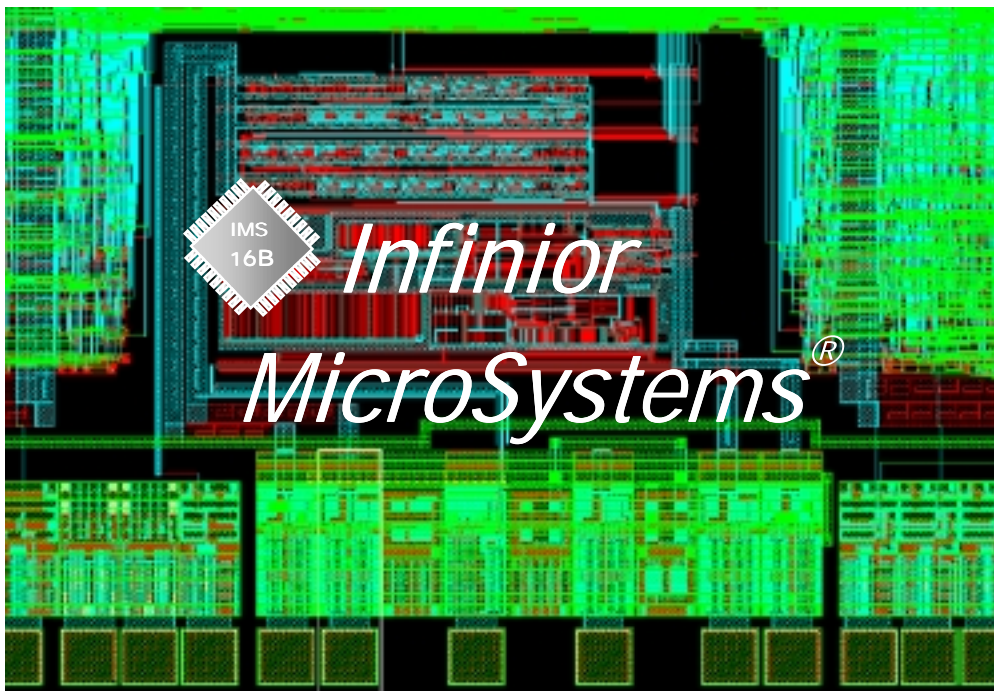


*Infinior MicroSystems'
Tahoe® 16bit Embedded processor*

*Software Development
User's Note*



January 2002

Infinior Microsystems

Infinior Microsystems reserves the right to modify any products described herein at any time without notice in order to improve function or design. Infinior Microsystems does not assume any liability arising from the application or the use of any product described herein; neither does it convey any license under patent rights nor imply the rights of others.

Copyright 2002 by Infinior Microsystems:

No part of this document may be reproduced in any form or means without the prior written permission of Infinior Microsystems.

IMS16B is a registered trademark of Infinior Microsystems.

The IMS16B is available from the following manufacturer:

Infinior MicroSystems
5F Jeongam Bldg, 769-12, Yeoksam2-Dong,
Gangnam-Gu, Seoul, Korea 135-928
Tel: + 82-2-6202-5500
FAX: + 82-2-6202-5555
www.infinior.com
sales@infinior.com

Contents

1. Introduction	4
1.1 Overview	4
2. Paradigm Environment	5
2.1 Borland's C/C++ Compiler and Paradigm Debug Software	5
2.1.1 Installing Paradigm Software	5
2.1.1.1 Installing Paradigm LOCATE	6
2.1.1.2 Installing Paradigm PDREM	6
2.1.1.3 Installing Paradigm Debug/RT-186	7
2.1.2 Building PDREM	7
2.1.3 Building An IMS16B Application Program	9
2.1.3.1 Building a demo program	9
2.2 Paradigm Windows Version	12
2.2.1 Building an IMS16B application	12
3. Peripherals Setting Examples	13
3.1 UART Initialization	13
3.1.1 Serial Port Control Register	13
3.1.2 Serial Port Baud Rate Divisor Register	13
3.1.3 UART Serial Interrupt Control Register	13
3.1.4 Interrupt Mask Register	14
3.1.5 Serial Port Status Register	14
3.1.6 Serial Port Receive Data Register	15
3.1.7 Serial Port Transmit Data Register	15
3.2 Timer Initialization	16
3.2.1 Timer Interrupt Control Registers	16
3.2.2 Timer 0 and Timer 1 Mode and Control Registers	16
3.2.3 Timer Count Registers	16
3.3 DMA Initialization	18

1**Introduction**

1.1 Overview

There are several commercial and free compilers and locators for IMS16B applications. Users can use any 186 processor supported compilers such as Borland C/C++, Turbo C/C++, Microsoft's C/C++, and [Linux 8086 development environment](#).

