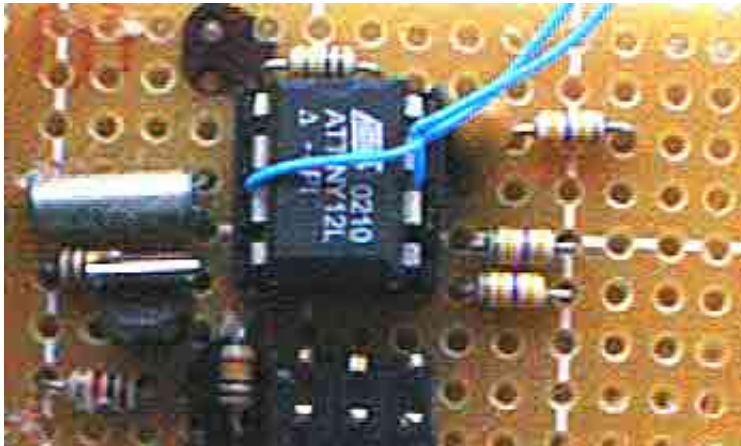


# Real Time Clock/Calendar/ Alarm with Interpreter for battery backed-up and battery powered operation with DS interface.

Based on the Atmel ATtiny12L-4PI microcontroller



This is the timekeeping test circuit. It includes a one-transistor circuit to switch between an external 5V power supply and a 3v battery. That loop of blue wire-wrapping wire is a lariat used to quickly and easily pull the chip out of the socket without damage and without having to reach for the removal tool.

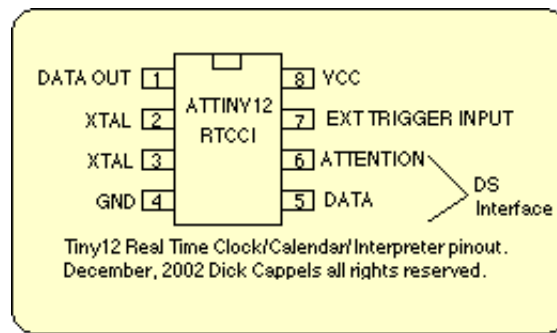
- **Low current operation (less than 60 uA at 3V -see data sheet)**
- **Alarm and external event triggered interpreter operation**
- **Capable of stand-alone timer and alarm use. No additional processor required once programmed.**
- **One external input pin and one open drain output pin for interpreter**

**Note: When**

**After programming, you must:**

- 1. Select the Low Frequency Crystal oscillator 67ms + 32k clock, and**
- 2. Disable Reset to free up pin 1 as open drain output.**

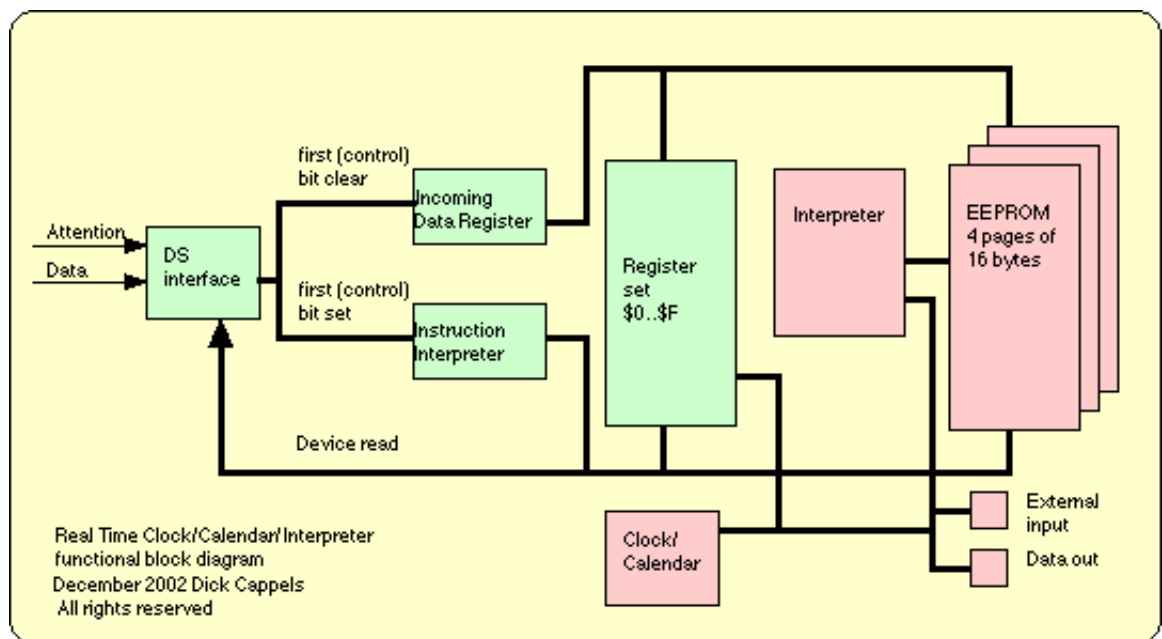
You might want to enable brownout detection if the EEPROM is important in your application, but be aware that activating brown out detection will increase current drain.



