INTERRUPT TIMED CLOCK

ABSTRACT: This laboratory will introduce the student to the generation of interrupts (using a programmable timer), the handling of an interrupt, and the priority of interrupts.

This laboratory will implement a digital clock using interrupts timed at 1/10th second intervals. A 68230 programmed interface/timer (PI/T) chip emulated by the debugger using debugger macros will be used to program the timer. Note that the 68230's interrupt MUST be reset by clearing the ZDS bit in the interrupt service routine. At the start of the program you should accept an input time (hours, minutes, seconds). Your program will consist of an interrupt service routine that will update the time and display it on the screen every 1/10-th of a second, i.e. HH:MM:SS.D. I WILL LEAVE IT TO YOUR OPTION AS TO WHETHER YOU IMPLEMENT A 12 HOUR OR A 24 HOUR CLOCK. This will require that you convert your clock data from hex to ASCII.

OPTIONAL: To make the program more interesting you can program it to beep once for every hour, i.e. beep once for 1:00:00.0, twice for 2:00:00.0, etc. The beeping sound can be generated by sending a $07 to your terminal.

The program should have the following programming sequence:

1. Request the input time
2. Store the input time
3. Initialize registers in the PI/T
4. Reset the 68230 status bits
5. Set up the interrupt vectors
6. Enable the timer
7. Enable interrupts by changing the status register
8. Generate displays and beeps as necessary
9. Interrupt routines
    a. reset
    b. terminate

The terminal display function can be done using HexOut.

SETTING UP THE 68230 PI/T

Since this is a complicated chip and the setting of certain registers requires a detailed knowledge of how the chip is connected to the 68000 I will provide you with a guide to setting up some of the 68230 control registers.

REGISTER	BIT(S)	SET TO	DESCRIPTION
PGCR ($3F)	7,6	0	
	5	1	enable H3 and H4 interrupts
	4	1	enable H1 and H2 interrupts
	3,2,1,0	1	make H1/H2/H3/H4 high true

PSRR ($18)	7,6,5	0	no DMA
	4,3	1	set up interrupt pins
	2,1,0	0	H1 > H2 > H3 > H4 priority
PIVR ($44)		68	H1 vector at (68+0)*4 = $110, H2 vector at (68+1)*4 = $114, H3 vector at (68+2)*4 = $118, H4 vector at (68+3)*4 = $11B
PADDR	all	1	all pins (LED's) are outputs
PBDDR	all	0	all pins (switches) are inputs
PACR ($82)	7	1	mode 0 submode 1X
	6	0	mode 0 submode 1X
	5,4,3	0	H2 is output pin (LED above H1)
	2	0	no H2 interrupts
	1	1	accept H1 interrupts
	0	x	don't care
PBCR ($82)	7	1	mode 0 submode 1X
	6	0	mode 0 submode 1X
	5,4,3	0	H4 is output pin (LED above H3)
	2	0	no H4 interrupts
	1	1	accept H3 interrupts
	0	x	don't care
PBDR		x	input port
PADR		x	output port
PSR			status register for H1,H2,H3,H4
	3	0/1	H4 status (set to 1 to clear interrupt)
	2	0/1	H3 status (set to 1 to clear interrupt)
	1	0/1	H2 status (set to 1 to clear interrupt)
	0	0/1	H1 status (set to 1 to clear interrupt)
TCR ($A1)	7	1	set appropriate pin functions
	6	0	
	5	1	
	4	0	reload counter when countdown through zero
	2	0	use pre-scaler ÷ 32
	1	0	
	0	0/1	stop/start timer
TIVR		64	64*4=$100, the address of the vector
CPRH/CPRM/CPRL			counter preload registers YOU FIGURE OUT FOR 1/10-th SECOND
INTERRUPTS			
TSR	0	0/1	status of timer (ZDS bit), 1 indicates rollover, set to 0 to clear timer interrupt