This application note provides users with step-by-step instruction for building and debugging an IMS16B application program with a paradigm locator.

This application note is divided to two parts:

1. Paradigm Environment (DOS Version & Windows Version)
2. Peripherals Setting Examples

Users can find detail information and sample codes from Infinior Microsystems(www.infinior.com)

2**Paradigm Environment**

This section provides users with information for building and debugging an IMS16B application program using Borland C/C++ and Microsoft's C/C++ compiler with a Paradigm Debug Software.

A Paradigm C++ 5.0 or above supports a C/C++ compiler, assembler, linker and locator, debugging and monitoring tools, and libraries.

Users also can use other commercial compiler, assembler, and linker with a paradigm locator. This section only describes the process of using Borland C/C++ compiler and linker with paradigm locator.

This section is divided to four parts:

1. Borland's C/C++ Compiler and Paradigm Debug Software
2. Paradigm Windows Version

2.1. Borland's C/C++ Compiler and Paradigm Debug Software

The user needs to install the following tools on his/her PC in order to build and debug an IMS16B application using Borland C/C++ with the Paradigm Debug Software:

- Borland C/C++ Compiler 5.0
- Paradigm LOCATE 5.11
- Paradigm Debug/RT-186 5.0
- Paradigm PDREMOTE/ROM 5.0a
- IMS16B Evaluation Board

The process of debugging an IMS16B application involve two basic steps: (1) building Paradigm Remote ROM Monitor (PDREM) and downloading it to the board; (2) building and debugging an IMS16B application.

2.1.1 Installing Paradigm Software

Before installing the Paradigm Software, Borland's C/C++ compiler and assembler should be installed. The Paradigm Software (Paradigm LOCATE, Paradigm PDREM, and Paradigm Debug/RT-186) is distributed by Paradigm Systems. To obtain a copy of any of these, please contact to Paradigm Systems at www.devtools.com.

This section provides only specific information for debugging an IMS16B application on the IMS16B Evaluation Board. When installing paradigm programs, users should follow the installation instruction of the appropriate manuals distributed by Paradigm

System.

2.1.1.1 Installing Paradigm LOCATE

Paradigm LOCATE serves as the locator. Paradigm LOCATE takes input from a relocatable .EXE file produced by the compiler/linker, assigns memory addresses defined in the .CFG file to produce an absolute executable file(.AXE, .HXE .BIN) suitable for the embedded design.

Figure1. shows how an absolute executable file is generated. User application files(.c, .cpp, .asm) and startup code are compiled with Borland C++ compiler and object files are produced. These generated object files and libraries are linked with Borland C++ linker. Paradigm LOCATE assigns memory addresses defined in the .CFG file and then produce an absolute executable file(.AXE, .HXE .BIN).

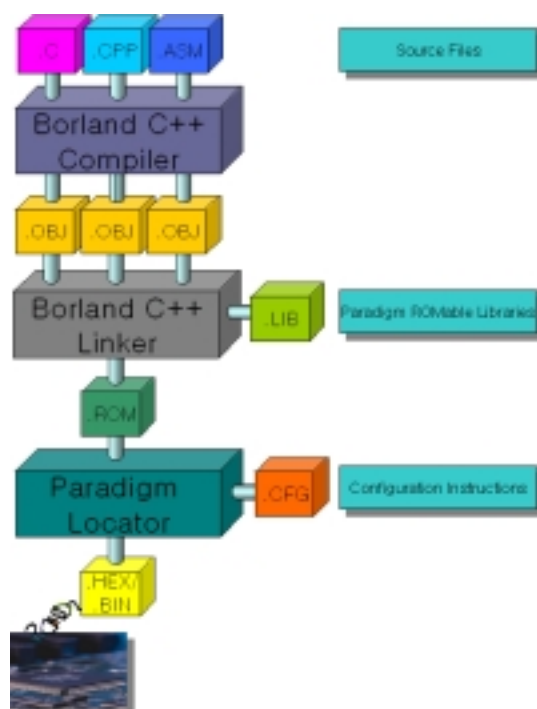


Figure 1 A process of making a binary file

2.1.1.2 Installing Paradigm PDREM

The installation program needs to know something about the software and hardware environment to retrieve correct source code for building PDREM. When installing PDREM, be sure to set the following parameters as indicated:

