MICROCONTROLLER-BASED SCHOOL TIMER

U.B. MUJUMDAR

The basic requirements of a real-time programmable timer generally used in schools and colleges for sounding the bell on time are:

- Precise time base for time keeping.
- Read/write memory for storing the bell timings.
- LCD or LED display for displaying real time as well as other data to make the instrument user-friendly.
- Keys for data entry.
- Electromechanical relay to operate the bell.

We are describing here a sophisticated, yet economical, school timer based on Motorola's 20-pin MC68HC705J1A microcontroller.

Description

The pin assignments and main features of the microcontroller are shown in Fig.1 and the Box, respectively. The complete system is divided into four sections, namely, the time keeping section, the input section (keyboard), the output (display, indicators, and relay driving) section, and power supply and battery backup.

The time-keeping section. Accurate time-keeping depends on the accuracy of time base used for driving the microcontroller. In this project, the microcontroller is driven by AT-cut parallel resonant crystal oscillator that is expected to provide a very stable clock. A 3.2768MHz crystal provides a time base to the controller. The frequency (f_{osc}) of the oscillator is internally divided by 2 to get the operating frequency (f_{op}). This high-frequency clock source is used to control the sequencing of CPU instructions.

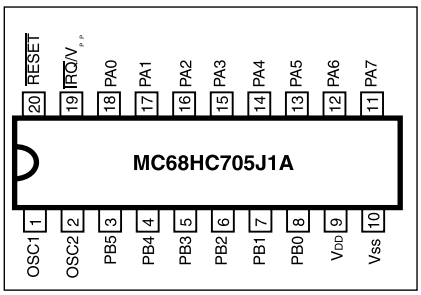

Fig. 1: MC68HC705J1A pin assignment

This provides the timing reference for timer functions.

The programmable timer status and control register (TSCR) is used for deciding the interrupt rate. It can be programmed to give interrupts after every 16,384, 3,2768, 65,536, or 131,072 clock cycles. In Table I, the control word is set to provide the interrupts after every 16,384 cycles. For a 32,768MHz crystal, the interrupt period will be 10 ms. Thus, timer interrupts will be generated after every 10 ms (100 Hz). That is, 100 interrupts will make 1 second.

Now time-keeping becomes very simple. As we are having a precise 1-second time count, a real-time clock can be easily built.

The MC68HC705J1A has a 64 byte RAM that is used for data storage. Real time (in terms of seconds, minutes,

Main features of MC68H705JIA

- 14 bidirectional input/output (I/O) lines. (All the bidirectional port pins are programmable as inputs or outputs.)
- 10mA sink capability on four I/O pins (PA0-PA3).
- 1,240 bytes of OTPROM, including eight bytes for user vectors.
- 64 bytes of user RAM.
- Memory-mapped I/O registers.
- Fully static operation with no minimum clock speed.
- Power-saving stop, halt, wait, and data-retention modes.
- Illegal address reset.
- A wide supply voltage range from - 0.3 to 7 volts.
- Up to 4.0MHz internal operating frequency at 5 volts.
- 15-stage multifunction timer, consisting of an 8-bit timer with 7-bit pre-scaler.
- On-chip oscillator connections for crystal, ceramic resonator, and external clock.

PARTS LIST

Semiconductors:

IC1	- 68HC705JIACP microcontroller
IC2	- CD4532 8-bit priority encoder
IC3	- 74LS138 3-line to 8-line decoder
IC4	- 74LS47 BCD-to-7-segment decoder/driver
T1-T3	- BC547/BC147 npn transistor
T4-T7	- 2N2907 pnp transistor
D1- D7	- 1N4007 diode
ZD1	- 5.6V, 0.5watt zener

Resistors (¼-watt, ±5% carbon, unless stated otherwise)

R1	- 210-ohm, 0.5 watt
R2	- 27-ohm
R3, R12-R14, R24-R27	- 1-kilo-ohm
R4-R8	- 100-kilo-ohm
R9 -R11, R23,R29	- 10-kilo-ohm
R15-R22	- 47-ohm
R28	- 10-mega-ohm

