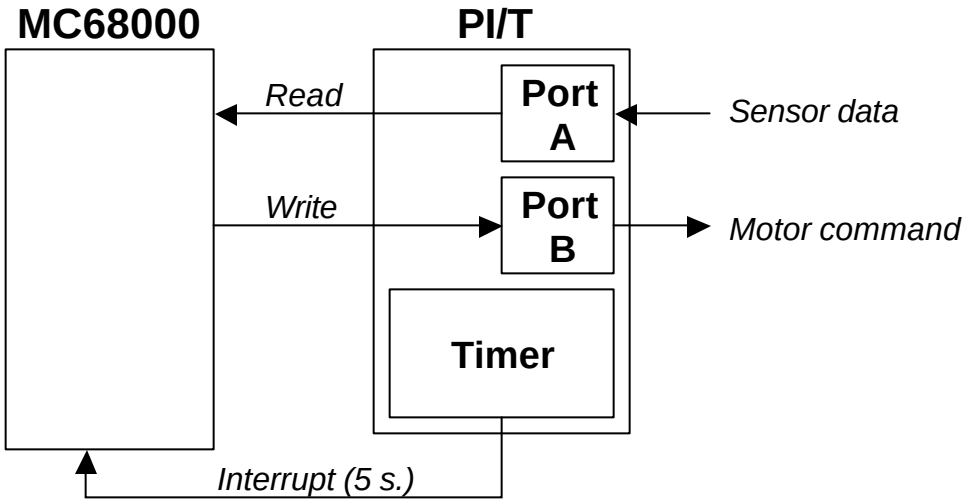
Lecture 2: MC68000 interrupt in C language

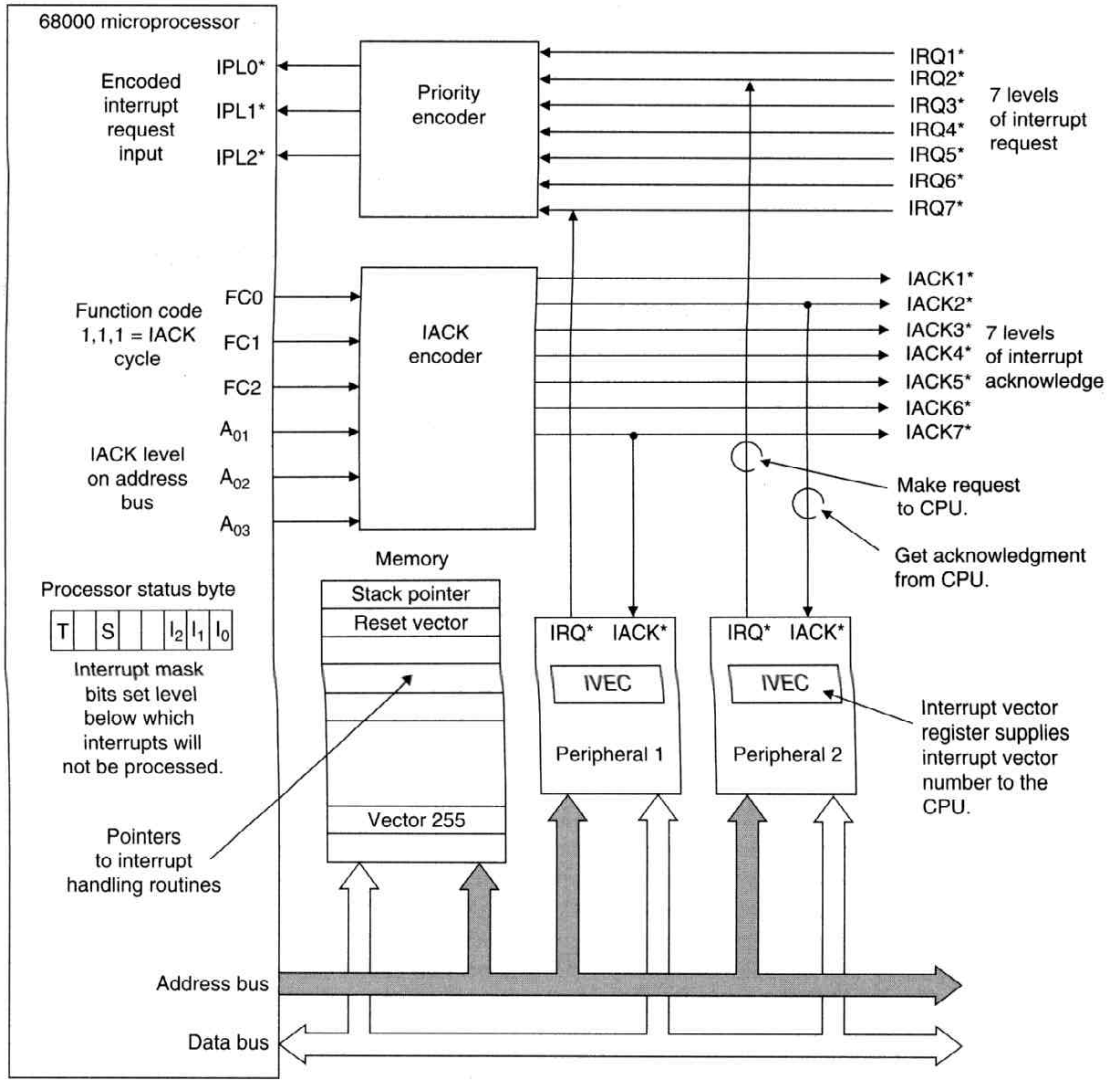
- **Problem definition**
- **Vectored interrupts**
- **The vector table**
- **Introduction to the PI/T**
- **Basic PI/T operation**
- **C language solution**

