

Using the MC68332 Periodic Interrupt Timer

By Mark Maiolani
Motorola Ltd
East Kilbride

INTRODUCTION

This application note demonstrates the use of the MC68332 periodic interrupt timer by implementing an interrupt driven real-time clock in software.

As well as detailing the use of the PIT, the general use and initialisation of interrupts on the MC68332 is covered, especially from the 'C' programming language. List files are also included to show the resultant assembly level program.

THE PERIODIC INTERRUPT TIMER

The Periodic Interrupt Timer, or PIT, provides a way of generating interrupts to the MC68332 core, i.e. the CPU32, at programmable regular intervals.

Essentially the PIT, shown in figure 1, consists of an 8-bit down-counter preceded by a + 4 prescaler, which

generates an interrupt and re-loads with a programmed value when zero is reached.

The 8-bit value to be re-loaded is stored in the Periodic Interrupt Timing Register (figure 2), as bits Pitr7-0. The PIT period can therefore be adjusted by modifying this value, or disabled by setting it to zero.

A prescaler bit, PTP, can extend the range of the PIT period, by switching an additional + 512 prescaler into the counter input when PTP = 1.

With a 32.768 kHz oscillator, the resultant period can be in the range 122µs to 15.94 seconds as shown in the example table of figure 3. The formula to calculate the PIT periods is:

$$\text{PIT period} = (\text{Pitr value} * 4) / (\text{EXTAL freq.} / \text{Prescaler})$$

where Prescaler = 512 if PTP = 1, or 1 for PTP = 0.

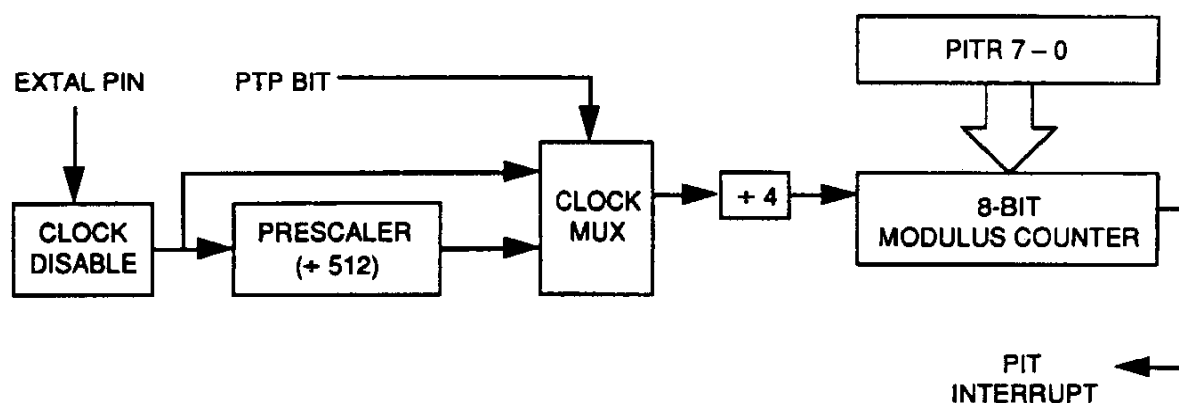


Figure 1. Periodic Interrupt Block Diagram

PITR

\$YFFA24

| | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|-----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | PTP | PITR 7 | PITR 6 | PITR 5 | PITR 4 | PITR 3 | PITR 2 | PITR 1 | PITR 0 |

Note: PTP takes the negated value of the MODCK pin on rising edge of RESET
 Y = F if MM bit of MCR is set, Y = 7 if MM bit is clear

