5 Development Board Manual

The MSA0654 Development Board contains a M30624FGLFP M16C 62 series microcontroller, along with a serial interface for connection to a PC and some switches and display devices. The unit allows M16C programs to be developed.

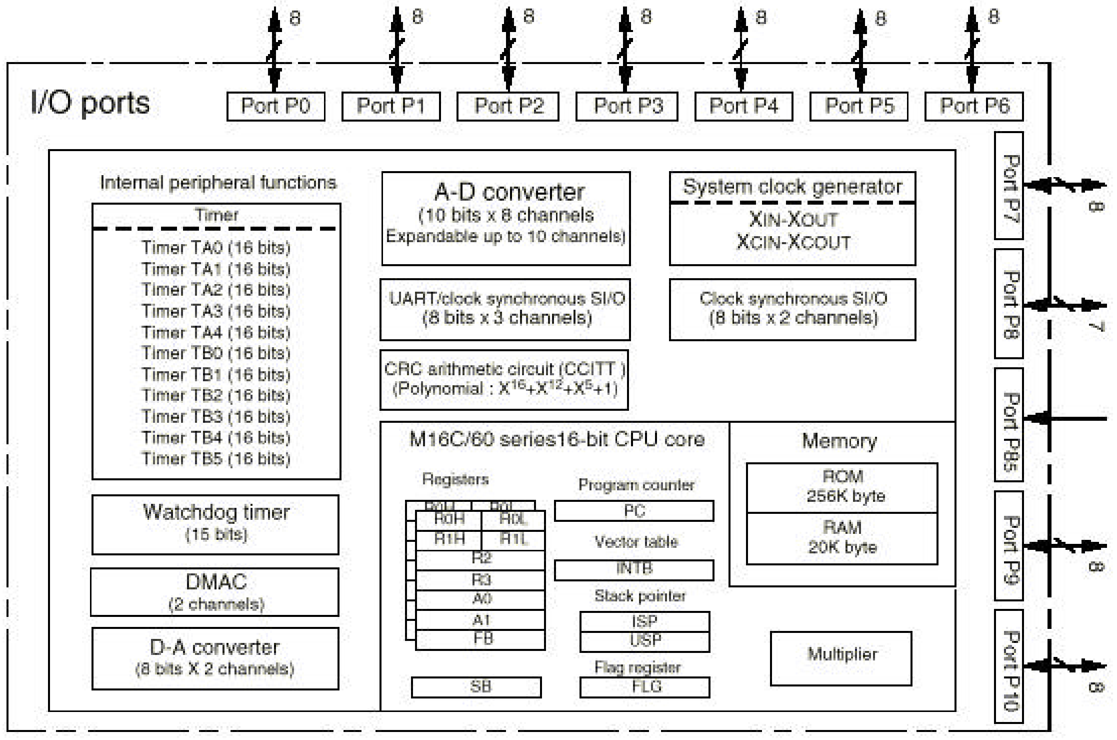
The unit also contains a monitor program. This monitor program is used with the C-SPY debugger program to allow advanced debugging of programs, by means such as breakpoints and single stepping. The fact that there is a monitor program on the microcontroller in addition to the user program has to be taken into account when writing user programs, since some of the chips resources must be reserved for the user program. These limitations are described here.

In addition to these limitations, the monitor program also adjusts some register values, as detailed in this document.

Note that it is possible to use the development board to construct a stand-alone system by replacing the monitor program with a user program programmed in to the flash memory on the microcontroller. Full details of reprogramming the monitor can be found in the M16C/62 StarterKit2 manual.

5.1 M30624FGLFP Microcontroller

The M16C/62 group accommodates certain units in a single chip. These units include ROM and RAM to store instructions and data and the central processing unit (CPU) to execute arithmetic/logic operations. Also included are peripheral units such as timers, serial I/O, D-A converter, DMAC, CRC calculation circuit, A-D converter, and I/O ports. The following explains each unit.



5.1.1 Memory

The M16C/62 group address space extends the 1M bytes from address 00000_{16} to $FFFFF_{16}$. From $FFFFF_{16}$ down is ROM. In the M30624FGLFP there is 256K bytes of internal ROM from C0000h to FFFFFh. The vector table for fixed interrupts such as the reset and NMI are mapped to $FFFDC_{16}$ to $FFFFF_{16}$. The starting address of the interrupt routine is stored here. The address of the vector table for timer interrupts, etc., can be set as desired using the internal register (INTB).