Problem

- **Setup the PI/T to read data from Port A and copy to Port B every 5 seconds**
  - Port A should be programmed for non-latched input
  - Port B should be programmed for single-buffered output
  - Timer should be setup to interrupt the MC68000 every 5 seconds



Vectored interrupts



Vectored interrupts

- **The 68000 provides two interrupt schemes**
  - **Vectored**: intended for modern 16-bit peripherals
  - **Auto-vectored**: intended for older 8-bit peripherals
- **There are seven levels of interrupts available**
- **The sequence of operations during a vectored interrupt request is the following**
  - A peripheral requires attention by asserting its interrupt request output (IRQ*)
  - The priority encoder produces a 3-bit code with the highest IRQ* line active and passes it to the 68000 on the IPL0*-IPL3* inputs
  - The 68000 compares the level of the interrupt with the interrupt mask flag ($I_2I_1I_0$) in the SR.
  - If the requested input is greater than ($I_2I_1I_0$), the interrupt is serviced, otherwise it is ignored
  - If the 698000 decides to service the interrupt:
    - The code 111 is placed on ($FC_2FC_1FC_0$) to inform the system that an interrupt is about to be serviced
    - The priority of the interrupt is placed on ($A_3A_2A_1$)
    - ($FC_2FC_1FC_0$) and ($A_3A_2A_1$) are passed to an interrupt acknowledge decoder which asserts one of the seven IACK* lines
  - The asserted IACK* line informs the interrupting device that it is about to be serviced
  - The peripheral whose interrupt level matches the asserted IACK* will “know” that it is going to be serviced
  - The peripheral then writes the IVEC vector onto the data bus (D_7D_0) and asserts the DTACK* line (DTACK stands for Data Transfer Acknowledge)
  - the active DTACK* terminates the IACK cycle and the 68000 will execute the interrupt handler pointed by the vector fetched from (D_7D_0)

