

Build the "7-uP" Alarm-Clock/Time-Switch

... with 7-day alarm schedule, USB link to PC for time sync and programming, a variety of alarm sounds, user-settable options and a host of other advanced features.

What's wrong with your alarm clock?

This project all began out of frustration with my alarm-clock/radio, which despite being a big-name brand, and not cheap, is very awkward to use and is lacking most of the features I would prefer to have.

Whenever I raise the issue among friends and work colleagues, it seems that all alarm-clocks (and clock-radios) fall short of user expectations in some way or other. Here's a summary of complaints I've often heard from people about their alarm clocks...

- Forgetting to switch on the alarm before going to bed (because it had to be switched off in the morning to silence it, of course).
- Clock-radio not tuned into a station properly, or the volume was turned way down low, so the alarm could not be heard when it activated.
- Tedious ritual to change the clock time or alarm time, typically using one button to set "hours" and another for "minutes". Holding down a button (in some cases two) causes the minutes or hours displayed to change automatically, but in most cases either too slow or too fast.
- Alarm time is the same every day, whether it's a weekday, Saturday or Sunday. (Most people need to wake up earlier on weekdays than on weekends and some people work part-time or shifts varying each day.)
- The "snooze" time interval is fixed (typically 7 to 9 minutes) and can't be adjusted.
- It's a pain having to press the snooze button to silence the alarm. It would be great to have the option for the alarm to sound for a few seconds, then go silent for a few minutes, and repeat the process until canceled somehow.
- The display is too bright at night or too dim in daylight.
- There's a switch to adjust the display brightness, but it's awkward to use. Why can't the clock adjust its brightness automatically to suit the ambient light level?
- I don't like waking up to the radio in the morning, but isn't there an alarm-clock that makes a more interesting sound? (Or, better still, a choice of different sounds.)
- Toddlers like to play with the buttons on the alarm-clock. Often the time or alarm settings get changed accidentally.
- The snooze button is too small, or badly located, or otherwise difficult to operate in the dark.
- It doesn't make the coffee.

How many of the foregoing complaints apply to your alarm-clock?

A trip to the local electronics store failed to find an alarm-clock meeting enough of the requirements on my wish list. So, being an Electronics Engineer by profession, I decided to design and build my "ideal alarm-clock".

An improved digital alarm-clock design

Following is a concept for a digital alarm-clock meeting my objectives...

My main focus was to combine enhanced functionality with convenience (ease of use), while attempting to address many of the deficiencies of commercial products. The design would not be compromised to compete on a cost basis with consumer mass-market products. Do-it-yourself construction cannot compete with mass-production, but there is great satisfaction derived from making something better.

An intuitive "local user interface" (control panel) comprising display, a number of push-buttons and a rotary encoder switch (knob) should be provided for operation of the clock

while not connected to a computer. The local controls would be designed primarily for alarm time setting and for selection of displayed information, rather than for advanced configuration. Programming of the 7-day time-switch ON/OFF schedule, if required, would be more conveniently done via a USB PC link using a Windows software application.

The front panel should have a large 4-digit display, normally showing the time-of-day, plus several dedicated back-lit enunciators, e.g. PM, 24hr, A1..A4, S1..S4, MON, TUE, WED, etc. The "alarm" enunciators (A1, A2, A3, A4) show the ON/OFF status of up to four daily alarm events. Another set of four enunciators (S1, S2, S3, S4) indicates the ON/OFF status of the time-switch outputs.

Display brightness must be variable, and preferably controllable by a choice of user-configurable options, e.g. ambient light sensor ("automatic"), or alarm event ("power-save mode"), or manually using the control panel.

The alarm function should have a variety of configurable options, e.g. choice of alarm sounds, initial volume, maximum volume, volume auto-increasing ("ramp-up" option), number of repeat alerts, interval between repeat alerts, etc, etc.