Target Processor	AMD Am186EM
Target Math coprocessor	None
Target Math coprocessor	Am186EM/internal
System Compiler	Borland C/C+ + 5.0

2.1.1.3 Installing Paradigm Debug/RT-186

The Paradigm Debug/RT-186 installation program will request information such as the serial communication port and baud rate. The specified baud rate defines the communication channel between Paradigm Debug/RT-186(pdrt186) and PDREM. Users should change baud rate and cpu clock.

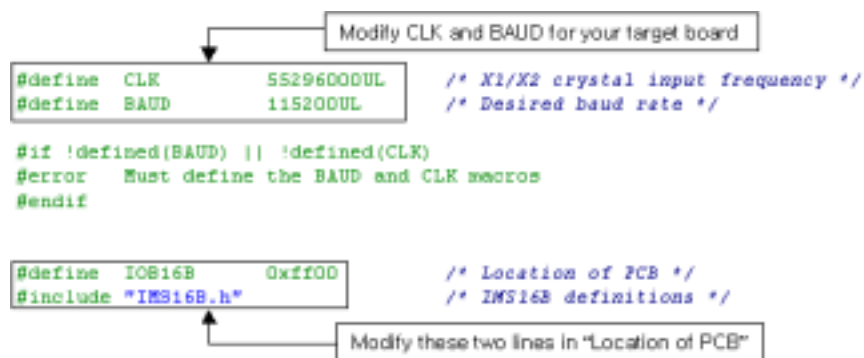


Figure 2 DCOMMS.C File for IMS16B Board

2.1.2 Building PDREM

A PDREM is a remote debugging program running at a ROM with a serial communication. It monitors all registers and memory, and supports step-by-step debugging.

In most cases, the files PDREM.CFG, TARGET.H and DCOMMS.C are all that will need to be modified.

PDREM.CFG contains the LOCATE configuration information to direct where PDREMOTE/ROM will be located as well as any chip select initialization required. You should check this file carefully to ensure that the initialization information is suitable for your hardware. The registers and instruction set is compatible with AMD186EM Processor. Therefore, user should write cputype as Am186EM at a PDREM.CFG file.

In the middle of a pdrem.cfg file, user should set chip select at initcode part. When a system is booted up, automatically jumps to 0xFFFF0 and then jumps to initcode(??LOCATE), and goes to code segment.

```

//
// Paradigm LOCATE configuration file for building the PDREM kernel.
// Copyright (C) 1994, 1995 Paradigm Systems. All rights reserved.
//
// This is an example configuration for the Infinior IMS16B
//
// Please ensure that the example values shown are appropriate for
// your target hardware.
//
absfile axe86
hexfile binary offset=0xe0000 size=128 fill=0xff

listfile segments publics // Create a segment map

(1)...
map 0x00000 to 0x003ff as reserved // Interrupt vector table
map 0x00400 to 0x00fff as rdwr // PDREMOTE/ROM data area
map 0x01000 to 0xdffff as reserved // Reserved for applications
map 0xe0000 to 0xfffff as rdoonly // PDREMOTE/ROM kernel ZPROG (

cputype Am186EH // Select the processor type

// The following initcode values are example values.
// The user should determine appropriate values for their
// specific hardware and substitute those values instead.

(2)...
initcode reset // Chip select initializations
// Reset vector
// 128K ROM, 3 wait states
// 256K RAM, 1 wait states
    uscs = 0xf73f \
    lscs = 0x183d \
    rpsc = 0x90bb \
    zscs = 0xc1f6 \
    pscs = 0x007b \

dup DATA ROMDATA // Make a copy of the initialized data

class CODE = 0xe000 // Modify to set the EPROM address
class DATA = 0x0040 // And the RAM address
class ??LOCATE = 0xffff0 // Chip select initialization code

order DATA // Fix the class ordering of DGROUP
CONST \
BSS \
BSSEND \
STACK \

order CODE // Code, initialized data in EPROM
ROMDATA ENDROMDATA \

output CODE // Write to the output file
ROMDATA ENDROMDATA \
??LOCATE \

```

Figure 3 pdrem.cfg File for IMS16B Evaluation Board

TARGET.H contains some conditional code, which if enabled, will control the selection of polled or interrupt driven serial interface (the default is polled) and whether the application kernel interface is linked in.