The vector table

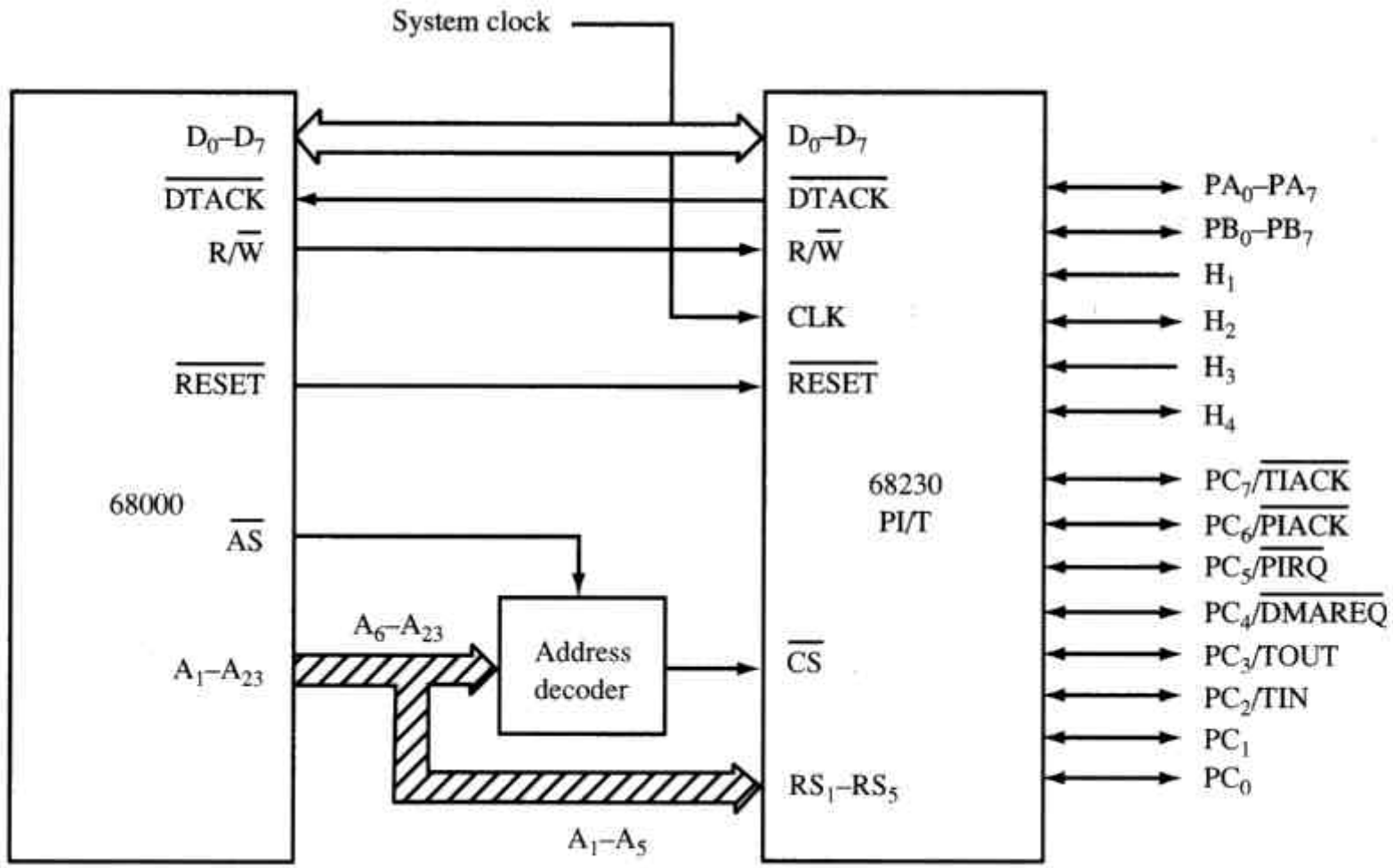
vector number (Decimal)	address (Hex)	assignment
0	0000	RESET: initial supervisor stack pointer (SSP)
1	0004	RESET: initial program counter (PC)
2	0008	bus error
3	000C	address error
4	0010	illegal instruction
5	0014	zero divide
6	0018	CHK instruction
7	001C	TRAPV instruction
8	0020	priviledge violation
9	0024	trace
10	0028	1010 instruction trap
11	002C	1111 instruction trap
12*	0030	not assigned, reserved by Motorola
13*	0034	not assigned, reserved by Motorola
14*	0038	not assigned, reserved by Motorola
15	003C	uninitialized interrupt vector
16-23*	0040-005F	not assigned, reserved by Motorola
24	0060	spurious interrupt
25	0064	Level 1 interrupt autovector
26	0068	Level 2 interrupt autovector
27	006C	Level 3 interrupt autovector
28	0070	Level 4 interrupt autovector
29	0074	Level 5 interrupt autovector
30	0078	Level 6 interrupt autovector
31	007C	Level 7 interrupt autovector
32-47	0080-00BF	TRAP instruction vectors**
48-63	00C0-00FF	not assigned, reserved by Motorola
64-255	0100-03FF	user interrupt vectors

NOTES:
* No peripheral devices should be assigned these numbers
** TRAP #N uses vector number 32+N