There should be a good selection of preset (built-in) alarm sounds, which can be customized by the user. A different alarm sound could be programmed for each alarm "event" occurring thru'out the 7-day alarm schedule, if the user so desired. Ideally, alarm sounds would be digital audio clips (PCM/MP3 encoded), downloadable via the USB link, giving wide-range low-distortion audio quality. (This could be an optional extra, as the added cost might not be justifiable to many hobbyists.)

The "snooze" function might have two modes of operation. In "classic" snooze mode, the alarm sound is silenced temporarily by a single push of the button, but is reactivated after a preset time interval (e.g. 10 minutes). Alarm activation repeats until canceled (e.g. by pressing the snooze button 3 times rapidly) or until a preset timeout expires.

In the alternative "auto-repeat" snooze mode, however, alarm activation would recur automatically at preset intervals (e.g. 10 minutes), for a number of times, without operator intervention, before canceling itself. A single press of the snooze button would cancel repeat activations, regardless.

In either snooze mode, it would not be necessary to switch off the alarm every morning and re-enable it again before going to bed every night!

In normal operating mode, the knob would be a manual brightness control for the display. In time-setting and configuration modes, however, the knob could well be used for convenient adjustment of displayed data.

A means to confirm the next pending alarm event, up to 48 hours in advance, must be provided. Perhaps pressing the snooze/cancel button (while there is no alarm active) could enter an "alarm confirmation" mode in which the time and day of the next pending alarm is displayed. And if the button was held down for longer than (say) two seconds, the sound assigned to the pending alarm event would be played to check the loudness level. Taking this concept a step further, the knob could be used to adjust the loudness, while a pair of buttons might be used to change the sound effect.

For technology enthusiasts, the clock should connect to a PC (via USB) and support a variety of accessories via rear-panel I/O connectors. These provide logic outputs which can be used to switch appliances on and off at preset times, and/or to switch on when the alarm activates. Examples of accessories are: alarm bell/chime, bed-side lamp, radio, TV, kettle, coffee-maker, computer, PC peripherals, etc. Each of the time-switch "channels" should be settable independently and may have ON/OFF times set differently for each day of the week.

A backup battery must be provided to maintain the clock time and date in the event of temporary power failure. When the battery voltage drops below the minimum usable voltage, a warning should be indicated.

All user-settable options, parameters, on/off/alarm weekly time schedule, etc, must be preserved in non-volatile memory. User configurations may be transferred to and from a PC using a Windows utility. While connected to a host PC, the clock should automatically

synchronize its time and date to the PC's internal real-time clock, which in turn may be synchronized to Internet time.

In addition to supporting a Windows "graphical user interface" (GUI) to configure clock options, and for programming the alarm schedule, etc, the USB port must allow downloading of firmware upgrades. This capability is essential to support feature enhancements and bug fixes.

Features of the "7-uP" Alarm-clock/Time-switch

Convenience and ease of use

- Intuitive local controls and setup menu
- Knob (rotary encoder switch) for easy time setting and data entry
- One-touch confirmation of pending alarm time and loudness

Multiple alarm "events" and weekly (7-day) schedule

- Program up to four alarm "events" in a 24-hour day
- Different alarm times each day of the week, if desired

Versatile snooze/cancel options

- "Classic mode" — alarm sounds continuously until deferred by button hit
- "Auto-repeat mode" — alarm recurs at periodic intervals, until canceled
- Adjustable snooze time interval (in either mode)

Variable brightness display, with choice of control modes, i.e.

- Ambient light sensor
- Alarm or timer activated
- Manual override — adjust brightness using knob

High quality, programmable alarm sounds

- Choose from a variety of "interesting" preset alarm sounds
- Customize the preset sounds or create your own sounds
- Assign a different sound to each alarm "event"

USB link to host PC #

- Synchronize the clock to PC system time and date (i.e. Internet time)
- Save and restore option settings, alarm event and time-switch schedules ^
- Download firmware upgrades (feature enhancements and bug fixes)

Time-switch and Alarm-switch accessory connections

- Four independent time-switch channels with 7-day ON/OFF schedule
- Control AC appliances (lamps, radio, TV/AV, computer & peripherals)*
- Alarm-activated output for controlling external wake-up devices
- Auxiliary (countdown) timer with accessory control output
- Switch PC peripherals on automatically when USB bus power is detected

* Requires optional 4-outlet solid-state relay board for controlling AC mains powered equipment.