Figure 2. Periodic Interrupt Timing Register

| PITR | PIT Period |
|--------|-----------------------------|
| \$0000 | Periodic Interrupt Disabled |
| \$0001 | 122 μ s |
| \$0002 | 244 μ s |
| \$0004 | 488 μ s |
| \$0008 | 977 μ s |
| \$000F | 1.83 ms |
| \$0020 | 3.90 ms |
| \$0040 | 7.88 ms |
| \$0080 | 15.6 ms |
| \$00A0 | 19.5 ms |
| \$00FF | 31.1 ms |
| \$0100 | Periodic Interrupt Disabled |
| \$0101 | 62.5 ms |
| \$0102 | 125 ms |
| \$0104 | 250 ms |
| \$0108 | 500 ms |
| \$0110 | 1 second |
| \$0120 | 2 seconds |
| \$0140 | 4 seconds |
| \$0180 | 8 seconds |
| \$01A0 | 10 seconds |
| \$01FF | 15.9 seconds |

Figure 3. Example PIT periods

CONFIGURING THE PIT INTERRUPT

The second PIT register is the Periodic Interrupt Control Register, or PICR (figure 4), which is used to configure the interrupt generated by the PIT. The Periodic Interrupt Request Level bits determine the priority of the interrupt from 1 to 7. If the PIRQL field is set to all zeros, the interrupt is disabled.

When the CPU32 detects an interrupt, it requests the number of the vector which contains the address of the exception handler routine. The vector number returned in response to a PIT interrupt is determined by the Periodic Interrupt Vector field in the PICR. This can be any vector number from 0 to 255, although normally it would be set to indicate one of the CPU32 user defined vectors, numbered from 64 to 255.

As the PIT is part of the System Integration Module in the MC68332, the main SIM Module Configuration Register (figure 5) also has to be initialised for the resultant interrupts to be handled correctly.

The Interrupt Arbitration Bits, IARB3-0, are used for arbitration when interrupts of the same level are generated simultaneously by different modules on the Inter Module Bus, such as the SIM and QSM. A zero value for a module's IARB field results in all interrupts that it generates being treated as spurious, whereas a value from 1 to \$F determines its priority on the IMB, from lowest to highest. It is recommended that each module on the IMB should be programmed with a different IARB number to allow the arbitration process to function as above.

| PICR | | | | | | | | | | \$YFFA22 | | | | | |
|------|----|----|----|----|------------|------------|------------|----------|----------|----------|----------|----------|----------|----------|----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | PIRQL 2 | PIRQL 1 | PIRQL 0 | PIV 7 | PIV 6 | PIV 5 | PIV 4 | PIV 3 | PIV 2 | PIV 1 | PIV 0 |

Figure 4. Periodic Interrupt Control Register

| MCR | | | | | | | | | | \$YFFA00 | | | | | |
|-------|-------|-------|----|-------|----|-----------|-----------|------|----|----------|---|-----------|-----------|-----------|-----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EXOFF | FRZSW | FRZBM | 0 | SLVEN | 0 | SHEN 1 | SHEN 0 | SUPV | MM | 0 | 0 | IARB 3 | IARB 2 | IARB 1 | IARB 0 |

Figure 5. Module Configuration Register

Freescale Semiconductor, Inc.

INITIALISATION OF THE CPU32 VECTOR TABLE

Before the PIT interrupt can be enabled, the address of the software routine to be executed in response to the interrupt, ie. the exception handler, has to be programmed into the correct vector table entry.

The starting address of the vector table is defined by the CPU Vector Base Register, in a similar manner to the 68010/20 MPUs.

As the table consists of 256 vectors (figure 6), where each vector is a byte address, the address of vector n can be calculated as:

$$\text{Address} = \text{VBR} + (n * 4)$$

If the PICR is set to assign the PIT interrupt to vector number 64, which is the first user-defined vector, then the four bytes starting at address $((\text{VBR}) + (64 * 4))$ should be programmed with the address of the PIT exception handler.