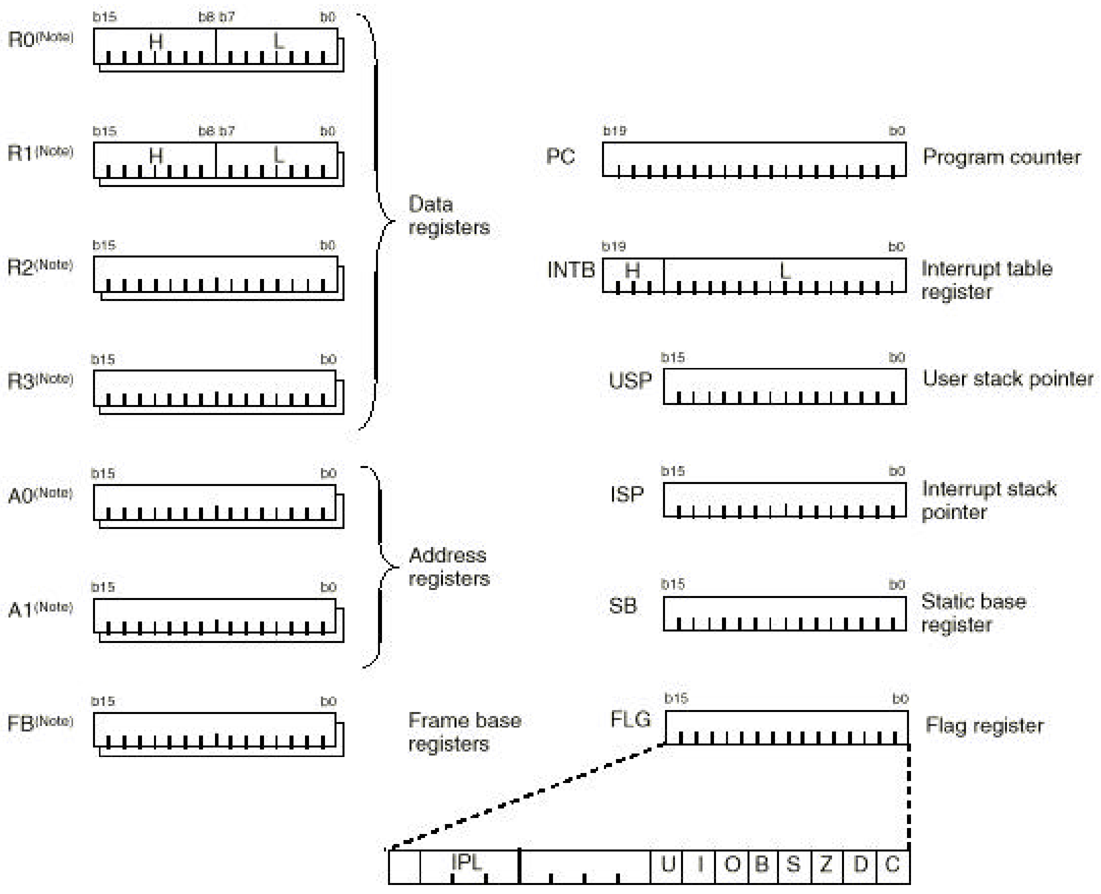
From 00400_{16} up is RAM. For example, in the M30624FGLFP, 20K bytes of internal RAM is mapped to the space from 00400_{16} to $053FF_{16}$. In addition to storing data, the RAM also stores the stack used when calling subroutines and when interrupts are generated.

The SFR area is mapped to 00000_{16} to $003FF_{16}$. This area accommodates the control registers for peripheral devices such as I/O ports, A-D converter, serial I/O, and timers, etc. Any part of the SFR area that is not occupied is reserved and cannot be used for other purposes.

The special page vector table is mapped to $FFE00_{16}$ to $FFFDB_{16}$. If the starting addresses of subroutines or the destination addresses of jumps are stored here, subroutine call instructions and jump instructions can be used as 2-byte instructions, reducing the number of program steps.

5.1.2 Central Processing Unit (CPU)

The CPU has a total of 13 registers shown below. Seven of these registers (R0, R1, R2, R3, A0, A1, and FB) come in two sets; therefore, these have two register banks.



Note: These registers consist of two register banks.

(1) Data registers (R0, R0H, R0L, R1, R1H, R1L, R2, and R3)

Data registers (R0, R1, R2, and R3) are configured with 16 bits, and are used primarily for transfer and arithmetic/logic operations.
Registers R0 and R1 each can be used as separate 8-bit data registers, high-order bits as (R0H/R1H), and low-order bits as (R0L/R1L). In some instructions, registers R2 and R0, as well as R3 and R1 can use as 32-bit data registers (R2R0/R3R1).

(2) Address registers (A0 and A1)

Address registers (A0 and A1) are configured with 16 bits, and have functions equivalent to those of data registers. These registers can also be used for address register indirect addressing and address register relative addressing.
In some instructions, registers A1 and A0 can be combined for use as a 32-bit address register (A1A0).

(3) Frame base register (FB)

Frame base register (FB) is configured with 16 bits, and is used for FB relative addressing.

(4) Program counter (PC)

Program counter (PC) is configured with 20 bits, indicating the address of an instruction to be executed.

(5) Interrupt table register (INTB)

Interrupt table register (INTB) is configured with 20 bits, indicating the start address of an interrupt vector table.

(6) Stack pointer (USP/ISP)

Stack pointer comes in two types: user stack pointer (USP) and interrupt stack pointer (ISP), each configured with 16 bits. Your desired type of stack pointer (USP or ISP) can be selected by a stack pointer select flag (U flag). This flag is located at the position of bit 7 in the flag register (FLG).

(7) Static base register (SB)

Static base register (SB) is configured with 16 bits, and is used for SB relative addressing.

(8) Flag register (FLG)

