Lecture 10: PI/T timer

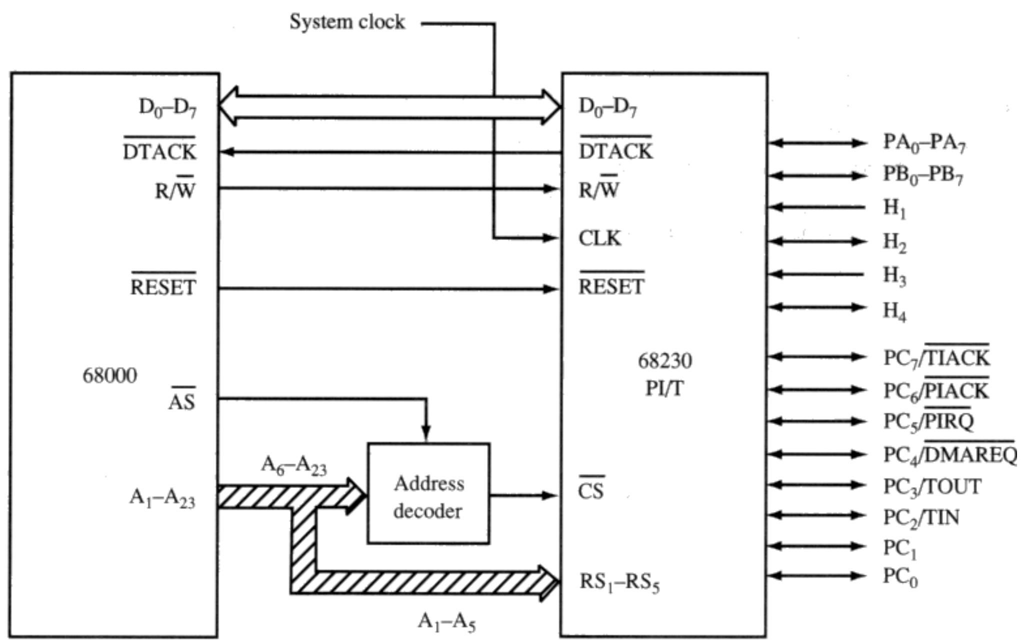
- **Introduction to the 68230 Parallel Interface/Timer**
- **Interface with the 68000**
- **PI/T Timer registers**
- **Timer Control Register**
  - Clock control
  - Zero-detect control
- **MOVEP instruction**
- **Examples**
  - Real-time clock
  - Square wave generator
- **Polling Vs. Interrupt**
- **Programming the PI/T in C language**

Introduction to the 68230

- **The 68230 PI/T (Parallel Interface/Timer) is a general-purpose peripheral**
  - Its primary function is a parallel interface
  - Its secondary function is a programmable timer
- **The PARALLEL INTERFACE provides 4 modes with various *handshaking* and *buffering* capabilities**
  - Unidirectional 8-bit
  - Unidirectional 16-bit
  - Bidirectional 8-bit
  - Bidirectional 16-bit
- **The PROGRAMMABLE TIMER provides a variety of OS services**
  - Periodic interrupt generation
  - Square wave generation
  - Interrupt after timeout
  - Elapsed time measurement
  - System watchdog
- **This lecture covers the (easier) programmable timer function**
  - The next two lectures will cover the parallel interface