Capacitors:

C1	- 350µF, 25V electrolytic
C2, C3	- 1µF, 16V electrolytic
C4, C5	- 27pF ceramic disk
C6	- 0.1µF ceramic disk

Miscellaneous:

S1-S5	- Push-to-on switch (key)
S6	- On/off switch
PZ1	- Piezo buzzer
RL1	- Relay 12V, 300-ohm, 1C/O
XTAL	- 3.2768MHz AT-cut crystal
X1	- 230V AC primary to 12V-0-12V, 500mA secondary transformer
DIS.1-DIS.4	- LTS542 common-anode display
	- 4 x 1.2V Ni-Cd cells

Timer. The basic function of a timer is the measurement or generation of time-dependant events. Timers usually measure time relative to the internal clock of the microcontroller. The MC68HC705J1A has a 15-stage ripple counter preceeded by a pre-scaler that divides the internal clock signal by 4.

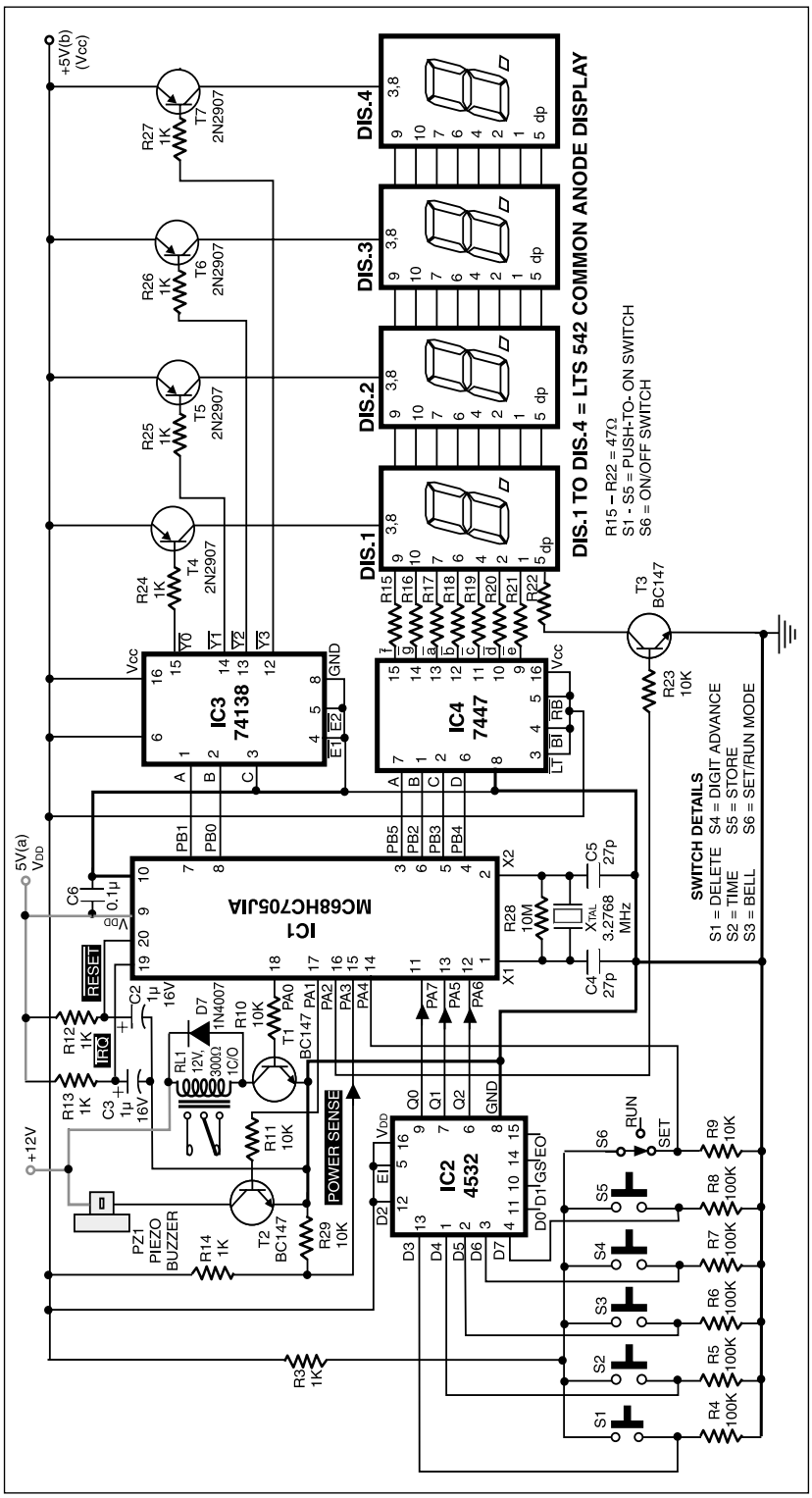

Fig. 2: Schematic diagram of the microcontroller-based school timer

hours, days of a month, and months) is stored in this RAM. Thus an accurate real-time clock is generated.

The input section. For setting the real-time clock and storing operating times, the timer requires to be programmed externally. Data is fed using the keyboard.

Press-to-on type keys are interfaced to the microcontroller using an 8-bit priority encoder CD4532. This encoder detects the key-press operation and generates the equivalent 3-bit binary data. Its truth table is shown in Table II. The priority encoder is interfaced to port A of the microcontroller.

Various keys used in the timer, along with their functions, are described below:

Time (4): For setting real time in minutes and hours.

Bell (5): For setting the bell's operating timings.

Digit Advance (6): Data setting is done digitwise (hour's digit followed by minute's digit). The Digit Advance key shifts the decimal point to the right.

Store (7): For storing the data (real time or bell time).

Delete (3): For deleting a particular bell timing.