PC link uses the USB-CDC "Virtual COM port" API, also accessible using HyperTerminal.

^ Facilitated by Windows GUI application software, to be offered by 3rd-party developers.

Technical Introduction

Essential hardware consists of a main board and display board, plus encoder switch (knob), speaker and snooze/cancel button. Surface-mount (SMT) components have been eliminated from the 2 boards comprising the basic hardware, to make assembly easier for hobbyists with lesser soldering skill and/or limited tool kit. The microcontroller chip (MCU) is socketted (52-pin PLCC) for the same reason.

The main board incorporates a simple, low-cost sound generator. Audio level (attack/decay amplitude envelope and overall volume) is controlled digitally by the MCU using pulse-width modulation (PWM). With the help of some tricky firmware, which can add frequency-modulation and/or amplitude-modulation effects, this arrangement can produce a good variety of alarm sounds ranging from quite pleasant to unbearably horrible (to suit all tastes). A selection of 16 pre-defined sounds is provided with the generic firmware. Alternative sounds may be customized by the user, by modifying the "sound shape" parameters stored in EEPROM.

The parts may be housed in a ready-made enclosure, or in a unique housing of your own design and construction. The "standard" enclosure is PAC-TEC model CM6-225, but a Chinese-made replica is available at a lower price. The circuit boards were designed to fit in this box. (*See Assembly Instructions in download package.*) Although the seven "control surface" push-buttons are normally fitted to the top of the display board, so that they are accessible along the front edge of the top panel, they may be located elsewhere, e.g. on a piece of prototyping board, wired back to the display board. Such an arrangement would allow creative variations from the "standard" enclosure design.

Hardware expansion is possible via an optional "mezzanine" board which connects to the main board via a ribbon-cable connector. This makes it possible to extend the functionality of the clock, or to use the hardware platform for a completely different purpose. A mezzanine board to generate hi-fidelity PCM/MP3 digital audio alarm sounds may be developed by the author in the future. The board will include an MP3 decoder chip and serial data-flash memory to hold alarm "sound clips".

The firmware is based on the author's "ALERT" (Low-End Real-Time) operating system. Project builders may choose to write (and share) their own firmware, or to download a generic version free-of-charge. The source code is available for readers who may wish to modify or extend the firmware for their own purposes.

The hardware is based on Atmel's AT89C5131 USB microcontroller. This MCU was chosen because it is available in a PLCC package (socketable), it has a generous amount of on-chip flash program memory (32kB) with separate data EEPROM (1kB), it has an on-chip USB peripheral module, it is popular and hence readily available at low cost (US\$8.60 from Digi-Key). Most importantly, the 'C5131 has a bootloader for flash programming via the USB port, requiring no additional programming device.

The 'C5131's only bad trait is that it has an 8051 core, which is (in the author's opinion) one of the worst MCU architectures ever devised! Not that it really matters if you're programming in C and there's enough flash PROM to compensate for the inefficient instruction set. (To be fair, the 8051 instruction set is quite efficient when working with the core "register file", i.e. RAM below 0xFF, but horribly inefficient working with "extended" data RAM.)

For DIY firmware developers, there is a free C compiler available for 8051 code development. (Google with "SDCC 8051 compiler".) Overall, I think the 89C5131 was a good choice for a DIY project, especially for hobbyists not skilled in SMD soldering. For a commercial product of similar design, I would choose a different MCU, e.g. Freescale MC9S08-JM60, Microchip PIC18F66J50, or maybe even a 32-bit ARM core MCU (since the cost is comparable with many 8-biters).

