF      = 0xBC; /* ADC 2 CONFIGURATION                      */
sfr ADC0L       = 0xBE; /* ADC 0 DATA - LOW BYTE                   */
sfr ADC2        = 0xBE; /* ADC 2 DATA - LOW BYTE                   */
sfr ADC0H       = 0xBF; /* ADC 0 DATA - HIGH BYTE                  */
sfr SMB0CN      = 0xC0; /* SMBUS 0 CONTROL                         */
sfr CAN0STA     = 0xC0; /* CAN 0 STATUS                            */
sfr SMB0STA     = 0xC1; /* SMBUS 0 STATUS                          */
sfr SMB0DAT     = 0xC2; /* SMBUS 0 DATA                           */
sfr SMB0ADR     = 0xC3; /* SMBUS 0 SLAVE ADDRESS                 */
sfr ADC0GTL     = 0xC4; /* ADC 0 GREATER-THAN REGISTER - LOW BYTE */
sfr ADC2GT      = 0xC4; /* ADC 2 GREATER-THAN REGISTER - LOW BYTE */
sfr ADC0GTH     = 0xC5; /* ADC 0 GREATER-THAN REGISTER - HIGH BYTE */
sfr ADC0LTL     = 0xC6; /* ADC 0 LESS-THAN REGISTER - LOW BYTE    */
sfr ADC2LT      = 0xC6; /* ADC 2 LESS-THAN REGISTER - LOW BYTE    */
sfr ADC0LTH     = 0xC7; /* ADC 0 LESS-THAN REGISTER - HIGH BYTE   */
sfr TMR2CN      = 0xC8; /* TIMER 2 CONTROL                         */
sfr TMR3CN      = 0xC8; /* TIMER 3 CONTROL                         */
sfr TMR4CN      = 0xC8; /* TIMER 4 CONTROL                         */
sfr P4          = 0xC8; /* PORT 4                                  */
sfr TMR2CF      = 0xC9; /* TIMER 2 CONFIGURATION                   */
sfr TMR3CF      = 0xC9; /* TIMER 3 CONFIGURATION                   */
sfr TMR4CF      = 0xC9; /* TIMER 4 CONFIGURATION                   */
sfr RCAP2L       = 0xCA; /* TIMER 2 CAPTURE REGISTER - LOW BYTE   */
sfr RCAP3L       = 0xCA; /* TIMER 3 CAPTURE REGISTER - LOW BYTE   */
sfr RCAP4L       = 0xCA; /* TIMER 4 CAPTURE REGISTER - LOW BYTE   */
sfr RCAP2H       = 0xCB; /* TIMER 2 CAPTURE REGISTER - HIGH BYTE  */
sfr RCAP3H       = 0xCB; /* TIMER 3 CAPTURE REGISTER - HIGH BYTE  */
sfr RCAP4H       = 0xCB; /* TIMER 4 CAPTURE REGISTER - HIGH BYTE  */
sfr TMR2L        = 0xCC; /* TIMER 2 - LOW BYTE                      */
sfr TMR3L        = 0xCC; /* TIMER 3 - LOW BYTE                      */
sfr TMR4L        = 0xCC; /* TIMER 4 - LOW BYTE                      */
sfr TMR2H        = 0xCD; /* TIMER 2 - HIGH BYTE                     */
sfr TMR3H        = 0xCD; /* TIMER 3 - HIGH BYTE                     */
sfr TMR4H        = 0xCD; /* TIMER 4 - HIGH BYTE                     */
sfr SMB0CR      = 0xCF; /* SMBUS 0 CLOCK RATE                     */
sfr PSW          = 0xD0; /* PROGRAM STATUS WORD                   */
sfr REF0CN      = 0xD1; /* VOLTAGE REFERENCE 0 CONTROL           */

```

```

sfr DAC0L      = 0xD2; /* DAC 0 REGISTER - LOW BYTE          */
sfr DAC1L      = 0xD2; /* DAC 1 REGISTER - LOW BYTE          */
sfr DAC0H      = 0xD3; /* DAC 0 REGISTER - HIGH BYTE         */
sfr DAC1H      = 0xD3; /* DAC 1 REGISTER - HIGH BYTE         */
sfr DAC0CN     = 0xD4; /* DAC 0 CONTROL                      */
sfr DAC1CN     = 0xD4; /* DAC 1 CONTROL                      */
sfr HVA0CN     = 0xD6; /* HVDA CONTROL REGISTER             */
sfr PCA0CN     = 0xD8; /* PCA 0 COUNTER CONTROL             */
sfr CAN0DATL   = 0xD8; /* CAN 0 DATA - LOW BYTE            */
sfr P5          = 0xD8; /* PORT 5                           */
sfr PCA0MD     = 0xD9; /* PCA 0 COUNTER MODE              */
sfr CAN0DATH   = 0xD9; /* CAN 0 DATA - HIGH BYTE           */
sfr PCA0CPM0    = 0xDA; /* PCA 0 MODULE 0 CONTROL           */
sfr CAN0ADR    = 0xDA; /* CAN 0 ADDRESS                     */
sfr PCA0CPM1    = 0xDB; /* PCA 0 MODULE 1 CONTROL           */
sfr CAN0TST    = 0xDB; /* CAN 0 TEST                         */
sfr PCA0CPM2    = 0xDC; /* PCA 0 MODULE 2 CONTROL           */
sfr PCA0CPM3    = 0xDD; /* PCA 0 MODULE 3 CONTROL           */
sfr PCA0CPM4    = 0xDE; /* PCA 0 MODULE 4 CONTROL           */
sfr PCA0CPM5    = 0xDF; /* PCA 0 MODULE 5 CONTROL           */
sfr ACC         = 0xE0; /* ACCUMULATOR                       */
sfr PCA0CPL5    = 0xE1; /* PCA 0 MODULE 5 CAPTURE/COMPARE - */
                                /* COMPARE - LOW BYTE               */
sfr XBR0        = 0xE1; /* CROSSBAR CONFIGURATION REGISTER 0 */
sfr PCA0CPH5    = 0xE2; /* PCA 0 MODULE 5 CAPTURE/COMPARE - HIGH */
                                /* BYTE                            */
sfr XBR1        = 0xE2; /* CROSSBAR CONFIGURATION REGISTER 1 */
sfr XBR2        = 0xE3; /* CROSSBAR CONFIGURATION REGISTER 2 */
sfr XBR3        = 0xE4; /* CROSSBAR CONFIGURATION REGISTER 3 */
sfr EIE1        = 0xE6; /* EXTERNAL INTERRUPT ENABLE 1       */
sfr EIE2        = 0xE7; /* EXTERNAL INTERRUPT ENABLE 2       */
sfr ADC0CN      = 0xE8; /* ADC 0 CONTROL                      */
sfr ADC2CN      = 0xE8; /* ADC 2 CONTROL                      */
sfr P6          = 0xE8; /* PORT 6                           */
sfr PCA0CPL2    = 0xE9; /* PCA 0 MODULE 2 CAPTURE/COMPARE - LOW BYTE */
sfr PCA0CPH2    = 0xEA; /* PCA 0 MODULE 2 CAPTURE/COMPARE - HIGH BYTE */
sfr PCA0CPL3    = 0xEB; /* PCA 0 MODULE 3 CAPTURE/COMPARE - LOW BYTE */
sfr PCA0CPH3    = 0xEC; /* PCA 0 MODULE 3 CAPTURE/COMPARE - HIGH BYTE */
sfr PCA0CPL4    = 0xED; /* PCA 0 MODULE 4 CAPTURE/COMPARE - LOW BYTE */
sfr PCA0CPH4    = 0xEE; /* PCA 0 MODULE 4 CAPTURE/COMPARE - HIGH BYTE */
sfr RSTSRC      = 0xEF; /* RESET SOURCE

```

```

sfr B      = 0xF0; /* B REGISTER */ */
sfr EIP1   = 0xF6; /* EXTERNAL INTERRUPT PRIORITY REGISTER 1 */ */
sfr EIP2   = 0xF7; /* EXTERNAL INTERRUPT PRIORITY REGISTER 2 */ */
sfr SPI0CN = 0xF8; /* SPI 0 CONTROL */ */
sfr CAN0CN = 0xF8; /* CAN 0 CONTROL */ */
sfr P7     = 0xF8; /* PORT 7 */ */
sfr PCA0L  = 0xF9; /* PCA 0 TIMER - LOW BYTE */ */
sfr PCA0H  = 0xFA; /* PCA 0 TIMER - HIGH BYTE */ */
sfr PCA0CPL0 = 0xFB; /* PCA 0 MODULE 0 CAPTURE/COMPARE - LOW BYTE */ */
sfr PCA0CPH0 = 0xFC; /* PCA 0 MODULE 0 CAPTURE/COMPARE - HIGH BYTE */ */
sfr PCA0CPL1 = 0xFD; /* PCA 0 MODULE 1 CAPTURE/COMPARE - LOW BYTE */ */
sfr PCA0CPH1 = 0xFE; /* PCA 0 MODULE 1 CAPTURE/COMPARE - HIGH BYTE */ */
sfr WDTCN  = 0xFF; /* WATCHDOG TIMER CONTROL */ */

/* 16-BIT sfr Definitions */
sfr16 DP      = 0x82;           // data pointer
sfr16 RCAP2   = 0xCA;          // Timer2 reload/capture value
sfr16 RCAP3   = 0xCA;          // Timer3 reload/capture value
sfr16 RCAP4   = 0xCA;          // Timer4 reload/capture value
sfr16 TMR2    = 0xCC;          // Timer2 counter/timer
sfr16 TMR3    = 0xCC;          // Timer3 counter/timer
sfr16 TMR4    = 0xCC;          // Timer4 counter/timer
sfr16 ADC0    = 0xBE;          // ADC0 data
sfr16 ADC0GT  = 0xC4;          // ADC0 greater than window
sfr16 ADC0LT  = 0xC6;          // ADC0 less than window
sfr16 DAC0    = 0xD2;          // DAC0 data
sfr16 DAC1    = 0xD2;          // DAC1 data
sfr16 CAN0DAT = 0xD8;          // CAN data window

/* BIT Registers */

/* TCON 0x88 */
sbit TF1     = TCON ^ 7;        /* TIMER 1 OVERFLOW FLAG */ */
sbit TR1     = TCON ^ 6;        /* TIMER 1 ON/OFF CONTROL */ */
sbit TF0     = TCON ^ 5;        /* TIMER 0 OVERFLOW FLAG */ */
sbit TR0     = TCON ^ 4;        /* TIMER 0 ON/OFF CONTROL */ */
sbit IE1     = TCON ^ 3;        /* EXT. INTERRUPT 1 EDGE FLAG */ */
sbit IT1     = TCON ^ 2;        /* EXT. INTERRUPT 1 TYPE */ */
sbit IE      = TCON ^ 1;        /* EXT. INTERRUPT 0 EDGE FLAG */ */
sbit IT0     = TCON ^ 0;        /* EXT. INTERRUPT 0 TYPE */ */

```

```

/* CPT0CN 0x88 */
sbit CP0EN    = CPT0CN ^ 7;          /* COMPARATOR 0 ENABLE           */
sbit CP0OUT   = CPT0CN ^ 6;          /* COMPARATOR 0 OUTPUT            */
sbit CP0RIF   = CPT0CN ^ 5;          /* COMPARATOR 0 RISING EDGE      */
                                         INTERRUPT                      */
                                         /*                                     */
sbit CP0FIF   = CPT0CN ^ 4;          /* COMPARATOR 0 FALLING EDGE     */
                                         INTERRUPT                      */
                                         /*                                     */
sbit CP0HYP1  = CPT0CN ^ 3;          /* COMPARATOR 0 POSITIVE          */
                                         HYSTERESIS 1                   */
sbit CP0HYP1  = CPT0CN ^ 3;          /* COMPARATOR 0 POSITIVE HYSTERESIS */
                                         1 */ sbit CP0HYP0 = CPT0CN
                                         ^ 2;          /* COMPARATOR 0
                                         POSITIVE HYSTERESIS 0 */
                                         /*                                     */
sbit CP0HYN1  = CPT0CN ^ 1;          /* COMPARATOR 0 NEGATIVE          */
                                         HYSTERESIS 1                   */
sbit CP0HYN0  = CPT0CN ^ 0;          /* COMPARATOR 0 NEGATIVE          */
                                         HYSTERESIS 0                   */

/* CPT1CN 0x88 */
sbit CP1EN    = CPT1CN ^ 7;          /* COMPARATOR 1 ENABLE           */
sbit CP1OUT   = CPT1CN ^ 6;          /* COMPARATOR 1 OUTPUT            */
sbit CP1RIF   = CPT1CN ^ 5;          /* COMPARATOR 1 RISING EDGE      */
                                         INTERRUPT                      */
                                         /*                                     */
sbit CP1FIF   = CPT1CN ^ 4;          /* COMPARATOR 1 FALLING EDGE     */
                                         EDGE INTERRUPT */
                                         /*                                     */
sbit CP1HYP1  = CPT1CN ^ 3;          /* COMPARATOR 1 POSITIVE          */
                                         HYSTERESIS 1                   */
sbit CP1HYP0  = CPT1CN ^ 2;          /* COMPARATOR 1 POSITIVE          */
                                         HYSTERESIS 0                   */
sbit CP1HYN1  = CPT1CN ^ 1;          /* COMPARATOR 1 NEGATIVE          */
                                         HYSTERESIS 1                   */
sbit CP1HYN0  = CPT1CN ^ 0;          /* COMPARATOR 1 NEGATIVE          */
                                         HYSTERESIS 0                   */

/* CPT2CN 0x88 */
sbit CP2EN    = CPT2CN ^ 7;          /* COMPARATOR 2 ENABLE           */
sbit CP2OUT   = CPT2CN ^ 6;          /* COMPARATOR 2 OUTPUT            */
sbit CP2RIF   = CPT2CN ^ 5;          /* COMPARATOR 2 RISING EDGE      */
                                         EDGE INTERRUPT */
                                         /*                                     */
sbit CP2FIF   = CPT2CN ^ 4;          /* COMPARATOR 2 FALLING EDGE     */
                                         /*                                     */

```

```

sbit CP2HYP1 = CPT2CN ^ 3;      /* COMPARATOR 2 POSITIVE
                                 HYSTERESIS 1 */  

sbit CP2HYP0 = CPT2CN ^ 2;      /* COMPARATOR 2 POSITIVE
                                 HYSTERESIS 0 */  

sbit CP2HYN1 = CPT2CN ^ 1;      /* COMPARATOR 2 NEGATIVE
                                 HYSTERESIS 1 */  

sbit CP2HYN0 = CPT2CN ^ 0;      /* COMPARATOR 2 NEGATIVE
                                 HYSTERESIS 0 */  
  

/* SCON0 0x98 */  

sbit SM00 = SCON0 ^ 7;          /* UART 0 MODE 0 */  

sbit SM10 = SCON0 ^ 6;          /* UART 0 MODE 1 */  

sbit SM20 = SCON0 ^ 5;          /* UART 0 MCE */  

sbit REN0 = SCON0 ^ 4;          /* UART 0 RX ENABLE */  

sbit TB80 = SCON0 ^ 3;          /* UART 0 TX BIT 8 */  

sbit RB80 = SCON0 ^ 2;          /* UART 0 RX BIT 8 */  

sbit TI0 = SCON0 ^ 1;           /* UART 0 TX INTERRUPT FLAG */  

sbit RI0 = SCON0 ^ 0;           /* UART 0 RX INTERRUPT FLAG */  
  

/* SCON1 0x98 */  

sbit S1MODE = SCON1 ^ 7;         /* UART 1 MODE */  

sbit MCE1 = SCON1 ^ 5;           /* UART 1 MCE */  

sbit REN1 = SCON1 ^ 4;           /* UART 1 RX ENABLE */  

sbit TB81 = SCON1 ^ 3;           /* UART 1 TX BIT 8 */  

sbit RB81 = SCON1 ^ 2;           /* UART 1 RX BIT 8 */  

sbit TI1 = SCON1 ^ 1;            /* UART 1 TX INTERRUPT FLAG */  

sbit RI1 = SCON1 ^ 0;            /* UART 1 RX INTERRUPT FLAG */  
  

/* IE 0xA8 */  

sbit EA = IE ^ 7;                /* GLOBAL INTERRUPT ENABLE */  

sbit IEGF0 = IE ^ 6;              /* */  

sbit ET2 = IE ^ 5;                /* TIMER 2 INTERRUPT ENABLE */  

sbit ES0 = IE ^ 4;                /* UART0 INTERRUPT ENABLE */  

sbit ET1 = IE ^ 3;                /* TIMER 1 INTERRUPT ENABLE */  

sbit EX1 = IE ^ 2;                /* EXTERNAL INTERRUPT 1 ENABLE */  

sbit ET0 = IE ^ 1;                /* TIMER 0 INTERRUPT ENABLE */  

sbit EX0 = IE ^ 0;                /* EXTERNAL INTERRUPT 0 ENABLE */  
  

/* IP 0xB8 */  

sbit PT2 = IP ^ 5;                /* TIMER 2 PRIORITY */  


```

```

sbit PS0      = IP ^ 4;          /* SERIAL PORT UART0 PRIORITY      */
sbit PT1      = IP ^ 3;          /* TIMER 1 PRIORITY                 */
sbit PX1      = IP ^ 2;          /* EXTERNAL INTERRUPT 1 PRIORITY    */
sbit PT0      = IP ^ 1;          /* TIMER 0 PRIORITY                 */
sbit PX0      = IP ^ 0;          /* EXTERNAL INTERRUPT 0 PRIORITY    */