USING THE PERIODIC INTERRUPT FROM THE 'C' LANGUAGE

The PIT (or indeed any other MC68332 interrupt) can be configured efficiently with very few 'C' instructions. This is shown in the example program '332RTC', where the vector table is initialised with the address of the exception handler, clock().

| Vector Offset | Vector Assignment | Vector Number |
|---------------|---|---------------|
| 000 | Reset: Initial Supervisor Stack Pointer | 0 |
| 004 | Reset: Initial Program Counter | 1 |
| 008 | Bus Error | 2 |
| 00C | Address Error | 3 |
| 010 | Illegal Instruction | 4 |
| 014 | Zero Divide | 5 |
| 018 | CHK Instruction | 6 |
| 01C | TRAPcc, TRAPV Instructions | 7 |
| 020 | Privilege Violation | 8 |
| 024 | Trace | 9 |
| 028 | Line 1010 Emulator | 10 |
| 02C | Line 1111 Emulator | 11 |
| 030 | Hardware Breakpoint | 12 |
| 034 | (Reserved, coproc protocol violation) | 13 |
| 038 | Format Error | 14 |
| 03C | Uninitialised Interrupt | 15 |
| 040 - 05C | (Unassigned and Reserved) | 16 - 23 |
| 060 | Spurious Interrupt | 24 |
| 064 - 07C | Level 1-7 Interrupt Autovectors | 25 - 31 |
| 080 - 0BC | TRAP #0-15 Instruction | 32 - 47 |
| 0C0 - 0E8 | Reserved for Coprocessor) | 48 - 58 |
| 0EC - 0FC | (Unassigned and Reserved) | 59 - 63 |
| 100 - 3FC | User Interrupt Vectors | 64 - 255 |

Figure 6. CPU32 Vector Table

The program line :

```
*(long *)((vecno * 4) + vbr) = (long)clock;
```

with the resultant assembly code :

```
move.l #clock,256
```

takes the address of the routine clock, converts it to a long value, and stores it in the location pointed to by the long value ((vecno * 4) + vbr).

Note that this program assumes that startup code has initialised the CPU VBR register to a fixed value, as it defines 'vbr' to be '0x00'. An alternative way to determine the value of the VBR, which is shown commented out in '332RTC', is to import its value directly from the startup code.

One important point to remember when dealing with interrupts in high level languages is that the exception handler must always be terminated by the assembly instruction 'Return from Exception', RTE, rather than the 'Return from Subroutine' or RTS instruction. With some Compilers a directive can be used to force the use of RTE to terminate a routine instead of RTS.

The program '332RTC' uses the '__mod2__' directive available on the Intral 332 compiler for this purpose.

Other methods of vectoring interrupts can be used, either involving user written assembly level exception handlers which will 're-vector' the interrupt to the handler routine via a JSR instruction, or alternative methods 'built-in' to the compiler.

Although these will have the disadvantage of increasing the response time to the interrupt, they will allow the exception handler to be called by the program itself, which is not possible if the routine terminates with RTE.

332RTC - GENERAL INFORMATION

The program '332RTC' was developed on the MC68332 BCC, and runs under the '332Bug' monitor. Because of this, the SIM MCR register is not modified, but is left in the state programmed by the monitor. As the PIT interrupt request level is programmed to level 6, the CPU32 interrupt mask must be programmed to 5 or less for the interrupt to be recognised. This may be achieved directly from the monitor or by including this function in the assembly startup code for '332RTC'.

The PIT interrupt, which is programmed to occur at 1Hz frequency, vectors to the routine clock. This updates the global time variables (hours, minutes and seconds) before printing a display of the time via a PRINTF instruction. The 'PRINTF' instruction from the Intral 332 compiler is directed to the MC68332 SCI port, and allows the messages to be viewed on a PC connected to the BCC or EVS RS232 port.