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

PI/T simplified interface with the MC68000



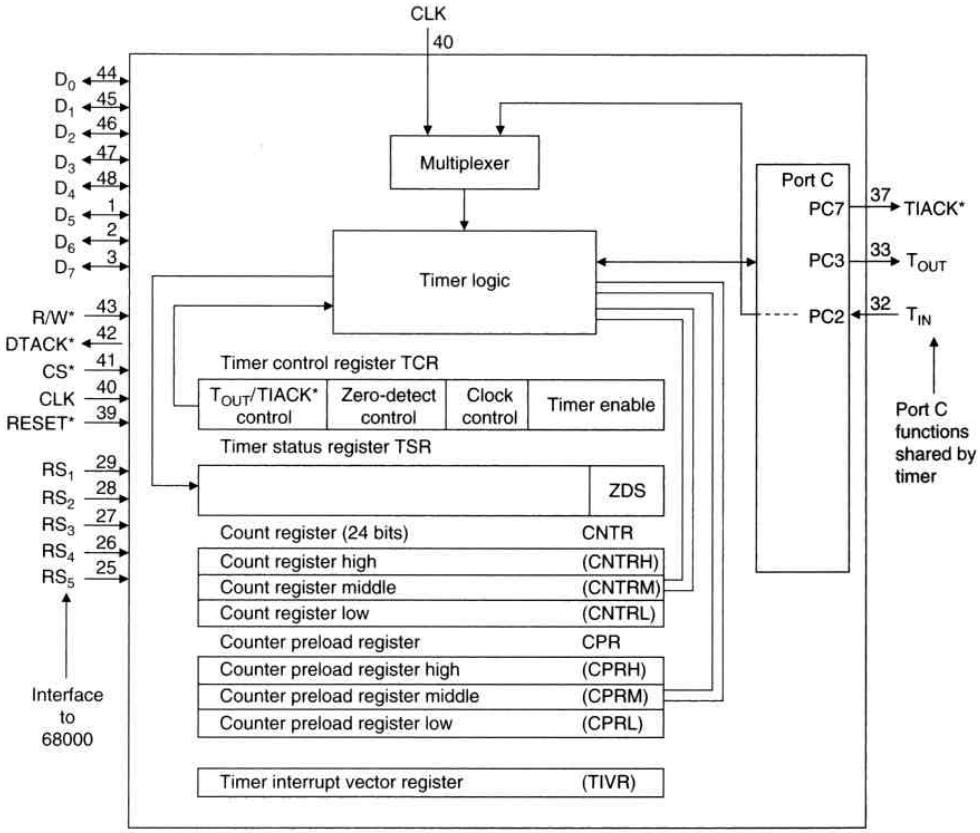
PI/T simplified interface with the MC68000

- **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 (TCR)**
  - Determines the operation modes of the timer
- **Timer Interrupt Vector Register (TIVR)**
  - Stores the interrupt vector number
- **Counter Preload Register (CPR)**
  - A 24-bit counter with the desired (by the programmer) number of counts measured in ticks
- **Counter Register (CNTR)**
  - A 24-bit counter down-counter that is **automatically** decremented with every tick
- **Timer Status Register (TSR)**
  - 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

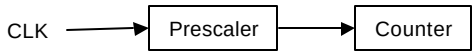
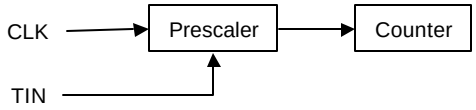
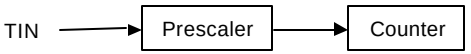
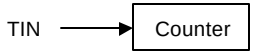
- **Timer Enable (TCR0)**
  - Turns the timer ON and OFF. The timer is disabled when the bit is cleared; it is enabled when set
    - To start the timer, place an 1 in TCR0
    - To stop the timer, place a 0 in TCR0
- **Clock Control (TCR1-2)**
  - The PI/T timer permits different clock pulse operations. 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.	