Pinout

The clock is based on the Atmel ATtiny12 processor which is ideal for this kind of application because of its low current drain when clocked by a 32768 Hz crystal.

## Architectural Description

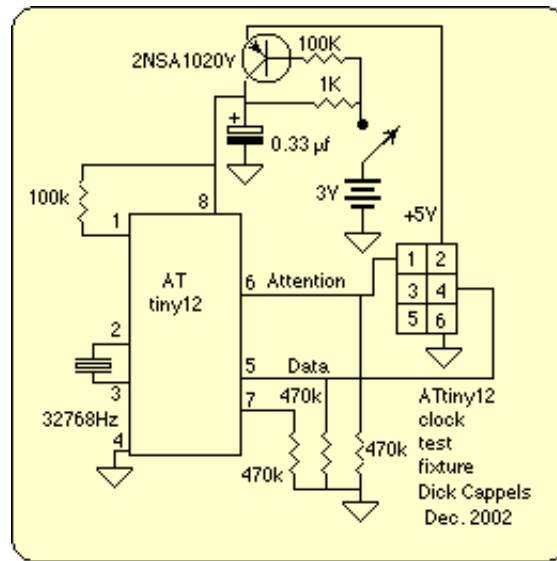


Functional block diagram

A DS Interface allows access to a register file, EEPROM, and an instruction interpreter. A separate interpreter interprets instructions stored in the EEPROM. EEPROM interpreter operation can be initiated by a clock/calendar alarm, an external event, or by command via the DS interface. The register set plays a central role in moving information between the DS interface and the clock and EEPROM interpreter. One external input can be sampled by the interpreter and one external open drain output can be driven by the interpreter.

The clock's output can be read from registers 3 through 8, but writing to these registers does not affect the clock unless the "write time buffers to clock" flag is set in the control register (register \$0F). See the Register Assignments section for more detail.

### Electrical considerations



Timekeeping test circuit

Be sure to read the electrical specifications in the Atmel ATtiny12 data sheet. One thing to keep in mind is that the open drain output on pin 1 should not be allowed to be pulled more than a few hundred millivolts above VCC (pin 8) because this might damage the chip or trigger the reset circuit.

Note that the 32768 watch crystal connects directly to the ATtiny12's pins without any capacitors or other components needed. The drive power to the crystal is very low -on the order of a micro watt I believe, so probing the crystal connections with a voltmeter, scope probe, or someone's fingers is likely to stop the oscillator.

