# Mod5234 Programmable Interrupt Timer

## Application Note

Revision 1.1
August 21, 2007
Document Status: Second Release

## *Table of Contents*

## *Introduction*

The Mod5234 has four programmable interrupt timer modules, PIT0-PIT3 (PIT0 is reserved for the uCOS timer). Each PIT is a 16-bit timer that provides precise interrupts at regular intervals with minimal processor intervention. The timer can either count down from the value written in the modulus register, or it can be a free-running down-counter.

This application note will describe the use of the timers, as well as provide an example program that uses one of the timers to create interrupts at a regular interval. The example is useful both for learning how to use interrupts as well as the PIT functions. Additional information concerning the PITs are available in the MCF5235 Reference Manual, Chapter 23 (Revision 2.0).

## *PIT Registers*

The PIT modules involve the use of three types of 16-bit registers: the PIT Control and Status Register (PCSR), PIT Modulus Register (PMR), and the PIT Count Register (PCNTR). Each PIT module has its own set of registers. The PCSR and PMR registers have read and write access while the PCNTR register only has read access. The following subsections describe these registers.

### *PIT Control and Status Register (PCSR)*

The PCSR registers configure the corresponding timer’s operation. You can enable/disable interrupts and the PITs, set the starting value for down-counting, determine when to reload a new starting value, and more with this register. Additional information about the use of this register can be found in section 23.2.1.1 of the reference manual.

An important note that requires attention is the configuration of the prescalar on bits 11-8 of the PSCR register. Configuring and generating the PIT clock requires knowing the system clock, PIT modulus value, and prescalar value for the desired timeout period. The following equation is used to calculate the timeout period for the PIT clock:

```
T = PIT Timeout Period [Seconds (s)]
P = PIT Prescalar
M = PIT Modulus Value
F = PIT Frequency [Hertz (Hz)]
S = System Clock [Hertz (Hz)]

To determine desired timeout period:
T = ( P x ( M + 1 ) x 2 ) / S

To determine PIT modulus value using desired PIT frequency
M = [S / ( 2 x P x F )] – 1
```

For example, lets say that a timeout period of about 0.001 second (1000 Hz) is needed. The system clock of the Mod5234 is 147.456 Mhz, so we have a value of 147,456,000 Hz for S. Using a system clock divisor value (prescalar value) of 16 is sufficient for 0.001 second. Knowing what prescalar value is needed depends on the desired timeout period. The PIT Modulus Register is only 16 bits, so the highest starting value it will count down from is 0xFFFF, or 65,535. If a larger time interval is used, such as 1 second, then a system clock divisor of 16 is not enough. The divisor would need to be larger, such as 32,768 (a bit configuration value of "1111" for the PCSR register from table 23-3) in order to keep the value of the modulus register under the maximum value. All that's left now is determining the modulus value, which comes out to be about 4607. With the prescalar bit configuration set to "0100" (from the prescalar table for system clock divisor of 16 of table 21-3), PIT Modulus Register written with the value 4607, and PCSR[1:0] written with "11" (reloads the down-counter from PMR once it reaches zero and enables the PIT), you get a PIT clock cycle of 0.001 second.

## *PIT Modulus Register (PMR)*

The 16-bit read/write PMR contains the timer modulus value that is loaded into the PIT counter when the count reaches 0x0000 and the PCSR[RLD] (reload) bit is set. Additional information on how to calculate the modulus value can be found in the previous section on the PIT Control and Status Register.

When the PSCR[OVW] (overwrite) bit is set, PMR is transparent, and the value written to PMR is immediately loaded into the PIT counter. The prescalar counter is reset (0xFFFF) anytime a new value is loaded into the PIT counter and also during reset. Reading the PMR returns the value written in the modulus latch. Reset initializes the PMR to 0xFFFF.

## *PIT Count Register (PCNTR)*

The 16-bit read-only PCNTR contains the counter value. Reading the 16-bit counter with two 8-bit reads is not guaranteed to be coherent. Writing to PCNTR has no effect, and write cycles are terminated normally.

