A Precise Real Time Clock

As μCs are being used more frequently in data analysis and in control applications, it is becoming increasingly important for the computer to know the actual time an event took place. To do this, the computer must have access to a real time clock — an internal or external device that keeps track of the time in hours, minutes, seconds, etc. This article describes the construction and interfacing of an accurate, remote real time clock with a μC system that required an accuracy of a hundredth of a second.

A Simple Clock

A very simple real time clock can be constructed which uses the line voltage to supply a short pulse which interrupts the μP. Most μPs have one or more pins that can interrupt the processor which can then go to an area of memory that services the interrupt. The program residing there would cause memory locations to be updated. The time is then determined by examining those memory locations and decoding the values (it is also possible to decode the information at interrupt before storing in the memory locations, but that uses much more processor time). Such a clock can measure to the nearest tenth of a second with an accuracy dependent on AC power frequency. This type of clock is quite adequate for most applications but not when a precise time (or time interval) is required that is relatively short (few seconds) or when the power system is unreliable.

More Precise Versions

A real time clock which is more precise can be built along two lines. One is to continue with an interrupt system but use a quartz oscillator to generate the interrupt signals. To obtain high precision (small time intervals), a larger and larger percentage of the processor's time is devoted to time measurement as the frequency of interruptions increases. To avoid wasting valuable processor time, a non-interrupt, real time clock was designed and interfaced. This device can keep track of the time and inform the computer only when requested. A non-interrupt clock once required external memory to keep track of the time. However, with the new LSI clock chips available, one cannot only construct a very precise clock but also do it very easily. The design presented below satisfies three requirements: 0.01 second resolution, ease of interfacing to the μC through 8 bit parallel ports and high accuracy.

Number	Segments lit abcdefg	Condensed code abefg	Decimal	Hex (converted)
0	xxxxxx	11110	30	F0
1	xx	01000	8	40
2	xx xx x	11101	29	E8
3	xxxx x	11001	25	C8
4	xx xx	01011	11	58
5	x xx xx	10011	19	98
6	x xxxxx	10111	23	B8
7	xxx	11000	24	C0
8	xxxxxxx	11111	31	F8
9	xxxx xx	11011	27	D8

Table 1: Numbers, their seven segment LED code, and the five segments ("condensed code") needed to uniquely determine them; the decimal equivalent of this latter, condensed code also appears. The decimal equivalent must be multiplied by 8 (2^3) and converted to hex for comparing to the information coming to the computer from the clock (see Figure 2). This converted hex code appears last. See Figure 1 for segment labeling.

Inputs								Outputs		
1 H	2 10M	3 M	4 10S	5 S	6 S/10	7 S/100	8 10H	C	B	A
							0	1	1	1
						0	1	0	0	0
					0	1	1	0	0	1
				0	1	1	1	0	1	0
			0	1	1	1	1	0	1	1
		0	1	1	1	1	1	1	0	0
	0	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	0

Table 2: Logic for a 1 of 8 priority encoder (74147). A low (0) to one of eight inputs (each corresponding to a digit in the "display") can be encoded to 3 bits on the output. Blanks in the input indicate either a high (1) or low may exist.

Clock Design

An Intersil 7045 was chosen since it afforded convenient interfacing

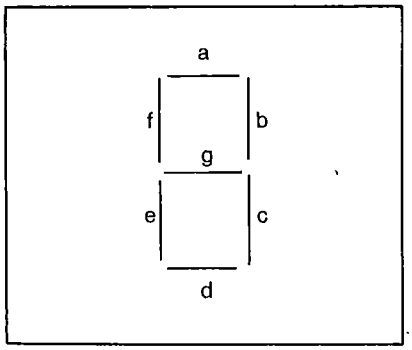

Figure 1: A seven segment LED digit and the normal labeling of the segments.

with several attractive options. This chip was built for stopwatch applications yet would accurately keep track of the time through 24 hours with a resolution of 0.01 seconds. It stores the time internally as hours, minutes, seconds and hundredths of a second. A very attractive feature was the "split" mode which allowed the output from the chip (the display) to freeze while the clock continued to measure the time. Thus one could read the digits without affecting the timing accuracy.