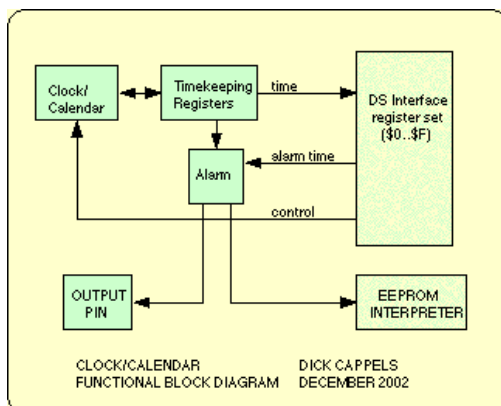
The timekeeping test fixture shown above allows the clock/calendar chip to run from a battery most of the time, and automatically switch to +5V when it is plugged into the DS Interface Tool. When the voltage applied to the connector is more than about a volt above the battery voltage, sufficient base current appears in the PNP transistor to saturate it. This circuit is also suitable for use in situations in which a battery backed up clock/calendar chip is on the same PC board as a host processor. The PNP transistor can be a 2N2907 or similar. Whatever transistor you choose, pay attention to the low current beta. Current flows through the 1K resistor to supply the ATtiny12 when operating from the battery, and current flow the other way to trickle charge the battery when external power is applied through the connector. This creates an interesting set of design tradeoffs and in the end, the maximum trickle charge current may narrowly limit your choices of transistor betas.

The 100k resistor provides a pull-up on pin 1, which has not active pull-up of its own, to facilitate monitoring of the digital output.

The 470 k resistors hold the inputs low when the test circuit is not plugged into the DS Test tool. Letting these inputs float is a bad idea in current sensitive situations since floating inputs can easily double this chips' current drain.

Sometimes real life results are more inspiring than specifications. After running continuously for two months on the test fixture above, this circuit agreed to within 2 minutes with my very expensive (\$35) Casio watch. That's less than 400 parts per billion ( $400 \times 10^{-9}$ ). I used some already-almost-dead penlight cells that would no longer run my CD player, so its been running from 2.4 volts and the battery voltage has not declined noticeably the last three months. On this particular board, the voltage drop across the 1 k resistor is only about 6 millivolts!

# Clock/Calendar/ALARM



## Summary:

The clock updates internal timekeeping registers once each second after which, if clock updates are enabled, the timekeeping registers are copied to the time registers in the DS Interface register set. Next, if enabled, the alarm compares the alarm time and alarm date stored in the DS Interface register set with the clock's internal timekeeping registers. When the alarm time and date agree with the time and date in the timekeeping registers, the alarm triggers the EEPROM interpreter if the enabled, otherwise it pulls the output pin (pin 1) low.

## Clock/Calendar Command Set

Instructions sent over the DS Interface are capable of setting the clock/calendar, setting the alarm, writing and reading data and programs to the EEPROM, running the EEPROM Interpreter and setting and reading chip status. The commands are sent as a byte preceded by a control bit set high. See the DS Protocol for details.

Command	Function
1R	Place contents of Incoming Data Register into register[R]
2R	Read contents of register[R] to DS port
3X	Dump time registers to DS port [Y:M:D:H:M:S]
4A	Write contents of data register to EEPROM location [A]
5A	Read contents of EEPROM location [A] to DS port
6X	Interpret contents of EEPROM
7X	Bless EEPROM (update checksum and ram flags)
8X	Read firmware version number to DS port (2 bytes)
9X	Set output pin high (Alarm Reset)

### Command 1R; Place contents of Incoming Data Register into register[R]

Write the data in the Incoming Data Register to the selected register by sending a command byte composed of the value \$1 in the upper nybble and the value (\$0..\$F) corresponding to the register to be written to in the lower nybble. For example, to write the data previously sent to the Incoming Data Register into the Control Register (register \$F), send the value \$1F as a command.

### Command 2R; Read contents of register[R] to DS port

Causes the chip to send the value of a specified register to the host via the DS Interface. This command is sent by sending a command byte composed of the value \$2 in the upper nybble and the value (\$0..\$F) corresponding to the register to be read from in the lower nybble. For example, to read the contents of the Control Register (register \$F), send the value \$2F as a command.

### Command 3X; Dump time registers to DS port [Y:M:D:H:M:S]

This command causes the chip to send six data bytes representing the contents of the time registers. This command is sent by sending a command byte composed of the value \$3 in the upper nybble and any value (\$0..\$F) in the lower nybble. For example, the commands \$30 and \$3A are both valid forms of this command and are indistinguishable from one-another by the chip.