/* SMB0CN 0xC0 */
sbit SMB0BUSY     = SMB0CN ^ 7; /* SMBUS 0 BUSY                     */
sbit ENSMB       = SMB0CN ^ 6; /* SMBUS 0 ENABLE                   */
sbit STA         = SMB0CN ^ 5; /* SMBUS 0 START FLAG               */
sbit STO         = SMB0CN ^ 4; /* SMBUS 0 STOP FLAG                */
sbit SI          = SMB0CN ^ 3; /* SMBUS 0 INTERRUPT PENDING FLAG  */
sbit AA          = SMB0CN ^ 2; /* SMBUS 0 ASSERT/ACKNOWLEDGE FLAG */
sbit FTE         = SMB0CN ^ 1; /* SMBUS 0 FREE TIMER ENABLE        */
sbit TOE         = SMB0CN ^ 0; /* SMBUS 0 TIMEOUT ENABLE          */

/* CAN0STA 0xC0 */
sbit BOFF        = CAN0STA ^ 7; /* Bus Off Status                  */
sbit EWARN        = CAN0STA ^ 6; /* Warning Status                 */
sbit EPASS        = CAN0STA ^ 5; /* Error Passive                  */
sbit RXOK         = CAN0STA ^ 4; /* Received Message Successfully   */
sbit TXOK         = CAN0STA ^ 3; /* Transmit a Message Successfully */
sbit LEC2         = CAN0STA ^ 2; /* Last error code bit 2           */
sbit LEC1         = CAN0STA ^ 1; /* Last error code bit 1           */
sbit LEC0         = CAN0STA ^ 0; /* Last error code bit 0           */

/* TMR2CN 0xC8 */
sbit TF2          = TMR2CN ^ 7; /* TIMER 2 OVERFLOW FLAG           */
sbit EXF2         = TMR2CN ^ 6; /* TIMER 2 EXTERNAL FLAG           */
sbit EXEN2        = TMR2CN ^ 3; /* TIMER 2 EXTERNAL ENABLE FLAG   */
sbit TR2          = TMR2CN ^ 2; /* TIMER 2 ON/OFF CONTROL          */
sbit CT2          = TMR2CN ^ 1; /* TIMER 2 COUNTER SELECT          */
sbit CPRL2        = TMR2CN ^ 0; /* TIMER 2 CAPTURE SELECT          */

/* TMR3CN 0xC8 */
sbit TF3          = TMR3CN ^ 7; /* TIMER 3 OVERFLOW FLAG           */
sbit EXF3         = TMR3CN ^ 6; /* TIMER 3 EXTERNAL FLAG           */
sbit EXEN3        = TMR3CN ^ 3; /* TIMER 3 EXTERNAL ENABLE FLAG   */
sbit TR3          = TMR3CN ^ 2; /* TIMER 3 ON/OFF CONTROL          */
sbit CT3          = TMR3CN ^ 1; /* TIMER 3 COUNTER SELECT          */

```

```

sbit CPRL3      = TMR3CN ^ 0;    /* TIMER 3 CAPTURE SELECT          */
/* */

/* TMR4CN 0xC8 */
sbit TF4        = TMR4CN ^ 7;    /* TIMER 4 OVERFLOW FLAG          */
sbit EXF4       = TMR4CN ^ 6;    /* TIMER 4 EXTERNAL FLAG          */
sbit EXEN4      = TMR4CN ^ 3;    /* TIMER 4 EXTERNAL ENABLE FLAG   */
sbit TR4        = TMR4CN ^ 2;    /* TIMER 4 ON/OFF CONTROL         */
sbit CT4        = TMR4CN ^ 1;    /* TIMER 4 COUNTER SELECT         */
sbit CPRL4      = TMR4CN ^ 0;    /* TIMER 4 CAPTURE SELECT         */

/* PSW 0xD0 */
sbit CY         = PSW ^ 7;     /* CARRY FLAG                   */
sbit AC         = PSW ^ 6;     /* AUXILIARY CARRY FLAG         */
sbit F0         = PSW ^ 5;     /* USER FLAG 0                  */
sbit RS1        = PSW ^ 4;     /* REGISTER BANK SELECT 1       */
sbit RS0        = PSW ^ 3;     /* REGISTER BANK SELECT 0       */
sbit OV         = PSW ^ 2;     /* OVERFLOW FLAG                */
sbit F1         = PSW ^ 1;     /* USER FLAG 1                  */
sbit P          = PSW ^ 0;     /* ACCUMULATOR PARITY FLAG     */
sbit Parity     = PSW ^ 0;     /* ACCUMULATOR PARITY FLAG     */

/* PCA0CN 0xD8 */
sbit CF         = PCA0CN ^ 7;   /* PCA 0 COUNTER OVERFLOW FLAG */
sbit CR         = PCA0CN ^ 6;   /* PCA 0 COUNTER RUN CONTROL BIT */
sbit CCF5       = PCA0CN ^ 5;   /* PCA 0 MODULE 5 INTERRUPT FLAG */
sbit CCF4       = PCA0CN ^ 4;   /* PCA 0 MODULE 4 INTERRUPT FLAG */
sbit CCF3       = PCA0CN ^ 3;   /* PCA 0 MODULE 3 INTERRUPT FLAG */
sbit CCF2       = PCA0CN ^ 2;   /* PCA 0 MODULE 2 INTERRUPT FLAG */
sbit CCF1       = PCA0CN ^ 1;   /* PCA 0 MODULE 1 INTERRUPT FLAG */
sbit CCF0       = PCA0CN ^ 0;   /* PCA 0 MODULE 0 INTERRUPT FLAG */

/* ADC0CN 0xE8 */
sbit AD0EN      = ADC0CN ^ 7;   /* ADC 0 ENABLE                  */
sbit AD0TM      = ADC0CN ^ 6;   /* ADC 0 TRACK MODE              */
sbit AD0INT     = ADC0CN ^ 5;   /* ADC 0 EOC INTERRUPT FLAG     */
sbit AD0BUSY    = ADC0CN ^ 4;   /* ADC 0 BUSY FLAG               */
sbit AD0CM1     = ADC0CN ^ 3;   /* ADC 0 CONVERT START MODE BIT 1 */
sbit AD0CM0     = ADC0CN ^ 2;   /* ADC 0 CONVERT START MODE BIT 0 */
sbit AD0WINT    = ADC0CN ^ 1;   /* ADC 0 WINDOW INTERRUPT FLAG  */
sbit AD0LJST    = ADC0CN ^ 0;   /* ADC 0 LJST FLAG               */

```

```

/* ADC2CN 0xE8 */
sbit AD2EN      = ADC2CN ^ 7;      /* ADC 2 ENABLE */          */
sbit AD2TM      = ADC2CN ^ 6;      /* ADC 2 TRACK MODE */        */
sbit AD2INT     = ADC2CN ^ 5;      /* ADC 2 EOC INTERRUPT FLAG */ */
sbit AD2BUSY    = ADC2CN ^ 4;      /* ADC 2 BUSY FLAG */         */
sbit AD2CM2     = ADC2CN ^ 3;      /* ADC 2 CONVERT START MODE BIT 2 */ */
sbit AD2CM1     = ADC2CN ^ 2;      /* ADC 2 CONVERT START MODE BIT 1 */ */
sbit AD2CM0     = ADC2CN ^ 1;      /* ADC 2 CONVERT START MODE BIT 0 */ */
sbit AD2WINT    = ADC2CN ^ 0;      /* ADC 2 WINDOW INTERRUPT FLAG */ */

/* SPI0CN 0xF8 */
sbit SPIF       = SPI0CN ^ 7;      /* SPI 0 INTERRUPT FLAG */        */
sbit WCOL       = SPI0CN ^ 6;      /* SPI 0 WRITE COLLISION FLAG */   */
sbit MODF       = SPI0CN ^ 5;      /* SPI 0 MODE FAULT FLAG */        */
sbit RXOVRN    = SPI0CN ^ 4;      /* SPI 0 RX OVERRUN FLAG */        */
sbit NSSMD1    = SPI0CN ^ 3;      /* SPI 0 SLAVE SELECT MODE BIT 1 */ */
sbit NSSMD0    = SPI0CN ^ 2;      /* SPI 0 SLAVE SELECT MODE BIT 0 */ */
sbit TXBMT     = SPI0CN ^ 1;      /* SPI 0 TX BUFFER EMPTY */        */
sbit SPIEN     = SPI0CN ^ 0;      /* SPI 0 SPI ENABLE */           */

/* CAN0CN 0xF8 */
sbit CANINIT    = CAN0CN ^ 0;      /* CAN Initialization bit */      */
sbit CANIE      = CAN0CN ^ 1;      /* CAN Module Interrupt Enable Bit */ */
sbit CANSIE     = CAN0CN ^ 2;      /* CAN Status change Interrupt
                                         Enable Bit */                  */
sbit CANEIE     = CAN0CN ^ 3;      /* CAN Module Error Interrupt
                                         Enable Bit */                  */
sbit CANIF      = CAN0CN ^ 4;      /* CAN Module Interrupt Flag */     */
sbit CANDAR    = CAN0CN ^ 5;      /* CAN Disable Automatic
                                         Retransmission bit */          */
sbit CANCCE     = CAN0CN ^ 6;      /* CAN Configuration Change Enable bit */ */
sbit CANTEST   = CAN0CN ^ 7;      /* CAN Test Mode Enable bit */       */

```

```
/* SFR PAGE DEFINITIONS */
#define CONFIG_PAGE 0x0F      /* SYSTEM AND PORT CONFIGURATION PAGE */
#define LEGACY_PAGE 0x00       /* LEGACY SFR PAGE */
#define TIMER01_PAGE 0x00     /* TIMER 0 AND TIMER 1 */
#define CPT0_PAGE 0x01         /* COMPARATOR 0 */
#define CPT1_PAGE 0x02         /* COMPARATOR 1 */
#define CPT2_PAGE 0x03         /* COMPARATOR 2 */
#define UART0_PAGE 0x00        /* UART 0 */
#define UART1_PAGE 0x01        /* UART 1 */
#define SPI0_PAGE 0x00          /* SPI 0 */
#define EMI0_PAGE 0x00          /* EXTERNAL MEMORY INTERFACE */
#define ADC0_PAGE 0x00          /* ADC 0 */
#define ADC2_PAGE 0x02          /* ADC 2 */
#define SMB0_PAGE 0x00          /* SMBUS 0 */
#define TMR2_PAGE 0x00          /* TIMER 2 */
#define TMR3_PAGE 0x01          /* TIMER 3 */
#define TMR4_PAGE 0x02          /* TIMER 4 */
#define DAC0_PAGE 0x00          /* DAC 0 */
#define DAC1_PAGE 0x01          /* DAC 1 */
#define PCA0_PAGE 0x00          /* PCA 0 */
#define CAN0_PAGE 0x01          /* CAN 0 */

/* BIT definitions for bits that are not directly accessible */

// ADC0CF
#define AMP0GN0    0x01
#define AMP0GN1    0x02
#define AMP0GN2    0x04
#define AD0SC0     0x08
#define AD0SC1     0x10
#define AD0SC2     0x20
#define AD0SC3     0x40
#define AD0SC4     0x80

// ADC2CF
#define AMP2GN0    0x01
#define AMP2GN1    0x02
#define AD2SC0     0x08
#define AD2SC1     0x10
#define AD2SC2     0x20
```

```
#define AD2SC3      0x40
#define AD2SC4      0x80

// AMX0CF
#define AIN01IC     0x01
#define AIN23IC     0x02
#define HVDA2C      0x04
#define PORT3IC     0x08

// AMX0PRT
#define PAIN0EN     0x01
#define PAIN1EN     0x02
#define PAIN2EN     0x04
#define PAIN3EN     0x08
#define PAIN4EN     0x10
#define PAIN5EN     0x20
#define PAIN6EN     0x40
#define PAIN7EN     0x80

// AMX0SL
#define AMX0AD0     0x01
#define AMX0AD1     0x02
#define AMX0AD2     0x04
#define AMX0AD3     0x08

// AMX2CF
#define PIN01IC     0x01
#define PIN23IC     0x02
#define PIN45IC     0x04
#define PIN67IC     0x08

// AMX2SL
#define AMX2AD0     0x01
#define AMX2AD1     0x02
#define AMX2AD2     0x04
```

```
// CKCON
#define SCA0      0x01
#define SCA1      0x02
#define T0M       0x08
#define T1M       0x10

// CLKSEL
#define CLKSL     0x01

// CPT0MD
#define CP0MD0    0x01
#define CP0MD1    0x02
#define CP0FIE    0x10
#define CP0RIE    0x20

// CPT1MD
#define CP1MD0    0x01
#define CP1MD1    0x02
#define CP1FIE    0x10
#define CP1RIE    0x20

// CPT2MD
#define CP2MD0    0x01
#define CP2MD1    0x02
#define CP2FIE    0x10
#define CP2RIE    0x20

// DAC0CN
#define DAC0DF0   0x01
#define DAC0DF1   0x02
#define DAC0DF2   0x04
#define DAC0MD0   0x08
#define DAC0MD1   0x10
#define DAC0EN    0x80

// DAC1CN
#define DAC1DF0   0x01
#define DAC1DF1   0x02
#define DAC1DF2   0x04
#define DAC1MD0   0x08
```

```
#define DAC1MD1 0x10
#define DAC1EN 0x80

// EIE1
#define ESPIO 0x01
#define ESMBO 0x02
#define EWADC0 0x04
#define EPCA0 0x08
#define CP0IE 0x10
#define CP1IE 0x20
#define CP2IE 0x40

// EIE2
#define ET3 0x01
#define EADC0 0x02
#define ET4 0x04
#define EWADC2 0x08
#define EADC2 0x10
#define ECANO 0x20
#define ES1 0x40

// EIP1
#define PSPIO 0x01
#define PSMB0 0x02
#define PWADC0 0x04
#define PPCA0 0x08
#define PCP0 0x10
#define PCP1 0x20
#define PCP2 0x40

// EIP2
#define PT3 0x01
#define PADC0 0x02
#define PT4 0x04
#define PWADC2 0x08
#define PADC2 0x10
#define PX7 0x20
#define EP1 0x40

// EMI0CF Bits
```

```
#define EALE0 0x01
#define EALE1 0x02
#define EMD0 0x04
#define EMD1 0x08
#define EMD2 0x10
#define PRTSEL x20

// EMI0CN
#define PGSEL0 0x01
#define PGSEL1 0x02
#define PGSEL2 0x04
#define PGSEL3 0x08
#define PGSEL4 0x10
#define PGSEL5 0x20
#define PGSEL6 0x40
#define PGSEL7 0x80

// EMI0TC Bits
#define EAH0 0x01
#define EAH1 0x02
#define EWR0 0x04
#define EWR1 0x08
#define EWR2 0x10
#define EWR3 0x20
#define EAS0 0x40
#define EAS1 0x80

// FLSCL
#define FLWE x01
#define FRAE 0x40
#define FOSE 0x80

// HVA0CN Bits
#define HVGAIN0 0x01
#define HVGAIN1 0x02
#define HVGAIN2 0x04
#define HVGAIN3 0x08
#define HVDAEN 0x80

// OSCICN
```

```
#define IFCN0 0x01
#define IFCN1 0x02
#define IFRDY 0x40
#define IOSCEN 0x80

// OSCXCN
#define XFCN0 0x01
#define XFCN1 0x02
#define XFCN2 0x04
#define XOSCMD0 0x10
#define XOSCMD1 0x20
#define XOSCMD2 0x40
#define XTLVLD 0x80

// PCA0MD
#define ECF 0x01
#define CPS0 0x02
#define CPS1 0x04
#define CPS2 0x08
#define CIDL 0x80

// PCA0CPM0
#define ECCF0 0x01
#define PWM0 0x02
#define TOG0 0x04
#define MAT0 0x08
#define CAPN0 0x10
#define CAPP0 0x20
#define ECOM0 0x40
#define PWM160 0x80

// PCA0CPM1
#define ECCF1 0x01
#define PWM1 0x02
#define TOG1 0x04
#define MAT1 0x08
#define CAPN1 0x10
#define CAPP1 0x20
#define ECOM1 0x40
#define PWM161 0x80
```

```
// PCA0CPM2
#define ECCF2    0x01
#define PWM2     0x02
#define TOG2     0x04
#define MAT2     0x08
#define CAPN2    0x10
#define CAPP2    0x20
#define ECOM2    0x40
#define PWM162   0x80

// PCA0CPM3
#define ECCF3    0x01
#define PWM3     0x02
#define TOG3     0x04
#define MAT3     0x08
#define CAPN3    0x10
#define CAPP3    0x20
#define ECOM3    0x40
#define PWM163   0x80

// PCA0CPM4
#define ECCF4    0x01
#define PWM4     0x02
#define TOG4     0x04
#define MAT4     0x08
#define CAPN4    0x10
#define CAPP4    0x20
#define ECOM4    0x40
#define PWM164   0x80

// PCA0CPM5
#define ECCF5    0x01
#define PWM5     0x02
#define TOG5     0x04
#define MAT5     0x08
#define CAPN5    0x10
#define CAPP5    0x20
#define ECOM5    0x40
#define PWM165   0x80
```

```
// PCON
#define IDLE    0x01
#define STOP    0x02

// PSCTL
#define PSWE    0x01
#define PSEE    0x02
#define SFLE    0x04

// REF0CN
#define REFBE   0x01
#define BIASE   0x02
#define TEMPE   0x04
#define AD2VRS  0x08
#define AD0VRS  0x10

// RSTSRC
#define PINRSF  0x01
#define PORSF   0x02
#define MCDRSF  0x04
#define WDTRSF  0x08
#define SWRSEF  0x10
#define C0RSEF  0x20
#define CNVRSEF 0x40