C SOURCE CODE - 332RTC.C

```

*      /* 332RTC.C 17/8/90
*
*      C demo showing use of periodic interrupt timer to implement a real
*      time clock in software. Demonstrates the use of interrupt driven
*      software in Introl C
*
*      The interrupt handler, clock(), updates the time variables, and also
*      prints the time for demonstration of operation
*
*      Periodic interrupt is programmed to level 6 so startup code must
*      set the interrupt mask to 5 or less
*
*      Written by:
*          Mark Maiolani, Motorola East Kilbride
*
*/

#include "332defs.h" /* General definitions */

#define pitr  0xfffffa24 /* Address of Pitr assuming MM bit =1 */
#define picr  0xfffffa22 /* .. .. PICR .. .. */
#define vecno 0x40 /* Vector number used */
#define vbr   0x00 /* Assume VBR =0 */
/* import vbr Or import from startup file */

/* Global Variables */
byte hours=0, minutes=0, seconds=0;

/* function prototypes */
void clock();

main()
(
/* Set up interrupt vector (number vecno) to point to routine clock */
*(long *) ((vecno * 4) + vbr) = (long)clock;

/* Set Pitr for 1 second period */
*(word *) (pitr) = 0x0110;

/* Set PICR for level 6 interrupt vector number 0x40 */
*(word *) (picr) = 0x0640;

/* Loop forever */
while (1);
)
void __mod2__clock()
(
seconds++;
if (seconds>59)
(
seconds=0;
minutes++;
if (minutes>59)
(
minutes=0;
hours++;
if (hours>12) hours=1;
)
)
)
printf("\r%02d:%02d %02d", hours, minutes, seconds);
)

```

MERGED C SOURCE AND ASSEMBLY OUTPUT - 332RTC.C

```

*
* /* 332RTC.C 17/8/90
*
* C demo showing use of periodic interrupt timer to implement a real
* time clock in software. Demonstrates the use of interrupt driven
* software in Intrlc C
*
* The interrupt handler, clock(), updates the time variables, and also
* prints the time for demonstration of operation
*
* Periodic interrupt is programmed to level 6 so startup code must
* set the interrupt mask to 5 or less
*
* Written by:
*      Mark Maiolani, Motorola East Kilbride
*
*/
*
*      #include "332defs.h"          /* General definitions */
*
*      #define pitr 0xfffffa24      /* Address of Pitr assuming MM bit =1 */
*      #define picr 0xfffffa22      /* ,, ,, PICR ,, ,, ,, */
*      #define vecno 0x40           /* Vector number used */
*      #define vbr 0x00            /* Assume VBR =0 */
* /*      import vbr                Or import from startup file */
*
* /*      Global Variables */
*      byte hours=0,minutes=0,seconds=0;
*
6 00000000          ds.w 0
7 00000000      hours:
8 00000000 00          dc.b 0
9 00000001          ds.w 0
10 00000002      minutes:
11 00000002 00          dc.b 0
12 00000003          ds.w 0
13 00000004      seconds:
14 00000004 00          dc.b 0
16                      section.text
*
* /*      function prototypes */
*      void clock();
*
*      main()
*
18 00000000      main:          fbegin
19 00000000 4e56fff0          link fp,#-16
20
*
*      {
* /*      Set up interrupt vector (number vecno) to point to routine clock */
*      *(long *) ((vecno * 4) + vbr) = (long)clock;
*
22 00000004 >21fc000000000100      move.l #clock,256
*
* /*      Set Pitr for 1 second period */
*      *(word *) (pitr) = 0x0110;

```

```

24 0000000c 31fc0110fa24      move.w #272,-1500

*
*      /*      Set PICR for level 6 interrupt vector number 0x40 */
*      * (word *) (picr) = 0x0640;

26 00000012 31fc0640fa22      move.w #1600,-1502
27 00000018                ?0.4

*
*      /*      Loop forever */
*      while (1);

29 00000018 60fe                bra      ?0.4

*
*      )

31 0000001a                ?_1
32
33 0000001a 4e5e                unlk    fp
34 0000001c 4e75                rts
35 0000001e                fend

*
*
*      void __mod2__clock()

38 0000001e clock: fbegin
39 0000001e 4e56ffcc          link    fp, #-52
40 00000022 48ee1f07ffd0      movem.l d0/d1/d2/a0/a1/a2/a3/a4, (-48, fp)
41 00000028 >45f9000000000    lea    minutes, a2
42 0000002e >47f9000000000    lea    hours, a3
43 00000034 >49f9000000000    lea    seconds, a4

*
*      {
*      seconds++;

45 0000003a 5214                add.b  #1, (a4)

*      if (seconds>59)

47 0000003c 0c14003b          cmp.b  #59, (a4)
48 00000040 6318                bls    ?1.10

*      {
*      seconds=0;

50 00000042 4214                clr.b  (a4)

*      minutes++;

52 00000044 5212                add.b  #1, (a2)

*      if (minutes>59)

54 00000046 0c12003b          cmp.b  #59, (a2)
55 0000004a 630e                bls    ?1.10

*      {
*      minutes=0;

57 0000004c 4212                clr.b  (a2)

*      hours++;

59 0000004e 5213                add.b  #1, (a3)