### Command 4A; Write contents of data register to EEPROM location [A]

Write the data in the Incoming Data Register to the selected EEPROM location by sending a command byte composed of the value \$4 in the upper nybble and the value (\$0..\$F) corresponding to the address on the current EEPROM page to be written to in the lower nybble. For example, to write the data previously sent to the Incoming Data Register into EEPROM location \$C on the current page, send the value \$4C as a command.

The current page may be changed by writing to bits 5 and 4 of the Control Register. See the section on the Control Register for details.

### Command 5A; Read contents of EEPROM location [A] to DS port

Causes the chip to send the value of a specified location in the EEPROM to the host via the DS Interface. This command is sent by sending a command byte composed of the value \$5 in the upper nybble and the value (\$0..\$F) corresponding to address in the current EEPROM page to be read from in the lower nybble. For example, to read the contents of EEPROM address \$C on the current page, send the value \$5C as a command.

### Command 6X; Interpret contents of EEPROM

This command causes the EEPROM Interpreter to begin interpretation of instructions in the EEPROM starting with the address stored in the Incoming Data Register. This command is sent by sending a command byte composed of the value \$6 in the upper nybble and any value (\$0..\$F) in the lower nybble. For example, the commands \$60 and \$6A are both valid forms of this command and are indistinguishable from one-another by the chip.

For example, to begin executing a program in EEPROM starting at EEPROM address \$20, send the data value \$20 to the chip, followed by the command \$60.

### Command 7X; Bless EEPROM (update checksum and ram flags)

This command causes the chip to calculate the sum, without carry, of contents of bytes in EEPROM locations \$00 through \$FE and to store the sum at EEPROM location \$3F. It also sets the CHECKSUM OK FLAG in the Control Register and sets internal flags so processes within the chip consider the EEPROM contents to be valid.

This command should be used to re-establish the checksum after writing to the EEPROM using the 4A command.

### Command 8X; Read firmware version number to DS port (2 bytes)

This command causes the chip to send two data bytes corresponding to the firmware revision to the host over the DS Interface. The first byte corresponds to the version number and the second byte corresponds to the revision level of that version.

This command is sent by sending a command byte composed of the value \$8 in the upper nybble and any value (\$0..\$F) in the lower nybble. For example, the commands \$80 and \$8A are both valid forms of this command and are indistinguishable from one-another by the chip.

### Command 9X; Set output pin high (Alarm Reset)

This command causes chip to set the state of the Data Output Pin (pin 1) high. It serves to reset the pin to its initial state after, for example, the pin is set low by the alarm or by the EEPROM interpreter.

This command is sent by sending a command byte composed of the value \$9 in the upper nybble and any value (\$0..\$F) in the lower nybble. For example, the commands \$90 and \$9A are both valid forms of this command and are indistinguishable from one-another by the chip.

## Register Assignments

Sixteen registers are accessible via read and write commands over the DS interface. They are as follows:

Register	Function
0	Incoming Data Register from DS port
1	Register A
2	Alarm mask
3	Year buffer
4	Month buffer
5	Day buffer
6	Hour buffer
7	Minute buffer
8	Second buffer
9	Year alarm
A	Month alarm
B	Day alarm
C	Hour alarm
D	Minute alarm
E	Second alarm
F	Control Register

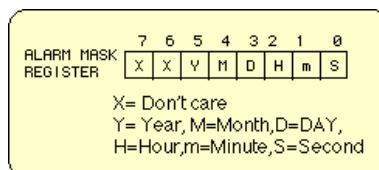
### Register 0; Incoming Data Register from DS port

This register serves as Incoming Data Register for the DS interface.

### Register A

This register can be tested and incremented by the EEPROM Interpreter. It can serve as an interface between the EEPROM interpreter and the DS interface and as a counter for the EEPROM interpreter. See the EEPROM interpreter instructions for more detail.