The CMOS clock chip can run on 5V and does not require a special power supply. The chip needs two input lines: one for reset and another for start/stop (this for the "display" in the split mode). The output of the chip is designed to go to a multiplexed seven segment, eight digit common cathode LED display. This means fifteen lines are needed to see the time on a display. However, many of these lines carry redundant information and it was possible to reduce the number of output lines to eight without losing any information. The eight output lines from the clock chip and the two input lines to the chip can share the same 8 bit parallel port, but, for simplicity and because two lines were free in another 8 bit port, this design uses two lines from one 8 bit port as input and eight lines from another as output from the chip.

A multiplexed display requires a high to each of the appropriate segments and a low to the cathode of that digit so that the segments light in the digit correctly (see **Figure 1** for segment code). Each digit is lit individually, but this occurs so fast that the eye would see all eight digits of the display lit at once. The width of each pulse to the display from the clock chip is about 0.1ms. The digits are lit in the following order: .01 sec, 10 hr, 1 sec, 10 min, 10 sec, 1 min, .1 sec, and 1 hr. All eight digits can thus be read in .8ms, which is much less than the clock's 10ms resolution.

The code for a seven segment display appears in **Table 1;** only five (a,b,e,f and g) of the seven segments are needed to uniquely specify the number. Since each digit is lit sequentially, the digit corresponding to the segments is determined by sensing which digit line is low. For example, when the time is 16:22:40.53 (hours: minutes: seconds), then, when the ten second digit would go low, the segments b,(c),f and g would go high to give a display of 4.

Instead of looking at each of the eight digit lines to see which is low, one can use a priority encoder (74147) to give a three bit output (2^3) indicating to which of the eight digit lines is low **(Table 2).** Thus, the seven segment lines have been reduced to five and the eight digit lines have been reduced to three resulting in one byte (for 8 bit processors) that contains the value of a particular digit. Eight bytes are required to fully specify the time.

Address Data	Assembled Code	Address Data	Assembled Code
BE00 08	PHP	BE51 2907	AND #$07
BE01 A207	LCX #$07	BE53 CDF6BF	CMP $BFF6
BE03 A900	LDA #$00	BE56 F0F6	BEQ $BE4E
BE05 9DF7BF	STA $BFF7,X	BE58 60	RTS
BE08 CA	DEX	BE59 A000	LDY #$00
BE09 D0FA	BNE $BE05	BE5B ADF5BF	LDA $BFF5
BE0B AD0EC7	LDA $C70E	BE5E 29F8	AND #$F8
BE0E 2907	AND #$07	BE60 C9F0	CMP #$F0
BE10 D0F9	BNE $BE0B	BE62 F029	BEQ $BE8D
BE12 A20A	LDX #$0A		
BE14 204ABE	JSR $BE4A	BE64 C8	INY
BE17 AD0EC7	LDA $C70E	BE65 C940	CMP #$40
BE1A 8DF5BF	STA $BFF5	BE67 F024	BEQ $BE8D
BE1D 2907	AND #$07	BE69 C8	INY
BE1F D0EA	BNE $BE0B	BE6A C9E8	CMP #$E8
BE21 ADF5BF	LDA $BFF5	BE6C F01F	BEQ $BE8D
BE24 A200	LDX #$00	BE6E C8	INY
BE26 2059BE	JSR $BE59	BE6F C9C8	CMP #$C8
BE29 8EF6BF	STX $BFF6	BE71 F01A	BEQ $BE8D
BE2C 204EBE	JSR $BE4E	BE73 C8	INY
BE2F A20A	LDX #$0A	BE74 C958	CMP #$58
BE31 204ABE	JSR $BE4A	BE76 F015	BEQ $BE8D
BE34 AD0EC7	LDA $C70E	BE78 C8	INY
		BE79 C998	CMP #$98
BE37 8DF5BF	STA $BFF5	BE7B F010	BEQ $BE8D
BE3A 2907	AND #$07	BE7D C8	INY
BE3C AA	TAX	BE7E C9B8	CMP #$B8
BE3D 8DF6BF	STA $BFF6	BE80 F00B	BEQ $BE8D
BE40 2059BE	JSR $BE59	BE82 C8	INY
BE43 8A	TXA	BE83 C9C0	CMP #$C0
BE44 2907	AND #$07	BE85 F006	BEQ $BE8D
BE46 D0E4	BNE $BE2C	BE87 C8	INY
BE48 28	PLP	BE88 C9F8	CMP #$F8
BE49 60	RTS		
BE4A CA	DEX	BE8A F001	BEQ $BE8D
BE4B D0FD	BNE $BE4A	BE8C C8	INY
BE4D 60	RTS	BE8D 98	TYA
BE4E AD0EC7	LDA $C70E	BE8E 9DF7BF	STA $BFF7,X
		BE91 60	RTS