Here, the figures within parentheses indicate the decimal equivalents of 3-bit binary data from the keyboard.

Set and run modes. Data setting is possible only in set mode. Set mode or run mode can be selected by toggle switch S6. By using a lock switch for S6, the timer can be protected from unauthorised data entry/storage.

In run mode if you press 'Bell' key once, the display shows the bell's various operating timings one after the other, in the same order in which these had been previously stored. In case you want to discontinue seeing all the bell timings, you may press 'Time' key at any stage to revert back to the display of real time.

The output section. Seven-seg-

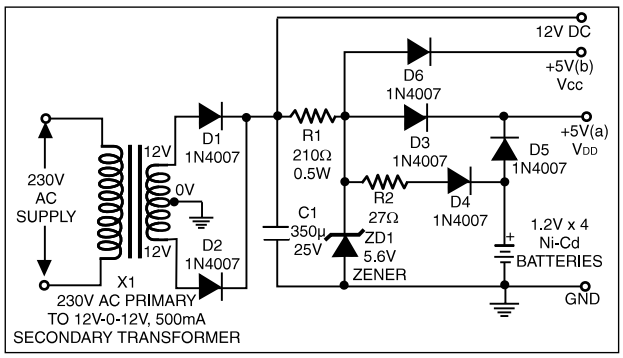

Fig. 3: Power supply circuit for the school timer

ment displays are used for data display. As LEDs are brighter, these have been used in the system. There are two techniques for driving the displays: (i) driving each display using a separate driver (like 74LS47 or CD4511) and (ii) using multiplexed displays.

The first technique works well, but practically it has two problems: it uses a large number of IC packages and consumes a fairly large amount of current. By using multiplexed display both the problems can be solved. In multiplexing, only one input is displayed at any given instant. But if you chop or alter inputs fast enough, your eyes see the result as a continuous display. With LEDs, only one digit is lighted up at a time. This saves a lot of power and also components, making the system economical.

Generally, displays are refreshed at a frequency of 50 to 150 Hz. Here, displays are refreshed at a frequency of 100 Hz (after every 10 ms). The display-refreshing program is an interrupt service routine program. BCD-to-7-segment decoder/driver 74LS47, along with transistor 2N2907, and 3-line-to-8-line decoder 74LS138 are used for driving common-anode displays.