These registers are located in memory as follows:

```
PGCR    $10001
PSRR    $10003
PADDR   $10005
PBDDR   $10007
PIVR    $1000B
PACR    $1000D
PBCR    $1000F
PADR    $10011
PBDR    $10013
PSR     $1001B
TCR     $10021
TIVR    $10023
CPRH    $10027
CPRM    $10029
CPRL    $1002B
TSR     $10035
```

HINTS/NOTES ON DEBUGGING YOUR LAB

1. Put your display routine in your main program loop. Display timing is not critical and will slow down your interrupt service routine. DO NOT call exceptions (TRAP's) in your interrupt service routine.

2. IMPORTANT NOTE: You have to set an option in the debugger to properly process interrupts. At the command line type:

    Debugger Options General Exceptions Normal

or

    Debugger Options General Exceptions Report

to enable interrupts.

The default is

    Debugger Options General Exceptions Stop

This command specifies what the debugger does on an exception. As one would guess Stop will simply display a message in the journal window and NOT process your exception. For debugging, I would recommend the Report option, it flashes a message in the journal window and processes the exception just as you would expect. When you actually run your clock program, use the Normal option which processes the exception the same way the Report option does BUT without the message to the journal window. This info is in the debugger manual on page 4-2.

3. Test a first version of your program without any fancy output. A major problem is often that your program is so complicated that you cannot debug it. Formatted output often causes problems.

4. You can examine the 68000 exception vector table using the debugger's Memory Display commands. You can also examine the 68230 registers. Note that the 68230 only has registers at odd addresses. Any attempt to display an even address (such as $10022) will generate a BUS ERROR.

Emulation Code User's Manual - the 68230 PI/T

Information on the 68230 PI/T can be found in the text book on pages 399-415. The information presented here is intended only to make writing code that uses the 68230 emulation easier.

What do I need to use the 68230 emulation?
You will need to load the file called "68230.com" into the debugger. This file can be loaded by typing, from the debugger command line: File Command 68230.com

I have put it into /home/courses/282/pub

How can I use the timer?
You can use the timer in two modes: interrupt mode or polled mode. The polled mode works similarly to the polled I/O from the keyboard. First, the timer is set up by writing to the counter preload registers and the timer control register in that order. Then the timer can be used by starting the timer, checking the timer status register until it signals the end of the given time, and then clearing the timer status register with a direct clear to reset the timer.

The interrupt mode operates much differently. To use this mode, the timer interrupt vector register must be properly set first. Then the counter preload registers can be set, and finally, the timer control register can be initialized. The interrupt vector register should not be changed from its initial value without stopping the time and starting the entire process from scratch.

In summary,
polled I/O mode

1. initialize counter preload registers
2. initialize timer control register
3. start the timer
4. poll the timer status register
5. when finished, direct clear the TSR. YOU MUST WRITE TO TRSW!!
6. to time the same period again, goto step 3.

interrupt I/O mode

1. initialize timer interrupt vector register
2. initialize counter preload registers
3. initialize timer control register

4. start the timer
5. The timer will continue timing the same period until stopped.
6. When finished, stop the timer by writing to the timer control register.

Steps 2 and 3 for the polled I/O case, or steps 3 and 4 for the interrupt case can be done together because they both involve writing to the timer control register.

How do I set the period I want to time?
You always set the period you want to time by writing to the three counter preload registers. As detailed in the text book, all three registers can be set at once using a single MOVEP instruction, although three MOVE.B instructions will also work. The timer ticks 125,000 times per second, so the proper number to load into the registers can be calculated by multiplying the number of seconds by 125,000.