Table 3: Machine Language subroutine to read and decode the data from the real clock. A Flow Chart is shown in Figure 3. All values are hexidecimal and are for the 6502 microprocessor. The # symbol refers to an immediate command and the $ symbol preceeds a number in hexidecimal. The starting address is BE00.

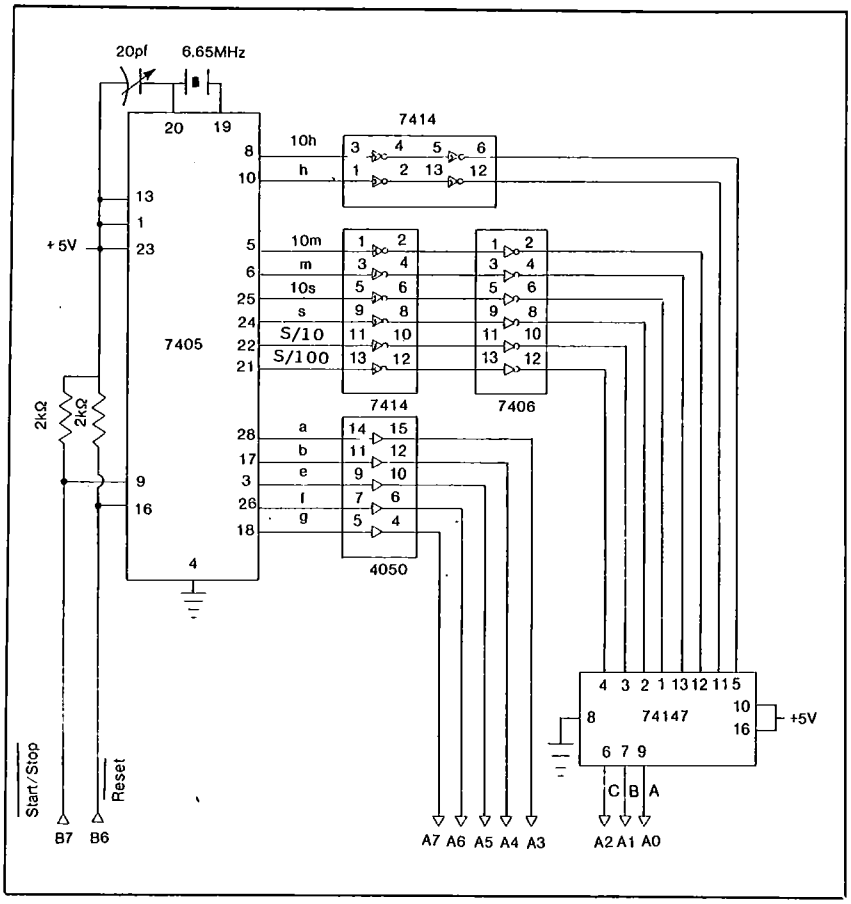

Figure 2: Electrical schematic for a non-interrupt, remote, real time clock. An Intersil ICM7045IPI stopwatch clock chip is used allowing eight digit accuracy: .01 sec (S/100) to 10 hr (10 HR) in a 24 hour format. Schmidt triggers (7414) and a 4050 are used to bring the 7405 signals to TTL levels. The priority encoder (74147) determines which digit (s, 10s, etc.) corresponds to the segment data presented to the computer. By reading eight consecutive values, the time is determined.