All components are readily available from the major online suppliers such as Farnell, Digi-Key, Mouser, etc. Most parts should also be stocked by your local hobby electronics store (probably at more inflated prices). Beware of high delivery fees when ordering online from overseas suppliers.

General operation

POWER SUPPLY REQUIREMENTS

The clock is designed to be powered from a 9V DC plug pack. The clock itself draws less than 100mA (with the display set to normal brightness). A switch-mode regulated plug-pack is recommended to minimize AC power consumption. The clock has a switch-mode 5V regulator IC for optimum efficiency.

The clock may also be powered from the USB port ($V_{bus} = 5V_{dc}$). The external DC supply (+VEX), normally available at the accessory sockets, comes directly from the 9V power supply input, so it is not available when running on USB power. Consequently, accessories which rely on the 9V DC supply will obviously not function. The backup battery keeps the clock running while disconnected from both USB and the 9V plug-pack.

BACKUP BATTERY

The clock is not intended to be battery powered in normal operation. The battery is intended only to prevent loss of time and date during the occasional brief AC mains power failure; also to allow the clock to be disconnected momentarily while being moved from one location to another, e.g. for connection to a host PC. While running solely from the backup battery, only the bare minimum clock functions are kept alive to maintain the time and date.

The backup battery comprises 3 'AA' size cells, which may be 1.5V alkaline types or 1.2V rechargeable types. The micro-controller runs on any battery voltage in the range 3.3V to 5V. Battery voltage is monitored continuously by the micro-controller when running on the main 9V DC supply. A switch is provided to disconnect the backup battery in case the clock may be powered down for a prolonged time.

ACCESSORY DC SUPPLY

The Alarm and Time-Switch Accessory sockets provide a protected DC supply, fed from the 9V DC inlet (plug-pack). The external DC supply is current limited to about 0.5A by a resettable fuse. The alarm-switched control output is capable of sourcing up to 0.5A to a DC-powered accessory (e.g. bell, lamp, radio, etc).

FIRMWARE DOWNLOAD

Firmware is installed into the device's Flash program memory via the USB link using Atmel's "FLIP" (Flexible In-system Programmer) PC utility. The latest version of FLIP software and user documentation can be downloaded from Atmel's website.

To start the device's USB bootloader, press and hold the "ISP" (In-System Programming) button, then press and release the reset button, then release the "ISP" button. The display should be blank. Plug in the USB cable to your PC. Start the FLIP application. Follow FLIP instructions from Atmel.

HOST COMMAND INTERFACE (HCI)

The firmware incorporates a communications protocol intended primarily for passing data to and from a host PC via the USB link. The protocol is based on a simple command-line user interface format, hence the term "host command interface" (HCI). Since the protocol uses printable ASCII characters, the HCI can be accessed using a terminal emulator (such as 'HyperTerminal') for human interaction. The HCI can be used interactively for setting clock options, parameters, etc. Advanced configuration options, not accessible via the "local user interface" (control panel), can be set using the HCI with HyperTerminal, or a custom Windows GUI application (TBD).

The device USB port appears to the host PC as a "virtual COM port". Windows application software can communicate with the clock through the standard Windows "COM port API" function calls (open, read, write, etc).

A command string is composed of a 2-letter command "mnemonic" and a number of user-supplied "arguments" (parameters). Some commands have no arguments. A single space must be inserted between command line arguments, where there is more than one (including the 2-letter command name). HCI input is generally not case-sensitive. There is no provision for command-line editing, but Escape or Ctrl-X will cancel the command line.

A brief command summary is given by the help command, "HE". Debug commands are provided to assist programmers developing their own firmware code, and for those interested in the internal workings of the micro-controller and firmware.

For further details on the HCI, refer to the User Guide.

Control panel operation - a brief overview

Figure 1 shows the standard layout of push-buttons along the top panel of the clock. The SNOOZE/CANCEL button is also located on the top panel. The buttons, LED display and encoder switch (knob) comprise the "local user interface". Most clock operations and settings are accessible via the local user interface. Some of the more esoteric user options are adjustable only via the USB link.