// SFRPGCN
#define SFRPGEN 0x01

// SMB0ADR
#define GC      0x01
#define SLV0   0x02
#define SLV1   0x04
#define SLV2   0x08
#define SLV3   0x10
#define SLV4   0x20
#define SLV5   0x40
#define SLV6   0x80
```

```
// SMB0STA
#define STA0    0x01
#define STA1    0x02
#define STA2    0x04
#define STA3    0x08
#define STA4    0x10
#define STA5    0x20
#define STA6    0x40
#define STA7    0x80

// SPI0CFG
#define RXBMT   0x01
#define SRMT    0x02
#define NSSIN   0x04
#define SLVSEL   0x08
#define CKPOL   0x10
#define CKPHA   0x20
#define MSTEN   0x40
#define SPIBSY   0x80

// SPI0CKR
#define SCR0    0x01
#define SCR1    0x02
#define SCR2    0x04
#define SCR3    0x08
#define SCR4    0x10
#define SCR5    0x20
#define SCR6    0x40
#define SCR7    0x80

// SSTA0
#define S0RCLK0 0x01
#define S0RCLK1 0x02
#define S0TCLK0 0x04
#define S0TCLK1 0x08
#define SMODO   0x10
#define TXCOL0  0x20
#define RXOVO   0x40
#define FE0     0x80
```

```
// TMOD
#define T0M0    0x01
#define T0M1    0x02
#define C_T0    0x04
#define GATE0   0x08
#define T1M0    0x10
#define T1M1    0x20
#define C_T1    0x40
#define GATE1   0x80

// TMR2CF
#define DCEN2   0x01
#define T2OE    0x02
#define TOG02   0x04
#define T2M0    0x08
#define T2M1    0x10

// TMR3CF
#define DCEN3   0x01
#define T3OE    0x02
#define TOG03   0x04
#define T3M0    0x08
#define T3M1    0x10

// TMR4CF
#define DCEN4   0x01
#define T4OE    0x02
#define TOG04   0x04
#define T4M0    0x08
#define T4M1    0x10

// XBR0
#define SMB0EN  0x01
#define SPI0EN  0x02
#define UART0EN 0x04
#define PCA0ME0  0x08
#define PCA0ME1  0x10
#define PCA0ME2  0x20
#define ECIOE   0x40
#define CP0E    0x80
```

```

// XBR1
#define CPIE      0x01
#define TOE       0x02
#define INTOE     0x04
#define T1E       0x08
#define INT1E     0x10
#define T2E       0x20
#define T2EXE     0x40
#define SYSCKE    0x80

// XBR2
#define CNVST0E   0x01
#define EMIFILE   0x02
#define UART1E    0x04
#define T4E       0x08
#define T4EXE     0x10
#define XBARE    0x40
#define WEAKPUD   0x80

// XBR3
#define T3E       0x01
#define T3EXE    0x02
#define CNVST2E   0x04
#define CP2E      0x08
#define CTXOUT    0x80

// CAN Protocol Register Index for CAN0ADR,
// from TABLE 18.1 of the C8051F040 data sheet
///////////////////////////////
#define CANCTRL    0x00      //Control Register
#define CANSTAT    0x01      //Status register
#define ERRCNT     0x02      //Error Counter Register
#define BITREG     0x03      //Bit Timing Register
#define INTREG     0x04      //Interrupt Low Byte Register
#define CANTSTR    0x05      //Test register
/////////////////////////////
//IF1 Interface Registers
#define BRPEXT     0x06      //BRP Extension Register

```

```
//////////  

#define IF1CMDRQST 0x08      //IF1 Command Rest Register  

#define IF1CMDMSK 0x09      //IF1 Command Mask Register  

#define IF1MSK1 0x0A      //IF1 Mask1 Register  

#define IF1MSK2 0x0B      //IF1 Mask2 Register  

#define IF1ARB1 0x0C      //IF1 Arbitration 1 Register  

#define IF1ARB2 0x0D      //IF1 Arbitration 2 Register  

#define IF1MSGC 0x0E      //IF1 Message Control Register  

#define IF1DATA1 0x0F      //IF1 Data A1 Register  

#define IF1DATA2 0x10      //IF1 Data A2 Register  

#define IF1DATB1 0x11      //IF1 Data B1 Register  

#define IF1DATB2 0x12      //IF1 Data B2 Register  

//////////  

//IF2 Interface Registers  

//////////  

#define IF2CMDRQST 0x20      //IF2 Command Rest Register  

#define IF2CMDMSK 0x21      //IF2 Command Mask Register  

#define IF2MSK1 0x22      //IF2 Mask1 Register  

#define IF2MSK2 0x23      //IF2 Mask2 Register  

#define IF2ARB1 0x24      //IF2 Arbitration 1 Register  

#define IF2ARB2 0x25      //IF2 Arbitration 2 Register  

#define IF2MSGC 0x26      //IF2 Message Control Register  

#define IF2DATA1 0x27      //IF2 Data A1 Register  

#define IF2DATA2 0x28      //IF2 Data A2 Register  

#define IF2DATB1 0x29      //IF2 Data B1 Register  

#define IF2DATB2 0x2A      //IF2 Data B2 Register  

//////////  

//Message Handler Registers  

//////////  

#define TRANSREQ1 0x40      //Transmission Rest1 Register  

#define TRANSREQ2 0x41      //Transmission Rest2 Register  

#define NEWDAT1 0x48      //New Data 1 Register  

#define NEWDAT2 0x49      //New Data 2 Register  

#define INTPEND1 0x50      //Interrupt Pending 1 Register  

#define INTPEND2 0x51      //Interrupt Pending 2 Register  

#define MSGVAL1 0x58      //Message Valid 1 Register
```

```
//////////  

#define IF1CMDRQST 0x08      //IF1 Command Rest Register  

#define IF1CMDMSK 0x09      //IF1 Command Mask Register  

#define IF1MSK1 0x0A      //IF1 Mask1 Register  

#define IF1MSK2 0x0B      //IF1 Mask2 Register  

#define IF1ARB1 0x0C      //IF1 Arbitration 1 Register  

#define IF1ARB2 0x0D      //IF1 Arbitration 2 Register  

#define IF1MSGC 0x0E      //IF1 Message Control Register  

#define IF1DATA1 0x0F      //IF1 Data A1 Register  

#define IF1DATA2 0x10      //IF1 Data A2 Register  

#define IF1DATB1 0x11      //IF1 Data B1 Register  

#define IF1DATB2 0x12      //IF1 Data B2 Register  

//////////  

//IF2 Interface Registers  

//////////  

#define IF2CMDRQST 0x20      //IF2 Command Rest Register  

#define IF2CMDMSK 0x21      //IF2 Command Mask Register  

#define IF2MSK1 0x22      //IF2 Mask1 Register  

#define IF2MSK2 0x23      //IF2 Mask2 Register  

#define IF2ARB1 0x24      //IF2 Arbitration 1 Register  

#define IF2ARB2 0x25      //IF2 Arbitration 2 Register  

#define IF2MSGC 0x26      //IF2 Message Control Register  

#define IF2DATA1 0x27      //IF2 Data A1 Register  

#define IF2DATA2 0x28      //IF2 Data A2 Register  

#define IF2DATB1 0x29      //IF2 Data B1 Register  

#define IF2DATB2 0x2A      //IF2 Data B2 Register  

//////////  

//Message Handler Registers  

//////////  

#define TRANSREQ1 0x40      //Transmission Rest1 Register  

#define TRANSREQ2 0x41      //Transmission Rest2 Register  

#define NEWDAT1 0x48      //New Data 1 Register  

#define NEWDAT2 0x49      //New Data 2 Register  

#define INTPEND1 0x50      //Interrupt Pending 1 Register  

#define INTPEND2 0x51      //Interrupt Pending 2 Register  

#define MSGVAL1 0x58      //Message Valid 1 Register
```

```
#define     MSGVAL2          0x59      //Message Valid
                           2           Register

#endif
// end of __C8051F040_H__
```



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



APPENDIX B – FURTHER EXAMPLE

We list here an interesting example for the C8051F040 target board using a single LED already found on the board to display the time and date. The LED displays the time flickering on and off using the Morse code. Every 15 minutes, it also transmits the date, whilst every 20 minutes it saves the date and time onto the scratch pad memory just in case the power goes out. The saved time is recovered once power is restored. Each press on button P3.7 of the C8051F040-TB target board, would advance the minute by 1. At minute 59, pressing P3.7 the minute would go back to 0 without advancing the hour.

B.1 Single LED clock display

The files PaulOS_F040.h, PaulOS_F040.c, F04x_Osc_Init.h and F04x_Osc_Init.c although part of the clock project, are not being included here since they are listed elsewhere in another previous chapter. Programs for reading and writing to the Flash memory are taken from examples given by Silicon Labs, as indicated on the files themselves.

The time can be initialised in the program and if the button P3.7 is held down when switching on the board, it will take the initial values as the time. Every 20 minutes, the program stores the time in the scratchpad memory so that if the supply is out, it will take these stored values as the initial time when the supply is restored.

When running, pressing the button P3.7 advances the minute by 1 each time it is pressed, and the time is once again stored immediately in the scratch pad area.

Also, if there is a GPS device connected to the RS232 serial port, the time will be synchronised to the GPS signal.

The listings are well remarked so as to make the program easy to follow and understand.

```

1 //-----
2 // MorseLedClockMemGarmin2.c
3 //      with auto summer/winter time update, button setting, flash memory storage
4 //      Board time can be set automatically via the serial port by means of
5 //      serial data coming from a Garmin GPS unit.
6 //      If no GPS signal is present at the serial port, time is taken from the
7 //      default or stored values.
8 //      Summer/Winter time change made optional by means of a [ #define _LEGAL_TIME_ ]
9 //      during compilation
10 //-----
11 // Copyright (C) 2018
12 //
13 // AUTH: PD
14 // DATE: 09 AUG 18
15 //
16 // This program flashes the green LED on the C8051F040 target board
17 // One task flickers the LED, one digit at a time to display the time
18 // using morse code to represent the digits
19 // Another task keeps track of the clock (24 hr clock with auto Summer/Winter time)
20 // whilst another task handles the push switch activity
21 // to increment the minutes for clock setting purposes.
22 //
23 // Every 3 seconds it checks the UART for any Garmin GPS signal to synchronise time/date
24 // Every 15 minutes it also transmits the Date
25 // At 6, 26, 46 min past the hour it stores the date/time info on Scratchpad memory
26 // in case of a power cut
27
28 // RS232 cable from the Garmin eTrex to C8051F020 board has to be the
29 // -----

```

B.2 MorseLedClockMemGarmin2.c

```
-----  
// MorseLedClockMemGarmin2.c  
// with auto summer/winter time update, button  
// setting, flash memory storage  
// Board time can be set automatically via  
// the serail port by means of  
// serial data coming from a Garmin GPS unit.  
// If no GPS signal is present at the serial  
// port, time is taken from the  
// default or stored values.  
// Summer/Winter time change made optional by  
// means of a [ #define _LEGAL_TIME_ ]  
// during compilation  
-----  
// Copyright (C) 2018  
//  
// AUTH: PD  
// DATE: 09 AUG 18  
//  
// This program flashes the green LED on the C8051F040 target board  
// One task flickers the LED, one digit at a time to display the time  
// using morse code to represent the digits  
// Another task keeps track of the clock (24 hr  
// clock with auto Summer/Winter time)  
// whilst another task handles the push switch activity  
// to increment the minutes for clock setting purposes.  
//  
// Every 3 seconds it checks the UART for any Garmin  
// GPS signal to synchronise time/date  
// Every 15 minutes it also transmits the Date  
// At 6, 26, 46 min past the hour it stores the  
// date/time info on Scratchpad memory  
// in case of a power cut  
  
// RS232 cable from the Garmin eTrex to C8051F040 board has to be the  
// null cable type (crossed wires) and not the straight one.  
// i.e. pin 2 on one socket connected  
// to pin 3 on the other socket and vice versa.  
//  
// Using Task Priority sorting with Clock Task  
// given top priority (task number 0)
```

```
// to try and keep the correct time as much as possible.

// Target: C8051F04x
//
//

//-----
// Includes
//-----

#include "C8051F040.h"          /* special function registers 8052 */
#include "PaulOS_F040.h"        /* PaulOS_F040 system calls definitions */
#include "F04x_Osc_Init.h"
#include "F040_FlashPrimitives.h"
#include "F040_FlashUtils.h"
#include "XonXoff_RRTOS.h"
#include "GPS_NMEA_Parser_2.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//-----
// Global CONSTANTS and definitions
//-----


#define _LEGAL_TIME_           // define only if legal summer
                           // time is being implemented
                           // remark if no auto summer time
                           // change is to be implemented

#define GMT 0
#define Flash_Storage_Area    0x0000
// Location for storing defaults on Flash, in Scratchpad (0x00 - 0x7F)
extern myTime udtTime;         // myTime defined in GPS_NMEA_
                               // Parser.c GPS_NMEA_Parser.h

sbit LED = P1^6;              // green LED: '1' = ON; '0' = OFF
sbit BUTTON = P3^7;

#define DATE_INTERVAL     15      // transmit date info every 15 minutes
```

```
#define DISPLAY_RATE_SEC 2          // pause between time info
                                    transmissions

#define DISPLAY_RATE_MSEC 500

#define MORSE_SPEED 10             // morse transmission speed
                                in words per minute

enum MEMORY {FLASH, SCRATCHPAD};

enum eMonths {JAN = 1, FEB, MAR, APR, MAY,
JUN, JUL, AUG, SEP, OCT, NOV, DEC};

enum eDays {MON = 1, TUE, WED, THU, FRI, SAT, SUN};

enum eTasks {BOARD_CLOCK, GENERATE_MORSE, CHECK_
BUTTON, RS232, STORE_TIME_DATE};

struct time {                      /* structure of the time record */
    unsigned int year;
    unsigned char month;
    unsigned char day;
    unsigned char hour;           /* hour */
    unsigned char min;            /* minute */
    unsigned char sec;            /* second */
    unsigned char winter;         /* 1 = winter time, 0 =
                                    summer (legal) time */
};

// initial time stamp
struct time board_time = { 2018, 12, 7, 12, 33, 0, 1 };
                           // storage for clock time values
                           //       yyyy, mm, dd, hh, mm, ss, winter
static code char daytab[2][13] = {
{0,31,28,31,30,31,30,31,31,30,31,30,31},
{0,31,29,31,30,31,30,31,31,30,31,30,31}
};

unsigned int data ChangeToSummerTime,ChangeToWinterTime;

char code daycode[][][4] = {
    {"XXX"}, 
    {"MON"}, 
    {"TUE"}, 
    {"WED"},
```

```
{ "THU" },  
{ "FRI" },  
{ "SAT" },  
{ "SUN" }  
};  
  
char code monthcode[][][4] = {  
    { "XXX" },  
    { "JAN" },  
    { "FEB" },  
    { "MAR" },  
    { "APR" },  
    { "MAY" },  
    { "JUN" },  
    { "JUL" },  
    { "AUG" },  
    { "SEP" },  
    { "OCT" },  
    { "NOV" },  
    { "DEC" }  
};  
  
//-----  
// Function PROTOTYPES  
//-----  
void PORT_Init (void);  
void DISABLE_Watchdog (void);  
void UpdateDay (void);  
void UpdateMonth (void);  
unsigned char day_of_week(unsigned int year,  
    unsigned char month, unsigned char day);  
unsigned int last_dom(unsigned int year, unsigned  
    char month, unsigned char dow);  
unsigned int day_of_year(unsigned int year,  
    unsigned char month, unsigned char day);  
unsigned char code *morse(uchar c);  
  
//-----  
// DISABLE_Watchdog  
//-----
```

```
//  
// Disables the watchdog timer  
//  
void DISABLE_Watchdog (void)  
{  
    char SFRPAGE_SAVE;  
  
    SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page  
    SFRPAGE = CONFIG_PAGE;          // Switch to configuration page  
    EA = 0;  
    WDTCN = 0xDE;  
    WDTCN = 0xAD;  
    EA = 1;  
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page  
}  
  
//-----  
// PORT_Init  
//-----  
//  
// Configure the Crossbar and GPIO ports  
//  
void PORT_Init (void)  
{  
    char SFRPAGE_SAVE;  
  
    SFRPAGE_SAVE = SFRPAGE;           // Save Current SFR page  
    SFRPAGE = CONFIG_PAGE;          // Switch to configuration page  
    XBR0 = 0x04;                    // Enable UART 0  
    XBR1 = 0x00;  
    XBR2 = 0x40;                    // Enable crossbar and weak pull-ups  
    P0MDOUT |= 0x01;                // enable TX0 as a push-pull output  
    P1MDOUT |= 0x40;                // enable P1.6 (LED) as  
                                    // push-pull output  
    P3MDOUT &= 0x7F;                // set P3.7 in open-drain  
                                    // output mode (default)  
    BUTTON = 1;                     // P3.7 as digital input  
                                    // (board push button)  
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page  
}
```

```
*****  
/* Function to update Days: */  
*****  
void UpdateDay (void){  
    unsigned char DaysinMonth;  
  
    switch (board_time.month)  
    {  
        case JAN: case MAR: case MAY: case JUL:  
            DaysinMonth = 31;  
            break;  
  
        case AUG: case OCT: case DEC:  
            DaysinMonth = 31;  
            break;  
  
        case APR: case JUN: case SEP: case NOV:  
            DaysinMonth = 30;  
            break;  
  
        case FEB:  
        if (((board_time.year%4==0) && (board_time.  
year%100 != 0)) || (board_time.year%400==0))  
            DaysinMonth = 29;  
        else  
            DaysinMonth = 28;  
        break;  
    }  
  
    if (board_time.day < DaysinMonth)  
    {  
        board_time.day++;  
    }  
    else  
    {  
        board_time.day = 1;  
        UpdateMonth();  
    }  
}
```