How do I initialize the timer control register?
You can initialize the timer control register by doing a MOVE.B command. The number written to the control register determines the mode that the timer operates in. The settings should be made as follows:

action on zero detect	first three bits	should be either 100 for polled or 101 for interrupt
counter load	fourth bit	can be 1 to rollover or 0 to reset from preload
clock control	sixth,seventh bits	should always be 00
timer enable	eighth bit	can be 1 to start or 0 to stop timer

Additional Guidelines
The operations should always be done in the order specified. This means that the timer interrupt vector register should never be initialized after you start the timer. Also, anytime you want to change a parameter, the timer should be stopped, the parameter changed, and then the timer restarted. Changing a parameter while the timer is running could do anything from having no effect to sending the 68000 into random code sections. it is very important that you stop and restart the timer when changing parameters to give the emulation code a chance to recognize your changes.

In addition to the above guidelines, it is important to keep in mind that this timer is emulated. Therefore, the timed periods will not always be accurate. Due to overhead involved in running the debugger, the timer may run slightly fast or slow. Thus, it is usually best to try and minimize overhead if you want the timer to approximate the real device. In other words, if you are trying to run a clock, timing every tenth of a second might create excessive overhead and force the timer to ruin very slowly, with each tenth actually taking about a quarter of a second. However, timing every minute would create very little overhead, so the timer might actually run faster than a real clock. The moral of the story is: set the counter preload registers as if there were 125,000 ticks per second and expect the timer to go a little faster or slower than you set it to.

Timer register addresses
The addresses of the emulated timer registers are:

```
TCR     timer control register                  $10021
TIVR    timer interrupt vector register         $10023
CPRH    counter preload-high                    $10027
CPRM    counter preload-medium                  $10029
CPRL    counter preload-low                     $1002B
TSR     timer status register                   $10035
TSRW    timer status register (write only)      $10037
```

If you copy the clocklab.com file to your directory, you will be able to use an easy command to start up the debugger without needed the File Command 68230.com discussed above, i.e., by typing:

db68k -c clocklab <your_prog_name>

This means all of the following files should be in your directory:

```
68230.com
acia.com
iodorm.com
clocklab.com
extra.com
```

The extra.com file just sets the usp and ssp. If you want to do this tyourself, you can skip this file.

HINTS/NOTES ON DEBUGGING YOUR LAB

1. DO NOT call exceptions (TRAP's) in your interrupt service routine.

2. IMPORTANT NOTE: You have to set an option in the debugger to properly process interrupts. At the command line type:

```
Debugger Options General Exceptions Normal
```

or

```
Debugger Options General Exceptions Report
```

to enable interrupts.

The default is

```
Debugger Options General Exceptions Stop
```

This command specifies what the debugger does on an exception. As one would guess Stop will simply display a message in the journal window and NOT process your exception. For debugging, I would recommend the Report option, it flashes a message in the journal window and processes the exception just as you would expect. When you actually run your clock program, use the Normal option which processes the exception the same way the Report option does BUT without the message to the journal window. This info is in the debugger manual on page 4-2.

3. Test a first version of your program without any fancy output. A major problem is often that your program is so complicated that you cannot debug it. Formatted output often causes problems.

4. You can examine the 68000 exception vector table using the debugger's Memory Display commands. You can also examine the 68230 registers. Note that the 68230 only has registers at odd addresses. Any attempt to display an even address (such as $10022) will generate a BUS ERROR.

LAB #6 WRITE UP FORMAT:

1. Include the normal title page.

2. Include a Pseudo code description of your program. Be specific about 68230 initialization procedures and turning on/off interrupts during routines in your pseudo code.

3. Include an ASSEMBLER listing of your complete program.

4. Include a short description in English of how your debugger screen should look. Also include a Debugger screen showing the clock.