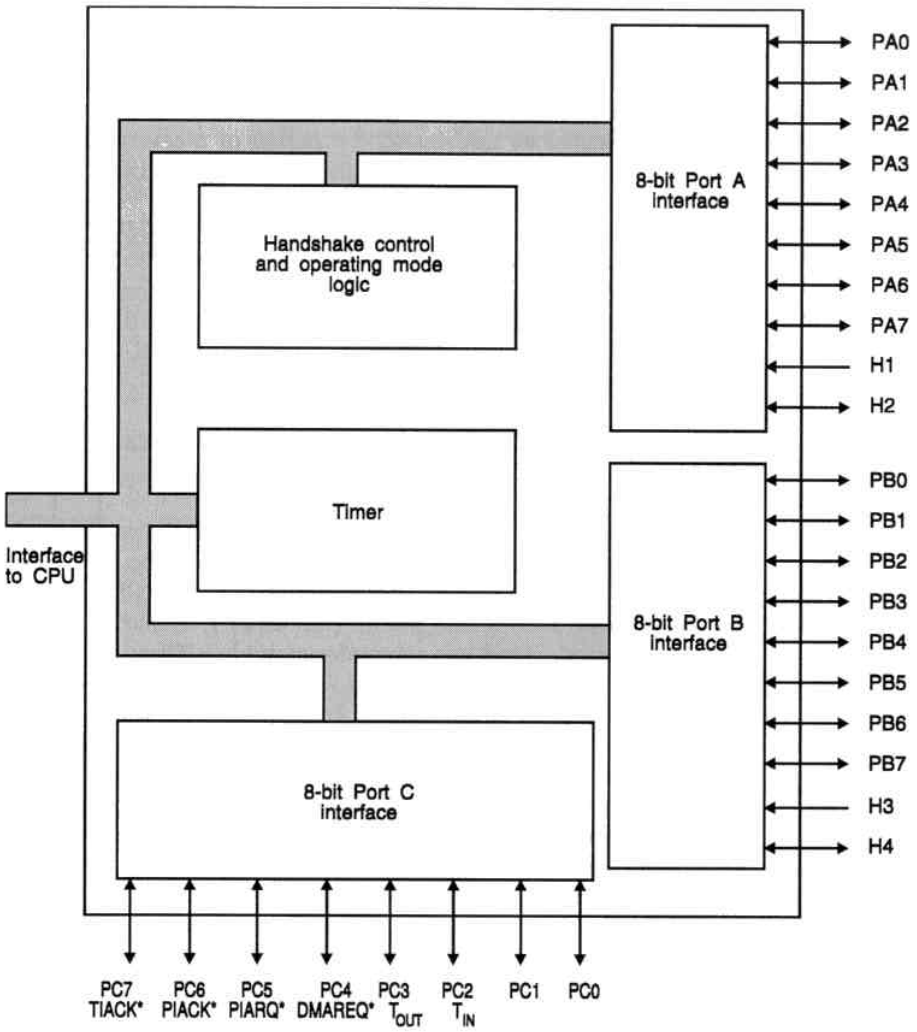
T_{OUT}/TIACK* control (TCR7-TCR5)

- **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.