```
*****
/* Function to update Month and year if necessary: */
*****
void UpdateMonth (void)
{
    if (board_time.month < DEC)
    {
        board_time.month++;
    }
    else
    {
        board_time.month = 1;
        board_time.year++;
        #ifdef _LEGAL_TIME_
        ChangeToSummerTime = last_dom(board_time.year, MAR, SUN);
        ChangeToWinterTime = last_dom(board_time.year, OCT, SUN);
        #endif
    }
}

/* last_dom: return day_of_year for last day (dow) of month (m) */
unsigned int last_dom(unsigned int year,
unsigned char month, unsigned char dow)
{
    unsigned char i, j, leap, maxdays;

    leap = (((year%4 == 0) && (year%100 != 0)) || (year%400 == 0));
    maxdays = daytab[leap][month];
    for (i = maxdays; i >= maxdays-6; i--)
        if (day_of_week(year,month,i) == dow)
            /* 1 = Monday, 7 = Sunday */
            j=i;
    return day_of_year(year,month,j);
}
/* day_of_year: set day number (1-366) of
year from day, month and year */
unsigned int day_of_year(unsigned int year,
unsigned char month, unsigned char day)
{
    int i, leap;
```

```

leap = (((year%4 == 0) && (year%100 != 0)) || (year%400 == 0));
for (i=1; i < month; i++)
    day += daytab[leap][i];
return day;
}
/* day_of_week: returns the day of the week, (1-7)
   ( 1 is Monday, 7 Sunday ) given year, month, day
*/
unsigned char day_of_week(unsigned int year,
unsigned char month, unsigned char day)
{
    unsigned char dowk;
    unsigned int y;
    static unsigned char t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
    y = year;
    y -= month < 3;
    dowk = (y + y/4 - y/100 + y/400 + t[month-1] + day) % 7;
    if (dowk==0) dowk = 7;
    return dowk;
}

/*
*****
*****MORSE CODE
A '.' signifies a mark with a duration of 1 dot time
A '-' signifies a mark with a duration of 3 dot time
A 'S' signifies a space with a duration of 2 dot time

Characters are in ASCII sequence order - only
CAPITAL letters are implemented
*/
/* function returning a pointer to first symbol of character */
unsigned char code *morse(uchar c){
static char code * code morsecode[] = { /* pointer in code to
                                         code char */

```

```

"S",
/* */
"S",
/* ^A = 1 */
"S",
/* ^B */
"S",
/* ^C */
"S",
/* ^D */
"S",
/* ^E = 5 */
"S",
/* ^F */
"S",
/* ^G */
"S",
/* ^H */
"S",
/* ^I */
"---",
/* ^J ---- => /n = 10 */
"S",
/* ^K */
"S",
/* ^L */
"---.",
/* ^M ---. => /r = 13 */
"S",
/* ^N */
"S",
/* ^O = 15 */
"S",
/* ^P */
"S",
/* ^Q */
"S",
/* ^R */
"S",
/* ^S */
"S",
/* ^T = 20 */
"S",
/* ^U */
"S",
/* ^V */
"S",
/* ^W */
"S",
/* ^X */
"S",
/* ^Y = 25 */
"S",
/* ^Z */
"S",
/* */
"S",
/* */
"S",
/* */
"S",
/* = 30 */
"S",
/* */
"S",
/* SPACE */
"S",
/* ! */
"-. . . -",
/* .- . . - => `` */
"S",
/* # */
"S",
/* $ */
"S",
/* % */
"S",
/* & */
"S",
/* ` */
"-. - - .",
/* - . - - . => ( = 40 */

```

```

"-.-.-.", /* -.-.- => ) */
"S", /* * */
"S", /* + */
"--.-.", /* --.- => , */
".--.-.", /* .--.- => - */
"--.-.", /* --.- => . */
"--.-.", /* --.- => / */
"-----", /* ----- => 0 = 48 */
"-----", /* ----- => 1 */
"---", /* --- => 2 = 50 */
"---", /* --- => 3 */
"---", /* --- => 4 */
"---", /* --- => 5 */
"---", /* --- => 6 */
"---", /* --- => 7 */
"---", /* --- => 8 */
"---", /* --- => 9 */
"---", /* --- => : */
"--.-.", /* --.- => ; */
"S", /* < = 60 */
"--.-.", /* --.- => = */
"S", /* > */
"--.-.", /* --.- => ? */
"--.-.", /* --.- => @ */
"--.", /* -- => A = 65 */
"--.", /* -- => B */
"--.", /* -- => C */
"--.", /* -- => D */
"--.", /* -- => E */
"--.", /* -- => F = 70 */
"--.", /* -- => G */
"--.", /* -- => H */
"--.", /* -- => I */
"--.", /* -- => J */
"--.", /* -- => K */
"--.", /* -- => L */
"--.", /* -- => M */
"--.", /* -- => N */
"--.", /* -- => O */
"--.", /* -- => P = 80 */
"--.-.", /* --.- => Q */

```

```

        ".-.",          /* .-. => R */
        "...",          /* ... => S */
        "-.",           /* - => T */
        ".-.",          /* ..- => U */
        "...-",         /* ...- => V */
        ".--",          /* .-- => W */
        "-..-",         /* -..- => X */
        "-.--",         /* -.-- => Y */
        "--..",         /* --.. => Z = 90 */
        "S",             /* [ */
        "S",             /* \ */
        "S",             /* ] */
        "S",             /* ^ */
        "S",             /* - */
        "S",             /* ` = 96 */
        "S",             /* ` */
        "S"              /* ` */

    };

    return(morsecode[c]);
}

/*
***** Task 0 'BoardClock' *****
void BoardClockTask (void)
{
    OS_PERIODIC_A(0,1,0);          /* Repeat every 1 second */

    while (1) {                   /* clock is an endless loop */
        if (++board_time.sec == 60) { /* increment the second */
            board_time.sec = 0;
            if (++board_time.min == 60) { /* increment the minute */
                board_time.min = 0;
                if (++board_time.hour == 24){ /* increment the hour */
                    board_time.hour = 0;
                    UpdateDay();
                }
            }
        }
    }
}

```

```
        }

#ifndef _LEGAL_TIME_
else if ((day_of_year(board_time.year,board_time.
month,board_time.day)==ChangeToWinterTime) \
&& (board_time.hour == 3) && (board_time.winter == 0))
    { board_time.hour = 2;
      board_time.winter = 1;
    }
else if ((day_of_year(board_time.year,board_time.
month,board_time.day)==ChangeToSummerTime) \
&& (board_time.hour == 2) && (board_time.winter == 1))
    { board_time.hour = 3;
      board_time.winter = 0;
    }
#endif
}

if ((board_time.min == 6) || (board_time.
min == 26) || (board_time.min == 46))
    OS_SIGNAL_TASK(STORE_TIME_DATE);
// store date-time just in case there is a power cut
}
OS_WAITP(); /* wait for 1 second to pass */
}

/*
-----  

Generates Morse Signal  

TASK 1  

-----*/
/*  

MORSE TRANSMISSION  

A '.' signifies a mark with a duration of 1 dot time  

A '-' signifies a mark with a duration of 3 dot time  

A 'S' signifies a space with a duration of 2 dot time  

Unit of time is the duration of 1 dot signal == U  

Hence:  

Duration of a dot, led on = U  

Duration of a Dash, led on = 3U  

Space (led off) between dots/dashes representing a character = U
```

```
Space (led off) between characters in the same word = 3U
Space (led off) between words = 7U

Actual timings modified slightly for better LED flickering
*/

void GenerateMorseTask (void)
{
    unsigned char data i;
    unsigned int data OneUnit, OneUnitPlusDelta,
    TwoUnits, ThreeUnits, FiveUnits;
    unsigned char xdata * data j;
    /* pointer in data area, referring to characters in xdata area */
    unsigned char data mcode[10];
    char xdata TxText[100],DateText[50];
#define Delta 70 // Delta ms added to off time after a symbol (. or -)
                // since led does not switch off abruptly
                // This makes the flickering more pronounced
// Set up Initial TX speed variable and set up delay variables once
    // Morse speed in words per minute
    OneUnit = (unsigned int)(1200/MORSE_SPEED);
    // Duration of 1 dot in milliseconds (used
        as a reference for transmission)
    OneUnitPlusDelta = OneUnit + Delta;
    TwoUnits = OneUnit + OneUnit;
    ThreeUnits = TwoUnits + OneUnit;
    FiveUnits = TwoUnits + ThreeUnits;

while(1)
{
    sprintf(TxText,"%bu %bu %bu %bu",board_time.hour/10,board_
    time.hour%10,board_time.min/10, board_time.min%10);
    if(board_time.min%DATE_INTERVAL==0)
    // transmit date info every DATE_INTERVAL minutes
    {
        sprintf(DateText," %s %02bu %s %u",
            daycode[day_of_week(board_time. year,board_
            time.month,board_time.day)],

            board_time.day,
            monthcode[board_time.month],
```

```
        board_time.year);
strcat(TxText, DateText);
}

j = TxText;
while(*j) /* do all TxText characters */
{
i = 0;
strcpy(mcode,morse(*j));
while(mcode[i]!='\0') /* do all symbols
(.) or (-) within a character */
{
switch(mcode[i])
{
case('.'):                                // transmit a DOT
LED = 1;
OS_WAITT_A(0,0,OneUnit);      // duration of 1 dot
/* Transmit a PAUSE between symbols of the same character */
LED = 0;
OS_WAITT_A(0,0,OneUnitPlusDelta); // duration of 1 dot
i++;
break;
case('-'):                                // transmit a DASH
LED = 1;
OS_WAITT_A(0,0,ThreeUnits); // duration of 3 dots
/* Transmit a PAUSE between symbols of the same character */
LED = 0;
OS_WAITT_A(0,0,OneUnitPlusDelta); // duration of 1 dot
i++;
break;
case('S'):                                // transmit a SPACE or
PAUSE (no transmission)
LED = 0;
OS_WAITT_A(0,0,FiveUnits); // duration of 5 dots
// (another 2 dot pause transmitted
// at the end of a word,
// bringing the total to 7 dot units duration)
i++;
break;
}
}
```

```

/* Transmit a PAUSE (3 dots duration total)
   between characters (same word) */
/* or 7 dots duration total between words (after a SPACE) */
   /* duration actually lengthened a bit since
      LED does not switch off abruptly, */
   /* but rather it dims down to off */
LED = 0;
OS_WAITT_A(0,0,ThreeUnits);      // duration of 3 dots
                                // (1 dot pause already included with the . or - symbol)
j++;                         /* get next character to transmit */
}
OS_WAITT_A(0,DISPLAY_RATE_SEC,DISPLAY_RATE_MSEC);
}

// =====
// ****
/* Task 2 - Polls P3.7 button */
/*          BUTTON returns 0 if pressed, 1 if not pressed */
/*          Software switch de-bouncing is also handled here */
// ****
void CheckButtonTask (void)
{
while(1)
{
    while (BUTTON == 1) {OS_DEFER();}

// swap tasks if button is not being pressed

    OS_WAITT_A(0,0,50);
// BUTTON = 0, hence wait 50ms (de-bounce delay) then check again

    if (BUTTON == 0)
// if button is still pressed, then it is a valid button press
    {
        LED = 1;
        while (BUTTON == 0) {OS_DEFER();} // wait for button release
        board_time.min = (++board_time.min)%60;
        board_time.sec = 0;
        LED = 0;
    }
}

```

```

        }

}

/***** Task 3 'RS232' ****/
/***** Task 3 'RS232' ****/


void RS232Task (void)
{
    char GPS_String[100];
    unsigned char timezone;
OS_WAITT_A(0,10,0);           // wait 10 seconds initially
    while(1)
    {
        GPS_String[0] = '\0';
        GetStringU0(GPS_String,90);
        // Get GPS string from buffer, terminated with a LF (\n) character
        // and store it in GPS_String.
        // Defer task in getbyteU0() called by
        // GetStringU0() in XonXoff_RTOS.c
        // If the sentence is still not available in the buffer
        // defer task instead and try again after 3s

        if (Parse_String(GPS_String, GPRMC) == 1)
        // if Date and Time received ok
        {
            LED = 1; // indicate acquisition of GPS signal
            board_time.day = udtTime.ucDay;
            board_time.month = udtTime.ucMonth;
            board_time.year = udtTime.uiYear;
            board_time.hour = udtTime.ucHours;
            board_time.min = udtTime.ucMinutes;
            board_time.sec = udtTime.ucSeconds;
            timezone = GMT + 2;

            #ifdef _LEGAL_TIME_
            ChangeToSummerTime = last_dom(board_time.year, MAR, SUN);
            ChangeToWinterTime = last_dom(board_time.year, OCT, SUN);
            if((day_of_year(board_time.year,board_time.
month,board_time.day)> ChangeToSummerTime)\
```

```

    && (day_of_year(board_time.year, board_time.
month, board_time.day) < ChangeToWinterTime))
    {
        timezone = GMT + 2;
        board_time.winter = 0;
    }
else
{
    timezone = GMT + 1;
    board_time.winter = 1;
}
#endif
//board_time.hour += timezone; //not required if Garmin corrects
                           to local time automatically
OS_SIGNAL_TASK(STORE_TIME_DATE); // store immediately
                                 retrieved values
LED = 0; // switch off GPS signal indicator
}
OS_WAITT_A(0,3,0); // wait 3 seconds and try again
}

}

/*****
 *      Task 4 to store Time/Date to cater for electricity outage      */
/*      KEIL is Big endian, high byte is stored first      */
/* FLASH_Byte_Update first erases whole page than writes a byte      */
/* FLASH_ByteWrite writes a byte on a pre-erased page      */
/* Data is stored in 6 consecutive locations in scratchpad      */
/* Since the FLASHwrite routines disable the interrupts, OS_DEFER()      */
is                                         */
/* interspersed between commands to give other tasks a chance to      */
execute                                         */
/* in particular the time task so as not to lose time */
/*****
void StoreTimeDateTask (void){
    while(1) {
OS_WAITS(0); // wait for a signal to execute
FLASH_Byte_Update ((unsigned int) Flash_Storage_
Area, board_time.year/256, SCRATCHPAD);
OS_DEFER();
}

```

```

FLASH_ByteWrite ((unsigned int) Flash_Storage_
Area + 1, board_time.year%256, SCRATCHPAD);
OS_DEFER();
FLASH_ByteWrite ((unsigned int) Flash_Storage_
Area + 2, board_time.month, SCRATCHPAD);
OS_DEFER();
FLASH_ByteWrite ((unsigned int) Flash_Storage_
Area + 3, board_time.day, SCRATCHPAD);
OS_DEFER();
FLASH_ByteWrite ((unsigned int) Flash_Storage_
Area + 4, board_time.hour, SCRATCHPAD);
OS_DEFER();
FLASH_ByteWrite ((unsigned int) Flash_Storage_
Area + 5, board_time.min, SCRATCHPAD);
OS_DEFER();
FLASH_ByteWrite ((unsigned int) Flash_Storage_
Area + 6, board_time.winter, SCRATCHPAD);
}

//-----
// MAIN Routine
//-----
void main (void) {
    unsigned char data yearHI, yearLO;

    DISABLE_Watchdog ();
    SYSCLK_Crystal_Init ();
    PORT_Init ();

    Init_UART0(4800, 1);      /* UART0 used to connect to
                               the Garmin GPS Rx */

    OS_INIT_RTOS(TICK_TIMER); /* initialise RTOS (Timer
                               2 interrupt), */
                               /* variables and stack */
    OS_CREATE_TASK(BOARD_CLOCK, BoardClockTask);
    OS_CREATE_TASK(GENERATE_MORSE, GenerateMorseTask);
    OS_CREATE_TASK(CHECK_BUTTON, CheckButtonTask);
    OS_CREATE_TASK(RS232, RS232Task);
    OS_CREATE_TASK(STORE_TIME_DATE, StoreTimeDateTask);
}

```

```

// read stored date/time values if button P3.7 is not pressed,
// otherwise if button is pressed,
// take default values as set when declaring
// time structure above (line 82)
// and store them immediately in flash scratchpad
if (BUTTON == 1)           // if button is not being pressed
{
    yearHI = FLASH_ByteRead ((unsigned int)
Flash_Storage_Area, SCRATCHPAD);
    yearLO = FLASH_ByteRead ((unsigned int) Flash_
Storage_Area + 1, SCRATCHPAD);
    board_time.year = yearHI * 256 + yearLO;
    board_time.month = FLASH_ByteRead((unsigned int)
Flash_Storage_Area + 2, SCRATCHPAD);
    board_time.day = FLASH_ByteRead ((unsigned int)
Flash_Storage_Area + 3, SCRATCHPAD);
    board_time.hour= FLASH_ByteRead ((unsigned int)
Flash_Storage_Area + 4, SCRATCHPAD);
    board_time.min = FLASH_ByteRead ((unsigned int)
Flash_Storage_Area + 5, SCRATCHPAD);
    board_time.sec = 0;
    OS_DEFER();
    board_time.winter = FLASH_ByteRead((unsigned int)
Flash_Storage_Area+ 6, SCRATCHPAD);
}
else                         // else if button is being pressed
    // use default values and
    // store them immediately as initial values

FLASH[Byte]_Update ((unsigned int) Flash_Storage_
Area, board_time.year/256, SCRATCHPAD);
FLASH[Byte]_Write ((unsigned int) Flash_Storage_
Area + 1, board_time.year%256, SCRATCHPAD);
FLASH[Byte]_Write ((unsigned int) Flash_Storage_
Area + 2, board_time.month, SCRATCHPAD);
FLASH[Byte]_Write ((unsigned int) Flash_Storage_
Area + 3, board_time.day, SCRATCHPAD);
FLASH[Byte]_Write ((unsigned int) Flash_Storage_
Area + 4, board_time.hour, SCRATCHPAD);
FLASH[Byte]_Write ((unsigned int) Flash_Storage_
Area + 5, board_time.min, SCRATCHPAD);

```

```
FLASH_ByteWrite ((unsigned int) Flash_Storage_  
Area + 6, board_time.winter, SCRATCHPAD);  
  
P1 = 0;  
#ifdef _LEGAL_TIME_  
ChangeToSummerTime = last_dom(board_time.year, MAR, SUN);  
ChangeToWinterTime = last_dom(board_time.year, OCT, SUN);  
#endif  
OSRTOS_GO(1); /* Start multitasking, with priority sorting */  
while (1)  
{  
    OS_CPU_IDLE(); /* Go to idle mode if  
        doing nothing, to conserve energy */  
}  
  
}
```



Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders



B.3 F040_FlashPrimitives.c

```
-----  
// F040_FlashPrimitives.c  
-----  
// Copyright 2004 Silicon Laboratories, Inc.  
//  
// This program contains several useful  
// utilities for writing and updating  
// FLASH memory.  
//  
// AUTH: BW & GP  
// DATE: 21 JUL 04  
//  
// Target: C8051F04x  
// Tool chain: KEIL C51 7.06  
//  
// Release 1.0  
//  
//-----  
// Includes  
//-----  
  
#include "F040_FlashPrimitives.h"  
#include "C8051F040.h"  
  
//-----  
// Structures, Unions, Enumerations, and Type Definitions  
//-----  
  
//-----  
// Global Constants  
//-----  
  
//-----  
// Function Prototypes  
//-----  
  
// FLASH read/write/erase routines  
void FLASH_ByteWrite (FLADDR addr, char byte, bit SCRATCHPAD);  
unsigned char FLASH_ByteRead (FLADDR addr, bit SCRATCHPAD);
```

```
void FLASH_PageErase (FLADDR addr, bit SCRATCHPAD);

//-----
// Global Variables
//-----

//-----
// FLASH Routines
//-----

//-----
// FLASH_ByteWrite
//-----

//  

// This routine writes <byte> to the linear FLASH address <addr>.  

