Real Time Clock Subroutine

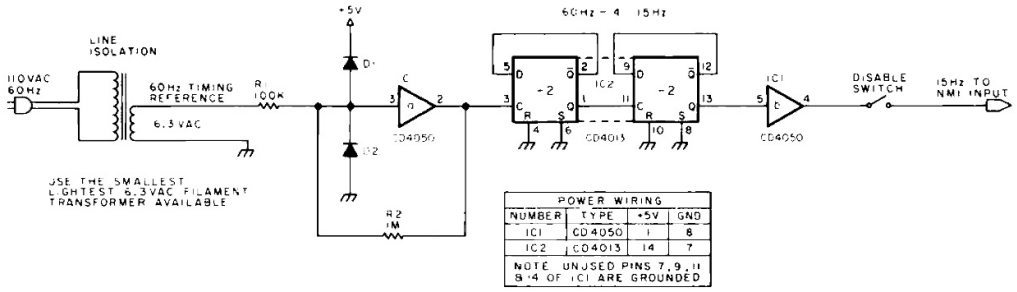
I wrote this subroutine a few years ago just as an exercise and thought you might like to see it.

As opposed to the more conventional straight line coding of real time clocks, this one uses a loop to increment each value and check it against the limit. Five zero page locations are required to hold the time and they must be properly initialized prior to use. Keep in mind that this is just a SUBROUTINE. The complete real time operating system must be written which uses RTCLK as the clock update routine.

Can you write one that is shorter?? If so, send it in and it may be published. How about one that's shorter AND faster?

```
HDE ASSEMBLER REV 2.2

 LINE#   ADDR   OBJECT     LABEL  SOURCE            PAGE 0001

01-0005  2000              ;6502 REALTIME CLOCK SUBROUTINE
01-0010  2000              ;WRITTEN BY E REHNKE 12,31,77
01-0015  2000              ;THE IDEA COMES FROM BYTE MAG
01-0020  2000              ;NOV. 77 P. 50. IT WAS CONVERTED
01-0025  2000              ;FROM A 6800 PROGRAM JUST TO SEE
01-0030  2000              ;HOW EASY IT BE TO DO.
01-0035  2000              ;
01-0040  2000              ;THIS ROUTINE IS SETUP TO USE
01-0045  2000              ;THE HARDWARE FROM BYTE, NOV.77
01-0050  2000              ;P. 72 AND CONSISTS OF THE 60 HZ.
01-0055  2000              ;LINE FREQ, DIVIDED DOWN AND
01-0060  2000              ;PROPERLY CONDITIONED TO FORM A
01-0065  2000              ;TTL COMPATIBLE 15 HZ. CLOCK WHICH
01-0070  2000              ;IS APPLIED TO THE INTERRUPT INPUT
01-0075  2000              ;ON KIM.
01-0080  2000              ;
01-0085  2000              ;THE FOLLOWING ZPAGE CLOCK REGS
01-0090  2000              ;MUST BE INITIALIZED TO THE
01-0095  2000              ;CORRECT TIME PRIOR TO STARTING
01-0100  2000              ;THE REAL TIME CLOCK.
01-0105  2000              ;THEY ARE AS FOLLOWS:
01-0110  2000              ;          $00E0= DAYS COUNTER
01-0115  2000              ;          $00E1= HOURS COUNTER
01-0120  2000              ;          $00E2= MINUTES COUNTER
01-0125  2000              ;          $00E3= SECONDS COUNTER
01-0130  2000              ;          $00E4= FRACTIONAL COUNTER
01-0135  2000              ;
01-0140  2000                     *=$E0
01-0145  00E0              CLKREG *=*+5
01-0150  00E5              ;
01-0155  00E5                     *=$2000
01-0160  2000  F8         CLOCK   SED               ;WORK IN THE DECIMAL MODE
01-0165  2001  A2 04              LDX #$4           ;INITIAL OFFSET
01-0170  2003              ;
01-0175  2003  18         CKLOOP  CLC
01-0180  2004  B5 E0              LDA CLKREG,X      ;GET THE TIME...
01-0185  2006  69 01              ADC #$1           ;DO A DECIMAL INCREMENT...
01-0190  2008  95 E0              STA CLKREG,X      ;AND PUT IT BACK.
01-0195  200A  DD 17 20           CMP TABLE,X       ;CHECK IT AGAINST THE LIMIT.
01-0200  200D  D0 07              BNE RETURN        ;NOT YET? THEN LEAVE.
01-0205  200F  A9 00              LDA #$0           ;IF SO, CLEAR THAT REGISTER.
01-0210  2011  95 E0              STA CLKREG,X
01-0215  2013  CA                 DEX               ;IS IT TIME TO LEAVE?
01-0220  2014  10 ED              BPL CKLOOP        ;NOT DONE, DO SOME MORE.
01-0225  2016  60         RETURN  RTS               ;BACK TO MAIN LINE.
01-0230  2017              ;
01-0235  2017  99         TABLE   .BYTE $99,$24,$60,$60,$15
01-0235  2018  24
01-0235  2019  60
01-0235  201A  60
01-0235  201B  15
01-0240  201C              FINISH .END


ERRORS = 0000

END OF ASSEMBLY = 201B
:ASM
```

Figure 1: A simple circuit which processes a 6.3 VAC reference signal derived from the power companies' 60 Hz grid to produce a digital logic level square wave at 15 Hz which can drive an interrupt line of a typical processor. The disable switch is optional and can be left out if the interrupt handlers are permanently loaded in ROM; otherwise, interrupts must be manually disabled while the systems software is bootstrapped into volatile memory.