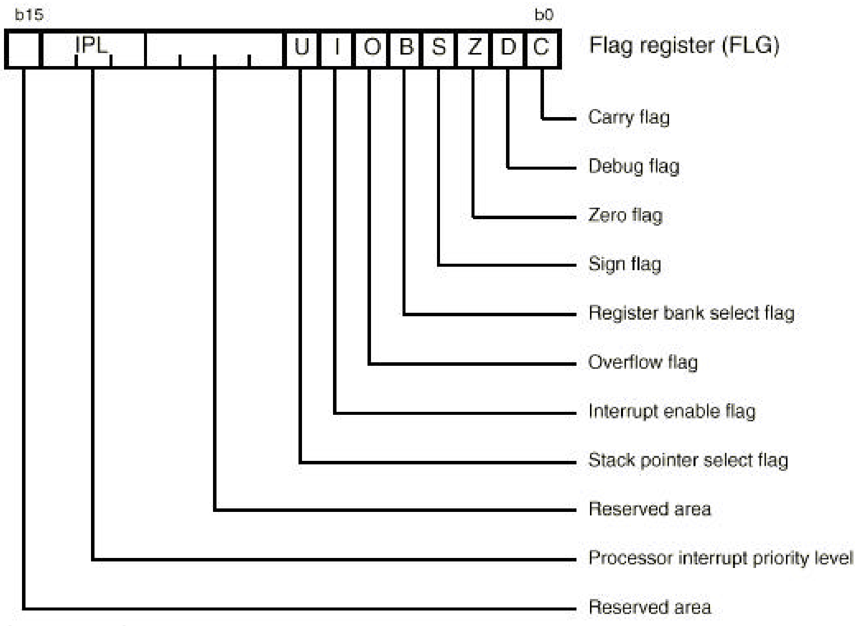
Flag register (FLG) is configured with 11 bits, each bit is used as a flag. The following explains the function of each flag:

- **Bit 0: Carry flag (C flag)**
  This flag retains a carry, borrow, or shift-out bit that has occurred in the arithmetic/logic unit.
- **Bit 1: Debug flag (D flag)**
  This flag enables a single-step interrupt.
  When this flag is “1”, a single-step interrupt is generated after instruction execution. This flag is cleared to “0” when the interrupt is acknowledged.
- **Bit 2: Zero flag (Z flag)**
  This flag is set to “1” when an arithmetic operation resulted in 0; otherwise, cleared to “0”.
- **Bit 3: Sign flag (S flag)**
  This flag is set to “1” when an arithmetic operation resulted in a negative value; otherwise, cleared to “0”.
- **Bit 4: Register bank select flag (B flag)**
  This flag chooses a register bank. Register bank 0 is selected when this flag is “0”; register bank 1 is selected when this flag is “1”.
- **Bit 5: Overflow flag (O flag)**
  This flag is set to “1” when an arithmetic operation resulted in overflow; otherwise, cleared to “0”.
- **Bit 6: Interrupt enable flag (I flag)**
  This flag enables a maskable interrupt.
  An interrupt is disabled when this flag is “0”, and is enabled when this flag is “1”. This flag is cleared to “0” when the interrupt is acknowledged.
- **Bit 7: Stack pointer select flag (U flag)**
  Interrupt stack pointer (ISP) is selected when this flag is “0” ; user stack pointer (USP) is selected when this flag is “1”.
  This flag is cleared to “0” when a hardware interrupt is acknowledged or an INT instruction of software interrupt Nos. 0 to 31 is executed.
- **Bits 8 to 11: Reserved area**
- **Bits 12 to 14: Processor interrupt priority level (IPL)**

Processor interrupt priority level (IPL) is configured with three bits, for specification of up to eight processor interrupt priority levels from level 0 to level 7.
If a requested interrupt has priority greater than the processor interrupt priority level (IPL), the interrupt is enabled.

- **Bit 15: Reserved area**

The C, Z, S, and O flags are changed when instructions are executed. See the software manual for details.



5.1.3 Other Registers

The following is a list of standard names for the microcontroller's registers. In order to use these names in C or Assembler Routines, you need to include a definition file such as M3062F.h

Processor Mode reg	**PM**	Processor Mode reg 0	**PM0**
		Processor Mode reg 1	**PM1**
System clock control reg	**CM**	System clock control reg 0	**CM0**
		System clock control reg 1	**CM1**
Protect reg	**PRCR**		
Watchdog timer start reg	**WDTS**	Watchdog timer control reg	**WDC**
Chip select control reg	**CSR**	Address match interrupt enable	**AIER**
Address match interrupt reg 0 (low)	**RMAD0L**	Address match interrupt reg 1 (low)	**RMAD1L**
Address match interrupt reg 0 (mid)	**RMAD0M**	Address match interrupt reg 1 (mid)	**RMAD1M**
Address match interrupt reg 0 (high)	**RMAD0H**	Address match interrupt reg 1 (high)	**RMAD1H**
DMA0 source pointer (word)	**SAR0**	DMA1 source pointer (long)	**SAR1**
DMA0 source pointer (low)	**SAR0L**	DMA1 source pointer (low)	**SAR1L**
DMA0 source pointer (mid)	**SAR0M**	DMA1 source pointer (mid)	**SAR1M**
DMA0 source pointer (high)	**SAR0H**	DMA1 source pointer (high)	**SAR1H**
DMA0 destination pointer (long)	**DAR0**	DMA1 destination pointer (long)	**DAR1**
DMA0 destination pointer (low)	**DAR0L**	DMA1 destination pointer (low)	**DAR1L**
DMA0 destination pointer (mid)	**DAR0M**	DMA1 destination pointer (mid)	**DAR1M**
DMA0 destination pointer (high)	**DAR0H**	DMA1 destination pointer (high)	**DAR1H**
DMA0 transfer counter	**TCR0**	DMA1 transfer counter	**TCR1**
DMA0 transfer counter (low)	**TCR0L**	DMA1 transfer counter (low)	**TCR1L**
DMA0 transfer counter (high)	**TCR0H**	DMA1 transfer counter (high)	**TCR1H**
DMA0 control reg	**DM0CON**	DMA1 control reg .	**DM1CON**
DMA0 interrupt control reg .	**DM0IC**	DMA1 interrupt control reg	**DM1IC**
Key input interrupt control reg	**KUPIC**		
AD conversion interrupt control reg	**ADIC**		