In multiplexed display, the current through the segments is doubled to increase the display's brightness. 74LS47 is rated for sinking a current of up to 24 mA. As the current persists for a very small time in multiplexed display, it is peaky and can be as high as 40 mA per segment.

The decimal point is controlled individually by transistor BC547, as 74LS47 does not support the decimal point. PA0 and PA1 bits of port A are used for controlling the electromechanical relay and buzzer, respectively.

Power supply and battery backup. The microcontroller and the associated IC packages require a 5V DC supply, while the relay and the buzzer require 12V DC supply. A simple rectifier along with zener diode-regulated power supply is used. The microcontroller is fed through a battery-backed power supply, so that in the case of power failure the functioning of the controller's timer section is not affected. During power failure the timer is taken to 'low power' mode (called 'wait' mode). In this mode the controller draws a very small current. So small Ni-Cd batteries can provide a good backup.

A simple diode-resistance (27-ohm, 1/4-watt) charger maintains the charge of the battery at proper charging rate.

Software

Motorola offers Integrated Development Environment (IDE) software for programming its microcontroller and complete development of the system. The development board comes with Editor, Assembler, and Programmer software to support Motorola's device programmer and software simulator. The ICS05JW in-circuit simulator along with development board (pod) forms a complete simulator and non-real-time I/O emulator for simulating, programming, and debugging code for a MC68HC705J1A/KJ1 family device.

When you connect the pod to your host computer and target hardware, you can use the actual inputs and outputs of the target system during simulation of the code. You can also use the ISC05JW software to edit and assemble the code in standalone mode, without input/output to/from pod. The pod (MC68HC705J1CS) can be interfaced to any Windows 3.x- or Windows 95-based IBM computer using serial port.

The software for the timer has been so developed that the system becomes as user-friendly as possible. The main constraint is read/write memory (RAM)

TABLE I

Timer Status and Control Register (TSCR)

Bit	7	6	5	4	3	2	1	0
Signal	TOF	RTIF	TOIE	RTIE	TOFR	RTIFR	RTI	RTO
Reset	0	0	0	0	0	0	1	1

TOF: Timer overflow flag
RTIF: Real-time interrupt flag
RTIE: Real-time interrupt enable
RTI and RTO: Real-time interrupt select bit.

RTI	RTO	Interrupt period
0	0	fop ÷ 2^{14}
0	1	fop ÷ 2^{15}
1	0	fop ÷ 2^{16}
1	1	fop ÷ 2^{17}

For 3.2768MHz crystal
Frequency of operation (fop)
= $3.2768 \times 10^6/2 = 1.638 \times 10^6$MHz
For RTI=RTO=0
Interrupt period = 10ms (100Hz)