DCOMMS.C contains the serial driver which will likely require some modification to select I/O port addresses or a different interrupt controller configuration.

When a user modified all the files referred here, simply run “make” at directory C:/pdrem as figure 4.

```

C:\WPDREM>make
MAKE Version 4.8 Copyright (c) 1987, 1996 Borland International
      tasm /nx pdrema.asm ;

Turbo Assembler Version 2.81 Copyright (c) 1988, 1998 Borland International

Assembling file:  pdrema.asm
PBREMOTE/ROM Startup Code
Error messages:   None
      bcc -c -ns -lc:\wbc5\include -Z -Od -f- -w -DBCPP50 -i- dcomms.c
Borland C++ 5.0 Copyright (c) 1987, 1996 Borland International
dcomms.c:asm /nx pdrema.asm ;
      bcc -c -ns -lc:\wbc5\include -Z -Od -f- -w -DBCPP50 -i- target.c
Borland C++ 5.0 Copyright (c) 1987, 1996 Borland International
target.c:
      tlink pdrema.obj dcomms.obj target.obj, pdrem.rom, pdrem.map, pdrem.lib

Turbo Link Version 7.1.38.1. Copyright (c) 1987, 1996 Borland International
      locate pdrem
Paradigm LOCATE - Version 5.11
Copyright (C) 1987-1996 Paradigm Systems. All rights reserved.

C:\WPDREM>

```

Figure 4 Run "Make" for PDREM at DOS Prompt

2.1.3 Building An IMS16B Application Program

The way to making user applications is almost same to making the Paradigm remote program(PDREM). The source files are compiled and linked by Borland tools to produce a ROM relocatable object file (.ROM). Paradigm LOCATE takes information defined in the .CFG file, and assigns code and data addresses to the .ROM file to produce an Absolute Executable file(.AXE). The .AXE file is then loaded and debugged using Paradigm Debug/RT pdrt186.

Besides building the ROMable runtime libraries, the Paradigm LOCATE installation program also installs other utilities needed for building an embedded program, such as startup code, library helpers, as well as examples showing how to build various application programs. Each example has a readme.txt file containing useful information. The example directories are placed in the directory C:\Wlocate\Wbcpp50\examples. Unfortunately, these examples were configured to run on an in-circuit emulator. This section shows how to make the demo program in the examples directory work with the IMS16B board. A User can download the demo source explained here from Infinior web site. (www.infinior.com/download/ims16b).

This section is divided to two parts:

- a) Building a demo program
- b) Invoking Paradigm Debug/RT

2.1.3.1 Building a demo program

The configuration file is the place where target-specific information such as memory

configuration is defined. Therefore a user should modify this configuration file depending on his/her target system. An IMS16B EVM Board has 128 MBytes ROM and 256 MBytes SRAM. A ROM area is from 0xE0000~0xFFFFF and a RAM area is from 0x00000~0x3FFFF. An interrupt vector table is allocated to memory 0x00000 to 0x003ff. A PDREM code is located in memory from 0xE0000 and a PDREM kernel data is located in between 0x00400 and 0x00FFF. Therefore, the memory from 0x00000 to 0x00fff should be reserved for PDREMOTE and interrupt vector table.