UART0 transmit interrupt control reg	**S0TIC**	UART1 transmit interrupt control reg	**S1TIC**
UART0 receive interrupt control reg	**S0RIC**	UART1 receive interrupt control reg	**S1RIC**
TimerA0 interrupt control reg	**TA0IC**	TimerB0 interrupt control reg	**TB0IC**
TimerA1 interrupt control reg	**TA1IC**	TimerB1 interrupt control reg	**TB1IC**
TimerA2 interrupt control reg	**TA2IC**	TimerB2 interrupt control reg	**TB2IC**
TimerA3 interrupt control reg	**TA3IC**	TimerB3 interrupt control reg	**TB3IC**
TimerA4 interrupt control reg	**TA4IC**	TimerB4 interrupt control reg	**TB4IC**
		TimerB5 interrupt control reg	**TB5IC**
Interrupt0 interrupt control reg	**INT0IC**	INT3 interrupt control reg	**INT3IC**
Interrupt1 interrupt control reg	**INT1IC**	INT4 interrupt control reg	**INT4IC**
Interrupt2 interrupt control reg	**INT2IC**	INT5 interrupt control reg	**INT5IC**
TimerA/B count start flags	**TABSR**		
Clock prescaler reset flag	**CPSRF**		
One-shot start flag	**ONSF**		
Trigger select reg	**TRGSR**		
Up- down-count selection flag	**UDF**		
TimerA0	**TA0**	TimerA1	**TA1**
TimerA0 (low byte)	**TA0L**	TimerA1 (low byte)	**TA1L**
TimerA0 (high byte)	**TA0H**	TimerA1 (high byte)	**TA1H**
TimerA2	**TA2**	TimerA3	**TA3**
TimerA2 (low byte)	**TA2L**	TimerA3 (low byte)	**TA3L**
TimerA2 (high byte)	**TA2H**	TimerA3 (high byte)	**TA3H**
TimerA4	**TA4**	TimerB0	**TB0**
TimerA4 (low byte)	**TA4L**	TimerB0 (low byte)	**TB0L**
TimerA4 (high byte)	**TA4H**	TimerB0 (high byte)	**TB0H**
TimerB1	**TB1**	TimerB2	**TB2**
TimerB1 (low byte)	**TB1L**	TimerB2 (low byte)	**TB2L**
TimerB1 (high byte)	**TB1H**	TimerB2 (high byte)	**TB2H**
TimerB3	**TB3**	TimerB4	**TB4**
TimerB3 (low byte)	**TB3L**	TimerB4 (low byte)	**TB4L**
TimerB3 (high byte)	**TB3H**	TimerB4 (high byte)	**TB4H**
TimerB5	**TB5**		
TimerB5 (low byte)	**TB5L**		
TimerB5 (high byte)	**TB5H**		
TimerA0 mode reg	**TA0MR**	TimerB0 mode reg	**TB0MR**
TimerA1 mode reg	**TA1MR**	TimerB1 mode reg	**TB1MR**
TimerA2 mode reg	**TA2MR**	TimerB2 mode reg	**TB2MR**
TimerA3 mode reg	**TA3MR**	TimerB3 mode reg	**TB3MR**
TimerA4 mode reg	**TA4MR**	TimerB4 mode reg	**TB4MR**
		TimerB5 mode reg	**TB5MR**
UART0 mode reg	**U0MR**	UART1 mode reg	**U1MR**
UART0 baud rate generator	**U0BRG**	UART1 baud rate generator	**U1BRG**
UART0 transmit buffer	**U0TB**	UART1 transmit buffer	**U1TB**
UART0 transmit buffer (low byte)	**U0TBL**	UART1 transmit buffer (low byte)	**U1TBL**
UART0 transmit buffer (high byte)	**U0TBH**	UART1 transmit buffer (high byte)	**U1TBH**
UART0 control reg	**U0C**	UART1 control reg	**U1C**
UART0 control reg 0	**U0C0**	UART1 control reg 0	**U1C0**
UART0 control reg 1	**U0C1**	UART1 control reg 1	**U1C1**
UART0 receive buffer	**U0RB**	UART1 receive buffer	**U1RB**
UART0 receive buffer (low byte)	**U0RBL**	UART1 receive buffer (low byte)	**U1RBL**
UART0 receive buffer (high byte)	**U0RBH**	UART1 receive buffer (high byte)	**U1RBH**
UART control reg 2	**UCON**		
UART2 transmit buffer reg	**U2TB**	UART2 transmit interrupt control reg	**S2TIC**
UART2 transmit buffer reg (low byte)	**U2TBL**	UART2 receive interrupt control reg	**S2RIC**
UART2 transmit buffer reg (high byte)	**U2TBH**	UART2 special mode reg 2	**U2SMR2**
UART2 receive buffer reg	**U2RB**	UART2 special mode reg	**U2SMR**
UART2 receive buffer reg (low byte)	**U2RBL**	UART2 transmit/receive mode reg	**U2MR**
UART2 receive buffer reg (high byte)	**U2RBH**	UART2 bit rate generator	**U2BRG**
UART2 transmit/receive control reg 0	**U2C0**	UART2 transmit/receive control reg 1	**U2C1**
DMA0 cause selection	**DM0SL**	DMA1 cause selection	**DM1SL**
CRC data reg	**CRCD**	CRC data reg (low byte)	**CRCDL**
CRC data reg (high byte)	**CRCDH**	CRC input reg	**CRCIN**
A/D reg 0	**AD0**	A/D reg 1	**AD1**