5. Answer the following questions
[Note: The 68230 is described in Section 12.4 of Ford & Topp in great detail.]

1. Describe the function of the 68230's TIVR. What is the vector number for your timer request? See Ford & Topp, p.758.

2. Describe the function of the zero detect field of the 68230's TCR.

3. When (if ever) does your main program terminate?

4. What does the instruction BCLR #0,TCON do? TCON is the address of the 68230's Timer Control Register.

Consider the following interrupt service routine (ISR) for a 68230 driven digital clock similar to your lab.

```
TIMER:     MOVE.W   #$2700,SR     ;(a)
           MOVE.L   A0,-(SP)
           ADDQ.W   #1,TOT_TIME
           LEA      TBASE,A0
           BSET     #0,(TSR,A0)  <---Why do you need this
                                     instruction?
           MOVE.L   (SP)+,A0
           RTE


TOT_TIME   DS.W     1
TBASE      EQU      $10021
TSR        EQU      20
```

5. Why do you need the indicated instruction?

6. What is the function of instruction (a)?

> In attempting to begin lab 6 we have encountered a number of problems:
>
> 1) How is input to be accepted from the user. The problem of
>decimal to hex conversion seems large. Are we supposed to accept it in
>the HH:MM:SS format as listed in the lab, and if so how is that input to
>be converted to hex so that it can be properly dealt with by the rest of
>the program.

As far as input is concerned, putting it into a register or memory with the debugger is perfectly fine and you do not need to convert to write an input conversion routine.
>
> 2) Along those same lines how is the information to be converted
>to ASCII for output? For example if you have counted 30 minutes and thus
>need to ouput a '30' to the screen how do you get a 1E hex to a 51 and a
>48 (decimal) for output as ASCII characters to the screen?

Now, this is more interesting and a lab requirement. The wasiest thing is to convert each number INDIVIDUALLY to an ASCII character, put them in memory somewhere, and use PutString to output them.
>
> 3) Also, it appears the 68230 has four interrupt lines (H1-H4)
>which are Port Interrupt Vectors. You have these listed as being set in
>the lab desciprtion, however are those the interrupts used or is it the
>TIVR interrupt that is used? You had also mentioned in class that the
>68000 should respond only to the Level 5 interrupt from the 68230 yet I
>cannot see how that relates to anything in how the 68230 is set up. It
>seems to me that upon counting down to 0 it will cause an interrupt which
>will jump to the vector at address $100 the way the 68230 is set up in
>the lab. Is this correct?

This is where you need to read section 12.4 of the textbook for a lot of detail. The H1 thru H4 interrupts are for other purposes. However, if you carefully read the description of the Port Interrupt Vector Register on page 762 of the text this should help. You may also want to look at the section of the text titled "A Standalone Timer Library" beginning on p.607. The routines initTimer there may be useful. Potentially more useful is the routine initIntTimer on page 759.

>In example 14.5, pg 760, what does the following line of code do:
>MOVE.L #clock,vClock.W
>
>I'm having trouble compiling this example because of this instruction. How
>is it linking the timer to the interrupt service routine

It does not link the timer to the interrupt service routine. It simply

loads the exception vector

it really is
   MOVE.L #clock,$100

our assembler probably does not like the vClock.w

Change it to MOVE.L #clock,$vclock

This instruction is putting the address of the interrupt service routine into the proper place in the exception vector table ($100) for a user interrupt vector of 64. The process of what is happening is described in detail on pages 776-777 of the text under Interrupt Acknowledge Bus Cycle.

We have gotten so much e-mail that we doubt that we can reply to all messages in the forseeable future. Some simple things you can do:

1. write a very small (minimalist) program to show that you can start the 68230 and handle interrupts. The following program is of that type and works very well.

```
        include   io.s          ;include the debugger i/o routines
        org       $1000         ;start program at $1000
Main:
        move.b    #2,D5         ;set variable seconds

        move.l    #intr,$100    ;load the address of the interrupt service
                                ;routine into the 68000 vector location
                                ;256 decimal = 64x4
        move.l    #125000,d0    ;put the counter preload into d0
        lea       $10025,a0     ;now put the address of the counter into a0
        movep.l   d0,(a0)       ;use movep to load the counter
        move.b    #64,$10023    ;load the vector # into the TIVR
        move.b    #$a1,$10021   ;load the TCR.   Setting $a just sets the
                                ;68230 to the appropriate mode.
                                ;The 1 in bit 0 starts the timer.

loop    nop                     ;now just do the same thing over n' over
        bra       loop

intr    add       #1,D5         ;this is the interrupt service routine
        rte                     ;done....

        end    Main
```

If this program does not work, you have other problems and should follow the following steps.