// Linear map is decoded as follows:  

// Linear Address Device Address  

// -----  

// 0x00000 - 0xFFFF 0x0000 - 0xFFFF  

//  

void FLASH_ByteWrite (FLADDR addr, char byte, bit SCRATCHPAD)  

{  

    char SFRPAGE_SAVE;                      // to preserve SFRPAGE  

    bit EA_SAVE;                            // to preserve EA  

    char xdata * pwrite;                    // FLASH write pointer  

    SFRPAGE_SAVE = SFRPAGE;                // preserve SFRPAGE  

    EA_SAVE = EA;                          // preserve EA  

    EA = 0;                                // disable interrupts  

    pwrite = (char xdata *) addr;          // initialize write  

                                         // pointer, CAST TO XDATA  

                                         // so as compiler uses  

                                         // MOVX command.  

    SFRPAGE = LEGACY_PAGE;  

    FLSCL |= FLWE;                        // enable FLASH writes/erases  

    PSCTL |= PSWE;                        // PSWE = 1
```

```
if (SCRATCHPAD) {
    PSCTL |= SFLE;                                // set SFLE
}

RSTSRC = 0x02;                                    // enable VDDMON as reset source
*pwrite = byte;                                   // write the byte

if (SCRATCHPAD) {
    PSCTL &= ~SFLE;                             // clear SFLE
}
PSCTL &= ~PSWE;                                  // PSWE = 0
FLSCL &= ~FLWE;                                 // disable FLASH writes/erases

SFRPAGE = SFRPAGE_SAVE;                          // restore SFRPAGE
EA = EA_SAVE;                                   // restore interrupts
}

//-----
// FLASH_ByteRead
//-----
//
// This routine reads a <byte> from the linear FLASH address <addr>.
//
unsigned char FLASH_ByteRead (FLADDR addr, bit SCRATCHPAD)
{
    char SFRPAGE_SAVE;                           // to preserve SFRPAGE
    bit EA_SAVE;                                // to preserve EA
    char code * data pread;                     // FLASH read pointer
    unsigned char byte;

    pread = (char code *) addr;                  // initialize read pointer,
                                                // CAST TO CODE
    SFRPAGE_SAVE = SFRPAGE;                      // preserve SFRPAGE
    EA_SAVE = EA;                               // preserve EA
    EA = 0;                                     // disable interrupts

    SFRPAGE = LEGACY_PAGE;

    if (SCRATCHPAD) {
```

```
    PSCTL |= SFLE;                      // set SFLE
}

byte = *pread;                      // read the byte

if (SCRATCHPAD) {
    PSCTL &= ~SFLE;                  // clear SFLE
}

SFRPAGE = SFRPAGE_SAVE;             // restore SFRPAGE
EA = EA_SAVE;                      // restore interrupts

return byte;
}
//-----
// FLASH_PageErase
//-----
//
// This routine erases the FLASH page containing
// the linear FLASH address
// <addr>.
//
void FLASH_PageErase (FLADDR addr, bit SCRATCHPAD)
{
    char SFRPAGE_SAVE;              // to preserve SFRPAGE
    bit EA_SAVE;                   // to preserve EA
    char xdata * data pwrite;      // FLASH write pointer

    SFRPAGE_SAVE = SFRPAGE;        // preserve SFRPAGE
    EA_SAVE = EA;                 // preserve EA
    EA = 0;                       // disable interrupts

    pwrite = (char xdata *) addr;  // initialize erase pointer,
                                  // CAST TO XDATA

    SFRPAGE = LEGACY_PAGE;

    FLSCL |= FLWE;                // enable FLASH writes/erases
    PSCTL |= (PSWE + PSEE);       // PSWE = 1; PSEE = 1
    if (SCRATCHPAD) {
```

```
    PSCTL |= SFLE;                      // set SFLE
}

RSTSRC = 0x02;                      // enable VDDMON as reset source
*pwrite = 0;                         // initiate page erase

if (SCRATCHPAD) {
    PSCTL &= ~SFLE;                  // clear SFLE
}

PSCTL &= ~(PSWE + PSEE);           // PSWE = 0; PSEE = 0
FLSCL &= ~FLWE;                   // disable FLASH writes/erases

SFRPAGE = SFRPAGE_SAVE;            // restore SFRPAGE
EA = EA_SAVE;                     // restore interrupts
}
```



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com



Click on the ad to read more

B.4 F040_FlashPrimitives.h

```
-----  
// F040_FlashPrimitives.h  
-----  
// Copyright 2004 Silicon Laboratories, Inc.  
//  
// This program contains several useful  
// utilities for writing and updating  
// FLASH memory.  
//  
// AUTH: BW & GP  
// DATE: 21 JUL 04  
//  
// Target: C8051F04x  
// Tool chain: KEIL C51 7.06  
//  
// Release 1.0  
//  
  
#ifndef F040_FLASHPRIMITIVES_H  
#define F040_FLASHPRIMITIVES _H  
  
-----  
// Includes  
-----  
  
-----  
// Structures, Unions, Enumerations, and Type Definitions  
-----  
  
typedef unsigned long ULONG;  
typedef unsigned int UINT;  
typedef unsigned char UCHAR;  
  
-----  
// Global Constants  
-----  
  
#ifndef FLASH_PAGESIZE  
#define FLASH_PAGESIZE 512  
#endif
```

```
#ifndef FLASH_SCRATCHSIZE
#define FLASH_SCRATCHSIZE 128
#endif

#ifndef FLASH_TEMP
#define FLASH_TEMP 0x0F800L
#endif

#ifndef FLASH_LAST
#define FLASH_LAST 0x0FA00L           // last page of FLASH
#endif

typedef UINT FLADDR;

//-----
// Exported Function Prototypes
//-----

// FLASH read/write/erase routines
extern void FLASH_ByteWrite (FLADDR addr, char byte, bit SCRATCHPAD);
extern unsigned char FLASH_ByteRead (FLADDR addr, bit SCRATCHPAD);
extern void FLASH_PageErase (FLADDR addr, bit SCRATCHPAD);

#endif // F040_FLASHPRIMITIVES_H
```

B.5 F040_FlashUtils.c

```
-----  
// F040_FlashUtils.c  
-----  
// Copyright 2004 Silicon Laboratories, Inc.  
//  
// This program contains several useful  
// utilities for writing and updating  
// FLASH memory.  
//  
// AUTH: BW & GP  
// DATE: 21 JUL 04  
//  
// Target: C8051F04x  
// Tool chain: KEIL C51 7.06  
//  
// Release 1.0  
//  
//-----  
// Includes  
//-----  
  
#include "F040_FlashPrimitives.h"  
#include "F040_FlashUtils.h"  
  
//-----  
// Structures, Unions, Enumerations, and Type Definitions  
//-----  
  
//-----  
// Global Constants  
//-----  
  
//-----  
// Function Prototypes  
//-----  
  
// FLASH read/write/erase routines  
void FLASH_Write (FLADDR dest, char *src,  
unsigned numbytes, bit SCRATCHPAD);
```

```

char * FLASH_Read (char *dest, FLADDR src,
unsigned numbytes, bit SCRATCHPAD);
void FLASH_Clear (FLADDR addr, unsigned numbytes, bit SCRATCHPAD);

// FLASH update/copy routines
void FLASH_BytE_Update (FLADDR dest, char databyte, bit SCRATCHPAD);
void FLASH_Update (FLADDR dest, char *src,
unsigned numbytes, bit SCRATCHPAD);
void FLASH_Copy (FLADDR dest, bit destSFLE, FLADDR
src, bit srcSFLE, unsigned numbytes);

// FLASH test routines
void FLASH_Fill (FLADDR addr, ULONG length,
UCHAR fill, bit SCRATCHPAD);

//-----
// Global Variables
//-----

//-----
// FLASH Routines
//-----

//-----
// FLASH_Clear
//-----
// 
// This routine erases <numbytes> starting from the FLASH addressed by
// <dest> by performing a read-modify-write
// operation using <FLASH_TEMP> as
// a temporary holding area. This function accepts <numbytes> up to
// <FLASH_PAGESIZE>.
//
void FLASH_Clear (FLADDR dest, unsigned numbytes, bit SCRATCHPAD)
{
    FLADDR dest_1_page_start;           // first address in 1st page
                                       // containing <dest>
    FLADDR dest_1_page_end;            // last address in 1st page
                                       // containing <dest>
    FLADDR dest_2_page_start;          // first address in 2nd page

```

```

                                // containing <dest>
FLADDR dest_2_page_end;           // last address in 2nd page
                                // containing <dest>
unsigned numbytes_remainder;     // when crossing page boundary,
                                // number of <src>
                                // bytes on 2nd page
unsigned FLASH_pagesize;         // size of FLASH page to update

FLADDR wptr;                     // write address
FLADDR rptr;                     // read address
unsigned length;

if (SCRATCHPAD) {                // update Scratchpad
    FLASH_pagesize = FLASH_SCRATCHSIZE;
} else {
    FLASH_pagesize = FLASH_PAGESIZE;
}

dest_1_page_start = dest & ~(FLASH_pagesize - 1);
dest_1_page_end = dest_1_page_start + FLASH_pagesize - 1;
dest_2_page_start = (dest + numbytes) & ~(FLASH_pagesize - 1);
dest_2_page_end = dest_2_page_start + FLASH_pagesize - 1;

if (dest_1_page_end == dest_2_page_end) {

    // 1. Erase Scratch page
    FLASH_PageErase (FLASH_TEMP, 0);

    // 2. Copy bytes from first byte of dest
    // page to dest-1 to Scratch page

    wptr = FLASH_TEMP;
    rptr = dest_1_page_start;
    length = dest - dest_1_page_start;
    FLASH_Copy (wptr, 0, rptr, SCRATCHPAD, length);

    // 3. Copy from (dest+numbytes) to dest_
    // page_end to Scratch page
}

```

```
wptr = FLASH_TEMP + dest - dest_1_page_start + numbytes;
rptr = dest + numbytes;
length = dest_1_page_end - dest - numbytes + 1;
FLASH_Copy (wptr, 0, rptr, SCRATCHPAD, length);

// 4. Erase destination page
FLASH_PageErase (dest_1_page_start, SCRATCHPAD);

// 5. Copy Scratch page to destination page
wptr = dest_1_page_start;
rptr = FLASH_TEMP;
length = FLASH_pagesize;
FLASH_Copy (wptr, SCRATCHPAD, rptr, 0, length);

} else {                                // value crosses page boundary
    // 1. Erase Scratch page
    FLASH_PageErase (FLASH_TEMP, 0);

    // 2. Copy bytes from first byte of dest
    //      page to dest-1 to Scratch page

    wptr = FLASH_TEMP;
    rptr = dest_1_page_start;
    length = dest - dest_1_page_start;
    FLASH_Copy (wptr, 0, rptr, SCRATCHPAD, length);

    // 3. Erase destination page 1
    FLASH_PageErase (dest_1_page_start, SCRATCHPAD);

    // 4. Copy Scratch page to destination page 1
    wptr = dest_1_page_start;
    rptr = FLASH_TEMP;
    length = FLASH_pagesize;
    FLASH_Copy (wptr, SCRATCHPAD, rptr, 0, length);

    // now handle 2nd page

    // 5. Erase Scratch page
    FLASH_PageErase (FLASH_TEMP, 0);
```

```
// 6. Copy bytes from numbytes remaining to
// dest_2_page_end to Scratch page

numbytes_remainder = numbytes - (dest_1_page_end - dest + 1);
wptr = FLASH_TEMP + numbytes_remainder;
rptr = dest_2_page_start + numbytes_remainder;
length = FLASH_pagesize - numbytes_remainder;
FLASH_Copy (wptr, 0, rptr, SCRATCHPAD, length);

// 7. Erase destination page 2
FLASH_PageErase (dest_2_page_start, SCRATCHPAD);

// 8. Copy Scratch page to destination page 2
wptr = dest_2_page_start;
rptr = FLASH_TEMP;
length = FLASH_pagesize;
FLASH_Copy (wptr, SCRATCHPAD, rptr, 0, length);
}

}

//-----
// FLASH_Update
//-----
// This routine replaces <numbytes> from
// <src> to the FLASH addressed by
// <dest>. This function calls FLASH_Clear()
// to handle the dirty work of
// initializing all <dest> bytes to 0xff's
// prior to copying the bytes from
// <src> to <dest>. This function accepts
// <numbytes> up to <FLASH_PAGESIZE>.
//
void FLASH_Update (FLADDR dest, char *src,
unsigned numbytes, bit SCRATCHPAD)
{
    // 1. Erase <numbytes> starting from <dest> (set to 0xFF)
    FLASH_Clear (dest, numbytes, SCRATCHPAD);

    // 2. Write <numbytes> from <src> to <dest>
```

```
    FLASH_Write (dest, src, numbytes, SCRATCHPAD);
}

//-----
// FLASH_Byte_Update
//-----
//
// This routine replaces 1 byte from <src> to the FLASH addressed by
// <dest>. This function calls FLASH_PageErase()
// to handle the dirty work of
// initializing all page <dest> bytes to 0xff's
// prior to copying the bytes from
// <src> to <dest>.
//
void FLASH_Byte_Update (FLADDR dest, char databyte, bit SCRATCHPAD)
{
    // 1. Erase page starting from <dest> (set to 0xFF)
    FLASH_PageErase (dest, SCRATCHPAD);

    // 2. Write databyte from <src> to <dest>
    FLASH_ByteWrite (dest, databyte, SCRATCHPAD);
}

//-----
// FLASH_Write
//-----
//
// This routine copies <numbytes> from <src>
// to the linear FLASH address
// <dest>.
//
void FLASH_Write (FLADDR dest, char *src,
unsigned numbytes, bit SCRATCHPAD)
{
    FLADDR i;

    for (i = dest; i < dest+numbytes; i++) {
        FLASH_ByteWrite (i, *src++, SCRATCHPAD);
    }
}
```

```
//-----
// FLASH_Read
//-----
//
// This routine copies <numbytes> from the
// linear FLASH address <src> to
// <dest>.
//
char * FLASH_Read (char *dest, FLADDR src,
unsigned numbytes, bit SCRATCHPAD)
{
    FLADDR i;

    for (i = 0; i < numbytes; i++) {
        *dest++ = FLASH_ByteRead (src+i, SCRATCHPAD);
    }
    return dest;
}

//-----
// FLASH_Copy
//-----
//
// This routine copies <numbytes> from <src>
// to the linear FLASH address
// <dest>.
//
void FLASH_Copy (FLADDR dest, bit destSCRATCHPAD,
FLADDR src, bit srcSCRATCHPAD,
unsigned numbytes)

{
    FLADDR i;

    for (i = 0; i < numbytes; i++) {

        FLASH_ByteWrite ((FLADDR) dest+i,
                         FLASH_ByteRead((FLADDR) src+i, srcSCRATCHPAD),
                         destSCRATCHPAD);
    }
}
```