Figure 1. Top Panel Button Legend <Button_legend_pic.JPG>

Unlike most alarm clocks, the display on this one shows the day-of-the-week. When setting an alarm, the user can choose which day(s) of the week the alarm will be activated. Other LED enunciators show the four alarm events' on/off status (A1..A4) and the four time-switch outputs' on/off status (S1..S4). The control panel allows manual over-ride of the time-switch outputs.

Figure 2. Front Panel Display Legend <Display_Legend_pic.JPG>

Most of the push-buttons have more than one function. The function performed by a button press is dependent on the context of the operation. For example, in normal time-of-day display mode, pressing the [+] or [-] button will select a different item for display, e.g. seconds, year, date (month & day), etc. In a setting mode, however, the [+] and [-] buttons may be used to increment or decrement a numeric value, or to scroll through a list of items, e.g. days of the week.

In general, when in a setting mode, a flashing digit or LED enunciator indicates an item that can be changed by a button press or by turning the knob. The EXIT button quits a setting operation without making any changes.

The table below presents a summary of "local user interface" (control panel) operations. The left column contains operations that can be performed from normal time-of-day display mode.

Button press or other input	Resulting action
TIMER/CLOCK button hit	Countdown Timer (initial or remaining time, mm.ss) is displayed. From here, the Countdown Timer can be set and started.
ALARM CHECK button hit	Alarm Check mode is entered. The scheduled time of a selected alarm event (A1..A4) is shown. From here, the selected alarm (time and day-of-the-week) may be re-programmed, or enabled or disabled. The [+] and [-] buttons select the day-of-the-week for the alarm to be set. The ON/OFF button toggles the alarm status. Pressing the SNOOZE button while in Alarm Check mode allows the alarm sound effect and loudness to be changed.
TIME-SWITCH button hit	Time-Switch control mode is entered. A selected time-switch channel (S1..S4) may be switched on or off (temporarily overriding the programmed time-switch schedule).
SET button held for > 3 sec	Time-of-Day (TOD) setting mode is entered. The knob adjusts the hours, then if the SET button is pressed again, adjusts the minutes. Press SET again to commit to the change, or press EXIT to quit without any change.
MENU button held for > 3 sec	Menu Setup mode is entered. From here, various user options and parameters can be adjusted. The [+] and [-] buttons scroll thru a list of menu items.
[+] or [-] button hit	The displayed item is changed from normal TOD to seconds or the date (year, month, day). Pressing [+] scrolls thru items in the sequence: Seconds (mm:ss), Year (20xx), Date (MM dd), then back to time-of-day. Pressing [-] reverses the sequence. The SET button can be pressed to set any of the displayed items.
SNOOZE/CANCEL button pressed	Alarm confirmation: The time of the next pending (enabled) alarm event, up to 48 hours in advance, is displayed. If the button is held down (> 2 sec), the alarm sound plays. Pressing SET while holding down the SNOOZE button allows the alarm sound effect and loudness to be changed.
KNOB rotated (while normal time of day is displayed)	Display brightness is adjusted. Depending on selected options, the setting may be retained until next adjusted, or it may change again automatically.

For full details on operation, see the User Guide in the docs download package.

Circuit Description – Main board

The main board schematic gives the impression of a very simple industrial controller with minimal external I/O interface connections. A notable exception is the audio generator circuitry, which is more complex than you would expect on a general-purpose programmable controller, for example.

The choice of micro-controller chip has already been explained in the technical introduction (above). The At89c5131 is a derivative of the Intel 8051 family and thus has many 8051 core characteristics, including the inexplicably bizarre I/O port configurations. Port 0, for example, is open-drain and therefore requires external pull-ups on pins to function as outputs. Other ports have internal weak pull-ups ($\sim 25\text{k}\Omega$) which do not provide sufficient output drive capacity for most purposes.

Nevertheless, the 89c5131 has many redeeming features including a very fast processor core. With a 24MHz crystal and "X2" clocking mode, the MCU core runs at 48MHz! The device incorporates many extra peripheral modules, e.g. USB, Timer T2, PCA (used for PWM outputs), SPI, and TWI (IIC).