A/D reg 0 (low byte)	**AD0L**	A/D reg 1 (low byte)	**AD1L**
A/D reg 0 (high byte)	**AD0H**	A/D reg 1 (high byte)	**AD1H**
A/D reg 2	**AD2**	A/D reg 3	**AD3**
A/D reg 2 (low byte)	**AD2L**	A/D reg 3 (low byte)	**AD3L**
A/D reg 2 (high byte)	**AD2H**	A/D reg 3 (high byte)	**AD3H**
A/D reg 4	**AD4**	A/D reg 5	**AD5**
A/D reg 4 (low byte)	**AD4L**	A/D reg 5 (low byte)	**AD5L**
A/D reg 4 (high byte)	**AD4H**	A/D reg 5 (high byte)	**AD5H**
A/D reg 6	**AD6**	A/D reg 7	**AD7**
A/D reg 6 (low byte)	**AD6L**	A/D reg 7 (low byte)	**AD7L**
A/D reg 6 (high byte)	**AD6H**	A/D reg 7 (high byte)	**AD7H**
A/D control reg	**ADCON**	A/D control reg 0	**ADCON0**
A/D control reg 1	**ADCON1**	A/D control reg 2	**ADCON2**
D-A control reg	**DACON**		
D-A reg 0	**DA0**	D-A reg 1	**DA1**
Port0 and Port1 reg	**P01**	Port2 and Port3 reg	**P23**
Port0 reg	**P0**	Port2 reg	**P2**
Port1 reg	**P1**	Port3 reg	**P3**
Port0 direction reg	**P0D**	Port2 direction reg	**P2D**
Port1 direction reg	**P1D**	Port3 direction reg	**P3D**
Port0 and Port1 direction reg	**P01D**	Port2 and Port3 direction reg	**P23D**
Port4 and Port5 reg	**P45**	Port6 and Port7 reg	**P67**
Port4 reg	**P4**	Port6 reg	**P6**
Port5 reg	**P5**	Port7 reg	**P7**
Port4 direction reg	**P4D**	Port6 direction reg	**P6D**
Port5 direction reg	**P5D**	Port7 direction reg	**P7D**
Port4 and Port5 direction reg	**P45D**	Port6 and Port7 direction reg	**P67D**
Port8 and Port9 reg	**P89**		
Port8 reg	**P8**		
Port9 reg	**P9**	Port10 reg	**P10**
Port8 direction reg	**P8D**	Port10 direction reg	**P10D**
Port9 direction reg	**P9D**		
Port8 and Port9 direction reg	**P89D**		
Pull-up reg 0 and 1	**PUR01**	Pull-up reg 1	**PUR1**
Pull-up reg 0	**PUR0**	Pull-up reg 2	**PUR2**
Data Bank reg	**DBR**		
SI/O3 interrupt control reg	**S3IC**	SI/O4 interrupt control reg	**S4IC**
Bus collision detection interrupt reg	**BCNIC**		
TimerB3,4, 5 count start flag	**TBSR**		
TimerA1-1 reg	**TA11**	TimerA2-1 reg	**TA21**
TimerA1-1 reg (low byte)	**TA11L**	TimerA2-1 reg (low byte)	**TA21L**
TimerA1-1 reg (high byte)	**TA11H**	TimerA2-1 reg (high byte)	**TA21H**
TimerA4-1 reg	**TA41**		
TimerA4-1 reg (low byte)	**TA41L**		
TimerA4-1 reg (high byte)	**TA41H**		
Three-phase PWM control reg 0	**INVC0**	Three-phase PWM control reg 1	**INVC1**
Three-phase output buffer reg 0	**IDB0**	Three-phase output buffer reg 1	**IDB1**
Dead time timer	**DTT**		
Timer B2 interrupt occurrence frequency	**ICTB2**		
Interrupt cause select reg	**IFSR**		
SI/O3 transmit/receive reg	**S3TRR**	SI/O4 transmit/reveive reg	**S4TRR**
SI/O3 control reg	**S3C**	SI/O4 control reg	**S4C**
SI/O3 bit rate generator	**S3BRG**	SI/O4 bit rate generator	**S4BRG**
Port control reg	**PCR**		
FLASH memory control 0 & 1	**FMCR**	FLASH memory control 1	**FMCR1**
FLASH memory control 0	**FMCR0**	FLASH ROM code protect function	**ROMCP**