```



```

*           if (hours>12) hours=1;

61 00000050 0c13000c          cmp.b  #12,(a3)
62 00000054 6304           bls   ?1.10
63 00000056 16bc0001         move.b #1,(a3)
64 0000005a          ?1.10
65                      section.strings

*           )
*           )
*           printf("\r%02d:%02d%02d",hours,minutes,seconds);

67 00000000          ds.w  0
68 00000000          ?S1
69 00000000 0d253032643a2530  dc.b  $0d,'%02d:%02d%02d',$0d
70                      section.text
71                      line 63
72 0000005a 7000          move.l #0,d0
73 0000005c 1014          move.b (a4),d0
74 0000005e 2e80          move.l d0,(sp)
75 00000060 7200          move.l #0,d1
76 00000062 1212          move.b (a2),d1
77 00000064 2f01          move.l d1,-(sp)
78 00000066 7400          move.l #0,d2
79 00000068 1413          move.b (a3),d2
80 0000006a 2f02          move.l d2,-(sp)
81 0000006c >48790000000000    pea   ?S1
82 00000072 >4eb90000000000    jsr   printf
83 00000078 4fef000c         lea   (12,sp),sp

*           )

85 0000007c          ?_2

86 0000007c 4cee1f07ffd0       movem.l (-48,fp),d0/d1/d2/a0/a1/a2/a3/a4
87 00000082 4e5e          unlk  fp
88 00000084 4e73          rte
89 00000086          fend
90                      import printf
91                      end

```

Section synopsis

```

1 00000005 (          5) .data
2 00000086 (         134) .text
3 00000010 (          16) .strings

```

Symbol table

```

.data      1 00000000 |
.text      2 00000000 | hours      E 1 00000000 | minutes  E 1 00000002 | printf  I 0 00000000
.strings   3 00000000 | clock      E 2 0000001e | main    E 2 00000000 | printf  I 0 00000000

```

Symbol cross-reference

```

.data      *4
.strings   *65
.text      *16          *70
clock      22          *38
hours      *7          42
main       *18
minutes    *10         41
printf     82          *90
seconds    *13         43

```

Freescale Semiconductor, Inc.

How to Reach Us:**Home Page:**

www.freescale.com

E-mail:

support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
 Technical Information Center, CH370
 1300 N. Alma School Road
 Chandler, Arizona 85224
 +1-800-521-6274 or +1-480-768-2130
 support@freescale.com

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
 Technical Information Center
 Schatzbogen 7
 81829 Muenchen, Germany
 +44 1296 380 456 (English)
 +46 8 52200080 (English)
 +49 89 92103 559 (German)
 +33 1 69 35 48 48 (French)
 support@freescale.com

Japan:

Freescale Semiconductor Japan Ltd.
 Headquarters
 ARCO Tower 15F
 1-8-1, Shimo-Meguro, Meguro-ku,
 Tokyo 153-0064
 Japan
 0120 191014 or +81 3 5437 9125
 support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
 Technical Information Center
 2 Dai King Street
 Tai Po Industrial Estate
 Tai Po, N.T., Hong Kong
 +800 2666 8080
 support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
 P.O. Box 5405
 Denver, Colorado 80217
 1-800-441-2447 or 303-675-2140
 Fax: 303-675-2150
 LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