The At89c5131 provides a separate flash program memory block (4kB) for a bootloader. The device ships with a USB bootloader pre-programmed into this "boot block", allowing user program code to be loaded into the main flash program area (32kB) via the USB port, i.e. requiring no additional flash programming hardware. If the processor is reset while the PSEN# pin is pulled low (using the "ISP" button), the bootloader is started instead of the user application code.

The UART TX and RX signals are routed to the expansion header (P1), also to a 4-pin header (P5), in case the user wants to attach an "RS232" serial interface adapter. The UART signals are also routed to an RS422/485 transceiver (U3 = SN75176) to support external accessories via a serial bus. The initial release of clock firmware does not include any feature which uses the RS422/485 serial bus, so the transceiver (U3) and bus port connector (J3) may be omitted until such time as might be needed.

Circuitry is provided to monitor the external DC supply voltage (+VEX). If it drops below about 6V, transistor Q1 will turn off and the logic signal PWRON# will be in the High state due to a pull-up resistor in the MCU. The firmware monitors this signal and turns on an enunciator (PF) if the supply drops below minimum required voltage. The signal PWRON# is also routed to the 5V switch-mode regulator IC (U2 = LM2594) EN# (enable) pin. If the external 9V power supply is removed, the 5V regulator is shut down, allowing the +5V rail to be energized either by USB power or battery power.

The USB 'Vbus' line is sensed by the circuit of transistor Q2. If USB power is not present, Q2 turns off and signal VBUSD# will float High. The firmware reads this signal and adapts its operating mode accordingly. The state of the USB Vbus signal can also be used by the firmware to control one (or more) of the accessory control outputs.

Accessory Logic Inputs and Outputs

Six logic outputs are provided on the accessory sockets J4 and J5. The four time-switch outputs are active low (current sinking) control signals which can sink up to 50mA each. These are intended to drive opto-couplers (isolated inputs) in a solid-state AC power board. The alarm-switched 9V DC output is active high (current sourcing) and can source up to 500mA (limited by the resettable fuse F1). The corresponding MCU output port pins are buffered with a CMOS hex-inverter (U4) because the port pins cannot source enough current to drive the inputs of the peripheral driver IC (U5 = ULN2003A).

Two logic inputs are implemented by comparators U6C and U6D. The logic threshold is set to +1.6V. The external inputs are buffered and filtered by RC networks to protect the MCU input port pins from nasty external transients (e.g. ESD).

Analog Inputs

The AT89c5131 MCU does not provide analog (ADC) inputs. The clock application needs two analog inputs to monitor the ambient light sensor and backup battery voltage. Analog-to-digital conversion is implemented by a firmware algorithm, making use of a pulse-width modulation (PWM) output from the MCU to provide a variable reference

voltage. A low-pass filter (R26, C21) removes the AC component (47kHz) from the PWM pulse train, so that the DC level is controlled by the PWM duty.

The PWM reference voltage is fed into the (–) inputs of two comparators U6A & U6B. The analog input voltages to be measured are fed into the (+) inputs of U6A & U6B. By changing the reference voltage at periodic intervals, the firmware can determine the input voltages. Conversion accuracy is limited by the resolution (and linearity and noise) of the PWM-generated reference voltage. About 1% error is the best we could hope for. Due to the filter time-constant, it takes a few milliseconds for the reference voltage to stabilize after a change in PWM duty. Consequently, the conversion time is quite slow compared to a hardware ADC. Fortunately, in this application, speed is not an issue.

Audio Generator and Power Amplifier

A square-wave oscillator is realized with a CMOS timer IC (U7 = TLC555). The frequency of the timer audio output signal is controlled by switching the resistance and capacitance of the RC timing network. Eight different frequencies in the range 500Hz to 2kHz (approx.) are selectable by MCU logic output signals FREQ2, FREQ1 and FREQ0. A CMOS quad analog switch (U8 = CD4066B) is used to switch the resistors (R27, R29) and capacitor (C25) in or out of circuit. The chosen frequencies are harmonically related.