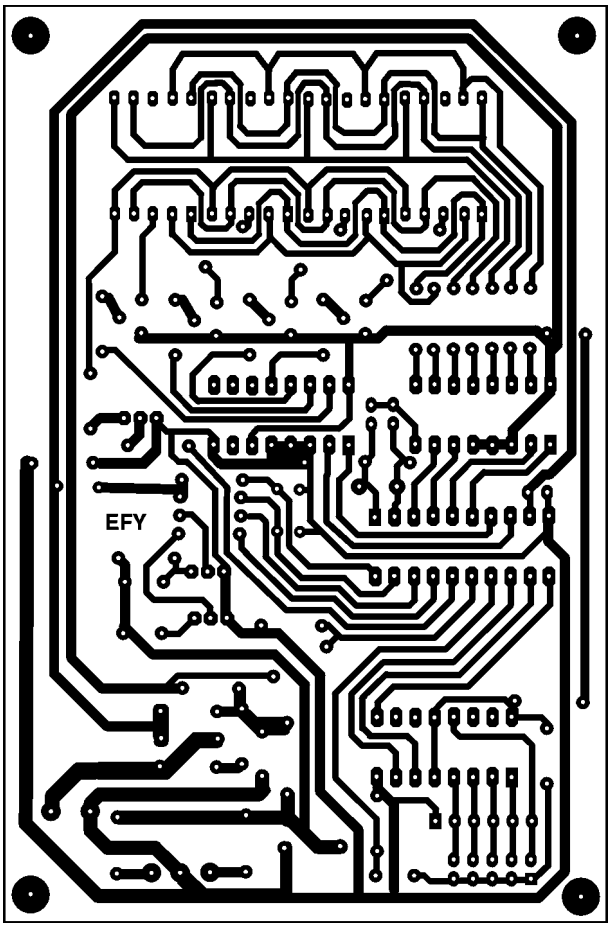

Fig. 4: Actual-size single-sided PCB for the circuits in Figs 1 and 2

space. As mentioned earlier, the microcontroller has only 64 byte RAM. About twenty bell operating timings are required to be stored. So the efficient use of RAM becomes essential.

The software routines for the timer, along with their Assembly language codes, are listed in a folder. (Note: This folder, containing source code (.asm) and listing file (.lst) will form part of the EFY-CD provided with the August issue. As files are quite large, it is not feasible to include them here.) Basically, the following functions are performed by the software program:

1. Initialisation of ports and the timer.
2. Reading of key-pressed data.
3. Storing of real time and bell timings.
4. Comparison of real time and bell time. If the two match, the bell rings.
5. Display of data.
6. Time-keeping.

For a user-friendly system, the associated software is required to perform many data manipulation tricks and internal branching. The operation and logic can be understood from the Assembly language listings. The software is mainly divided into the following modules:

Keyboard. When a key is pressed, CD4532 sends the corresponding data. After reading the data, the controller decides on the action. 'Set/ Run' key (S6) is connected to port PA4.

Bell. This part of the program is used for displaying the bell operating timings stored in the RAM. The operating timings are displayed one by one with a delay of 5 seconds between two consecutive timings.

Set. The real time and bell timings are stored using this part of the software. Data is entered digitwise; for example, 08:30 a.m. will be stored as 0, followed by 8, followed by 3, and finally 0. Data is stored in 24-hour format.

Data fed from the keyboard is converted into equivalent hex and stored in RAM. Any particular operating timing can be deleted from the memory using 'Delete' key, provided the timing is already stored in the memory.

Run. Here the real time is compared with bell operating time. If the two match, the relay is operated.

DataCon. This part of the software is used for finding out the decimal equivalent of hex data. The microcontroller manipulates the hex data and converts it into BCD format for display.

Timer. The timer of the microcontroller is initialised to give an interrupt after every 10 ms. A real-time clock is generated using the interrupt. Also the display is refreshed during the interrupt service routine.

For real-time systems battery backup is very essential, because power failure affects the time keeping. In interrupt service routine, the availability of power supply is checked. If the power is available, displays are refreshed and the timer operates normally. However, during the power-failure period, displays are off and system is taken to 'low power' mode. In this mode only the timer part of the microcontroller remains activated while operations of all other peripherals are suspended.

This considerably reduces the power consumption. When the supply gets restored, the controller starts operating in normal fashion.

Operating procedure

When the power is switched on, the display shows 12.00. Two settings are required in the timer: (a) setting of real time and (b) setting of bell operating timings. For setting real-time clock 'Time' key is used, while for setting bell timings 'Bell' key is used.

Storing of real time. To store real time, say, 05:35 p.m., flip 'Run'/'Set' key (S6) to set mode. The display will show '0.000'. Press 'Time' key. Further pressing of 'Time' key will increment the data, like 0.000, 1.000, 2.000, and thereafter it will repeat 0.000, etc. To select the digit, press 'Digit Advance'. This stores the present digit and the next digit is selected as indicated by the decimal pointer. Data is stored in 24-hour format. The time to be stored is 17.35, of which the first digit will be 1.000. The second, third, and fourth digits can be stored in similar fashion. After the fourth digit, press 'Digit Advance' key once more. The display will show 1735 (with no decimal). Now press 'Store' to store the data.