Parallel I/O general description

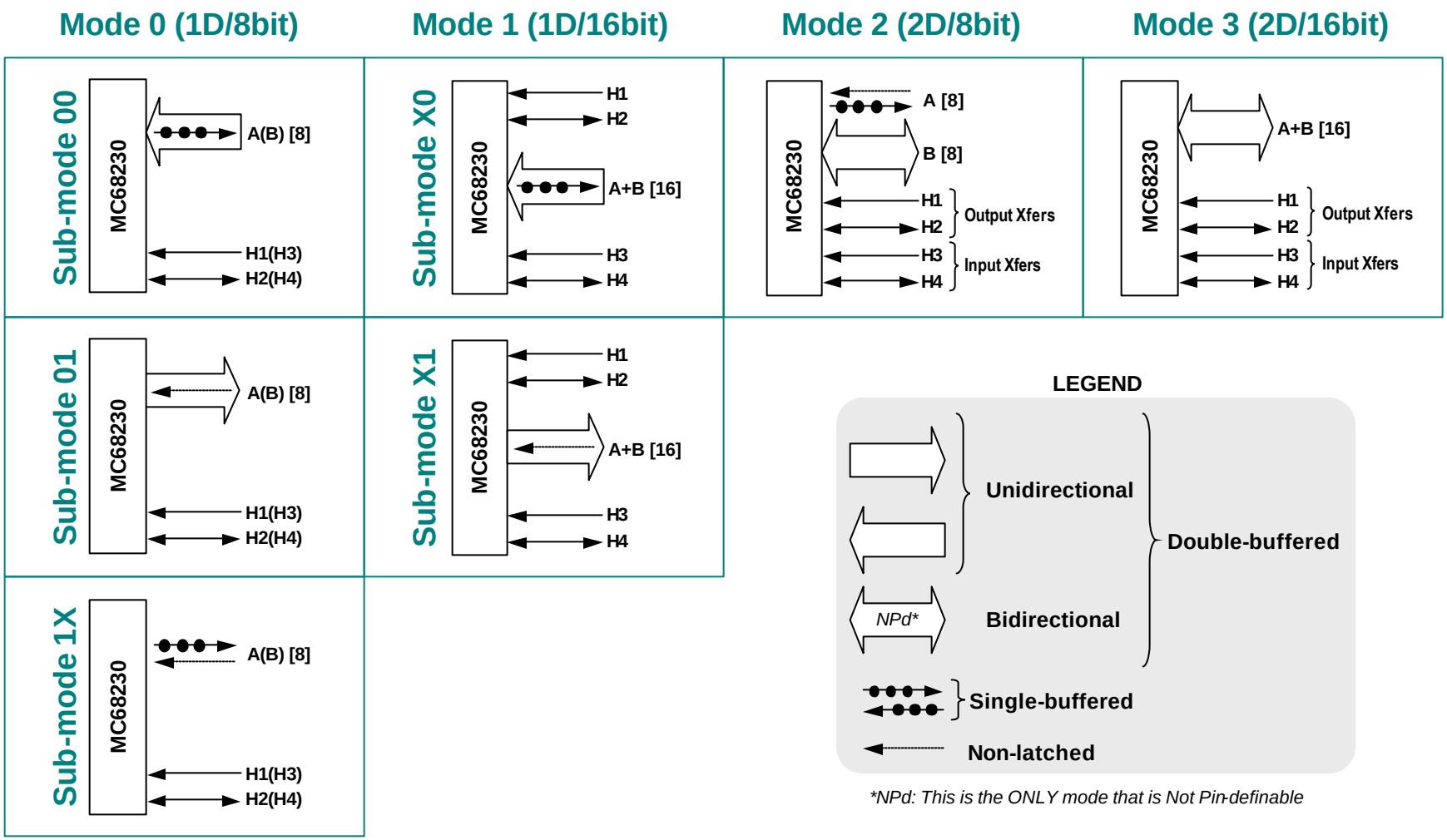
- **The parallel function has three ports**
  - Two independent 8-bit ports (and B)
  - A third dual-function port C
- **Ports A and B be can be used as I/O pc with various *handshaking* and *bufferin* capabilities in four different modes**
  - Mode 0: Unidirectional 8-bit
  - Mode 1: Unidirectional 16-bit
  - Mode 2: Bidirectional 8-bit
  - Mode 3: Bidirectional 16-bit
- **Port C can be used as**
  - a simple 8-bit port without handshaking or double-buffering
  - an interrupt interface to the timer
  - an interrupt interface to parallel I/O
  - a support interface for DMA operation



Brief overview of parallel I/O registers

- **Port General Control Register (PGCR)**
  - Selection of I/O modes (0, 1, 2 and 3) and handshaking signals (H1, H2, H3 and H4)
- **Port Service Request Register (PSSR)**
  - Selection of Port C functions: DMA requests, IRQ/IACK signals and handshaking signal priority
- **Port {A,B,C} Data Direction Register (PxDDR)**
  - Selection of individual port bits as inputs or outputs
- **Port Interrupt Vector Register (PIVR)**
  - Storage of vector number for vectored interrupts
- **Port {A,B} Control Register (PxCR)**
  - Selection of port sub-modes and handshake signals operation
- **Port {A,B,C} Data Register (PxDR)**
  - Contents of the I/O ports
- **Port {A,B} Alternate Data Register (PxADR)**
  - Instantaneous logic levels of the I/O pins of the port
- **Port Status Register (PSR)**
  - Status information of the handshake signals

The PI/T's modes of operation



*NPd: This is the ONLY mode that is Not Pin-definable

Port General Control Register

- **PGCR7-PGCR6**
  - Select the operating mode of the PI/T
- **PGCR5-PGCR4**
  - Enables the handshake pairs H3-H4 and H1-H2. These bits have to be set before we can make use of the control inputs and outputs. Doing this avoids spurious operation of the handshake lines before the PI/T has been fully configured
- **PGCR3-PCGR0**
  - Determine the *sense* of the four handshake lines. These control lines can be programmed to be active-low or active-high

Bit	PGCR7	PGCR6	PGCR5	PGCR4	PGCR3	PGCR2	PGCR1	PGCR0
Function	Port mode control		H34 enable	H12 enable	H4 sense	H3 Sense	H2 Sense	H1 Sense

PGCR7	PGCR6	Port mode control
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