PI/T simplified interface with the MC68000

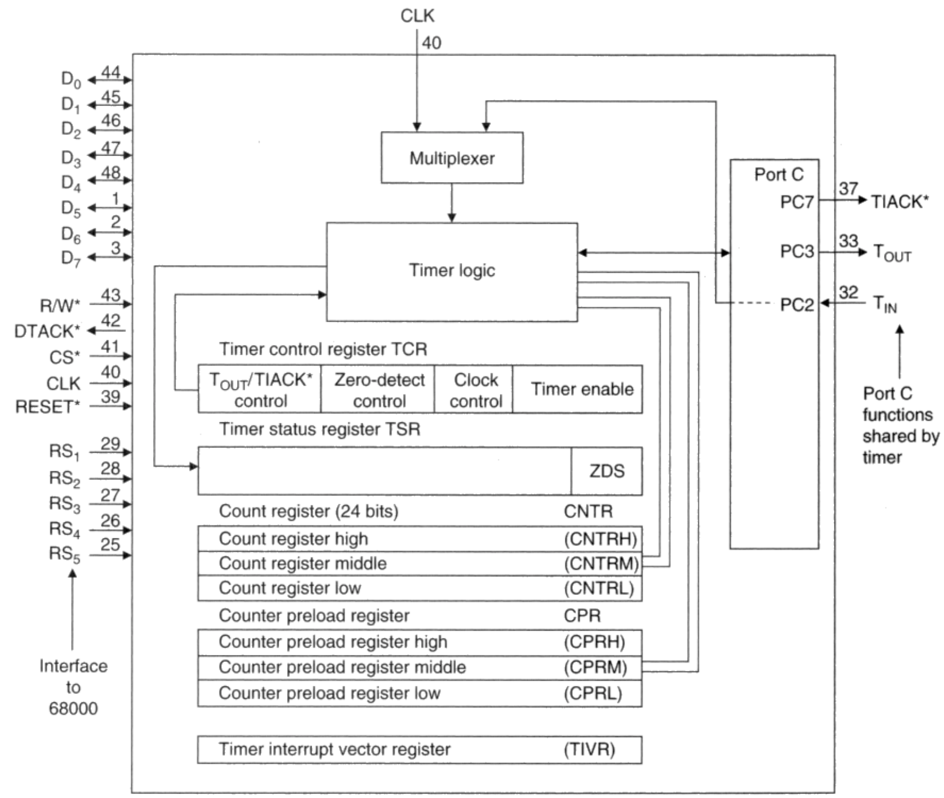
- **An address decoder places the PI/T at a given location within the address space of the processor**
  - On the SBC68K, the PI/T base address is $FE8000
- **The 68230 is programmed and used by reading and writing data to the correct memory-mapped locations (registers)**
- **The 68230 contains 23 internal registers, which are are selected by the state of 5 register-select inputs (RS_1-RS_5) connected to the address bus (A_1-A_5)**
  - Notice that ALL the registers are located at ODD memory locations
  - Only 9 of the 23 registers are used for the programmable timer function
- **Data to the internal registers is transferred through the data bus (D_0-D_7)**
- **There are three internal ports**
  - Port A and Port B are used for parallel interface
  - Port C is shared by timer and parallel interface
- **Handshaking is accomplished through lines H_1-H_4**



PI/T timer registers

- **Timer Control Register**
  - Determines the operation modes of the timer
- **Timer Interrupt Vector Register**
  - Stores the interrupt vector number
- **Counter Preload Register**
  - A 24-bit counter with the desired (by the programmer) number of counts measured in ticks
- **Counter Register**
  - A 24-bit counter down-counter that is **automatically** decremented with every tick
- **Timer Status Register**
  - Determines the status of the timer
  - Only Bit #0 (Zero Detect Status or ZDS) is used
  - In order to clear the ZDS bit after a zero-detect YOU MUST WRITE A 1 to it (YES, the ZDS bit is cleared by writing a ONE to it)

Register and Mnemonic		Acc.	Offset
Timer Control Register	TCR	R/W	$21
Timer Interrupt Vector Register	TIVR	R/W	$23
Counter Preload Register High	CPRH	R/W	$27
Counter Preload Register Middle	CPRM	R/W	$29
Counter Preload Register Low	CPRL	R/W	$31
Counter Register High	CNTRH	R	$2F
Counter Register Middle	CNTRM	R	$31
Counter Register Low	CNTRL	R	$33
Timer Status Register	TSR	R/W	$35



Timer Control Register

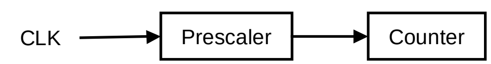
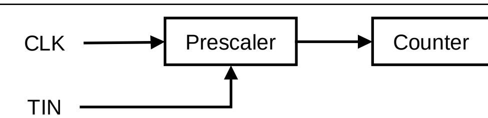
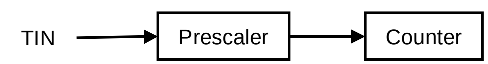
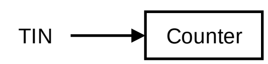
- **Timer Enable (TCR0)**
  - Turns the timer ON and OFF. The timer is disabled when the bit is cleared; it is enabled when set
    - To start the timer, place an 1 in TCR0
    - To stop the timer, place a 0 in TCR0
- **Clock Control (TCR1-2)**
  - The PI/T timer permits different clock pulse operations. When the field is 00, every 32 CPU clock cycles become 1 timer tick.
- **Counter Load (TCR4)**
  - After completing its countdown, the tick counter is either reset from the Counter Preload Register (CPR) or it rolls over to $FFFFFF
    - Writing a 0 on TCR4 causes a reload from the CPR
    - Writing a 1 on TCR4 causes a roll-over to $FFFFFF.
- **Action on Zero Detect (TCR5-7)**
  - The timer can select from a series of actions when the tick counter reaches 0.