2. When you are getting debugger problems with files opening and so forth. Consider the following. Start the debugger without any arguments, i.e. just type db68k. When the debugger has started up type File Command clocklab.com. If the files load correctly then your .com files are NOT the cause of the problem. If the files do not load correctly copy

new files from /home/courses/282/pub Also if you get a file access error for 68230a.com be sure that you have changed the 68230.com in clocklab.com to 68230a.com

3. If your com files pass test #2 above and you get an error message when you try to run your program. Specifically, you might get something like

```
  Invalid vector number. Must be 0 to 255, or none.
00000000|
>  0005000
>00000000|
>  0005001
>00000000|
>  0005001Program Interrupt Add Once ccyc,2,TIVR
```

or anything else that is referring to TIVR. I suspect that these errors are caused by a global variable inside the debugger interferring with a variable of the same name in your program. The global variable is changing causing changes to the values you are loading into the 68230 and interferring with the macros. Change the name of the variables in your program and THIS problem SHOULD go away.

Sorry this took so long but I had my computer crash twice while typing this up and lost both previous messages I sent. You can modify the suggest grading scheme as appropriate.

LAB #6 WRITE UP FORMAT (as sent to the students):

1. Include the normal title page.

10 POINTS

2. Include a Pseudo code description of your program. Be specific about 68230 initialization procedures and turning on/off interrupts during routines in your pseudo code.

30 POINTS Be sure that the pseudocode describes how the 68230 is initialized and how the program works in detail. This might be accompanied by a write-up.

>This laboratory will implement a digital clock using interrupts timed at >1/10th second intervals. A 68230 programmed interface/timer (PI/T) chip >emulated by the debugger using debugger macros will be used to program the >timer.

We will accept either 1 (preload = 125000) or 0.1 (preload = 12500) second interrupts. The Interrupt Service Routine (ISR) can include the display or the display can be in the main program. The 68230 should be initialized once exernal to the ISR. The ISR should not reset and restart the 68230 everytime an interrupt occurs. Several students have done this.

-10 points if they continually reset the 68230 unless they specifically mention that this was done to overcome problems with the debugger macros.

You should look to see if they reset the interrupt mask during the ISR. This wil only work if the program was started in supervisor mode.

They may have reset the ZDS bit in the ISR. This is not necessary, but don't take off any points for it.

The Lab assignment also said:
>At the start of the program you
>should accept an input time (hours, minutes, seconds).

-5 points if they did not use HexIn or $10040/$10042 polled i/o

3. Include an ASSEMBLER listing of your complete program.

10 POINTS

4. Include a short description in English of how your debugger screen should look. Also include a Debugger screen showing the clock.

According to the lab description:
>Your program will
>consist of an interrupt service routine that will update the time and
>display it on the screen every 1/10-th of a second, i.e. HH:MM:SS.D. I WILL
>LEAVE IT TO YOUR OPTION AS TO WHETHER YOU IMPLEMENT A 12 HOUR OR A 24 HOUR
>CLOCK. This will require that you convert your clock data from hex to
>ASCII.

We will accept almost any format

20 POINTS TOTAL FOR DISPLAY:
-5 points if no debugger screen of the display format
-8 points for display using HexOut instead of a digital format

GENERAL POINTS:
This laboratory will implement a digital clock using interrupts timed at 1/10th second intervals. A 68230 programmed interface/timer (PI/T) chip emulated by the debugger using debugger macros will be used to program the timer. Note that the 68230's interrupt MUST be reset by clearing the ZDS bit in the interrupt service routine. At the start of the program you should accept an input time (hours, minutes, seconds). Your program will consist of an interrupt service routine that will update the time and display it on the screen every 1/10-th of a second, i.e. HH:MM:SS.D. I WILL LEAVE IT TO YOUR OPTION AS TO WHETHER YOU IMPLEMENT A 12 HOUR OR A 24 HOUR
CLOCK. This will require that you convert your clock data from hex to ASCII.