Audio amplitude is controlled by a PWM signal generated by the MCU (CEX1), fed to the control input of analog switch (U8D) which “chops” the square-wave audio signal. The duty cycle of the PWM signal determines the effective audio output level. The PWM frequency is about 47kHz; well out of range of human hearing and out of range of the speaker too. (The speaker itself acts as a filter to remove the 47kHz PWM signal.) The PWM pulse duty is variable over a 256:1 range using the 8-bit duty register in the MCU. This is an adequate dynamic range for the application, but a 10-bit (or higher) resolution would have been really nice.

The audio signal remains a square-wave (chopped at 47kHz), right up to the speaker driver transistor Q6. Hence, the power amplifier is a class-D design. Transistor Q5 is necessary to boost the base drive current required to turn on Q6. The PWM analog switch (U8D) is not capable of passing currents higher than about 2mA. Using a class-D power amplifier not only provides higher audio output power level than a linear amplifier, but it has another major advantage in this application. When the audio generator is “muted” (by firmware putting the MUTE signal High), the PWM switch (U8D) is off and hence the output transistor Q6 is off, so there can be no current whatsoever flowing in the speaker, no matter how much noise is present on the supply rail (+VIN). This is a very important consideration for an appliance which may be sitting closer than one metre to your ears while you are trying to sleep.

The audio level is updated by the firmware “sound synthesizer” driver routine every two milliseconds. The firmware driver is capable of synthesizing an amplitude “envelope” for the alarm sounds. User-settable parameters determine the “attack” (slope), “sustain” (time), “release” (slope) and “hold-off” (time), in similar manner to the analog sound synthesizers of the 1970’s (for which the author has a fond nostalgia). The “hold-off” time determines the interval of silence between successive “plays” of an alarm sound. (This is not the same as the “snooze interval”, which is independently settable.)

In addition, the firmware provides a means to modulate the audio amplitude and/or frequency with (virtual) modulating signals, the period of which can be set to any integer multiple of two milliseconds (i.e. 2, 4, 6, 8 ... ms), up to 500ms.

In summary, a simple low-cost audio generator circuit in combination with a smart firmware driver routine delivers a diverse range of appealing sound effects (and some not so appealing!).

Circuit Description – Display board

The display uses a 4-digit 7-segment LED display because it is a low-cost solution giving adequate readability, and because the technology lends itself to variable-brightness capability. A white or multi-colour backlit graphical LCD panel would have been more aesthetic, but the extra cost was considered unjustified, and continuity of supply of LCD panels can be an issue.

The 4-digit numeric display is realized using two 2-digit devices (Avago HDSP-523X), available in 3 colours: red, green or yellow. Pin-compatible parts are made by other opto-electronics manufacturers. Backlit enunciators are realized by 24 discrete LEDs, with 5mm round diffuse lens of various colours, i.e. red, green or yellow. Don't be tempted to fit blue or white LEDs for any of the enunciators. Their forward voltage drop is too high (typically about 4V) to be used in this circuit design.

The 7-segment display and enunciator LEDs are multiplexed in a common cathode 7 x 8 matrix. The cathodes are driven (active low) by a 7-element "peripheral driver" IC (U5 = ULN2003A), which is an array of 7 Darlington transistors with TTL/CMOS logic compatible inputs.

The anode (segment) lines are driven by an 8-bit CMOS shift-register. Looking at the display schematic, astute readers will notice there are no current-limiting resistors in series with the anode lines. The design relies on the source resistance of the CMOS outputs (approx. 60Ω) to limit the LED current. The ULN2003 outputs have a minimum low-state voltage near 1V (because of the Darlington configuration). The LEDs' forward voltage drop is about 2V. That accounts for 3V, which when subtracted from the 5V supply, leaves 2V on the CMOS outputs' equivalent resistance (60Ω). The LED peak current is therefore limited to about 33mA ($=2V/60\Omega$).