### Register 2; Alarm Mask



The Alarm Mask Register is meant to reduce the number of conditions that need to be met for the alarm to trigger. By writing a zero into one or more flags, the corresponding units will be ignored when the clock checks to see if the conditions for an alarm have occurred. The Alarm Mask Register is initialized to \$FF during power-on initialization.

Consider this example. The registers are set as follows:

Register 2 AlarmMask Register = \$01 (all but seconds ignored)  
 Register E Alarm Seconds = \$20  
 Register F Control Register = \$40 (Alarm Enabled, clock notstopped)

Each minute, when the value in the seconds buffer (register 8) equals \$20, the alarm will trigger.

### Registers 3,4,5,6, 7, and 8; Time Buffers

These data are updated by the clock in binary format with ranges as follows:

Reg 3 Year \$00..\$FF  
 Reg 4 Month  
 Reg 5 Day \$00..\$1C,\$1D,\$1E,\$1F depending upon the month and whether it is a leap year or not.  
 Reg 6 Hour: \$00..\$18  
 Reg 7 Minute: \$00..\$3B  
 Reg.8 Second: \$00..\$3B

Think of these read/write registers as the time input/output register for the clock. The clock updates these buffers each second to agree with the clock's own registers. They can also be written and when the "write time buffer to clock" flag is set, the values in these registers are copied to the clock's own registers.

The once each second updates to these registers from the clock can be stopped by setting the "clock stop" flag in the control register. Stopping the clock is necessary for both setting the clock and avoiding the problems that can result if the counters are incremented after some but not all registers have been read.

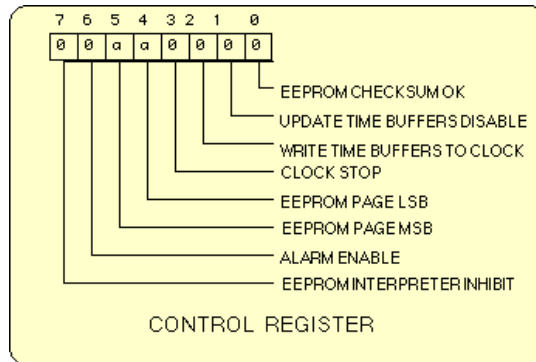
During chip initialization, the seconds register is written with the invalid value \$FF, which is then overwritten about a second later by the clock. Thus, any read of the time that returns \$FF in the seconds register is to be regarded as invalid as it was read before the registers were updated by the clock.

### Registers 9, A, B, C, D, and E, Alarm Registers

These registers hold the value that is compared with the clock each second, provided that the alarm is enabled. The range of valid values is given below. During chip initialization, seconds is written with the invalid value \$FF to assure that the alarm will not trigger until this register is programmed. Writing an invalid value to one of the registers while the corresponding bit in the Alarm Mask Register will keep the alarm from triggering.

Reg 9 Year alarm \$00..\$FF  
 Reg A Month alarm \$00..\$0C  
 Reg B Day alarm \$00..\$1C,\$1D,\$1E,\$1F depending upon the month and whether it is a leap year or not  
 Reg C Hour alarm \$00..\$18  
 Reg D Minute alarm \$00..\$3B  
 Reg E Second alarm \$00..\$3B

### Register F, Control Register



The control register holds flags that control activity of the clock, alarm, and EEPROM Interpreter. All bits may be read and written via the DS Interface. Some bits may be written by processes within the chip. The Control Register is initialized to \$00 during power-on initialization.

Bit	Function
0	EEPROM CHECKSUM OK
1	UPDATE TIME BUFFERS DISABLE
2	WRITE TIME BUFFERS TO CLOCK
3	CLOCK STOP
4	EEPROM PAGE Least Significant Bit
5	EEPROM PAGE Most Significant Bit
6	ALARM ENABLE
7	EEPROM INTERPRETER INHIBIT

#### BIT 0; EEPROM CHECKSUM OK

If set to 1, the sum, without carry, of contents of bytes in EEPROM locations \$00 through \$FE was found to be equal to the value stored in EEPROM location \$3F. This bit is set or cleared as appropriate during chip initialization. It is set to 1 during execution of the Bless EEPROM instruction, which calculates the current checksum, stores the checksum in the last EEPROM location, sets the EEPROM CHECKSUM OK flag, and sets internal flags.