## *Program Example*

The general procedure for setting up a PIT typically involves three steps: 1) Define an interrupt service routine, 2) Configure the PIT, and 3) Start the PIT. The following example program will configure an interrupt to trigger one PIT request event approximately every 1/1000th second (1000 Hz) via the PIT1 module.

```
/*********************************************************************
 * This example program exercises the programmable interrupt timer   *
 * the MCF5234 CPU.                                                  *
 *********************************************************************/

#include "predef.h"
#include <stdio.h>
#include <ctype.h>
#include <startnet.h>
#include <autoupdate.h>
#include <dhcpclient.h>
#include <smarttrap.h>
#include <taskmon.h>
#include <sim5234.h>
#include <cfinter.h>
#include <utils.h>
#include <pins.h>

//
// Function prototypes - Instruct the C++ compiler not to mangle the
// function names
//
extern "C"
{
   void UserMain( void *pd );

   //
   // This function sets up the 5234 interrupt controller
   //
   void SetIntc( int intc, long func, int vector, int level,
                 int prio );
}

const char *AppName = "MOD5234 PIT Example";   // App name for IPSetup
volatile DWORD pitr_count;                     // Global count variable

//////////////////////////////////////////////////////////////////////
// INTERRUPT - PIT interrupt service routine
//
INTERRUPT( my_pitr_func, 0x2600 )
{
   static WORD led_count;        // For incrementing carrier board LEDs
   WORD tmp = sim.pit[1].pcsr;   // Get PIT1 Control & Status Register
                                 // data

   //
   // Clear PIT1 - Refer to table 23-3 for more information on what
   // bits are being cleared and set
   //
   tmp &= 0xFF0F;           // Bits 4-7 cleared
   tmp |= 0x0F;             // Bits 0-3 set
   sim.pit[1].pcsr = tmp;

   //
   // You can add you ISR code here
   // - Do not call any RTOS function with pend or init in the function
   //   name
   // - Do not call any functions that perform a system I/O read,
   //   write, printf, iprintf, etc.
   //
   putleds( led_count++ );   // Increment carrier board LEDs
   pitr_count++;             // Increment when an interrupt occurs

   //
   // Toggle MOD5234 pin J2-48 to view the interrupts on an
   // oscilloscope. One cycle will be twice the time period. This
   // feature uses the NetBurner Pins Class, so you need to include
   // pins.h.
   //
   ( J2[48] ) ? J2[48] = 0 : J2[48] = 1;
}

////////////////////////////////////////////////////////////////////////
// SetUpPITR - PIT setup function. See chapter 23 of the 5235 reference
// manual for details
//
void SetUpPITR( int pitr_ch, WORD clock_interval, BYTE pcsr_pre /* See
                table 23-3 in the reference manual for bits 8-11 */ )
{
   WORD tmp;

   if ( ( pitr_ch < 1 ) || ( pitr_ch > 3 ) )
   {
      iprintf( "*** ERROR - PIT channel out of range ***\r\n" );
      return;
   }

   //
   // Populate the interrupt vector in the interrupt controller. The
   // SetIntc() function is supplied by the NetBurner API to make the
   // interrupt control register configuration easier
   //
   SetIntc( 0, ( long ) &my_pitr_func, 36 + pitr_ch, 2 /* IRQ2 */, 3 );

   //
   // Configure the PIT for the specified time values
   //
   sim.pit[pitr_ch].pmr = clock_interval;   // Set PIT modulus value
   tmp = pcsr_pre;
   tmp = ( tmp << 8 ) | 0x0F;
   sim.pit[pitr_ch].pcsr = tmp;   // Set system clock divisor to 16 and
                                  // set bits [3:0] in PCSR
}

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

```
// UserMain
//
void UserMain( void *pd )
{
   InitializeStack();
   if ( EthernetIP == 0 ) GetDHCPAddress();
   OSChangePrio( MAIN_PRIO );
   EnableAutoUpdate();
   EnableSmartTraps();
   EnableTaskMonitor();

   //
   // Let us make PIT happen at 1000 Hz. The base clock is
   // 147,456,000 Hz, so the equation is:
   //
   //                   System_Clock_Frequency / 2
   // PMR Value  =  -------------------------------  -  1
   //                  Prescalar * Desired_Frequency
   //
   //                  147456000 / 2
   // PMR Value  =  ---------------  -  1
   //                    16 * 1000
   //
   // PMR Value  =  4607
   //
   // Note that the PIT Count Register is a 16-bit counter, so the
   // clock count maximum is 65,535
   //
   SetUpPITR( 1 /* Use PIT1 */, 4607 /* Wait 4607 clocks */, 4 /*
              Divide by 16 from table 21-3 (2^4) */ );

   iprintf( "Application started\r\n" );
   pitr_count = 0;

   while ( 1 )
   {
      OSTimeDly( TICKS_PER_SECOND );
      iprintf( "PITR Count = %ld\r\n", pitr_count );
   }
}
```