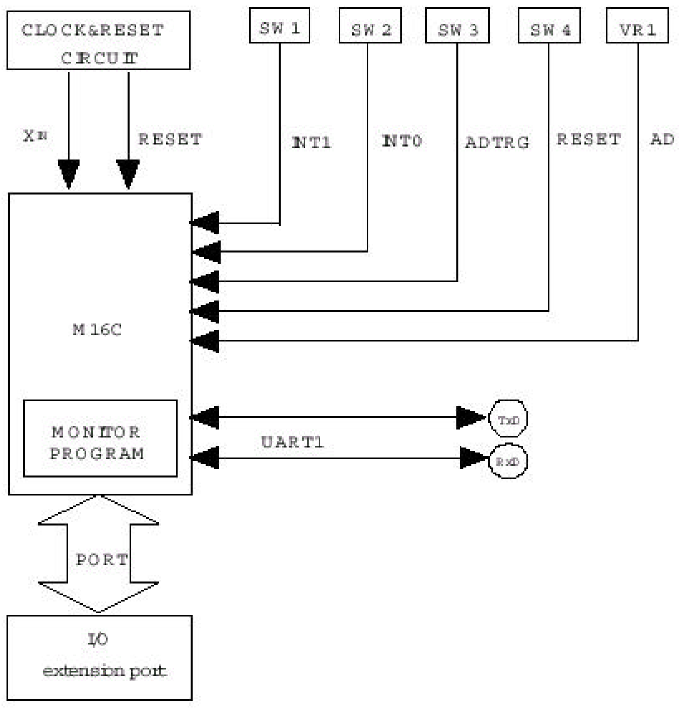
5.2 Standard Interrupt Vectors

0	INTB (l) to INTB+3 (h)	BRK (not maskable)
4	INTB+16 (l) to INTB+19 (h)	INT3
5	INTB+20 (l) to INTB+23 (h)	Timer B5
6	INTB+24 (l) to INTB+27 (h)	Timer B4
7	INTB+28 (l) to INTB+31 (h)	Timer B3
8	INTB+32 (l) to INTB+35 (h)	Selectable to SI/O4 or INT5
9	INTB+36 (l) to INTB+39 (h)	Selectable to SI/O3 or INT4
10	INTB+40 (l) to INTB+43 (h)	Bus collision detection
11	INTB+44 (l) to INTB+47 (h)	DMA0
12	INTB+48 (l) to INTB+51 (h)	DMA1
13	INTB+52 (l) to INTB+55 (h)	Key input interrupt
14	INTB+56 (l) to INTB+59 (h)	A to D
15	INTB+60 (l) to INTB+63 (h)	UART2 transmit or I^{2}C NACK
16	INTB+64 (l) to INTB+67 (h)	UART2 receive or I^{2}C ACK
17	INTB+68 (l) to INTB+71 (h)	UART0 transmit
18	INTB+72 (l) to INTB+75 (h)	UART0 receive
19	INTB+76 (l) to INTB+79 (h)	UART1 transmit
20	INTB+80 (l) to INTB+83 (h)	UART1 receive
21	INTB+84 (l) to INTB+87 (h)	Timer A0
22	INTB+88 (l) to INTB+91 (h)	Timer A1
23	INTB+92 (l) to INTB+95 (h)	Timer A2
24	INTB+96 (l) to INTB+99 (h)	Timer A3
25	INTB+100 (l) to INTB+103 (h)	Timer A4
26	INTB+104 (l) to INTB+107 (h)	Timer B0
27	INTB+108 (l) to INTB+111 (h)	Timer B1
28	INTB+112 (l) to INTB+115 (h)	Timer B2
29	INTB+116 (l) to INTB+119 (h)	INT0
30	INTB+120 (l) to INTB+123 (h)	INT1
31	INTB+124 (l) to INTB+127 (h)	INT2
32 to 63	INTB+128 on	Software interrupts (not maskable)

The above interrupts operate via the variable vector table. There are additional interrupts (Undefined, overflow, Address match, single-step, watchdog timer, DBC, and NMI) which operate via the fixed interrupt table. However, these are used by the monitor and should not be used by user programs.

5.3 On Board Peripherals

The MSA0654 Development Board contains four switches, two seven segment LED displays and a variable resister.

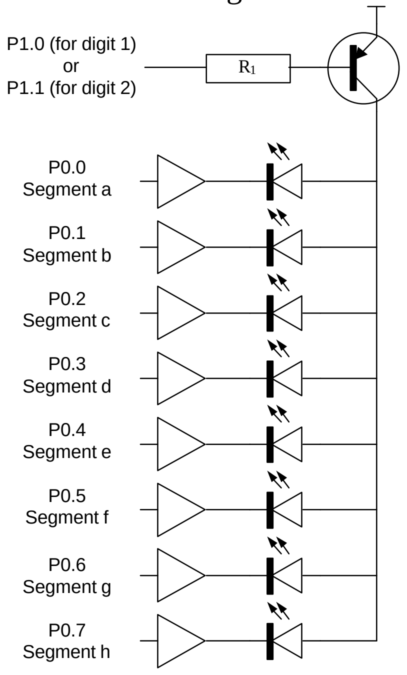


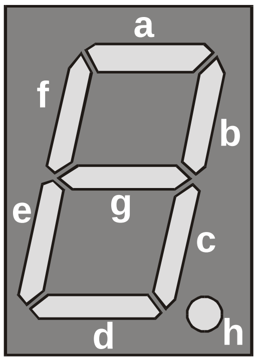
5.3.1 Analogue Input

The potentiometer, marked AD near the centre of the board, is connected to the AN0 pin on the M16C. The potentiometer is connected to ground and 5V, allowing any voltage between these two points to be applied to the AN0 pin.

5.3.2 Seven Segment Displays

The board has two seven segment displays. To reduce the number of I/O pins which are required these displays are multiplexed, so that the segments to display are put out on port P0, and then the digit is selected using the lowest two bits of port P1. The segment drives are active low, so writing a 0 to them switches on the relevant segment. The allocations are given below.





5.3.3 Switches

There are four switches on the board, each of which when activated connect the relevant input terminal to ground. When a switch is not pressed, a pull up resistor brings the line high.