This bit must be set in order for the EEPROM Interpreter to run in response to chip initialization, an external event, or an alarm.

#### Bit 1; UPDATE TIME BUFFERS DISABLE

While this bit is set, updates of the timer registers (registers 3 through 8) by the clock are suspended. This is useful when reading the time registers to prevent some digits from changing before the entire set of registers can be read. It is also useful when writing to the time buffers to set the clock.

#### Bit 2; WRITE TIME BUFFERS TO CLOCK

When this bit is set to 1, the contents of the time buffers (registers 3 through 8) are copied to the clock's internal timekeeping registers. Disable the time buffer update before doing this.

#### **Bit 3; CLOCK STOP**

While this bit is set of 1, the timekeeping function of the clock stops and updates to time registers (registers 3 through 8) are suspended.

#### **Bit 4 ; EEPROM PAGE Least Significant Bit, and Bit 5 EEPROM PAGE Most Significant Bit**

Control Register bits 4 and 5 constitute the two upper bits of the EEPROM address via the DS Interface as follows:

Bit 5	Bit 4	Address range of write (\$4A) and read (\$5A) instruction
0	0	\$00..\$0F
0	1	\$10..\$1F
1	0	\$20..\$2F
1	1	\$30..\$3F

#### **Bit 6; ALARM ENABLE**

While this bit is set to one, the contents of the alarm registers (registers 9 through E) are compared with the clock's internal timekeeping registers each second. When a match occurs, if this bit is set, the alarm will be triggered.

#### **Bit 7; EEPROM INTERPRETER INHIBIT**

The EEPROM Interpreter will not start interpreting programs stored in EEPROM if this bit is not set to 1. This bit can be set by grounding the Data Out Pin (pin 1) during chip initialization—thus providing a means of recovery in case a program is written to the EEPROM that interferes with user access via the DS Interface.

## **Alarm**

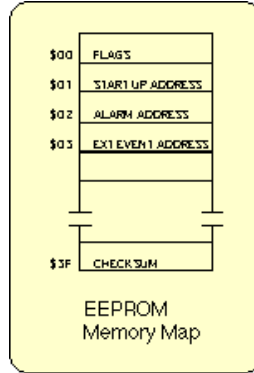
The alarm will be triggered when the contents of the alarm registers (registers 9 through E) match the timekeeping registers in the clock (and consequently the time registers, registers 3 through 8), provided that the Alarm Enable bit in the control register (register \$F, bit 6) is set to 1. If the flag that is EEPROM location \$00, bit 5 is set to 1, and the EEPROM checksum is valid, the EEPROM Interpreter will begin execution of the program that starts at the address contained in EEPROM location \$02. If the flag that is EEPROM location 0, bit 5 is not set, or the EEPROM checksum is not valid, then when the alarm is triggered, it will make the Data Output Pin (pin 1, open drain) low.

**Note: in order for the chip to work, after programming, you must:**

- 1. Select the Low Frequency Crystal oscillator 67ms + 32k clock, and**
- 2. Disable Reset to free up pin 1 as open drain output.**

# EEPROM AND EEPROM INTERPRETER

## On-Chip EEPROM



The 64 byte EEPROM on the ATtiny12 is accessible by the host via the DS Interface. Five of the locations within the EEPROM have been allocated, four related to the EEPROM interpreter and one for the EEPROM checksum, which is checked during chip initialization. The entire EEPROM space may be allocated by the programmer for nonvolatile data storage for the host and for program and constant storage for the EEPROM Interpreter.