The remaining chips bring the logic pulses from the clock chip to TTL levels. The 7405 does not provide sufficiently distinct output voltage levels to drive the 74147. A Schmidt trigger (7414) performs this task but inverts the signal in the process. An inverter (7406) follows the 7414 and provides the correct logic levels for the 74147. The 4050 is a CMOS to TTL driver. The complete schematic is shown in **Figure 2.** the 20 pF variable capacitor connected to the crystal can be varied to provide accurate time. An accurate frequency meter can be connected to one of the outputs of the 7414 (i.e. pin 2) and the capacitor varied for a period of 1.25 msec; this will provide an accuracy of 1 second/week.

```
1      REM THIS IS MICROSOFT BASIC ON OHIO SCIENTIFIC C4PMF
100    POKE 8955,0: POKE 8956, 190: POKE 8960, 189: REM SET LOCATION OF USR (X)
1100   REM SUBROUTINE FOR TIME
1110   R = PEEK (50954) AND 56: REM RECALL CURRENT VALUES ON THIS PORT
1120   POKE 50954, 192 + R: POKE 50954, 64 + R: REM UPDATE "DISPLAY"
1130   X = USR(X): REM CALL MACHINE LANGUAGE SUBROUTINE
1140   FOR L = 0 TO 7
1150   T(L) = PEEK (49143 + L): REM PULL VALUES OF TIME DIGITS FROM MEMORY
1160   NEXTL
1170   REM T(0) CONTAINS S/100 VALUE, T(1) CONTAINS S/10 VALUE, ETC.
1180   RETURN
```

Table 4: A BASIC program to read the current time from the clock. The program assumes the clock has already been reset and may have been started. The start/stop line is toggled in lines 1110 and 1120 by outputting to the clock by way of the port located at 50954_{10}. The machine language routine (Table 3) is accessed at 1130 and retrieves the values from memory and stores them in an array T(L). From this array one can readily form the time in hours, minutes and seconds. A test can be added to change dates and months.

Procedure And Software

In order to use this real time clock, the reset line is brought momentarily low; the first negative slope on the Start/Stop (abbreviated S/S) line starts the time. If the S/S line is brought high and then low, the "display" then freezes while the clock chip continues timing. The computer can read this time by a machine language program (**Table 3**; the multiplexed output from the clock is too fast for most BASICs.) Each time the S/S line is brought high and then low, the "display" is updated to the current time. Thus, in use, the computer toggles the S/S line and goes to the machine language subroutine that reads the time digits. The computer does this only when it needs to know the current time.

In our application, the computer utilizing this real time clock does data acquisition and control using the BASIC language. BASIC can transfer to a machine language routine and back again. Our computer uses a 6502 μP so that the machine language subroutine in Table 3 is specific for this processor. The program has the processor read the parallel port (configured as memory location 50958_{10} or $C70E_{16}$) and decode the information to determine the value for each digit. This value is stored in a table in memory where the 0.01 sec digit's value is located in 49143_{10} ($BFF7_{16}$), the 0.1 sec digit's value is in location 49144_{10} ($BFF8_{16}$) and so on. A flow chart of the program is shown in **Figure 3** and will allow a corresponding routine to be written for other μPs. The portion of the BASIC program that calls the machine language routine is shown in **Table 4.**