```
//-----  
// FLASH_Fill  
//-----  
  
// This routine fills the FLASH beginning at  
// <addr> with <length> bytes.  
//  
void FLASH_Fill (FLADDR addr, ULONG length,  
unsigned char fill, bit SCRATCHPAD)  
{  
    FLADDR i;  
  
    for (i = 0; i < length; i++) {  
        FLASH_ByteWrite (addr+i, fill, SCRATCHPAD);  
    }  
}
```



The image shows a woman with long blonde hair, wearing a blue tank top, smiling while wearing a black VR headset. To her right is a red rectangular advertisement for MT Højgaard. The ad features the company logo (a stylized 'M' and 'H') and the text 'MT Højgaard'. Below this, in large white capital letters, is the slogan 'BEDRE LØSNINGER'. Underneath the slogan, there is a block of Danish text: 'I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?' At the bottom of the ad, the website 'mth.dk/vorestilgang' is listed.

B.6 F040_FlashUtils.h

```
-----  
// F040_FlashUtils.h  
-----  
// Copyright 2004 Silicon Laboratories, Inc.  
//  
// This program contains several useful  
// utilities for writing and updating  
// FLASH memory.  
//  
// AUTH: BW & GP  
// DATE: 21 JUL 04  
//  
// Target: C8051F04x  
// Tool chain: KEIL C51 7.06  
//  
// Release 1.0  
//  
  
#ifndef F040_FLASHUTILS_H  
#define F040_FLASHUTILS_H  
  
-----  
// Includes  
-----  
  
#include "F040_FlashPrimitives.h"  
  
-----  
// Structures, Unions, Enumerations, and Type Definitions  
-----  
  
-----  
// Global Constants  
-----  
  
-----  
// Exported Function Prototypes  
-----
```

```
// FLASH read/write/erase routines
extern void FLASH_Write (FLADDR dest, char *src,
unsigned numbytes, bit SCRATCHPAD);
extern char * FLASH_Read (char *dest, FLADDR
src, unsigned numbytes, bit SCRATCHPAD);
extern void FLASH_Clear (FLADDR addr,
unsigned numbytes, bit SCRATCHPAD);

// FLASH update/copy routines
extern void FLASH_Byte_Update (FLADDR dest,
char databyte, bit SCRATCHPAD);
extern void FLASH_Update (FLADDR dest, char *src,
unsigned numbytes, bit SCRATCHPAD);
extern void FLASH_Copy (FLADDR dest, bit
destSCRATCHPAD, FLADDR src, bit srcSCRATCHPAD,
unsigned numbytes);

// FLASH test routines
extern void FLASH_Fill (FLADDR addr, ULONG
length, UCHAR fill, bit SCRATCHPAD);

#endif // F040_FLASHUTILS_H
```

B.7 GPS_NMEA_Parser_2.c

```
/*
-----*
GPS_NMEA_Parser_2.c: Garmin GPS Serial Link to C8051F040 Board
    UART0 under Interrupt control, with XON-XOFF controlled buffers.

    RS232 cable from the Garmin eTrex to
    C8051F040 board has to be of the
    null cable type (crossed wires) and not the straight one.
    i.e. pin 2 on one socket connected
    to pin 3 on the other socket and vice versa.

    March 2017
    Paul Debono
-----*/
#include "C8051F040.h"
#include <stdio.h>
#include "GPS_NMEA_Parser_2.h"
#include <string.h>
#include <stdlib.h>

myTime udtTime;
cLocation udtLocation;
bit ValidData;
// #define PrintEnabled
// uncomment above definition if you need to
// print results (usually for testing)

/*
-----*
-----*/
// Improved strtok function, able to handle empty fields elegantly.
// Hence we can test for empty fields without problem
char *improved_strtok(char *buf, const char *delim) {
    char *p;
    static char *secret_state;

    p = buf ? buf : secret_state;
    if (p == NULL)
        return (NULL);
    /*
     * Do *not* skip (span) initial delimiters. Find end of
```

```

        * non-delimiter "word" part, and if
        there is anything after
        * that make sure the next strtok(NULL)
        will pick up from there.
    */
secret_state = p + strcspn(p, delim);
if (*secret_state)
    *secret_state++ = 0;
else
    secret_state = NULL;
return (p);
}

/*****



// Get the separate fields from the GPS sentence and
// call function to decode string accordingly
bit Parse_String(unsigned char *stringRead,
unsigned char SentenceType )
{
    unsigned char xdata * xdata tokenPtr;
    unsigned char field;

    /* Analyze GPS sentence that was collected from Rx buffer */

    tokenPtr = improved_strtok(stringRead,
        ","); // first field (sentence ID)
    field = 0;
    ValidData = 0; // ValidData is then set by the
                    Decode_XXXX_Sentence routine

    switch (SentenceType)
    {
        case GPGGA:
        if ( strcmp(tokenPtr, "$GPGGA") == 0)
        {
            while (tokenPtr!=NULL)
            {
                if (*tokenPtr) Decode_GPGGA_
                    Sentence(tokenPtr, field );

```

```

// decode if not empty field
    tokenPtr = improved_strtok(NULL, ",*");
// continue tokenising string (delimiters ,
or *), note NULL first argument
    ++field;
}
}

break;

case GPRMC:
if ( strcmp(tokenPtr, "$GPRMC") == 0)
{
while (tokenPtr!=NULL)
{
if (*tokenPtr) Decode_GPRMC_
Sentence(tokenPtr, field );
}

// decode if not empty field
    tokenPtr = improved_strtok(NULL, ",*");
// continue tokenising string (delimiters ,
or *), note NULL first argument
    ++field;
}
}

break;

case GPGLL:
if ( strcmp(tokenPtr, "$GPGLL") == 0)
{
while (tokenPtr!=NULL)
{
if (*tokenPtr) Decode_GPGLL_
Sentence(tokenPtr, field );
}

// decode if not empty field
    tokenPtr = improved_strtok(NULL, ",*");
// continue tokenising string (delimiters ,
or *), note NULL first argument
    ++field;
}
}

break;

```

```

        case GPZDA:
            if ( strcmp(tokenPtr, "$GPZDA") == 0 )
            {
                while (tokenPtr!=NULL)
                {
                    if (*tokenPtr) Decode_GPZDA_
                        Sentence(tokenPtr, field );
                    // decode if not empty field
                    tokenPtr = improved_strtok(NULL, ",*");
                    // continue tokenising string (delimiters ,
                    or *), note NULL first argument
                    ++field;
                }
            }
            break;
        }

        return(ValidData); // ValidData is set by
        the Decode_XXXX_Sentence routine
    }

=====
/*
$GPRMC

Recommended minimum specific GPS/Transmit data

eg1. $GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

    225446      Time of fix 22:54:46 UTC
    A           Navigation receiver warning A = OK, V = warning
    4916.45,N   Latitude 49 deg. 16.45 min North
    12311.12,W   Longitude 123 deg. 11.12 min West
    000.5       Speed over ground, Knots
    054.7       Course Made Good, True
    191194      Date of fix 19 November 94
    020.3,E     Magnetic variation 20.3 deg East
    *68         mandatory checksum
*/
void Decode_GPRMC_Sentence(char *SubString, char field )
{

```

```
unsigned long xdata utcTime;
unsigned long xdata utcHour, cetHour;
unsigned long xdata utcMinutes, cetMinutes;
unsigned long xdata utcSeconds, cetSeconds;
unsigned long xdata utcDate, utcYear, utcMonth, utcDay;

float xdata      latitude;
int  xdata      latDegrees;
float xdata      latMinutes;

float xdata      longitude;
int  xdata      longDegrees;
float xdata longMinutes;

switch (field)
{
    case 0: // sentence identifier
        #ifdef PrintEnabled
            printf("%02bu. Sentence ID =
                    %s\n", field, SubString);
        #endif
        break;

    case 1: // UTC Time and convert also to CET
        sscanf(SubString, "%ld", &utcTime);
        utcHour = (utcTime/10000); // extract Hours from long
        utcMinutes = (utcTime - (utcHour*10000))/100; // extract
                                                minutes
                                                from long
        utcSeconds = utcTime - (utcHour*10000)
                    - (utcMinutes*100);
        // extract seconds from long
        if(utcHour >= 0 && utcHour <= 22)
            cetHour = utcHour + 1;
        else cetHour = utcHour - 23;

        cetMinutes = utcMinutes;
        cetSeconds = utcSeconds;
        udtTime.ucHours = (unsigned char)cetHour;
        udtTime.ucMinutes = (unsigned char)cetMinutes;
```

```
        udtTime.ucSeconds = (unsigned char)cetSeconds;
// NB: %02ld formats long to print 2 chars
wide, padding with 0 if necessary
#endif PrintEnabled
printf("%02bu. Time: %02ld:%02ld:%02ld
UTC = %02ld:%02ld:%02ld CET\r\n", \
field, utcHour, utcMinutes, utcSeconds,
cetHour, cetMinutes, cetSeconds);
#endif
break;

case 2: // Navigation Receiver Warning
#ifndef PrintEnabled
printf("%02bu. Navigation Receiver
Warning: %s\r\n", field, SubString);
printf(" A = Data OK\r\n");
printf(" V = Void warning\r\n");
#endif
if (SubString[0] == 'A')
    ValidData = 1;
else
    ValidData = 0;
break;

case 3: // Get latitude: ddmm.mmmm
sscanf(SubString, "%f", &latitude);
latDegrees = (int)(latitude/100);
latMinutes = (float)(latitude - latDegrees*100);
#ifndef PrintEnabled
printf("%02bu. Latitude: %02d DEG : %2.4f MIN\
r\n", field, latDegrees, latMinutes);
#endif
break;

case 4: // Get latitude Cardinal direction (N-S)
#ifndef PrintEnabled
printf("%02bu. Latitude Cardinal direction:
%s\r\n", field, SubString);
#endif
break;
```

```

        case 5: // Get longitude: dddmm.mmmm
        sscanf(SubString, "%f", &longitude);
        longDegrees = (int)(longitude/100);
        longMinutes = (float)(longitude - longDegrees*100);
                #ifdef PrintEnabled
printf("%02bu. Longitude: %03d DEG : %2.4f MIN\
r\n", field, longDegrees, longMinutes);
                #endif
        break;

        case 6: // Get longitude Cardinal direction (E-W)
                #ifdef PrintEnabled
printf("%02bu. Longitude Cardinal
direction: %s\r\n", field, SubString);
                #endif
        break;

        case 7: // Speed over ground
                #ifdef PrintEnabled
printf("%02bu. Speed over ground: %s
knots\r\n", field, SubString);
                #endif
        break;

        case 8: // Course made good, True
                #ifdef PrintEnabled
printf("%02bu. Course made good, True:
%s\r\n", field, SubString);
                #endif
        break;

        case 9: // UTC Date of Fix
        sscanf(SubString, "%ld", &utcDate);
        utcDay = (utcDate/10000); // extract Day from long
        utcMonth = (utcDate - (utcDay*10000))/100; // extract
                                         Month from long
        utcYear = 2000 + utcDate - (utcDay*10000)
        - (utcMonth*100);
// extract Year from long
        udtTime.ucDay = (unsigned char)utcDay;
        udtTime.ucMonth = (unsigned char)utcMonth;

```

```
        udtTime.uiYear = (unsigned int)utcYear;
#ifndef PrintEnabled
printf("%02bu. UTC Date of Fix: %02ld/%02ld/%04ld\
r\n", field, utcDay, utcMonth, utcYear);
#endif
break;

case 10: // Get Magnetic Variation
#ifndef PrintEnabled
printf("%02bu. Magnetic Variation: %s
degrees\r\n", field, SubString);
#endif
break;

case 11: // Get longitude Cardinal direction (E-W)
#ifndef PrintEnabled
printf("%02bu. Longitude Cardinal direction of above
variation : %s\r\n", field, SubString);
#endif
break;

case 12: // Mode indicator
#ifndef PrintEnabled
printf("%02bu. Mode indicator: %s\n", field, SubString);
printf(" A = Autonomous. Sat system used in non-
differential mode in position fix\r\n");
printf (" D = Differential. Sat system used in
differential mode in position fix\r\n");
printf (" E = Estimated (dead reckoning) mode\r\n");
printf ("F = Float RTK. Sat system used in real
time kinematic mode, floating integers\r\n");
printf (" M = Manual input mode\r\n");
printf (" N = No fix. Satellite system not used
in position fix, or fix not valid\r\n");
printf (" P = Precise. Satellite system used in precision mode.\r\n");
printf (" R = RT Kinematic. Sat system used in
RTK mode with fixed integers\r\n");
printf (" S = Simulator mode\r\n");
#endif
break;
```

```
        case 13: // Checksum
            #ifdef PrintEnabled
                printf("%02bu. Checksum: %s\r\n",
                       field, SubString);
            #endif
            break;

        default:
            #ifdef PrintEnabled
                printf("\r\n");           // end of param cases
                                         for GPGGA
            #endif
            break;
    } // end of field cases
}

// =====

void Decode_GPGGA_Sentence(char *SubString, char field )
{
    unsigned long xdata utcTime;
    unsigned long xdata utcHour, cetHour;
    unsigned long xdata utcMinutes, cetMinutes;
    unsigned long xdata utcSeconds, cetSeconds;

    float xdata latitude;
    int xdata    latDegrees;
    float xdata   latMinutes;

    float xdata    longitude;
    int xdata      longDegrees;
    float xdata    longMinutes;

    switch (field)
    {
        case 0: // sentence identifier
            #ifdef PrintEnabled
                printf("%02bu. Sentence ID =
                       %s\n", field, SubString);
            #endif
    }
}
```

```
#endif
ValidData = 1;
break;

case 1: // UTC Time and convert also to CET
    sscanf(SubString, "%ld", &utcTime);
    utcHour = (utcTime/10000); // extract Hours from long
    utcMinutes = (utcTime - (utcHour*10000))/100; // extract
                                                minutes
                                                from long
    utcSeconds = utcTime - (utcHour*10000) - (utcMinutes*100);
// extract seconds from long
    if(utcHour >= 0 && utcHour <= 22)
        cetHour = utcHour + 1;
    else cetHour = utcHour - 23;
    cetMinutes = utcMinutes;
    cetSeconds = utcSeconds;
// NB: %02ld formats long to print 2 chars
// wide, padding with 0 if necessary
#ifndef PrintEnabled
printf("%02bu. Time: %02ld:%02ld:%02ld UTC
= %02ld:%02ld:%02ld CET\n",
field, utcHour, utcMinutes, utcSeconds,
cetHour, cetMinutes, cetSeconds);
#endif
break;

case 2: // Get latitude: ddmm.mmmm
sscanf(SubString, "%f", &latitude);
latDegrees = (int)(latitude/100);
latMinutes = (float)(latitude - latDegrees*100);
#ifndef PrintEnabled
printf("%02bu. Latitude: %02d DEG : %2.4f
MIN\n",
field, latDegrees, latMinutes);
#endif
break;

case 3: // Get latitude Cardinal direction (N-S)
#ifndef PrintEnabled
```

```
printf("%02bu. Latitude Cardinal direction:  
%s\n", field, SubString);  
    #endif  
    break;  
  
case 4: // Get longitude: dddmm.mmmm  
sscanf(SubString, "%f", &longitude);  
longDegrees = (int)(longitude/100);  
longMinutes = (float)(longitude - longDegrees*100);  
    #ifdef PrintEnabled  
printf("%02bu. Longitude: %03d DEG : %2.4f  
MIN\n", field, longDegrees, longMinutes);  
    #endif  
    break;  
  
case 5: // Get longitude Cardinal direction (E-W)  
    #ifdef PrintEnabled  
printf("%02bu. Longitude Cardinal  
direction: %s\n", field, SubString);  
    #endif  
    break;  
  
case 6: // Get Quality of Service  
    #ifdef PrintEnabled  
printf("%02bu. Quality of Service:  
%s\n", field, SubString);  
    #endif  
    break;  
  
case 7: // Get Number of Satellites  
    #ifdef PrintEnabled  
printf("%02bu. Number of Satellites:  
%s\n", field, SubString);  
    #endif  
    break;  
  
case 8: // Precision  
    #ifdef PrintEnabled  
printf("%02bu. Horizontal dilution of  
precision: %s\n", field, SubString);
```

```
#endif
break;

case 9: // Get GPS antenna altitude
#ifndef PrintEnabled
printf("%02bu. GPS antenna altitude:
%s", field, SubString);
#endif
break;

case 10: // Get units of above parameter
#ifndef PrintEnabled
printf(" %s\n", SubString);
#endif
break;

case 11: // Get Geoidal separation
#ifndef PrintEnabled
printf("%02bu. Geoidal separation: %s", field, SubString);
#endif
break;

case 12: // Get units of above parameter
#ifndef PrintEnabled
printf(" %s\n", SubString);
#endif
break;

case 13: // Get Age of DGPS data
#ifndef PrintEnabled
printf("%02bu. Age of DGPS data: %s
seconds\n", field, SubString);
#endif
break;

case 14: // Differential station ID (4
characters before asterisk)
#ifndef PrintEnabled
printf("%02bu. Differential station
ID: %s\n", field, SubString);
#endif
```

```
#endif
        break;
    case 15: // Checksum
        #ifdef PrintEnabled
            printf("%02bu. Checksum: %s\n", field, SubString);
        #endif
        break;

    default:
        #ifdef PrintEnabled
            printf("\n");
        #endif
        // end of param
        cases for GPGGA
    #endif
    break;
} // end of field cases
}

// =====

void Decode_GPGLL_Sentence(char *SubString, char field )
{
    unsigned long xdata utcTime;
    unsigned long xdata utcHour, cetHour;
    unsigned long xdata utcMinutes, cetMinutes;
    unsigned long xdata utcSeconds, cetSeconds;

    float xdata      latitude;
    int xdata       latDegrees;
    float xdata     latMinutes;

    float xdata      longitude;
    int xdata       longDegrees;
    float xdata     longMinutes;

    switch (field)
    {
        case 0: // sentence identifier
            #ifdef PrintEnabled
                printf("%02bu. Sentence ID = %s\n", field, SubString);
            #endif
    }
}
```