Keep in mind the LEDs are multiplexed 8:1, and dimmed using PWM, so the peak current is allowed to be near the LEDs' maximum rating. The average LED current will be much lower than this — about 2mA in normal viewing conditions. Variable brightness is achieved by applying a variable duty 47kHz pulse waveform (PWM) to the Output Enable (OE#) input of the LED cathode driver register (U4). The range of PWM duty, and hence brightness, is 256:1. At minimum duty (1/256), the display is still bright enough to be readable in low ambient light conditions.

The LED display requires two 8-bit output ports and the push-buttons need an 8-bit input port. MCU I/O port expansion is realized by 8-bit CMOS shift-registers with parallel data latches and serial data transfer logic. The shift-registers are (almost) directly compatible with the MCU 'SPI' (serial peripheral interface) bus. With a tiny sprinkling of "glue" logic, implemented by a quad tri-state buffer (U2 = 74HC125), the display output registers (U3, U4 = 74HC595) and button input register (U1 = 74HC165) connect easily to the SPI bus. Note that buffers U2A and U2B are wired to function as inverters, with the aid of pull-down resistors on their (tri-state) outputs. Inverting the polarity of the SPI clock signal (SCK) to the display registers allows the same SPI clocking mode to be used for simultaneous read (slave input) and write (slave output).

Using the SPI bus minimizes the number of signals required to interface the display board to the main board. The display and button registers share a common SPI "slave-select" signal (SSDISP#). The display registers are written and the button register is read in the same 2-byte SPI data transfer cycle. Since the button input register is only one byte wide, the second byte of the received data is ignored by the firmware.

The slave-select signal SSDISP# is inverted to obtain a shift/load signal SH/LD#. While the input port is de-selected (SSDISP# High), parallel data are accepted by the input latch. During an SPI transfer, SSDISP# is put Low, SH/LD# goes High, so the input latch is inhibited and the serial clock is enabled, i.e. the input data is held constant while being shifted out.

Output registers (74HC595) are comprised internally of an 8-bit serial-in/parallel-out SR with an 8-bit D-type latch. Output pins are driven from the latch bits. Whenever the "latch clock" signal LCLK is pulsed, the rising edge causes the 8 bits in the internal shift register to be transferred to the output latch. While LCLK remains High, the bits in the output latch remain unchanged. Bits in the internal shift register change whenever any SPI transfer cycle (read or write) is executed, regardless of whether the SPI slave select signal is asserted or not. In other words, the shift register data will change when other devices on the SPI bus, if any, are selected and written to. This doesn't matter, so long as LCLK remains static while there is garbage data in the internal shift-register. Connecting LCLK to the display slave-select, SSDISP#, ensures the desired behaviour.

The display board has a phototransistor (Q1 = BPW85C) which senses the ambient light level. The current flowing in the emitter resistor, and hence the voltage across it,

increases with increasing illumination of the transistor junction. The emitter voltage is monitored by a comparator on the main board using a primitive analog-to-digital conversion technique, as noted earlier.

Provision is made on the display board for mounting a rotary encoder switch (S9). Project builders who choose to deviate from the "standard" enclosure arrangement might want to relocate the encoder switch. This is easily done by cutting the display board into two pieces, separating off the smaller piece with the encoder switch. For this reason, wiring terminations to the encoder switch are separate from the other display board connections (8-way pad strip).

Internally, the encoder switch has two sets of contacts which make and break as the shaft is rotated. Using pull-up resistors to the +5V supply rail, the two switch outputs (A and B) generate square pulse trains in a quadrature phase relationship (i.e. 90 degrees out of phase with each other). Each output produces 24 pulses per revolution of the shaft, but the relative phases of the two pulse outputs is different (by 180 degrees) depending on the direction of rotation. A firmware decoding routine removes noise from the signals and keeps track of changes in the encoder shaft position.

Assembly Instructions

See HTML file in docs download package.