Mode	TCR7	TCR6	TCR5	TCR4	TCR3	TCR2	TCR1	TCR0
1	1	X	1	0	X	00 or 1X		1
2	0	1	X	0	X	00 or 1X		1
3	1	X	1	1	X	00 or 1X		1
4	0	0	X	1	X	0	0	1
5	0	0	X	1	X	0	X	1
6	1	X	1	1	X	0	1	1
	T_{OUT}/TIACK* control			ZD control	Not used	Clock control		Timer enable

Mode 1: Real-time clock
Mode 2: Square wave generator
Mode 3: Interrupt after timeout
Mode 4: Elapsed time measurement
Mode 5: Pulse counter
Mode 6: Period measurement

Clock control (TCR2-TCR1)

- **The counter can be decremented from three different signals**
  - T_{IN}, the external clock input
  - The output of a 5-bit prescaler driven by CLK and enabled by T_{IN}
  - CLK, the system clock (prescaled)
- **The 5-bit prescaler allows us to divide the counter frequency by 32**
- **The SBC68K clock runs at 8MHz (125×10^{-9} seconds per count), so 1 second will require 250,000 CLK ticks (mode 00)**

TCR_2	TCR_1	Clock Control	Example
0	**0**	**PC_2/T_{IN} is a port C function. The counter clock is prescaled by 32, thus the counter clock is CLK/32. The timer enable bit determines whether the timer is in the run or halt state.**	
0	**1**	**PC_2/T_{IN} is a timer input. The prescaler is decremented on the falling edge of CLK and the counter is decremented when the prescaler rolls over from \$00 to \$1F (31_{10}) Timer is in the run state when BOTH timer enable bit and TIN are asserted.**	
1	**0**	**PC_2/T_{IN} is a timer input and is prescaled by 32. The prescaler is decremented following the rising transition of TIN after being synchronized with the internal clock. The 24-bit counter is decremented when the prescaler rolls over from \$00 to \$1F. The timer enable bit determines whether the timer is in the run or halt state.**	
1	**1**	**PC_2/T_{IN} is a timer input and prescaling is not used. The 24-bit counter is decremented following the rising edge of the signal at the T_{IN} pin after being synchronized with the internal clock. The timer enable bit determines whether the timer is in the run or halt state.**	

T_{OUT}/TIACK* control (TCR7-TCR5)

- **Bits 7-5 of the Timer Control Register control the way the PI/T timer behaves on a zero-detect (ZDS=1)**
  - Whether interrupts are supported (vectored, auto-vectored or none)
  - How does the PC3/T_{OUT} output pin behave
  - How is the PC7/TIACK* input pin interpreted