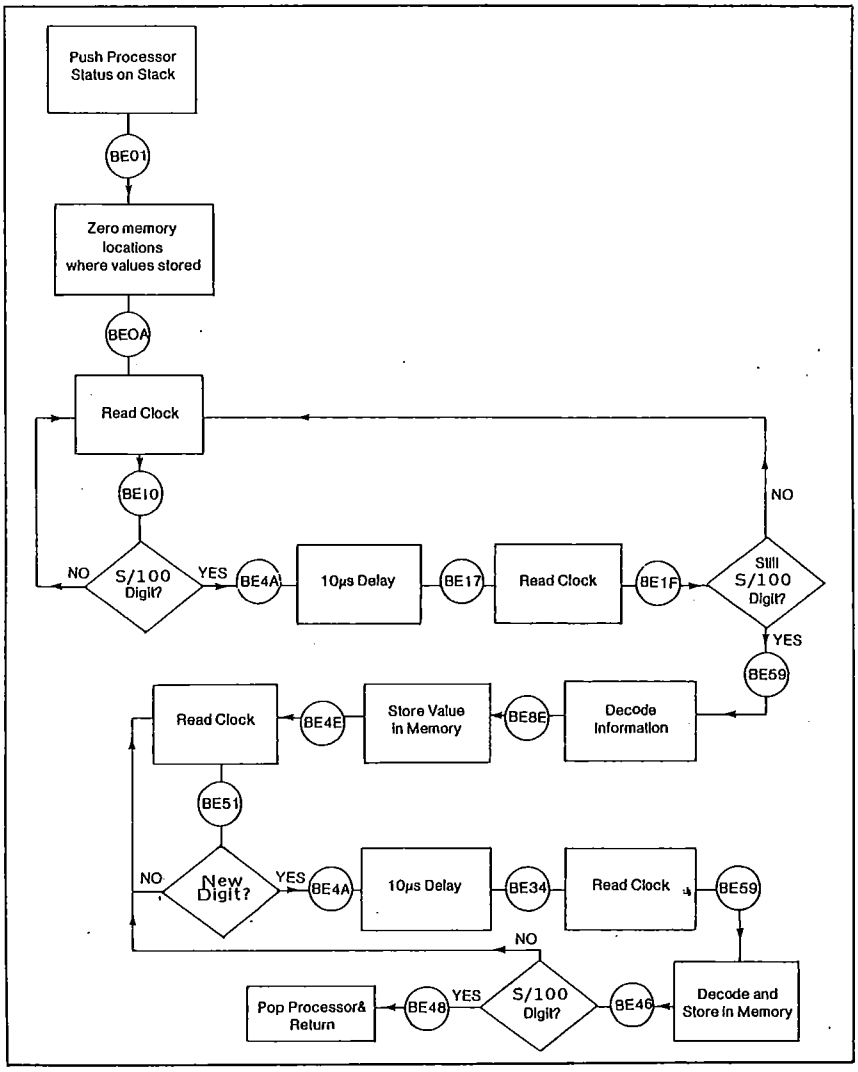

Figure 3: Flow Chart of the machine language subroutine that reads the parallel input port connected to the real time clock. The routine decodes the information in the 8 bit word and determines the value of the digit and stores that in a corresponding memory location. The Hex memory locations of the instructions in Table 3 are shown adjacent to each box. The first loop finds the .01s (S/100) digit (the 10μs delay insures stable voltage levels) while the second loop waits for the next digit, and decodes and stores it. This continues until the S/100 digit appears again. The time to read and store all eight digits can vary from 0.8ms to 1.5ms depending on which digit is being output by the clock when this subroutine begins.

The parameter T(L) contains the time with T(0) having the value of the S/100 digit and T(7) having the value of the 10 hour digit.

The real time clock has been working very well for the last six months. It makes efficient use of the processor's time, yet allows high resolution in timing. Successive timings as short as 180ms can be done using BASIC while timings of 0.01sec (resolution of the clock) are possible using only machine language. The six chips needed to construct this clock make it inexpensive and easy to construct. The interface to the computer is straightforward and allows μCs to access the current time for calculations or recording events.

Finally, I'd like to express my appreciation to Research Corporation for funding the research which led, in part, to this design.

D.T. Jacobs, Dept. of Physics, The College of Wooster, Wooster, OH 44696.