```
break;

case 1: // Get latitude: ddmm.mmmm
sscanf(SubString, "%f", &latitude);
latDegrees = (int)(latitude/100);
latMinutes = (float)(latitude - latDegrees*100);
#ifndef PrintEnabled
printf("%02bu. Latitude: %02d DEG : %2.4f
MIN\n", field, latDegrees, latMinutes);
udtLocation.dLatitude = latitude/100.0;
#endif
break;

case 2: // Get latitude Cardinal direction (N-S)
#ifndef PrintEnabled
printf("%02bu. Latitude Cardinal direction:
%s\n", field, SubString);
#endif
break;

case 3: // Get longitude: dddmm.mmmm
sscanf(SubString, "%f", &longitude);
longDegrees = (int)(longitude/100);
longMinutes = (float)(longitude - longDegrees*100);
#ifndef PrintEnabled
printf("%02bu. Longitude: %03d DEG : %2.4f
MIN\n", field, longDegrees, longMinutes);
udtLocation.dLongitude = longitude/100.0;
#endif
break;

case 4: // Get longitude Cardinal direction (E-W)
#ifndef PrintEnabled
printf("%02bu. Longitude Cardinal
direction: %s\n", field, SubString);
#endif
break;

case 5: // UTC Time and convert also to CET
sscanf(SubString, "%ld", &utcTime);
```

```
utcHour = (utcTime/10000); // extract Hours from long
utcMinutes = (utcTime - (utcHour*10000))/100; // extract
                                                minutes from long
utcSeconds = utcTime - (utcHour*10000) - (utcMinutes*100);
// extract seconds from long
if(utcHour >= 0 && utcHour <= 22) cetHour = utcHour + 1;
else cetHour = utcHour - 23;
cetMinutes = utcMinutes;
cetSeconds = utcSeconds;
// NB: %02ld formats long to print 2 chars
      wide, padding with 0 if necessary
#ifndef PrintEnabled
printf("%02bu. Time: %02ld:%02ld:%02ld UTC
= %02ld:%02ld:%02ld CET\n", \
      field, utcHour, utcMinutes, utcSeconds,
      cetHour, cetMinutes, cetSeconds);
#endif
break;

case 6: // Status (1 character before asterisk)
#ifndef PrintEnabled
printf("%02bu. Status: %s\n", field, SubString);
printf(" A = OK\r\n");
printf(" V = warning\r\n");
#endif
if (SubString[0] == 'A')
    ValidData = 1;
else
    ValidData = 0;
break;

case 7: // Checksum
#ifndef PrintEnabled
printf("%02bu. Checksum: %s\n", field, SubString);
#endif
break;

default:
#ifndef PrintEnabled
printf("\n");

```

```

#endif
break;
} // end of field cases

}

//=====
void Decode_GPZDA_Sentence(char *SubString, char field )
{
    unsigned long xdata utcTime;
    unsigned long xdata utcHour, cetHour;
    unsigned long xdata utcMinutes, cetMinutes;
    unsigned long xdata utcSeconds, cetSeconds;

    switch (field)
    {
        case 0: // sentence identifier
            #ifdef PrintEnabled
            printf("%02bu. Sentence ID =
%s\n", field, SubString);
            #endif
            ValidData = 1;
            break;

        case 1: // UTC Time and convert also to CET
            sscanf(SubString, "%ld", &utcTime);
            utcHour = (utcTime/10000); // extract Hours from long
            utcMinutes = (utcTime - (utcHour*10000))/100; // extract
                                            minutes
                                            from long
            utcSeconds = utcTime - (utcHour*10000) - (utcMinutes*100);
// extract seconds from long
            if(utcHour >= 0 && utcHour <= 22) cetHour = utcHour + 1;
            else cetHour = utcHour - 23;
            cetMinutes = utcMinutes;
            cetSeconds = utcSeconds;
            udtTime.ucHours = (unsigned char)utcHour;
            udtTime.ucMinutes = (unsigned char)utcMinutes;
            udtTime.ucSeconds = (unsigned char)utcSeconds;
    }
}

```

```
// NB: %02ld formats long to print 2 chars
     wide, padding with 0 if necessary
#define PrintEnabled
printf("%02bu. Time: %02ld:%02ld:%02ld
UTC = %02ld:%02ld:%02ld CET\n", \
       field, utcHour, utcMinutes, utcSeconds,
       cetHour, cetMinutes, cetSeconds);
#endif
break;

case 2: // Day
#define PrintEnabled
printf("%02bu. Day: %s\n", field, SubString);
#endif
udtTime.ucDay = (unsigned char)atoi(SubString);
break;

case 3: // Month
#define PrintEnabled
printf("%02bu. Month: %s\n", field, SubString);
#endif
udtTime.ucMonth = (unsigned char)atoi(SubString);
break;
case 4: // Year
#define PrintEnabled
printf("%02bu. Year: %s\n", field, SubString);
#endif
udtTime.uiYear = atoi(SubString);
break;

case 5: // Local Zone - hour
#define PrintEnabled
printf("%02bu. Local Zone - hour:
%s\n", field, SubString);
#endif
break;

case 6: // Local Zone - minutes
#define PrintEnabled
```

```
        printf("%02bu. Local Zone - minutes:  
        %s\n", field, SubString);  
    #endif  
    break;  
  
    case 7: // Checksum  
        #ifdef PrintEnabled  
        printf("%02bu. Checksum: %s\n", field, SubString);  
    #endif  
    break;  
  
    default:  
        #ifdef PrintEnabled  
printf("\n");                                // end of param cases for GPGGA  
        #endif  
        break;  
    } // end of field cases  
}
```



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.



B.8 GPS_NMEA_Parser_2.h

```
/* GPS_NMEA_Parser_2.h */

#ifndef GPS_NMEA_H
#define GPS_NMEA_H

typedef struct
{
    unsigned int uiYear;
    unsigned char ucMonth;
    unsigned char ucDay;
    unsigned char ucHours;
    unsigned char ucMinutes;
    unsigned char ucSeconds;
} myTime;

typedef struct
{
    int iYear;
    int iMonth;
    int iDay;
    double dHours;
    double dMinutes;
    double dSeconds;
} cTime;

typedef struct
{
    double dLongitude;
    double dLatitude;
} cLocation;

char *improved_strtok(char *buf, const char *delim);
void Decode_GPRMC_Sentence(char *SubString, char field );
void Decode_GPGGA_Sentence(char *SubString, char field );
void Decode_GPGLL_Sentence(char *SubString, char field );
void Decode_GPZDA_Sentence(char *SubString, char field );
bit Parse_String(char *StringName, unsigned char SentenceType);

enum sentences {GPGGA,GPRMC,GPGLL,GPZDA};

#endif
```

B.9 PaulOS_F040_Parameters.h

```
#define NOOFTASKS           5      // Number of tasks used in
                                    application program

#define STACKSIZE            0x0F  // Number of bytes to allocate
                                    for the stack
// Usually no need to change this parameter

/*
*****
*****
```

```
#define STAND_ALONE_ISR_00 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_01 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_02 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_03 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_04 1 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_05 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_06 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_07 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_08 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_09 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_10 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_11 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_12 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_14 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_15 0 // set to 1 if using this interrupt
                           as a stand alone ISR
```

```
#define NOOFTASKS          5 // Number of tasks used in
                             application program

#define STACKSIZE           0x0F // Number of bytes to allocate
                             for the stack
// Usually no need to change this parameter

/*
*****
*****
```

```
#define STAND_ALONE_ISR_00 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_01 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_02 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_03 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_04 1 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_05 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_06 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_07 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_08 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_09 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_10 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_11 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_12 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_14 0 // set to 1 if using this interrupt
                            as a stand alone ISR
#define STAND_ALONE_ISR_15 0 // set to 1 if using this interrupt
                            as a stand alone ISR
```

```
#define STAND_ALONE_ISR_16 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_17 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_18 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_19 0 // set to 1 if using this interrupt
                           as a stand alone ISR
#define STAND_ALONE_ISR_20 0 // set to 1 if using this interrupt
                           as a stand alone ISR

/*
*****
*/
#endif
```



**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

 **Inspired**

B.10 XonXoff_RRTOS.c

```

/*
**      XonXoff_RRTOS.c
**
**      Modified by Paul Debono (2007) from
**      RTMS 2.0 I/O: serial.c v1.0.0 (16/03/05)
**      By Sasha Jevtic (sjevtic@ece.northwestern.edu)
**
**      Implements an interrupt controlled serial port interface
**      and works in conjunction with PaulOS RTOS
**      TRANSMITS, RECEIVES AND REACTS TO XON/
**      XOFF characters for flow control
**      on the internal UART
**
*/
#include "XonXoff_RRTOS.h"
#include "C8051F040.h"
#include "Paulos_F040.h"      /* Paulos_F040 system calls definitions */

#include <string.h>

typedef enum {NORMAL, FULL, DRAINING, EMPTY} RECV_STATE_T;

/* The way the pointers work is as follows:
   Pointers are always incremented by 1, modulus LEN
   That is if LEN = 32, pointers range from 0 to 31

Tx Buffer:
Data for transmission is placed in the buffer
in the location pointed to by 't_in'.
Data is passed on to SBUF for transmission
via UART using pointer 't_out'.
In each case, the pointer is moved AFTER
reading or writing from/to buffer,
Thus t_out points to the next character to be sent to SBUF.
   t_in points to the location where the
   next character will be placed
               in Tx buffer for future transmission

If AFTER incrementing 't_out',
   't_in' and 't_out' point to the same location,
```

then the buffer is empty.

If AFTER incrementing 't_in',
't_in' = 't_out' point to the same location,
then the buffer is full

Rx Buffer:

Data is received in the buffer in the location pointed to by 'r_in'
Data is read from the buffer using pointer 'r_out'
In each case, the pointer is moved AFTER
reading or writing from/to buffer
Thus r_out points to the next character to be read.
r_in points to the location where the
next character will be placed
when it is received.

If AFTER incrementing 'r_out',
'r_in' and 'r_out' point to the same location,
then the buffer is empty.

If AFTER incrementing 'r_in',
'r_in' = 'r_out' point to the same location,
then the buffer is full

SERIAL INTERRUPTS ARE DISABLED WHEN HANDLING BUFFERS

Meanings of receiver states (and related policies) are as follows:

- NORMAL: Receive buffer is empty or filling, but is not full.
Reception should proceed normally.
- FULL: Receive buffer has been marked full. We need to send an XOFF character so that the sender allows us to relieve our buffer.
- EMPTYING: Receive buffer is full or emptying, but is not empty. An XOFF character has already been sent, so reception should be suspended until the buffer is empty. If we permitted

reception prior to completely emptying the buffer, we would put ourselves in a situation where it is very likely that the buffer would soon fill up again. This would be inefficient, as we wish to keep the XON/XOFF : data byte ratio very low.

This hysteresis helps to achieve that goal.

- EMPTY: Receive buffer has been marked empty. We need to send an XON character so that the sender begins sending us data again.

It is worth noting that for Windows communication, a significant receive buffer is required. Furthermore, the threshold level at which XOFF is sent must be significantly below the buffer's capacity. This is required for a couple of reasons.

First, there might already be an incoming byte working its way through the UART at the time that XOFF is sent. Even assuming an instantaneous response from the DTE to our XOFF, if space does not become available in the buffer prior to the firing of the receive interrupt for this byte, it will be lost.

Secondly, and moreover, Windows seems to be very slow in responding to our XOFF; that is, it continues to send data for a significant period before honoring our stop request. So, we need space to buffer that incoming data.

Received data integrity is a priority.

*/

```
#ifdef SMALLBUFFER      // use IDATA, character size pointers
unsigned char data t_out;           /* transmission buffer start index */
unsigned char data t_in;            /* transmission buffer end index */
char idata TxBuffer[TXLEN];        /* storage for transmission buffer */

// It seems we need 128 bytes of buffer to
// run at 57600 or 115200 kbps.
// when receiving characters from a PC (Windows)
// These values depend on sender reaction time
// (to XOFF) and on the baudrate

unsigned char data r_out;           /* receiving buffer start index */
unsigned char data r_in;            /* receiving buffer end index */
char idata RxBuffer[RXLEN];        /* storage for receiving buffer */

#else // LARGEBUFFER - use XDATA, integer size pointers

unsigned int data t_out;           /* transmission buffer start index */
unsigned int data t_in;            /* transmission buffer end index */
char xdata TxBuffer[TXLEN];        /* storage for transmission buffer */

// It seems we need 128 bytes of buffer to
// run at 57600 or 115200 kbps.
// when receiving characters from a PC (Windows)
// These values depend on sender reaction time
// (to XOFF) and on the baudrate

unsigned int data r_out;           /* receiving buffer start index */
unsigned int data r_in;            /* receiving buffer end index */
char xdata RxBuffer[RXLEN];        /* storage for receiving buffer */

#endif // buffer size

// should be bit
unsigned char TxBfull;             /* flag: marks transmit buffer full */
unsigned char TxActive;             /* flag: marks transmitter active */
unsigned char TxStop;               /* flag: marks XOFF character */
unsigned char TxBusy;               /* flag: marks transmitter busy */
unsigned char TxBempty;              /* flag: marks transmit buffer empty */

unsigned char RxBempty;              /* flag: marks receive buffer empty */
unsigned char RxBfull;               /* flag: marks receive buffer full */
```

```
RECV_STATE_T recvstate;           /* receiver state, for flow control */

enum {FALSE,TRUE} CONDITION_T;
#ifndef SMALLBUFFER
unsigned char RxBufUsed()
{
    unsigned char size;
    if(r_in >= r_out)
        size = (r_in - r_out);
    else
        size = (RXLEN - (r_out - r_in));
    return(size);
}

bit RxBufOverTHLD()
{
    return(RxBufUsed() > RXThreshHold ? TRUE : FALSE);
}

#else
unsigned int RxBufUsed()
{
    unsigned int size;
    if(r_in >= r_out)
        size = (r_in - r_out);
    else
        size = (RXLEN - (r_out - r_in));
    return(size);
}

bit RxBufOverTHLD()
{
    return(RxBufUsed() > RXThreshHold ? TRUE : FALSE);
}

#endif
/*********************************************
/*      putbufU0: write a character to SBUF or transmission buffer */
/*                      Buffer length must be a power of 2 */
/*********************************************
```

```
void putbufU0(char c)
{
    static char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE;                      // Save Current SFR page
    SFRPAGE = UART0_PAGE;
    if (!TxBfull)                                /* transmit only if buffer not
                                                   full */ */
    {
        /*
         Note that if buffer is full, waiting
         is handled by putchar routine
         which calls putbufU0.
        */
        ESO = 0;                                     /* disable serial interrupt */

        if (!TxActive && !TxStop) /* if transmitter not active: */
        {
            TxActive = 1;                /* transfer the first character direct */
            SBUFO = c;                  /* to SBUF to start transmission */
        }
        else                                     /* otherwise: */
        {
            TxBuffer[t_in] = c; /* transfer char to transmit buffer */
            t_in = ++t_in & (TXLEN-1); /* at the same time
                                         incrementing */
            TxBempty = 0;               /* the pointer in circular fashion */
            if (t_in == t_out)
            {
                TxBfull = 1;           /* set flag if buffer is full */
            }
        }
        ESO = 1;                                     /* enable serial interrupt */
    }
    SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

/*****************************************/
/*      putchar:                         */
/*      Places character in Tx buffer if there is space */
/*      otherwise wait (space created by serial ISR) */
/*      */
```

```
*****
char putchar(char c)
{
    if (c == '\n')                                /* expand new line character:      */
    {
        /* can omit if not needed          */
        while (TxBfull)
        {
            OS_DEFER(); /* wait for transmission buffer space */
        }
        /* which is cleared by the serial ISR */
        putbufU0 (0x0D);           /* send CR before LF for <new line> */
    }

    while(TxBfull)
    {
        OS_DEFER();
    }
    /* wait for transmission buffer space */

    putbufU0(c);           /* send character */
    return(c);             /* return character: ANSI requirement */
}

*****
/*      _getkey:                                */
/*      Read data from Rx buffer (using scanf say)   */
/*      Buffer length must be a power of 2          */
/*      Waits (DEFERS) if no character, until one is received via serial */
/*      ISR                                         */
*****
```

```
char _getkey(void)
{
    char data c;
    static char SFRPAGE_SAVE;

    while (RxBempty)
    {
        OS_DEFER();
    }
```

```
}

/*
    wait (DEFER in RTOS) for a character in the Rx buffer.
    In the mean time, any RI interrupt will fill this Rx buffer,
    and therefore RX buffer will no longer remain empty
*/
SFRPAGE_SAVE = SFRPAGE;                      // Save Current SFR page
SFRPAGE = UART0_PAGE;
ES0 = 0;
c = RxBuffer[r_out];                         /* take next char from buffer */
r_out = ++r_out & (RXLEN-1);
if (r_out == r_in) RxBempty = 1;
if ((recvstate == DRAINING) &&           /* if RX was full and
                                             now emptying...*/
    (RxBempty))                            /* ...and RX actually has
                                             emptied */
{
    recvstate = EMPTY;                     /* prepare to send XON to
                                             sending device */
    TIO = 1;                             /* force TI so as to send XON */
}
ES0 = 1;
SFRPAGE = SFRPAGE_SAVE;                      // Restore SFR page
return(c);
}