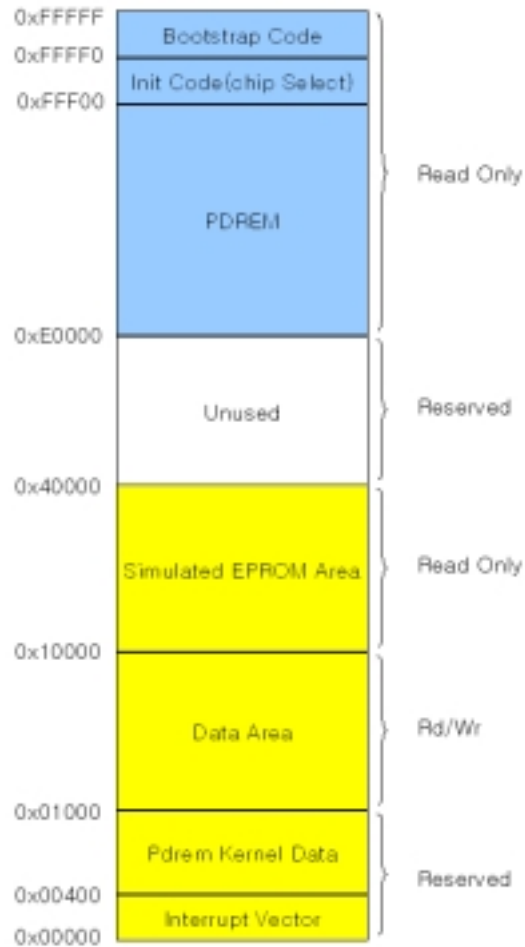


Figure 5 Memory Map for PDREM & Application

From 0x01000 to 0x0FFFFF is for a demo application data and from 0x10000 to 0x3FFFFF is for a demo application code. Take a look at figure 5. and modify it for your target-specific applications.

```

//
// Paradigm LOCATE configuration file for Borland/Turbo C++ application.
// This configuration file creates an .AXE file suitable for debugging
// using Paradigm DEBUG/RT with PDREMOTE.
//

#include "fardata.cfg"           // Access the far data definitions

absfile axe86                   // Output for Paradigm DEBUG
listfile segments              // Absolute segment map

map 0x00000 to 0x00fff as reserved // PDREMOTE and interrupt vector table
map 0x01000 to 0x0ffff as rdwr    // System RAM area (60KB RAM)
map 0x10000 to 0x3ffff as ronly   // Simulated EPROM area (192KB RAM)
map 0x40000 to 0xfffff as reserved // No access allowed

#define DATA_START 0x0100      // Start of application data
#define CODE_START 0x4000      // Start of application code

cputype As186EM

dup DATA ROMDATA              // Make a copy of initialized data

class CODE = CODE_START        // Assume loading at address 40000H
class DATA = DATA_START      // Data at address 01000H

order DATA                     \ // RAM class organization
    BSS BSSEND                 \
    STACK                      \
    _FARDATACLASSES           \
// FARHEAP                     \ // Order FARHEAP here if necessary

order CODE                     \ // EPROM class organization
    INITDATA EXITDATA         \
    ROMDATA ENDROMDATA        \
    _ROMFARDATACLASSES       \

output CODE                    \ // Classes in the output file(s)
    INITDATA EXITDATA         \
    ROMDATA ENDROMDATA        \
    _ROMFARDATACLASSES       \