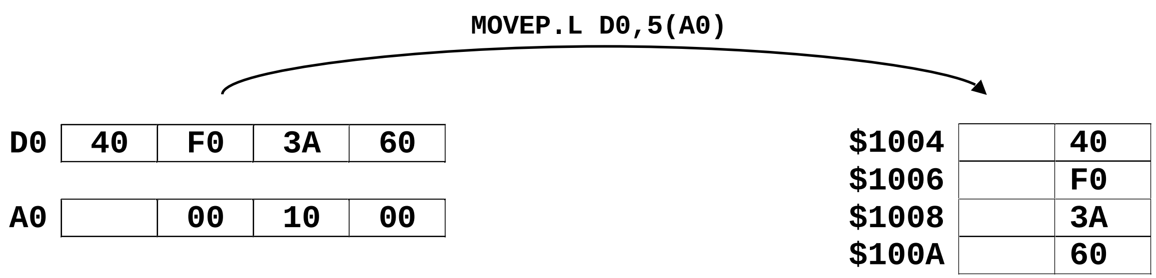
TCR_7	TCR_6	TCR_5	Timer response (simplified)	Timer response (detailed)
0	0	X	Use timer pins for the operation of I/O port C	PC3/T_{OUT} and PC7/TIACK* are port C functions
0	1	X	Toggle a square wave with each expiration of the timer	PC3/T_{OUT} is a timer function. In the run state T_{OUT} provides a square wave which is toggled on each zero-detect. The T_{OUT} pin is high in the halt state. PC7/TIACK* is a port C function.
1	0	0	No vectored interrupt generated on a count of 0	PC3/T_{OUT} is a timer function. In the run or halt state T_{OUT} is used as a timer request output. Timer interrupt is disabled, the pin is always three-stated. PC7/TIACK* is a port C function. Since interrupt requests are negated, PI/T produces no response to an asserted TIACK*.
1	0	1	Generate a vectored interrupt on a count of 0	PC3/T_{OUT} is a timer function and is used as a timer interrupt request output. The timer interrupt is enabled and T_{OUT} is low (IRQ* asserted) whenever the ZDS bit is set. PC7/TIACK* is used to detect the 68000 IACK cycle. This combination operates in the vectored-interrupt mode.
1	1	0	No autovectored interrupt generated on a count of 0	PC3/T_{OUT} is a timer function. In the run or halt state it is used as a timer interrupt request output. The timer interrupt is disabled and the pin always three-stated. PC7/TIACK* is a port C function.
1	1	1	Generate an auto-vectored interrupt on a count of 0	PC3/T_{OUT} is a timer function and is used as a timer interrupt request output. The timer interrupt is enabled and T_{OUT} is low whenever the ZDS bit is set. PC7/TIACK* is a port C function. This combination operates in an autovectored interrupt mode.

MOVEP instruction

- **The MOVEP instruction is provided to allow transfer of data to alternate bytes in memory**
  - This is very useful for 68000-based peripherals
- **Instruction format**

```
MOVEP.size Di,d(Aj)
MOVEP.size d(Aj),Di
```

- **Example**

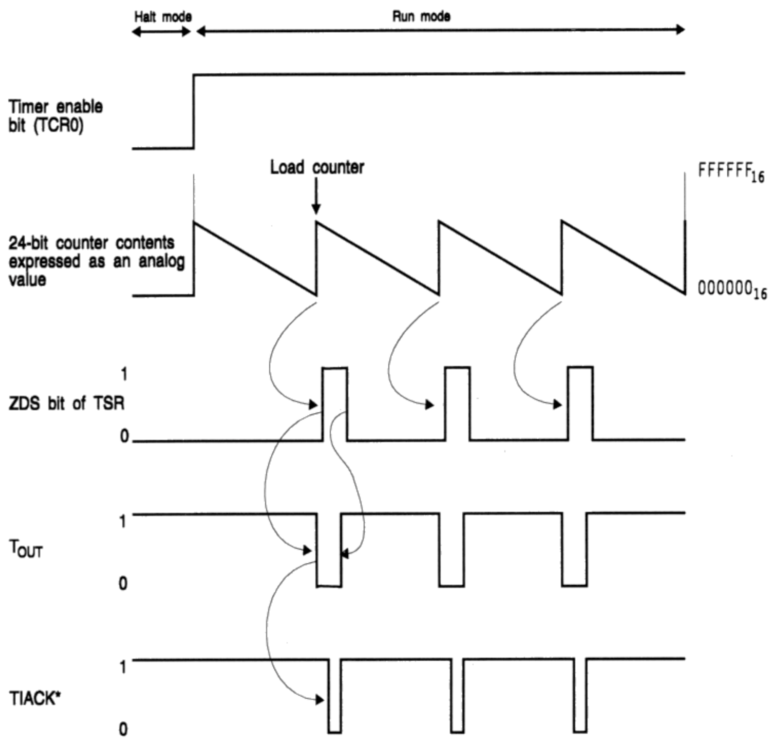


Example: Real-time clock

- **The PI/T generates an interrupt at periodic intervals**
- **Hardware configuration**
  - T_{OUT} MUST BE connected to one of the 68000's IRQ* lines
  - TIACK* MUST BE connected to the appropriate 68000's IACK* line
- **The counter is reloaded from CPCR on each zero-detect**
  - The ZDS MUST be cleared by the interrupt handler to remove the interrupt request
- **Sample assembly code**