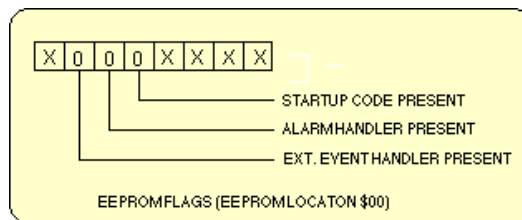
From the point of view of access via the DS Interface, the EEPROM is organized as 4 pages of 16 bytes. Only the current page is directly addressable by the DS Interface. The current page is determined by control register bits 4 and 5. The entire 64 byte space appears as one continuous block to the interpreter during execution.

Read and Write access as well as generation of a checksum, if desired, are controlled with EEPROM write commands (\$4A commands), EEPROM read commands (\$5A commands), and the Bless EEPROM command (command \$7X).

Take note that the EEPROM in the ATtiny12 has a limited number of write cycles, beyond which the manufacturer does not guarantee operation. See the manufacturer's data sheet for details.

## EEPROM Interpreter

The Clock/Calendar/Alarm chip is equipped with an interpreter that will interpret instructions stored in the 64 byte on-chip EEPROM. The interpreter can be used for simple tasks in response to power-on initialization, an alarm, an external event, or a command via the DS Interface. Each of the first three mechanisms (power-on, alarm, and external event) has a corresponding location in the EEPROM where the start address is stored, and an enable flag (zero=true) in EEPROM location zero. There is an EEPROM Interpreter Inhibit bit in the Control Register, and in order for a routine to execute, both the EEPROM inhibit bit in the control register and bit corresponding to that particular routine in EEPROM location \$00 must be clear,



A **ZERO** written to a flag bit indicates that the corresponding code is present in the EEPROM. The use of a zero instead of a one to indicate that code is present stems from the fact that the EEPROM's erased state is \$FF. This EEPROM location must be written by using the EEPROM write instruction (\$40).

PROGRAM	START VECTOR LOCATION	EEPROM LOCATION \$00 BIT
Power up initialization program	EEPROM location \$01	4
Alarm program	EEPROM location \$02	5
External event program	EEPROM location \$03	6

The bits in EEPROM location \$00 corresponding to the three pre-defined events must be written with a **ZERO** in order to enable their respective pieces of code.

For example, if the only program in EEPROM memory was one to handle external events and it started at EEPROM location \$20, the first four memory locations would be programmed like this:

EEPROM LOCATION	VALUE AT LOCATION	COMMENT
\$00	\$BF	FLAG TO INDICATE EXTERNAL EVENT CODE IS PRESENT
\$01	XX	START UP PROGRAM -NOT PRESENT SO DON'T CARE
\$02	XX	ALARM PROGRAM -NOT PRESENT SO DON'T CARE
\$03	\$20	EXTERNAL EVENT PROGRAM STARTS AT \$20

## Interpreter Command set

```

$00  Reset clock to zero
$02  Output pin high (reset alarm)
$03  Output pin low
$04  OR [next byte] with control register
$05  AND [next byte] with control register
$0C  Set Alarm to [next byte]
$0D  Increment registerA
$0E  Load alarm mask with [next byte]
$0F  Load register A with [next byte]
$C0  Jump to [next byte] if registerA = [3rd byte]
$C1  Jump to [next byte] if registerA - [3rd byte]
$80  Jump to [next byte] unconditionally
$81  Jump to [next byte] if register = data register
$82  Jump to [next byte] if register - data register
$83  Jump to [next byte] if input pin is high
$84  Jump to [next byte] if input pin is low
$FF  Stop execution of program (end of program)

```

### \$00 Reset clock to zero

Causes all of the time registers (registers 3 through 8) to be set to zero and their contents to be copied to the clock's timekeeping registers. Instruction completes after the clock's timekeeping registers have been cleared, which occurs within one second after the instruction is called.

Example program that resets the clock when the alarm is triggered

EEPROM LOCATION	VALUE AT LOCATION	COMMENT
\$00	\$DF	FLAG TO INDICATE ALARM CODE IS PRESENT
\$01	\$XX	START UP PROGRAM -NOT PRESENT SO DON'T CARE
\$02	\$03	ALARM PROGRAM STARTS AT \$0A
...		
\$0A	\$00	INSTRUCTION CODE TO SET CLOCK TO ZERO
\$0B	\$FF	INSTRUCTION CODE TO STOP PROGRAM EXECUTION

### \$02 Output pin high (reset alarm)

### \$03 Output pin low

Causes the output pin (pin 1) to be set high (\$02) or low (\$03), depending upon the instruction. The output pin is open drain and needs an external pull-up.