```

Figure 6 demo.rt

The makefiles in this and in most other example directories are designed to build different object files, depending on the value of the DEBUG variable in the makefile. If the value of DEBUG=0, the configuration file demo.rm will be chosen to build the standalone version; if the value of DEBUG=1, the configuration file demo.rm will be chosen to build the emulator version; if the value of DEBUG=2, the configuration file demo.rt will be chosen to build the PDREM version. To build a PDREM version of demo.c, simply change the value of the DEBUG variable to 2.

```

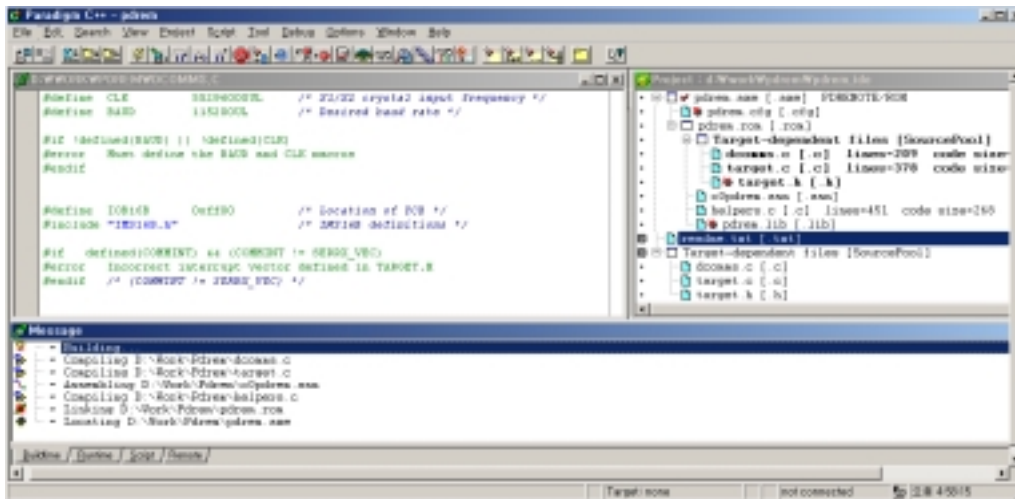
MODEL      = 1          # s, m, c, l, h
CPU        = 1          # 0 - 8086, 1 - 80186 or better
FLOAT      = 0          # 0 - none, 2 - emulator, 3 - coprocessor
FARDATA    = 0          # 0 - none, 1 - normal, 2 - compressed
EXCEPTIONS = 0          # 0 - disabled, 1 - enabled

DEBUG      = 2          # 0 - none, 1 - debug EPROM, 2 - debug FOREMOTE
OPTIMIZE   = 0          # 0 - none, 1 - size, 2 - speed
WARNINGS   = 1          # 0 - none, 1 - all

CODESTRING = 0          # Put string literals in code segment
DUPSTRING  = 1          # Duplicate string merged
CHECKSTACK = 1          # Check for stack overflow
STACK      = 1024       # Application stack size (in bytes)
    
```

2.2. Paradigm Windows Version

If you have Paradigm C++ 5.0 or above, you can make pdrem with Paradigm IDE.



2.2.1 Building an IMS16B application.

The way to make a program is almost same as Paradigm DOS version except that it supports more compatible environment.

When users want to know more detailed information about Paradigm C/C++ 5.0 or above, please refer to Paradigm User's Guide and Quick Start Guide.

3

Peripherals Setting Examples

3.1 UART Initialization

3.1.1 Serial Port Control Register (SPCT, Offset 80h)

The Serial Port register controls both transmit and receive sections of the serial port.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	TXI E	RXI E	LO OP	BR K	BRK VAL	PMO DE1	PMO DE0	WLG N	STP	TMO DE	RSI E	RM ODE

Reset value : 0000_0000_0000_0000b

Serial Port Control Register(SPCT,offset 80h)

- Transmit Enable (b2→1)
- Receive Data Ready Interrupt Enable (b10→1)
- Receive Enable(b0→1)
- .8 Data Bit (b4→1)
- .1 Stop Bit (b3→0)
- None Parity(b6,b5→00)

```
0000_0100_0001_0101 = 0x0415
    outport(SPCT, 0x0415);
```

3.1.2 Serial Port Baud Rate Divisor Register (SPBAUD, Offset 88h)

This register specifies a clock divisor for the generation of the serial clock that controls the serial port.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BAUDDIV[15:0]															

Reset value : 0000_0000_0010_0110b

$$BAUDDIV_{(decimal)} = (\text{Internal Processor Clock} / (32 * \text{Baud Rate})) - 1$$

Serial Port Baud Rate Divisor Register(SPBAUD,offset 88h)

Baud Rate Divisor (Processor Clock : 11.0592MHz x 2 | 115kBPS)

For example, if user's Internal Processor Clock is 11.0592MHz x 2 and UART Baud Rate is 9600bps, then he/she can get BAUDIV=0x47.

$$BAUDDIV = (11.0592\text{MHz} * 2 / (32 * 9600\text{bps})) - 1 = 0x47$$

```
0000_0000_0100_0111 = 0x47
    outport(SPBAUD, 0x47);
```

3.1.3 UART Serial Interrupt Control Register (SPICON, Offset 44h)

The Serial Port Interrupt Control register controls the operation of the asynchronous serial port interrupt source. This interrupt is assigned to interrupt type 14h.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	MSK	PR2	PR1	PR0

Reset value : 0000_0000_0000_1111b

UART Serial Interrupt Control Register (SPICON, Offset 44h)