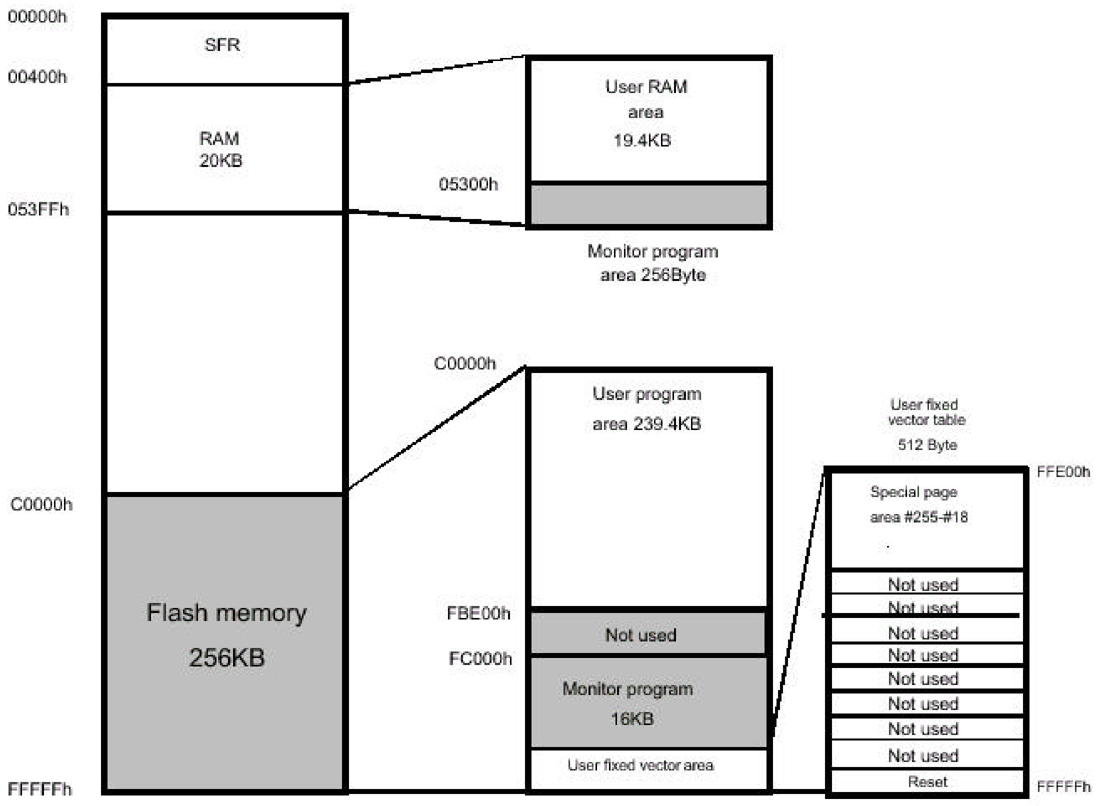
Switch	Input Line	Alternative Function
SW1	P8.2	INT0
SW2	P8.3	INT1
SW3	P9.7	ADTRG
SW4	RESET	

5.3.4 Restrictions on On-chip Peripherals

UART1 is used by the monitor to communicate with the host PC. This means that its transmit and receive interrupts are used for communication between the monitor program and the host computer and the UART cannot be used in user programs.

5.4 Memory Map

The monitor uses 256 bytes of RAM, leaving 19.4Kbyte of internal RAM for use between 00400h and 053FFh. 16Kbytes of FlashROM are also used by the monitor, leaving 239.4Kbyte of FlashROM between C0000h and FBE00h (see below). Note, however, that the IAR compiler available in the lab is restricted to 16Kbyte of code, although other versions of the compiler are available which allow full use of the memory. It is planned to upgrade to a version which will remove this restriction in the near future, but the 16K limit will not be a problem for the projects developed in the course as long as you do not try to use the full version of the library functions.



The monitor uses internal memory mode and therefore writes a '1' to bit 3 of the Processor Mode Register 1. Even without the monitor, memory expansion mode cannot be used unless resistor R12 is removed from the board.

5.5 Register Usage

The monitor uses a number of registers. In some cases these registers may not be altered by program or the monitor may fail to operate. In other cases, it is necessary to maintain the values of certain bits to ensure correct operation.

Register Name	Available for Use	Value after a reset
Processor mode register 0	No. Do not modify this register	Initialised to 00h. Processor mode: Single-chip mode
Processor mode register 1	When changing this register in the user program, always be sure to set bit 3 to 1.	Initialised to 08h.
System clock control register 0		Initialised to 08h.
System clock control register 1		Initialised to 20h. Selected main clock divide ratio: Not divide
ISP (interrupt stack pointer)	Set a value below 0530016h. Values 0530016h through 053FF16h are used by the monitor program.	Initialised to 044FFh.
UART1 Transmit/Receive Mode Register	No. Do not modify this register	Initialised to 05h.
UART1 Transfer Speed Register	No. Do not modify this register	Initialised to 1Ah.
UART1 Transmit/Receive Control Register 0	No. Do not modify this register	Initialised to 10h.

Register Name	Available for Use	Value after a reset
UART1 Transmit/Receive Control Register 1	No. Do not modify this register	Initialised to 05h.
UART1 Interrupt Control Register 0	No. Do not modify this register	Initialised to 07h
UART transmit/receive control register 2	Don't change bits 0, 2, 4, 5, and 6	Initialised to 03h
UART1 bit rate generator	No. Do not modify this register	
UART1 transmit buffer register	Don't write any data to this register.	
UART1 receive buffer register	Don't read this register.	
Protect register	If the monitor program starts immediately after Protect Register bit 2 (Port P9 Direction Register and SI/O3,4 Control Register write enable bit) is set to 1 (enabled), a write to some address by the monitor program occurs, so that the P9 Direction Register write enable bit is reset to 0 (disabled). Consequently, the P9 Direction Register cannot be written to in the following cases: 1) When a break to at the instruction that sets the write enable bit to 1 occurs 2) When Go, Step, Over, or Return to the instruction that sets the write enable bit to 1 is executed 3) When the P9 Direction Register is operated on from the dump window, etc.	
Flag register	Write to the D flag and I flag is ignored. (Always D flag is 0, I flag is 1)	