NUMBER	TYPE	+5V	GND
IC1	CD4050	1	8
IC2	CD4013	14	7

Adding an Interrupt Driven Real Time Clock

James R Sneed
13831 NE 8th, Apt 86
Bellevue WA 98005

Whenever a computer is interacting with the real world, either through sensors or actuators, a real time clock can be valuable. Using a real time clock, the computer can run programs at specified times or intervals, or the computer may record the times at which events are sensed.

There are two basic types of real time clocks used in computing systems: the external (hardware) clock and the internal (software) clock. An external clock uses hardware to keep track of time, and periodically or on command transmits the time to the computer. */Robert Grappel's article on page 68 of this issue shows one approach to such a clock . . .***CH**/ An internal software clock has hardware which interrupts the computer at regular intervals, and software which keeps track of time by incrementing a register whenever the computer receives a timing interrupt.

The hardware clock imposes a small software burden on the computer, and being separate from the computer, it need not be reset whenever the computer is shut off. The software clock imposes a larger software burden on the computer, and the clock must be initialized if the computer has been completely halted or had its power shut off. In applications where the computer operates continuously, the advantages of the software clock due to hardware simplicity outweigh its disadvantages due to increased software burden, and the software clock is the logical choice for a real time clock.

There are two key considerations involved in selecting the interrupt rate for the software clock. First, where the interrupt clock is derived by dividing a higher frequency clock, such as a 1 MHz computer clock, hardware simplicity favors as high an interrupt rate as possible, but the computational overhead of interrupt response increases with increasing interrupt rate. Second, a low interrupt rate produces a low computational burden but decreases timekeeping resolution and programming flexibility. Since my system requires no routines to be performed more often than 15 times per second, I decided that a 15 Hz interrupt derived by dividing the 60 Hz power line frequency by 4 would be an adequate interrupt rate. This gives a minimum event to event resolution of 67 ms.

Listing 1: Interrupt handler. This routine contains the overhead needed to field an NMI interrupt on a 6502 processor, save the state of the processor, call an interrupt processing subroutine, restore the state of the processor, and return from the interrupt event. If the jump at location 206 is replaced by NOP operations, this program will spin its wheels 15 times a second, doing nothing in response to the 15 Hz signal produced by the circuit of figure 1. With the exception of the JSR at location 206, this routine is independent of the location in memory of the software discussed in this article.

Hexadecimal Address	Hexadecimal Code	Op	Commentary
0200	48	PHA	Push accumulator onto stack
0201	8A	TXA	Transfer X register to accumulator
0202	48	PHA	Push X register onto stack
0203	98	TYA	Transfer Y register to accumulator
0204	48	PHA	Push Y register onto stack
0206	20 00 00	JSR	Call CLOCK
0209	68	PLA	Pull Y register from stack
020A	A8	TAY	Transfer accumulator to Y register
020B	68	PLA	Pull X register from stack
020C	AA	TAX	Transfer accumulator to X register
020D	68	PLA	Pull accumulator from stack
020E	40	RTI	Return from interrupt
FFFA	00 02		Interrupt address vector

The circuit in figure 1 produces the 15 Hz interrupts. The 60 Hz signal is taken from the secondary of a 6.3 V filament type transformer. (The term is a hangover from vacuum tube days when many tubes had 6.3 V filaments somewhat like incandescent light bulbs). The input to IC1A, a CMOS buffer, is clamped between 5 V and ground by diodes D1 and D2, which can be any silicon small signal diodes at hand. Resistor R2 provides positive feedback to produce about a half a volt of hysteresis in the switching of the buffer. This hysteresis reduces false interrupts due to line voltage fluctuations and transients. The two D type flip flops in IC2 are used as cascaded divide-by-two circuits. The 15 Hz output from IC2 is buffered to drive TTL loads by IC1B. To prevent runaway power consumption and the resulting chip destruction, the unused inputs of the CMOS integrated circuits are grounded.