TABLE II
Truth Table for Priority Encoder CD4532

Keys	E1	D7	D6	D5	D4	D3	D2	D1	D0	Q2	Q1	Q0
Store	1	1	X	X	X	X	X	X	X	1	1	1
Digit Adv.	1	0	1	X	X	X	X	X	X	1	1	0
Bell	1	0	0	1	X	X	X	X	X	1	0	1
Time	1	0	0	0	1	X	X	X	X	1	0	0
Delete	1	0	0	0	0	1	X	X	X	0	1	1

Storing of bell timings. The procedure to store bell operating timings is similar to that of setting real time. The only difference is that here data is changed by 'Bell' key in place of 'Time' key. Any number of bell timings (<20) can be stored in the same fashion. If the number of bell operating timings exceeds 20, the timer will not accept any new bell timing until one of the previously stored timings is deleted.

Deletion of bell operating timings. For deleting a particular timing, first store this timing using the steps given above. Then press 'Delete' key to delete the specific data from the memory.

Display of real time. If 'Run'/'Set' key is taken to run mode, real time will be displayed.

Checking of bell operating times. For checking the bell operating times, press bell key in 'Run' mode only. The stored bell operating timings will be displayed one by one with a delay of 5 seconds between two consecutive timings.

Programming

There are two ways to program the EPROM/OTPROM (one-time programmable ROM):

1. Manipulate the control bits in the EPROM programming register to program the EPROM/OTPROM on a byte-by-byte basis.

2. Program the EPROM/OTPROM with Motorola's MC68HC705J in-circuit simulator.

The author has used the second method for programming the OTPROM. (***EFY note.*** Readers who wish to acquire a Pod for 705KJ1/J1A microcontrollers, along with the required software, may contact Vinay Chaddha at gvc@vsnl.com.)

An actual-size, single-sided PCB for the circuits in Figs 2 and 3 is shown in Fig. 4, with its component layout shown in Fig. 5.

❑

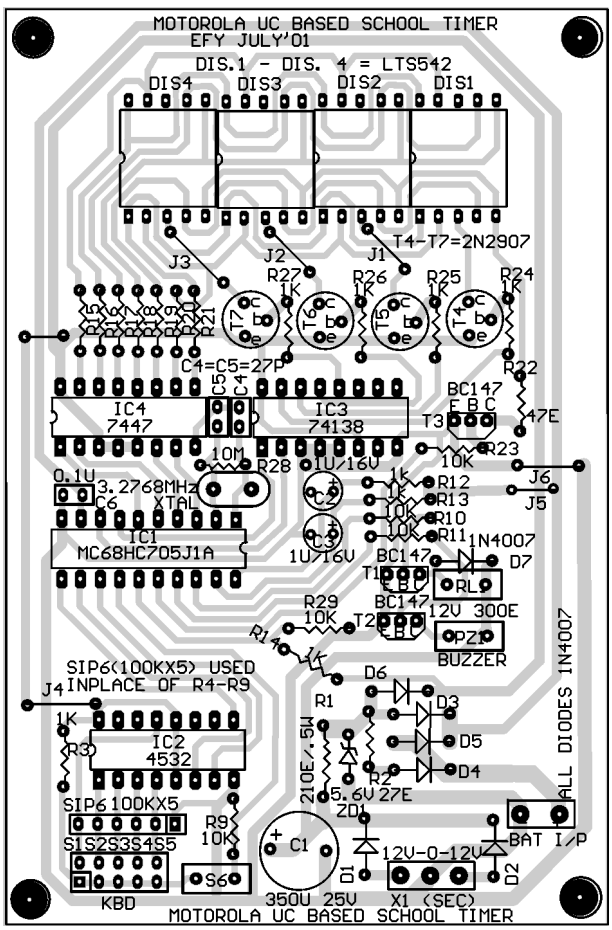

Fig. 5: Component layout for the PCB