PGCR5	H3, H4 control
0	H34 disable
1	H34 enable

PGCR4	H1, H2 control
0	H12 disable
1	H12 enable

PGCR3-0	Sense (Assertion Level)
0	LOW
1	HIGH

Port Data Direction Registers

- **Port Data Direction Registers: PADDR,PBDDR and PCDDR**
  - Select the direction and buffering characteristics of each of the appropriate port pins
    - A logical ONE makes the corresponding pin act as an OUTPUT
    - A logical ZERO makes the corresponding pin act as an INPUT
  - Port C behaves in the same fashion and determines whether each dual-function chosen for port C operation is an input or an output

Bit	PADDR7	PADDR6	PADDR5	PADDR4	PADDR3	PADDR2	PADDR1	PADDR0
Function	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Bit	PBDDR7	PBDDR6	PBDDR5	PBDDR4	PBDDR3	PBDDR2	PBDDR1	PBDDR0
Function	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Bit	PCDDR7	PCDDR6	PCDDR5	PCDDR4	PCDDR3	PCDDR2	PCDDR1	PCDDR0
Function	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Port Service Request Register

- **PSSR6-PSSR5 (service request)**
  - Determines whether the PI/T generates an interrupt or a DMA request
- **PSSR4-PSSR3 (operation select)**
  - Determines whether two of the dual-function pins belong to port C or perform special-purpose functions
- **PSSR2-PSSR0 (interrupt-priority control)**

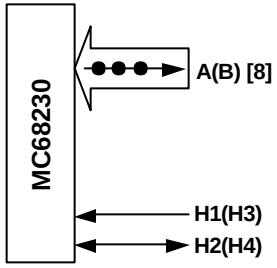
Bit	PSRR7	PSRR6	PSRR5	PSRR4	PSRR3	PSRR2	PSRR1	PSRR0
Function	´	SRVRQ (DMA control)		Interrupt control		Port interrupt priority		

PSRR6	PSRR5	Interrupt pin function	
0	´	PC4/DMAREQ* = PC4	DMA not used
1	0	PC4/DMAREQ* = DMAREQ*	Associated with double-buffered transfers controlled by H1 H1 does not cause interrups in this mode
1	1	PC4/DMAREQ* = DMAREQ*	Associated with double-buffered transfers controlled by H3 H3 does not cause interrups in this mode

PSRR4	PSRR3	Interrupt pin function	
0	0	PC5/PIRQ* = PC5	No interrupt support
		PC6/PIACK* = PC6	No interrupt support
0	1	PC5/PIRQ* = PIRQ*	Autovectored interrupt supported
		PC6/PIACK* = PC6	Autovectored interrupt supported
1	0	PC5/PIRQ* = PC5	
		PC6/PIACK* = PIACK*	
1	1	PC5/PIRQ* = PIRQ*	Vectored interrupt supported
		PC6/PIACK* = PIACK*	Vectored interrupt supported

PSRR2	PSRR1	PSRR0	Order of priority interrupt Higuest ¬ ¾ ¾ ¾ ® Lowest			
0	0	0	H1S	H2S	H3S	H4S
0	0	1	H2S	H1S	H3S	H4S
0	1	0	H1S	H2S	H4S	H3S
0	1	1	H2S	H1S	H4S	H3S
1	0	0	H3S	H4S	H1S	H2S
1	0	1	H3S	H4S	H2S	H1S
1	1	0	H4S	H3S	H1S	H2S
1	1	1	H4S	H3S	H2S	H1S

Mode 0, sub-mode 00



- **Data flow**
  - Double-buffered input or
  - Single-buffered output
- **Applications**
  - Normally used to receive data from devices such as A/D converters
- **Handshaking**
  - Data is latched into the input register by the asserted edge of H1
  - H2 behaves according to its programming function defined below
- **Port B behaves identically (using H3 and H4)**

Bit	PACR7	PACR6	PACR5	PACR4	PACR3	PACR2	PACR1	PACR0
	0	0	H2 Control			H2 Int.	H1 Control	

¬ ¾ ¾ ¾ ¾ ¾ ¾ ®

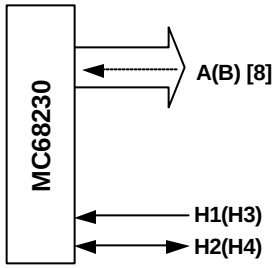
Sub-mode 00

PACR5	PACR4	PACR3	H2 Control	H2S
0	X	X	Input pin: Edge-sensitive input	Set on asserted edge
1	0	0	Output pin: negated	Always clear
1	0	1	Output pin: asserted	Always clear
1	1	0	Output pin: interlocked input handshake	Always clear
1	1	1	Output pin: pulsed input handshake	Always clear