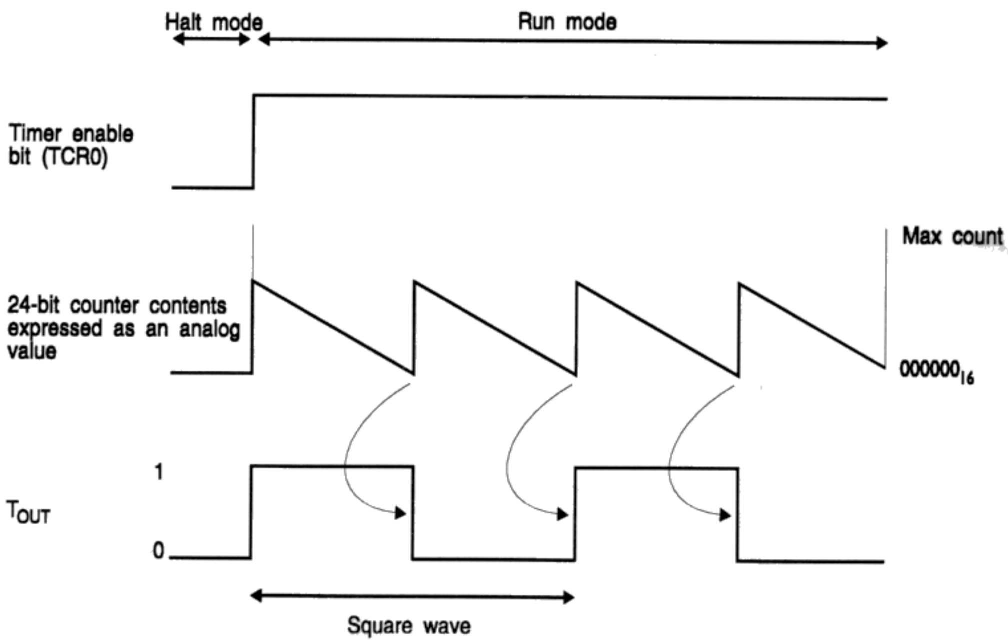
```
PIT     EQU     $FE8000         ;PI/T base address on the SBC68K
TCR     EQU     $21             ;offset to the timer control reg
TIVR    EQU     $23             ;offset to the timer int. vector reg
CPR     EQU     $25             ;offset to the counter preload reg
TSR     EQU     $35             ;offset to the timer status reg
TIME    EQU     4*timevec       ;location of the PI/T int. handler

setup   LEA     PIT,A0
        MOVE.B  #timevec,TIVR(A0)       ;load the interrupt vector
        MOVE.L  #$00FFFFFF,D0           ;setup the maximum count
        MOVEP.L D0,CPR(A0)              ;and load it into the CPR
        MOVE.B  #$10100001,TCR(A0)      ;set up the TCR
        RTS
```



Example: Square wave generator

- **The timer produces a square wave at its T_{OUT} output pin**
  - No interrupts are generated (supported)
- **Hardware configuration**
  - T_{OUT} MUST NOT be connected to an IRQ line or else the 68000 will be interrupted when T_{OUT} goes to LOW
  - TIACK* is ignored by the PI/T timer in this mode
- **The SBC68K has a jumper (JP6) that allows us to configure the way 68000 and PI/T are connected (SBC68K User's Manual, pp. 5-18)**
- **The TCR7 bit is cleared to allow the T_{OUT} pin to be toggled each time the counter rolls down to zero**
- **The period of the wave is determined by the valued loaded on the counter preload register**



Polling Vs. Interrupt

- **An alternative to programming interrupts is to create a polling loop**
  - The CPU periodically reads the ZDS bit off the PI/T
  - When ZDS=1 the CPU executes the code originally written for the interrupt handler
  - Unless the CPU has nothing else to do between timeouts of the PI/T timer, polling is a waste of CPU cycles!

polling.c

```
main () {
  set_up_pit_polling();

  while (1) {
    while (zds!=1) {
      /* do nothing until timeout */
      }
    clear_zds();
    perform_operation();
    }
}
```

interrupt.c

```
isr() {
  clear_zds();
  perform_operation();
}

main () {
  set_up_pit_interrupt(isr);

  while (1) {
    /* do something useful, isr()
       takes care of the timeout */
  }
}
```