5.6 Interrupts

The monitor program uses interrupts for its operation. In order not to interfere with the monitor, user programs should not disable interrupts, and should not set the Interrupt Priority Level (IPL) to 7 (but see below). In particular, most of the interrupts to which the fixed vector table relates (Undefined, overflow, BRK instruction, Address match, single-step, watchdog timer, DBC, and NMI) are inhibited by the monitor, and will look like a REIT function to a user program, even if a different routine is specified in the program. Therefore, these functions are not available to the user program, and the UND and INTO instructions should not be used in the program (which means that und_instruction() and interrupt_on_overflow() should not be used in the C code, either, since these intrinsic functions generate those instructions). The exception is the Reset vector at FFFFCh to FFFFFh, which will work normally.

All interrupts located in the variable vector table are available to the user with the exception of the UART1 transmit/receive interrupts, which are used by the monitor program. When using INTB to set up the variable vector table, set **0FCB6BH** at the addresses (software interrupt numbers 19, 20) that correspond to the UART1 transmit/receive interrupts to point to the correct place within the monitor code. The simplest way to do this is to include the file msa0654.s34 in the project.

The monitor system turns off interrupts 1 to 6, so in order to interrupt the monitor routines, it is necessary to use priority level 7 interrupts for your code. Note that this will make the ISR impossible to debug by single stepping, so debug it using a lower interrupt priority, and then change to level 7 when you have it working.
The monitor's STEP function requires interrupts, and so cannot be used when interrupts are disabled. It is necessary to disable interrupts when changing when changing the Interrupt Control Register, so this part of the code can not be single stepped. Interrupts will also be disabled if your program contains an interrupt service routine. If this is the case, ensure that interrupts are not disabled for more than more than 260μs, you should set the I flag to 1 to re-enable interrupts at the beginning of the interrupt routine.

5.7 Stop and Wait Modes

Stop and Wait modes cannot be used on the development system.

5.8 Breakpoints and Single Stepping

Breakpoints can be set on most instructions, especially in C code, but care is required in some cases. It is not possible to set a breakpoint on the instruction following an LDC instruction. Also, if you set a break on an INT instruction, GO will not work correctly. This is due to the fact that breakpoints are implemented by replacing the instruction with a software interrupt returning control to the monitor, and replacing an INT with another INT causes problems restarting.

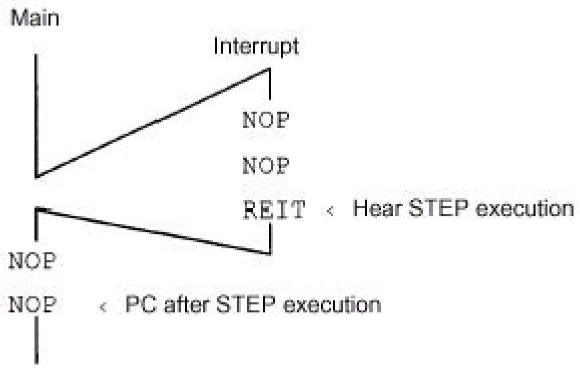
For similar reasons, it is not possible to single step into an interrupt routine. For example, when single stepping from instruction A in the following example, it would normally be expected that the debugger would stop on instruction B. In fact, it runs through the interrupt routine and stops at instruction C.

```
        NOP
        NOP
A       INT #3          STEP Skipped over when STEP is executed.
C       NOP
        JMP MAIN
        INT_3:
B       NOP             Address at which program execution ought to stop
        NOP
        NOP
        REIT
```

In order to step into an interrupt routine, you need to set a Break in the code at the start of the routine, and Go to it (rather than Stepping). The following will stop the code at B, allowing you to continue by single stepping.

```
        NOP
        NOP             GO, rather than STEP, from this statement
        INT #3
        NOP
        JMP MAIN
        INT_3:
B       NOP             Place Break here
        NOP
        NOP
        REIT
```

When single stepping past an REIT, JMPS or JSRS instruction, the following instruction will also be executed (i.e., the debugger will execute two instructions)



If interrupts are disabled within the code, then single stepping will execute right through the code without stopping until interrupts are enabled again. For example, when single stepping from A, execution will stop on instruction C, not instruction B.

```
A       FCLR I                  ; Disable interrupt
B       AND #00H , 0055H        ; Change Timer Interrupt 1
        NOP                     ; Clear instructions in pipeline
        NOP
C       FSET I                  ; Enable Interrupt
```

5.9 Communication Problems during Debug

For communications to operate with the development system, the monitor must be running. Sometimes this is not the case and a communication error occurs. To recover, press the reset button on the board and the reset button on the C-SPY menu. However, the program you downloaded may be corrupted and so it might be necessary to download the program again. The simplest way to do this is to close C-SPY and choose Debugger from the Project menu in the **IAR Embedded Workbench**.

The fact that the monitor is not operating may be due to the fact that the user program is running away and not returning control to the monitor. Since the monitor uses interrupts to allow this to happen, interrupts have to be enabled and you cannot disable them for your program (other than for short periods). Be careful if you have an interrupt routine which lasts more than 260µs, you should set the I flag to 1 to re-enable interrupts at the beginning of the interrupt routine. If an error occurs during downloading, press the reset button in case the program has started to run.