The nonmaskable interrupt of the 6502 is edge triggered; that is, the processor receives an interrupt whenever the voltage on the nonmaskable interrupt line goes from high (>2.4 V) to low (<2.4 V). The nonmaskable interrupt line can then stay low without generating another interrupt. When the processor receives a nonmaskable interrupt it jumps to the memory address stored at FFFA and FFFB, and pushes the address from which it was interrupted and the processor status onto the stack so that it can return to the preinterrupt computation as soon as it has processed the interrupt. A switch is shown between the 15 Hz interrupt and the NMI line so that interrupts can be disabled after power is applied until the interrerupt handler for NMI has been loaded in volatile memory. If the interrupt handler is in read only memory, this switch can be omitted.

The contents of the accumulator and the X and Y registers should be saved by software when the interrupt is received and control switches to the interrupt handler program. This is done by pushing them onto the stack using appropriate instructions. Once the preinterrupt state has been safely preserved, the processing done as a result of the interrupt is performed. After the interrupt program has been completed, the preinterrupt contents of the Y and X registers and the accumulator are restored by pulling them off the stack. The processor then pulls the preinterrupt processor status and program address from the stack and returns to the previous computation. Listing 1 is a sample interrupt handler.

Listing 2 is a 24 hour clock generated in software by accumulating 15 Hz interrupts. This program contains only relative jumps and so is easily relocatable, either in volatile memory, EROM or PROM.

The operation of the program real time

Listing 2: Time of day clock. If the jump at line 206 in the interrupt handler of listing 1 references the CLOCK routine, locations C4 to C7 in memory address space are continously updated with hours, minutes, seconds and 1/15 seconds respectively as the 15 Hz interrupts invoke its action. The 6502 code of this routine has been constructed to use relative branches only, so that it can be relocated anywhere in memory address space at the convenience of its user without modification of the object code.

CLOCK (Real Time Clock)

Hexadecimal Address	Hexadecimal Code	Label	Op	Operand	Commentary
0000	F8	CLOCK	SED		Set decimal mode
0001	18		CLC		Clear carry
0002	A5 C7		LDA	FSEC	Load seconds fraction
0004	69 01		ADC	1	Incr seconds fraction
0006	85 C7		STA	FSEC	Store seconds fraction
0008	38		SEC		Set carry
0009	E9 15		SBC	15	Subtract 15
000B	D0 2C		BNE	END	If not 15, go to end
000D	85 C7		STA	FSEC	Reset seconds fraction
000F	A5 C6		LDA	SEC	Load seconds
0011	18		CLC		Clear carry
0012	69 01		ADC	1	Incr seconds
0014	85 C6		STA	SEC	Store seconds
0016	38		SEC		Set carry
0017	E9 60		SBC	60	Subtract 60
0019	D0 1E		BNE	END	If not 60, go to end
001B	85 C6		STA	SEC	Reset seconds
001D	A5 C5		LDA	MIN	Load minutes
001F	18		CLC		Clear carry
0020	69 01		ADC	1	Incr minutes
0022	85 C5		STA	MIN	Store minutes
0024	38		SEC		Set carry
0025	E9 60		SBC	60	Subtract 60
0027	D0 10		BNE	END	If not 60, go to end
0029	85 C5		STA	MIN	Reset minutes
002B	A5 C4		LDA	HOURS	Load hours
002D	18		CLC		Clear carry
002E	69 01		ADC	1	Increment hours
0030	85 C4		STA	HOURS	Store hours
0032	38		SEC		Set carry
0033	E9 24		SBC	24	Subtract 24
0035	D0 02		BNE	END	If not 24, go to end
0037	85 C4		STA	HOURS	Reset hours
0039	D8	END	CLD		Clear decimal mode
003A	60		RTS		Return
00C4		HOURS			Storage for hours
00C5		MIN			Storage for minutes
00C6		SEC			Storage for seconds
00C7		FSEC			Storage for seconds/15

CLOCK is straightforward. Time is stored in BCD in the first page of memory: hours in 00C4, minutes in 00C5, seconds in 00C6, and 1/15 seconds in 00C7. When an interrupt is received and the preinterrupt state saved, the interrupt handler will call the real time CLOCK at 0000 (location 0206 in listing 1). The second's fraction is incremented and compared to 15. If it is less than 15 the processor will jump to the end of the clock program for return, but if it equals 15 the second's fraction is reset to zero and the seconds are incremented. Seconds, minutes and hours are handled similarly, counting modulo 60, 60 and 24 respectively. At the end of the program the processor returns to the interrupt handler. The clock can be set simply by loading the desired time into the time memory locations.

By comparing desired program times with the time of the real time CLOCK program, the processor may perform programs at any desired interval, up to one day, which is expressable as a multiple of 1/15 second. As an example, a program to be performed once per second would be executed only at those times when CLOCK has counted the second's fraction equal to zero.

It is important that the real time CLOCK should not impose an unreasonable computational burden on the computer. Using a 15 Hz interrupt and the program shown here, this criterion is satisfied. When run in a computer using a 6502 processor with a 1 MHz clock, the interrupt service requires about 1100 μs per second. This 0.1% cannot be called an excessive burden on the computer.■