MSK → 0, Priority 6 : PR2→1,PR1→1,PR0→0

0000_0000_0000_0110 = 0x06

output(SPICON, SPICON_PR1| SPICON_PR2);

3.1.4 Interrupt Mask Register (IMASK, Offset 28h)

This register is for masking interrupts. Programming a bit in the IMASK register has the effect of programming the MSK bit in the associated control register. Therefore, when a user wants to mask UART interrupt, he/she can change a value either in an SPICON register or an IMASK register bit 10.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	I2C	1	1	UART	WDT	I4	I3	I2	I1	I0	D1	D0	0	TMR

Reset value : 0011_1111_1111_1101b

Interrupt Mask Register (IMASK, Offset 28h)

b10:1→0

output(IMASK, inport(IMASK) & ~IMASK_UART);

```
void Uart_Init(void)
{
    disable();
    output(SPCT, 0x0415); // Serial Port Control Register
    output(SPBAUD, 0x0047); // Processor Clock : 11.0592MHz x 2 | 9600BPS
    output(SPICON, SPICON_PR1| SPICON_PR2);
    // or output(IMASK, inport(IMASK) & ~IMASK_UART);
}
```

Example 1 Uart Init Code

```
void interrupt far URT_RX_Isr (void)
{
    disable();
    Putch(Getch());
    output(EOI, EOI_SERIAL);
    enable();
}
```

Example 2 UART Rx Code

3.1.5 Serial Port Status Register (SPSTS, Offset 82h)

The Serial Port register indicates the status of transmits and receives sections of the

serial port.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	TEMT	THRE	RDR	BRKI	FER	PER	OER

Reset value : 0000_0000_0110_0000b

6	Transmit Empty	The TEMT bit is 1 when the transmitter has no data to transmit and the transmit shift register is empty. This indicates to software that it is safe to disable the transmit section. This bit is read-only.
4	Receive Data Ready	When the RDR bit is 1, receive buffer register contains data that can be read. When the RDR bit is 0, receive buffer register does not contain valid data. This bit is read-only. If receive interrupts are enabled by the RMODE and RXIE fields, a serial port interrupt request is generated when the THRE bit is 1. Reading receive buffer register resets the RDR bit.

3.1.6 Serial Port Receive Data Register (SPRD, Offset 86h)

This register contains data received over the serial port. The receiver is double-buffered, and the receive section can be receiving a subsequent frame of data in the receive shift register (which is not accessible to software) while the receive data register is being read by software.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0								RDATA[7:0]

Reset value : Not Defined

```

unsigned char Getch (void)
{
    while (!(inport(SPSTS) & 0x0010));

    return ((unsigned char )(inport(SPRD) & 0x00FF));
}
    
```

3.1.7 Serial Port Transmit Data Register (SPTD, Offset 84h)

Software writes this register with data to be transmitted on the serial port. The Transmitter is double-buffered, and the transmit section copies data from the transmit data register to the transmit shift register(which is not accessible to software) before transmitting the data.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0								TDATA[7:0]

Reset value : 0000_0000_0000_0000b

```

void Putch (unsigned char byChar)
{
    disable();
    // Wait until the transmitter has no data to transmit
    while (!(InPort(SPSTS) & 0x0040));
    outport(SPTD, (unsigned int)byChar);
    if (byChar == 0x0A)
        Putch (0x0D);
    enable();
}
    
```

Example 3 Transmit one byte data to UART (simple code)

3.2 Timer Initialization

3.2.1 Timer Interrupt Control Registers(TCUCON, Offset 32h)

The three timer interrupts are assigned to interrupt type 08h, 12h, and 13h. All three timer interrupts are configured through TCUCON, offset 32h.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	MSK	PR2	PR1	PR0

Reset value : 0000_0000_0000_1111b

Timer Interrupt Control Registers(TCUCON, Offset 32h)

MSK → 0, Priority 2 : PR2→0,PR1→1,PR0→0

0000_0000_0000_0010 = 0x02

3.2.2 Timer 0 and Timer 1 Mode and Control Registers (T0CON, Offset 56h, T1CON, Offset 5Eh)

These registers control the functionality of timer0 and timer1. A detail information about this register can be found at chapter 10 in an IMS16B USER's Guide.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	INH	INT	RIU	0	0	0	0	0	0	MC	RTG	P	EXT	ALT	CONT