PACR2	H2 interrupt
0	H2 interrupt disabled
1	H2 interrupt enabled

PACR1	PACR0	H1 Control
0	X	H1 interrupt and DMA request disabled
1	X	H1 interrupt and DMA request enabled
X	X	H1S status set anytime data is available in the double-buffered input path

Mode 0, sub-mode 01



- **Data flow**
  - Double-buffered output or
  - Non-latched input
- **Applications**
  - Normally used to send data to devices such as D/A converters or printers
- **Tables are almost identical to 0/00 except for PACR0**
  - If PACR0=0, H1S is set when port A is half-empty
  - If PACR0=1, H1S is set when port A is full-empty
- **Port B control is identical (using H3 and H4, of course)**

Bit	PACR7	PACR6	PACR5	PACR4	PACR3	PACR2	PACR1	PACR0
	0	1	H2 Control			H2 Int.	H1 Control	

¬ ¾ ¾ ¾ ¾ ¾ ¾ ¾ ®

Sub-mode 01

PACR5	PACR4	PACR3	H2 Control	H2S
0	X	X	Input pin: Edge-sensitive input	Set on asserted edge
1	0	0	Output pin: negated	Always clear
1	0	1	Output pin: asserted	Always clear
1	1	0	Output pin: interlocked input handshake	Always clear
1	1	1	Output pin: pulsed input handshake	Always clear

PACR2	H2 interrupt
0	H2 interrupt disabled
1	H2 interrupt enabled

PACR1	PACR0	H1 Control
0	X	H1 interrupt and DMA request disabled
1	X	H1 interrupt and DMA request enabled
X	0	H1S status set if either initial or final output latches can accept data and cleared otherwise
X	1	H1S status set if both initial and final output latches are empty and cleared otherwise

Solution

```
/* Timer Register Addresses */

#define tmr  ((unsigned char*) 0xFE8021) /* Timer Base Address */
#define tcr  (( unsigned char*) tmr)     /* Timer Control Reg */
#define tivr (( unsigned char*) tmr+2)   /* Timer Interrupt Vector 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 */

/* Parallel I/O Register Addresses */

#define PGCR  ( unsigned char*)0xFE8001   /* PI/T General Control Reg */
#define PSRR  ( unsigned char*)0xFE8003   /* PI/T Service Routine Reg */
#define PIVR  ( unsigned char*)0xFE800B   /* PI/T Interrupt Vector Reg */
#define PSR   ( unsigned char*)0xFE801B   /* PI/T Status Reg */
#define PACR  ( unsigned char*)0xFE800D   /* PI/T Port A Control Reg */
#define PADDR ( unsigned char*)0xFE8005   /* Port A Data Direction Reg */
#define PADR  ( unsigned char*)0xFE8011   /* Port A Data Reg */
#define PBCR  ( unsigned char*)0xFE800F   /* Port B Control Reg */
#define PBDDR ( unsigned char*)0xFE8007   /* Port B Data Direction Reg */
#define PBDR  ( unsigned char*)0xFE8013   /* Port B Data Reg */

void isr() {
  printf("Five secs has passed\n");

  *pbdr = *padr ; /* This is really the main job of isr *
                   It copies the content Port A data register (our input port)
                   and then places it to Port B (our output port)*/

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

  asm(" rte");
}
```

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

  asm("          move.w          #$2400,SR");
  asm("          movea.l         #$20000,SP);

  *PGCR =  0x0F;                  /* disable Port A & B */
  *PADDR = 0x00;                  /* Set Port A as input */
  *PBDDR = 0xFF;                  /* set Port B as Output  */
  *PSRR =  0x00;                  /* set PI/T for no Interrupts */
  *PBCR =  0x00; /*0r 0x80*/      /* Set Port B Control */
  *PACR =  0x40; /*0r 0x80*/      /* Set Port A Control */


  /****Prepare CPU for an interrupt processing**/

  *tivr   = 70;
  vtable  = (long *) (70*4);
  *vtable = isr;

  /****Set up timer control register*/

  *tcr = 0xA0;                    /* Set Timer Mode */

  *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;

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


  while (1) {
    /* Create an infinite loop which does nothing*/
  }
}
```