Example program that resets the clock when the alarm is triggered

EEPROM LOCATION	VALUE AT LOCATION	COMMENT
\$00	\$DF	FLAG TO INDICATE ALARM CODE IS PRESENT
\$01	\$XX	START UP PROGRAM -NOT PRESENT SO DON'T CARE
\$02	\$0A	ALARM PROGRAM STARTS AT \$0A
...		
\$0A	\$02	INSTRUCTION CODE TO SET DATA OUTPUT PIN HIGH
\$0B	\$FF	INSTRUCTION CODE TO STOP PROGRAM EXECUTION

### \$04 OR [next byte] with control register

### \$05 AND [next byte] with control register

Causes the byte in the control register to be replaced with the content of the control register to be ORed (in the case of instruction \$04) or ANDed (in the case of instruction \$05) with the constant following the instruction. Execution continues with the byte after the constant.

Example program that clears the Alarm Enable flag (bit 6) in the Control Register:

EEPROM LOCATION	VALUE AT LOCATION	COMMENT
\$00	\$DF	FLAG TO INDICATE ALARM CODE IS PRESENT
\$01	\$XX	START UP PROGRAM -NOT PRESENT SO DON'T CARE
\$02	\$0A	ALARM PROGRAM STARTS WITH AT \$0A
...		
\$0A	\$05	INSTRUCTION CODE TO AND NEXT BYTE WITH CONTROL REGISTER
\$0B	\$BF	BIT 6 OF CONTROL REGISTER CLEARED



These instructions cause a jump to the address contained at the location following the instruction, provided that the associated condition is true. Execution resumes at the location following the constant if the jump is not taken.

Example: Jump to location \$20 if the input pin (pin 7) is low.

```
$10          $84          INSTRUCTION TO JUMP IF INPUT PIN IS LOW
$06          $20          ADDRESS TO JUMP TO IF INPUT PIN IS LOW
$07          $XX (PROGRAM EXECUTION CONTINUES HERE IF INPUT PIN IS HIGH)
. . .
$20          $XX (PROGRAM EXECUTION JUMPS TO HERE IF INPUT PIN IS LOW)
```

**\$FF Stop execution of program (end of program)**

When this instruction is encountered, the EEPROM Interpreter stops program execution.

## Liability Disclaimer and intellectual property notice

(Summary: No warranties, use these pages at your own risk. You may use the information provided here for personal and educational purposes but you may not republish or use for any commercial purpose).

I neither express nor imply any warranty for the quality, fitness for any particular purpose or use, or freedom from patents or other restrictions on the rights of use of any software, firmware, hardware, design, service, information, or advice provided, mentioned, or made reference to in these pages. By utilizing or relying on software, firmware, hardware, design, service, information, or advice provided, mentioned, or made reference to in these pages, the user takes responsibility to assume all risk and associated with said activity and hold Richard Cappels harmless in the event of any loss or expense associated with said activity. The contents of this web site, unless otherwise noted, is copyrighted by Richard Cappels. Use of information presented on this site for personal, non-profit educational and non-commercial use is encouraged, but unless explicitly stated with respect to particular material, the material itself may not be republished or used directly for commercial purposes. For the purposes of this notice, copying binary data resulting from program files, including assembly source code and object (hex) files into semiconductor memories for personal, non-profit educational or other non-commercial use is not considered republishing. Entities desiring to use any material published on this page for commercial purposes should contact the respective copyright holder(s).

**For information on a commercial license, please contact Dick Cappels at [projjects@cappels.org](mailto:projjects@cappels.org)**

Contents ©2002 Richard Cappels All Rights Reserved. <http://www.projects.cappels.org/>

-end of document-