3.2.3 Timer Count Registers (T0CNT Offset 50h, T1CNT Offset 58h, T2CNT Offset 60h)

These registers can be incremented by one every four internal clocks. Timer0 and Timer1 can also be configured to increment based on TMRIN0 and TMRIN1 external signals, or timer2 can rescale them. Each timer is serviced on every fourth clock cycle. Therefore, a timer can operate at a maximum speed of on-quarter of the internal clock frequency.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TC[15:0]															

Reset value : 0000_0000_0000_0000b

```

void Timer0_Init(void)
{
    outport(TCUCON,0x02); // Priority 2, Mask disable => Timer Interrupt Possible
    // or
    // outport(IMASK, inport(IMASK) & ~IMASK_TMR);
    outport(TDCON, TDCON_EN | TDCON_INH | TDCON_INT | TDCON_CONT);
    // Initial Value of TOCNT is 0.
    // This value is incremented by 1 at every 4 clocks
    outport(TDCNT, 0x0000);
    // The Value at TOCNT register is compared with a value at TOCMPA register.
    // If it reaches the value stored at TOCMPA, a timer0 interrupt is occurred.
    outport(TOCMPA, 0xD800);
}

```

Example 4 Timer0 Init Code

At every seconds, it displays 0~9 through an UART.

```

void interrupt far Timer0_isr (void)
{
    disable();
    // 100 can be defined according to a value
    // at TOCMPA and Internal Clock Speed.
    if(count % 100 == 0)
    {
        Putch(sec_count+ '0');
        sec_count++;
        if(sec_count > 9) sec_count = 0;
    }
    count++;
    outport(EOI, EOI_TIMER0);
    enable();
}

```

Example 5 Timer0 ISR Example Code

```

void main(void)
{
    Uart_Init(); // Uart Initialization
    Timer0_Init(); // Timer0 Initialization
    setvect(SERIAL_TYPE, URT_RX_Isr); // SERIAL_TYPE: 0x14
    setvect(TIMER_0_TYPE, Timer0_isr); // TIMER_0_TYPE: 0x08
    enable(); // Interrupt Enable
}

```

Example 6 Uart&Timer Test Main Program

3.3 DMA Initialization

Direct Memory Access(DMA) permits transfer of data between memory and peripherals without CPU involvement. The DMA unit in the IMS16B microcontroller provides two high-speed DMA channels. Data transfers can occur between memory and I/O spaces(e.g., memory to I/O(or within the same space(e.g., memory-to-memory or I/O-to-I/O). This section provides simple DMA Transfer example from memory to memory.

When users write a program for transferring data from memory to memory, users only need to fill out source and destination addresses and number of data to transfer.

The DMA transfer count register (DTC) specifies the number of DMA transfers to be performed. Up to 64 Kbytes or 64K words can be transferred with automatic termination

Change Start Bit (CHG) must be set to 1 during a write to allow modification of the ST bit. When change start bit is set to 0 during a write, ST is not altered when writing the control word.

The DMA channel is started when the start bit (ST) is set to 1. This bit can be modified only when the change start bit is set to a 1 during the same register writes.

```

01: void write_dma0(const unsigned long src, const unsigned long dst, unsigned int nBytes)
02: {
03:     unsigned long p_src = FP_MAK(FP_SEGMENT(src),FP_OFFSET(src));
04:     unsigned long p_dst = FP_MAK(FP_SEGMENT(dst),FP_OFFSET(dst));
05:     unsigned int src_h = p_src >> 16;
06:     unsigned int src_l = (unsigned int)p_src;
07:     unsigned int dst_h = p_dst >> 16;
08:     unsigned int dst_l = (unsigned int)p_dst;
09:
10:     // Write Source Address
11:     output(D0SRCH, src_h);
12:     output(D0SRCL, src_l);
13:
14:     // Write Destination Address
15:     output(D0DSTH, dst_h);
16:     output(D0DSTL, dst_l);
17:
18:     // Number of data
19:     output(D0TC, nBytes);
20:
21:     // Set Start DMA Channel
22:     output(D0CON, inport(D0CON)| CHG| ST);
23: } ? end write_dma0 ?

```

Figure 7 DMA Test Code (Mem-to-Mem)