55
30
---
5. Answer the following questions

5 POINTS EACH. The student answers do not need to be as detailed as mine.

[Note: The 68230 is described in Section 12.4 of Ford & Topp in great detail.]

1. Describe the function of the 68230's TIVR. What is the vector number for

your timer request? See Ford & Topp, p.758.

The 68230 TIVR contains the vector number that the 68230 returns to the 68000 as part of the interrupt acknowledge cycle. This vector is used with the exception vector table to determine the address of the routine that should service this interrupt.

2. Describe the function of the zero detect field of the 68230's TCR.

According to Ford & Topp, page 603 the Action on Zero Detect field controls what happens when the timer counts down from zero. In Chapter 12 the 68230 was programmed with %100 which simply tells the counter to count and do nothing when it passes through zero. However, in Chapter 14 we typically used %101 which says to generate vectored interrupts on zero detect. Thus, everytime the counter passes through zero it will generate an interrupt. The 68000 will then ask the 68230 for the vector number which was previously programmed into TIVR to determine the address of the appropriate Interrupt Service Routine.

3. When (if ever) does your main program terminate?

An interrupt driven main program should never terminate. It should continually keep running in an infinite loop, or something similar.

4. What does the instruction BCLR #0,TCON do? TCON is the address of the 68230's Timer Control Register.

This is the 68230 timer enable bit. The 68230 timer is enabled when this bit is set to 1; it is disabled when the bit is set to zero. Specifically, the given instruction then stops the 68230's timer.

Consider the following interrupt service routine (ISR) for a 68230 driven digital clock similar to your lab.

```
TIMER:    MOVE.W   #$2700,SR    ;(a)
          MOVE.L   A0,-(SP)
          ADDQ.W   #1,TOT_TIME
          LEA      TBASE,A0
          BSET     #0,(TSR,A0)  <---Why do you need this
                                    instruction?
          MOVE.L   (SP)+,A0
          RTE


TOT_TIME  DS.W     1
TBASE     EQU      $10021
```

```
TSR      EQU     20
```

5. Why do you need the indicated instruction?

This was Problem 19(f), Chapter 14 of Ford & Topp.
According to F&T this instruction negates the timer interrupt request. This is incorrect.

According to page 605 of F&T, this is the Zero Detect Status (ZDS) bit. If ZDS=1 the counter has reached zero, otherwise the timer is still running or is not enabled. After the ZDS has been set to 1 by the counter reaching 0 it must be manually reset to 0. If it is not reset to zero, it will continually remain at one and cannot be used to detect any further zero crossings of the timer. Note that the timer continues to run. The TSR only monitors the zero crossings, the TCR controls the timer.

6. What is the function of instruction (a)?

The instruction labeled (a) changes the interrupt mask to %111 which disables all exteral interrupts. Note that the 68000 MUST be in supervisor mode for you to be able to do this.

> After going over the reports I've got so far, I found the
>following situations existing:
> 1. A few students still didn't give the proper format of pseu-
>docode. They wrote either the source file with comments for most
>instructions or the paragraphs of statements. So, I'd like to split
>pseudocode description into two parts: one for the format, the other
>for the logic(sometimes this part will actually graded by reading
>the listing instead of the pseudocode). Is that OK?

That seems fair. I would take off 5 points for the format and allocate the rest for the logic - the most important part.

> 2. Some students stated that they couldn't terminate their programs,
>However, you have instructed0 them how to do that. Shall I take off
>any points for that?

That's OK. Since we have implemented Exception processing the normal Trap #0 wll not work for them. So, don't take anything off for that.