Example: Programming interrupts for the PI/T in C language

```
/* This code will setup the 68320 to generate an interrupt every
   5 seconds. The interrupt service routine isr()clears the ZDS
   bit so the 68320 stops asserting the IRQ* line since its
   interrupt request has been serviced */

#define tmr  ((unsigned char*) 0xFE8021) /* Timer Base Address   */
#define tcr  (( unsigned char*) tmr)     /* Timer Control Reg    */
#define tivr (( unsigned char*) tmr+2)   /* Timer Int. Vect. Reg */
#define cprh (( unsigned char*) tmr+6)   /* Preload Hi Reg       */
#define cprm (( unsigned char*) tmr+8)   /* Preload Mid Reg      */
#define cprl (( unsigned char*) tmr+10)  /* Preload Lo Reg       */
#define cnrh (( unsigned char*) tmr+14)  /* Counter Hi Reg       */
#define cnrm (( unsigned char*) tmr+16)  /* Counter Mid Reg      */
#define cnrl (( unsigned char*) tmr+18)  /* Counter Lo Reg       */
#define tsr  (( unsigned char*) tmr+20)  /* Timer Status Reg     */
#define tvector 0x40                     /* Timer Vector reg     */
#define tmrcntrl 0x80                    /* Timer Mode           */


void isr() {

  /* so we get feedback when this function gets called */
  printf("Five secs has passed\n");

  /* reset the ZDS bit */
  *tsr = 0x01;

  /* return to main() */
  asm(" rte");
}
```

```
main () {
  long *vtable;
  int count=1250000;

  /* set supervisor mode and interrupt mask to 4 */
   asm("       move.w  #$2400,SR");

  /* setup the stack pointer */
  asm("           movea.l  #$20000,SP");

  /* setup timer control register  */
  *tcr = 0xA0;

  /* setup vector table entry */
  *tivr = 70;
  vtable = (long *) (70*4);
  *vtable = isr;

/* load the counter preload register */
  *cprl = (unsigned char) count;
  count = count >> 8;      /* shift right 8 bits */
  *cprm = (unsigned char) count;
  count = count >> 8;      /* shift right 8 bits */
  *cprh = (unsigned char) count;

  /* Start timer */
  *tcr = 0xA1;

  while (1) {
    /*do the regular control loop*/
    }
  }
```

Example: Programming a polling loop for the PI/T in C language

```
/* This code will setup a polling loop so the 68320 continuously
   checks the ZDS bit of the PI/T timer */

#define tmr  ((unsigned char*) 0xFE8021) /* Timer Base Address   */
#define tcr  (( unsigned char*) tmr)     /* Timer Control Reg    */
#define tivr (( unsigned char*) tmr+2)   /* Timer Int. Vect. Reg */
#define cprh (( unsigned char*) tmr+6)   /* Preload Hi Reg       */
#define cprm (( unsigned char*) tmr+8)   /* Preload Mid Reg      */
#define cprl (( unsigned char*) tmr+10)  /* Preload Lo Reg       */
#define cnrh (( unsigned char*) tmr+14)  /* Counter Hi Reg       */
#define cnrm (( unsigned char*) tmr+16)  /* Counter Mid Reg      */
#define cnrl (( unsigned char*) tmr+18)  /* Counter Lo Reg       */
#define tsr  (( unsigned char*) tmr+20)  /* Timer Status Reg     */
#define tvector 0x40                     /* Timer Vector reg     */
#define tmrcntrl 0x80                    /* Timer Mode           */



/* The isr() function is not needed anymore since the
   code it used to execute is now performed by main()
   after it reads that the ZDS bit has been set to 1 */
```

```
main () {
  int count=1250000;

  /* set supervisor mode and interrupt mask to 4 */
  asm("       move.w  #$2400,SR");

  /* setup the stack pointer */
  asm("           movea.l  #$20000,SP");

  /* setup timer control register  */
  *tcr = 0x80;

  /* load the counter preload register */
  *cprl = (unsigned char) count;
  count = count >> 8;      /* shift right 8 bits */
  *cprm = (unsigned char) count;
  count = count >> 8;      /* shift right 8 bits */
  *cprh = (unsigned char) count;

  /* Start timer */
  *tcr = 0x81;

  while (1) {
    while ( !(*tsr&1) ) {
      /* check until ZDS goes high */
      }
    printf("Five secs has passed\n");

    /* reset the ZDS bit */
    *tsr = 0x01;
  }
}
```