/*****************
/*          serial: serial receiver / transmitter interrupt      */
/*          Buffer length must be a power of 2 */                  */
/*****************/

serial () interrupt 4 //using 1           /* use registerbank 1 for
                                             interrupt */
{
    char c;
    bit start_trans = 0;
    static char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE;                      // Save Current SFR page
    SFRPAGE = UART0_PAGE;

    if (RI0)                                     /* if receiver interrupt */

```

```
{  
    c = SBUFO;                      /* read character */  
    RI0 = 0;                         /* clear interrupt request flag */  
  
    switch (c)                       /* process character */  
    {  
  
        case CTRL_S:                /* XOFF */  
            TxStop = 1;             /* if Control+S was rcvds stop  
                                         transmission */  
            break;  
  
        case CTRL_Q:                /* XON */  
            start_trans = TxStop;   /* if Control+Q was rcvd start  
                                         transmission */  
            TxStop = 0;  
            break;  
  
        default:                    /* put all other characters into RxBuffer */  
            if (!RxBfull)  
            {  
                RxBuffer[r_in] = c;  
                r_in = ++r_in & (RXLEN-1); /* increment circular pointer */  
                RxBempty = 0;  
            }  
  
            /* check if RX above XOFF threshold */  
            if (RxBufOverTHLD())  
            {  
                if (recvstate == NORMAL) /* prevent "oscillations" */  
                {  
                    recvstate = FULL;      /* prepare to send XOFF */  
                    RxBfull = 1;  
                }  
            }  
            break;  
    }  
    if (TxBusy)  
    {  
        SFRPAGE = SFRPAGE_SAVE; /* do not send  
                                 anything until transmitter is free */  
    }  
}
```

```

        return;
    }
    /* It will return to the ISR when TI=1 after tx */
}                                // end of RIO

if (TIO ||                         /* if transmitter interrupt */
    start_trans ||                  /* or we received an XON
                                    and must start */
    (recvstate == FULL) ||          /* or we need to send an XOFF */
    (recvstate == EMPTY))           /* or we need to send an XON */

{
    if (TIO)
    {
        TxBusy = 0;                /* TX interrupt means Tx
                                       not busy anymore */
        TIO = 0;                   /* clear interrupt request flag */
    }

    if (recvstate == FULL)         /* we need to send an XOFF */
    {
        TxBusy = 1;                /* send XOFF command to
                                       other sender */
        SBUFO = CTRL_S;            /* slowly wait for RX
                                       buffer to drain */
        recvstate = DRAINING;
    }

    else if (recvstate == EMPTY)   /* we need to send an XON */
    {
        TxBusy = 1;                /* send XON to sender */
        SBUFO = CTRL_Q;            /* we are back in business,
                                       receiving */
        recvstate = NORMAL;
    }

    else if (!TxBempty)           /* if more characters
                                    in buffer and */
    {
        if ((!TxStop) && /* if not received Control+S (XOFF) */

```

```

        (recvstate == NORMAL)) /* and receive buffer
                               isn't overwhelmed */
    {
        TxBusy = 1;
        SBUFO = TxBuffer[t_out]; /* transmit character */
        t_out = ++t_out & (TXLEN-1); /* increment circular
                                       pointer */
        if (t_out==t_in) TxBempty =1;
        TxBfull = 0; /* clear 'TxBfull' flag */
    }
}
else TxActive = 0; /* if all chars transmitted, then
                     Tx is not active */
}
SFRPAGE = SFRPAGE_SAVE;
}

void Init_UART0(unsigned int baudrate, unsigned char Timer)
{
    static char SFRPAGE_SAVE;
    switch (Timer)
    {
//-----
// Setup RS232 UART in the Cygnal chip. UART0 uses timer 1 in mode 2
// 8-BIT AUTO-RELOAD.
// to generate baud rate. SMOD0 = 0 (divisor
32) and T0M = 1 (use SYSCLOCK)
// so baud rate formula is
// Baud rate = 22.1184 MHz /(32 * (256 - TH1))
// TH1 = 256 - [22118400/(32 * BR)]
//      = 256 - (691200/BR)
// Serial interrupt enabled
// SYSCLOCK is defined in PaulOS_F040.h          (usually 22118400UL)
//-----
        case 1:
#message "Using Timer 1 for UART0 baudrate" // compile time message
        SFRPAGE_SAVE = SFRPAGE;                      // Preserve SFRPAGE
        SFRPAGE = UART0_PAGE;
        SCON0 = 0x50;           // 8-bit variable baud rate, REN enabled;
        SSTA0 = 0x00;           // Clear all flags; baud rate
        // doubler disabled (SMOD0 = 0 for low baudrates)
    }
}

```

```

SFRPAGE = TIMER01_PAGE;
CKCON |= T1M;           // T1M = 1, use SYSCLOCK for timer 1
TMOD &= 0x0F;          // Clear Timer 1 control bits
TMOD |= 0x20;          // Set Timer 1 to mode 2
TH1 = 256 - (unsigned char)(SYSCLOCK/baudrate/32UL);
TL1 = TH1;
TR1 = 1;               // start timer 1
TI0 = 1;               // Indicate TX0 ready, for UART 0
SFRPAGE = SFRPAGE_SAVE; // Restore SFRPAGE
break;

case 2:
//      #message "Using Timer 2 for UART0
// baudrate" // compile time message
SFRPAGE_SAVE = SFRPAGE; // Preserve SFRPAGE
SFRPAGE = TMR2_PAGE;
TMR2CN = 0x04;          // Timer in 16-bit auto-
                        // reload up timer
// mode
TMR2CF = 0x08;          // SYSCLK is time base; no output;
// up count only
RCAP2 = 65536 - (SYSCLOCK/baudrate/16UL);
TMR2 = RCAP2;
TR2 = 1;                // Start Timer2

SFRPAGE = UART0_PAGE;
SCON0 = 0x50;            // 8-bit variable baud rate;
// 9th bit ignored; RX enabled
// clear all flags
SSTA0 = 0x05; // Clear all flags;
// Use Timer2 as RX and TX baud rate
// source;
IP |= 0x10;
break;
}

// Make sure we start in a clean state.

TxBempty = 1;
TxBfull = 0;
TxActive = 0;

```

```
TxStop      = 0;
TxBusy     = 0;

RxBempty   = 1;
RxBfull    = 0;

recvstate  = NORMAL;
t_out       = 0;
t_in        = 0;
r_out       = 0;
r_in        = 0;

SFRPAGE   = UART0_PAGE;
ES0 = 1;           /* enable serial interrupts */
EA = 1;           /* enable global interrupts */
SFRPAGE = SFRPAGE_SAVE; // Restore SFRPAGE
}

/*********************************************
// Writes to software buffer ONLY if there is space.
// Wait (DEFER in RTOS) for some free buffer space if full
// Send Carriage return before a linefeed (can be omitted)

char sendbyteU0 (char c)
{
    if (c == '\n')          /* expand new line character: */
    {
        /* can omit if not needed */
        while (TxBfull)
        {
            OS_DEFER();
        }
        /* wait for transmission buffer space */
        /* which is cleared by the serial ISR */
        putbufU0 (0x0D);      /* send CR before LF
                                for <new line> */
    }

    while(TxBfull)
    {
        OS_DEFER();
    }
}
```

```

/* wait for transmission buffer space */

putbufU0(c);                                /* send character      */
return(c);
}

/********************* */

char getbyteU0 (void)
{
// Wait (DEFER under RTOS) for key press
// Retrieves a character from the software RX buffer, if available.
// The character from the buffer is returned.
/*                               Buffer length must be a power of 2      */

char c;
static char SFRPAGE_SAVE;

while (RxBempty)
{
    OS_DEFER();
}
/*
    wait (DEFER under RTOS) for a character in Rx buffer,
    in the mean time, any RI interrupt will fill buffer,
    and therefore the RX buffer will no longer remain empty
*/
SFRPAGE_SAVE = SFRPAGE;                      // Save Current SFR page
SFRPAGE = UART0_PAGE;
ES0 = 0;
c = RxBuffer[r_out];                         /* take next char from buffer */
r_out = ++r_out & (RXLEN-1);
if (r_out == r_in) RxBempty = 1;
if ((recvstate == DRAINING) && /* if RX was full and now
                                         emptying... */
(RxBempty))                                /* ...and RX actually has emptied */
{
    recvstate = EMPTY;                      /* prepare to send XON */
    TI0 = 1;                             /* force TI so as to send
                                         XON in ISR */
}
ES0 = 1;

```

```
SFRPAGE = SFRPAGE_SAVE;           // Restore SFRPAGE
return(c);
}

/********************************************/

// Get a string from buffer, if available, terminated by a CR or
// the first 'Length' characters if string is longer.
// Stay waiting (DEFER under RTOS) if nothing in buffer.
// This routine can be modified as required.

void GetStringU0(char StringName[],unsigned char Length)
{
    unsigned char i = 0;
    char c;
    while (i <= (Length-2))
    {
        c = getbyteU0(); /* wait (DEFER in RTOS)
        for some character in buffer */
        if (c == '\r')
        {
            StringName[i] = '\n';
            StringName[i+1] = '\0';
            i = Length;
        }
        else if (c == '\n')
        {
            StringName[i] = '\n';
            StringName[i+1] = '\0';
            i = Length;
        }
        else if (c != 0) StringName[i] = c; // store character in string
        i++;
        if (i == Length) StringName[i] = '\0';
    }
}

/********************************************/
/********************************************/
// For use independently not via printf
```

```
// Send string terminated with a nul character \0
void PrintStringU0 (char s[])
{
    unsigned char i = 0;
    while (s[i] != '\0')
    {
        sendbyteU0(s[i++]);
    }
}
```

Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

B.11 XonXoff_RRTOS.h

```
/*
XonXoff_RRTOS.h
*/

#ifndef __XONXOFF_H__
#define __XONXOFF_H__


#include "C8051F040.h"


//#define CTRL_Q 'q'      // for testing
//#define CTRL_S 's'      // for testing

/* These are the actual XON / XOFF characters */
#define CTRL_Q 0x11      // XON
#define CTRL_S 0x13      // XOFF


// Some commonly used non-printing ASCII codes.
#define CTRL_C 0x03
#define DEL 0x7F
#define BACKSPACE 0x08
#define CR 0x0D
#define LF 0x0A
#define BELL 0x07


/* Use LEN values as POWERS OF TWO in order to work correctly
as a circular buffer with bit AND operations */
#ifndef SMALLBUFFER
#define TXLEN 32
#define RXLEN 16
#define RXThreshHold 8 /* Rx Threshhold for sending XOFF */
#else
#define TXLEN 1024
#define RXLEN 1024
#define RXThreshHold 512 /* Rx Threshhold for sending XOFF */
#endif


void putbufU0(char c);
char putchar(char c);
char _getkey(void);
void Init_UART0(unsigned int br, unsigned char t);
```

```
void PrintStringU0(char s[]);
void GetStringU0(char StringName[],unsigned char Length);
char getbyteU0(void);
char sendbyteU0(char c);

#endif      // end of __XONXOFF_H__
```

BIBLIOGRAPHY

- Ayala, K. J. 1999.** *8051 Microcontroller: Architecture, Programming, and Applications*. s.l. : Delmar Thomson Learning, 1999.
- Barnett, R. H. 1994.** *The 8051 Family of Microcontrollers*. 1st ed. s.l. : Prentice Hall PTR, 1994.
- Blaut, J. 2004.** *8051 RTOS*. B.Sc. Electrical Engineering Thesis, University of Malta. 2004. p. B.Sc. Thesis, B.Sc. Elec. Eng. Thesis.
- Calcutt, D. M., Cowan, F. J. and Parchizadeh, G. H. 1998.** *8051 Microcontrollers Hardware, Software and Applications*. London : Arnold, 1998.
- Chew, M. T. and Gupta, G. S. 2005.** *Embedded Programming with Field-Programmable Mixed-Signal Microcontrollers*. s.l. : Silicon Laboratories, 2005.
- Debono, P. P. 2013a.** *PaulOS: Part I - An 8051 Real-Time Operating System*. 1st ed. s.l. : bookboon.com, 2013a. ISBN: 978-87-403-0449-7.
- . 2013b. *PaulOS: Part II - An 8051 Real-Time Operating System*. 1st ed. s.l. : bookboon.com, 2013b. ISBN: 9788740304503.
- . 2015. *PaulOS-F020: An RTOS for the C8051F020*. 2015. p. 204.
- Huang, H. 2009.** *Embedded System Design with the C8051*. Stanford : Cengage Learning, 2009.



MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang

- Intel.** 1993. *MCS51 Microcontroller Family User's Manual*. Santa Clara, CA, USA : Intel Corporation, 1993.
- Labrosse, J. J.** 2002. *Embedded Systems Building Blocks*. [ed.] U. S. A. San Francisco. 2nd ed. s.l. : CMP Books, 2002.
- . 2002. *MicroC/OS-II, The Real-Time Kernel*. 2nd ed. San Francisco, CA, USA : CMP Books, 2002.
- Labs, Silicon.** 2003. AN122 - Annotated 'C' Examples for the 'F02x' Family. Austin, TX, USA : Silicon Laboratories Inc., 2003.
- . 2003. C8051F020 Data Sheet. Austin, TX, USA : Silicon Laboratories Inc., 2003.
- . 2004. C8051F040 Data Sheet. Austin, TX, USA : Silicon Laboratories Inc., 2004. p. 328.
- Mackenzie, I. S.** 1998. *The 8051 Microcontroller*. 3rd ed. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1998.
- Mazidi, M. A. and Mazidi, J. G.** 1999. *The 8051 Microcontroller and Embedded Systems with Disk*. 1st ed. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1999.
- Pont, M. J.** 2002. *Embedded C*. 1st ed. Boston : Addison-Wesley Longman Publishing Co., Inc., 2002.
- . 2002. *Patterns for Time-Triggered Embedded Systems: Building reliable applications with the 8051 family of microcontrollers*. Boston : Addison-Wesley Longman Publishing Co., Inc., 2002.
- Predko, M.** 1999. *Programming and Customizing the 8051 Microcontroller*. New York, NY, USA : McGraw-Hill, Inc., 1999.
- Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment*. **Liu, C. L. and Layland, J. W.** 1973. 1973, J. ACM, Vol. 20, pp. 40-61.
- Schultz, T. W.** 2004. *C and the 8051*. s.l. : Pagefree Publishing, 2004.
- . 1999. *C and the 8051 (volume II): building efficient applications*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1999.
- . 1997. *C and the 8051: Hardware, Modular Programming and Multitasking with Cdrom*. 2nd ed. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1997.
- Stewart, J. W.** 1999. *The 8051 Microcontroller: hardware, software and interfacing*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1999.
- Thorne, M.** 1986. *Programming the 8086/8088 for the IBM PC and compatibles*. Redwood City, CA, USA : Benjamin-Cummings Publishing Co., Inc., 1986.

INDEX

The index presented here is not the creation of a well thought out index. Since this book would be presented in an electronic form, the reader would be far better served by using a free and easily available text search, usually built-in within the e-reader software.

C

C8051F040TB 13
co-operative 8, 9, 27, 29, 39, 136

D

development boards 14

E

Examples
PaulOS RTOS - C 50

I

ISR
stand-alone - PaulOS 46

M

memory
code area 16
external 17
internal data 18
on-chip 18
organisation 15

P

PaulOS 8, 9, 26, 27, 29, 30, 31, 33, 41, 46, 47, 50, 74, 79,
136, 137, 138, 139, 142, 153, 158, 227, 285, 306
OS_CPU_DOWN() 46
OS_CREATE_TASK() 32, 33, 35
OS_DEFER() 31, 32, 44
OS_INIT_RTOS() 32, 33
OS_KILL_IT() 31, 32, 43
OS_PAUSE_RTOS() 45
OS_PERIODIC() 32, 33
OS_PERIODIC_A() 45
OS_RESUME_RTOS() 45
OS_RESUME_TASK() 32, 33
OS_RTOS_GO() 32, 33, 35
OS_RUNNING_TASK_ID() 31, 37
OS_SCHECK() 31, 33, 37
OS_SIGNAL_TASK() 32, 33, 38

OS_WAITI() 32, 40
OS_WAITP() 31, 32, 39
OS_WAITS() 32, 41
OS_WAITS_A() 45
OS_WAITT() 32, 42
OS_WAITT_A() 45
ready 30
running 29
stand-alone ISR 46
waiting 29
programming 8, 9, 15, 17, 26, 39, 133
pitfalls 5, 9, 133, 136, 308
tips 9, 133, 308

S

SFR 8, 9, 14, 18, 22, 24, 26, 34, 45, 46, 134
paging 8, 9, 14, 18, 26, 134
SFRPAGE 26
SFRPGCN 26

T

tips
Crossbar Setup 135
C tips 137
interrupts 136
programming 133
ram size 133
SFRs 134
System Clock 135
Watchdog Timer 135

END NOTES

¹ The original idea for this RTOS came from the book “C and the 8051 – Building Efficient Applications – Volume II” by Thomas W. Schultz and published by Prentice Hall (0-13-521121-2). In this book, Prof. Schultz discusses the development of two real-time kernels. The first one is the RTKS which I corrected and developed into PaulOS co-operative RTOS. The second one is the RTKB which I also corrected, modified and developed into MagnOS pre-emptive RTOS. Both operating systems, RTKS and RTKB as written in the book are not fully functional, contain some errors and lack some essential components. I did correspond with Prof. Schultz and sent him my modifications and final versions of the programs which he later acknowledged in the 3rd edition of the book “C and the 8051”, again published by Prentice-Hall (0-58961-237-X). So I am particularly grateful to Prof. Schultz for being the catalyst of my increased interest in RTOSs.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

