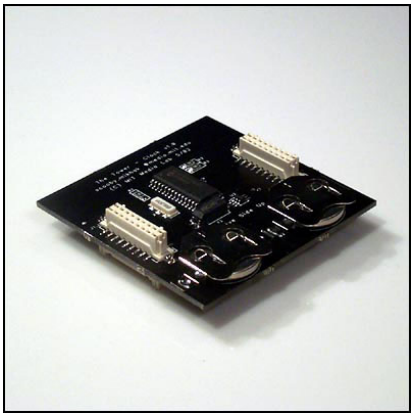
Clock Layer



Description

The Clock layer contains a real-time-clock integrated circuit, as well as batteries to back up stored time and date information. The clock is capable of remembering year, month, date, day, hour, minute, and second, and can easily be used as a timer event-trigger for time-driven processes.

Hardware Detail

The Maxim MAX6900 real-time-clock chip communicates with the on-board PIC via the I^2C serial protocol, but it is important to note that it is not on the main Tower serial bus. To avoid address conflicts with the rest of the system, the PIC processor on the layer independently communicates with the clock chip using two of the PIC's standard I/O lines. Two 20mm 3-volt coin cell batteries are also on-board to provide clock backup for an estimated 10-15 years, so that the time data will not have to be re-entered every time the Tower is powered on. These batteries are type CR2032. Please note that the layer will not function without these batteries installed, even when the Tower is powered.

Layer Code

The include file for the Clock layer contains eight functions, two low-level read/write functions for directly accessing registers on the clock chip, two higher-level functions for setting and getting the current date and time, a function for starting a timer, a function for getting the timer value, and two lookup table functions, one for the day of the week, and another for the month of the year. In addition, three global variables are defined within the file, which are used exclusively by the timer functions to track the amount of elapsed time. *(All of these functions are written for the PIC Foundation. The include files used with other foundations differ slightly, due to the presence of local variables.)*

For reading and writing data, it is important to know what is contained in each clock memory location. A table of the register assignments is shown below:

Memory Location	Description
$00	Seconds
$01	Minutes
$02	Hours
$03	Date
$04	Month
$05	Day
$06	Year
$07	Control
$08	(unused)
$09	Century

The **clock-write** function takes two arguments, the clock memory location, and the data value to be written. The clock memory location contains a specific field corresponding to the current time, and can be a value from 0 to 9. A total of three arguments are being sent to the Clock layer. First a “0” is sent, indicating that a write function is being performed. Then, the memory location and data value are sent. It is important to note that all data values are stored in BCD (Binary Coded Decimal) format, where the high four bits represents the tens digit of the value, and the low four bits represent the ones digit. The conversion between decimal and BCD is done in the following two functions as the data value is sent or received.

```
to clock-write :addr :data
    i2c-start
    i2c-write-byte $0e
    i2c-write-byte 3
    i2c-write-byte 0
    i2c-write-byte :addr
    i2c-write-byte ((:data / 10) * 16) + (:data % 10)
    i2c-stop
end
```

The **clock-read** function operates similarly to the write function, sending the clock memory location as an argument to the layer. The first argument in this case however, is a “1”, signifying a memory read. After the request is sent to perform the clock memory read, the result is read out of the layer’s transmit buffer as a single byte. Actually two bytes are read out, but the first is ignored, as it is known to be a “1”, representing the number of arguments to follow.

```
to clock-read :addr
    i2c-start
    i2c-write-byte $0e
    i2c-write-byte 2
    i2c-write-byte 1
    i2c-write-byte :addr
    i2c-stop
    i2c-start
    i2c-write-byte $0f
    ignore i2c-read-byte 1
    seti2c-byte i2c-read-byte 0
    i2c-stop
    output ((i2c-byte / 16) * 10) + (i2c-byte % 16)
end
```

The **set-time** function is used to set the entire time and date memory at the same time. It takes seven total arguments, corresponding to each data field for the time registers, and then simply calls the **clock-write** function with these arguments and the corresponding clock memory locations. The arguments are, in order, the year (which will be internally split into separate year and century values), the day of the week (Sunday=1 to Saturday=7), the month (January=1 to December=12), the day of the month, the hours (24-hour time), the minutes, and the seconds.

```
to set-time :year :day :month :date :hour :minute :second
    clock-write 0 :second
    clock-write 1 :minute
    clock-write 2 :hour
    clock-write 3 :date
    clock-write 4 :month
    clock-write 5 :day
    clock-write 6 (:year % 100)
    clock-write 9 (:year / 100)
end
```

The **get-time** function is used to print out all of the day and time information to the console in a single, easy to read sentence. The function essentially reads out all of the values from the timekeeping registers, and formats them nicely using a number of **print** and **print-string** function calls. The **type** function is similar to **print**, only it does not put a carriage return at the end of the number when it prints it. Look-up functions are used to get the string values for the day of the week and month names. Those functions are defined further down in this document.

```
to get-time
    print-string "|Today is |
    print-string get-day clock-read 5
    print-string "|, |
    print-string get-month clock-read 4
    print-string "| |
    type clock-read 3
    print-string "| |
    type ((clock-read 9) * 100) + clock-read 6
    print-string "|, |
    setn clock-read 2
    ifelse (n = 0)
        [type 12]
        [
            ifelse (n > 12)
                [type n - 12]
                [type n]
        ]
    print-string "|:|
    print-clock clock-read 1
    print-string "|:|
    print-clock clock-read 0
    print-string "| |
    ifelse (n > 11)
        [print-string "PM]
        [print-string "AM]
    cr
end
```

The **get-day** function is the look-up function for the name of the day of the week. The function takes a single argument, the number corresponding to the day, which ranges from 1-7. A series of if-statements are used to check the argument value and when a match is found, the text string is output.

```
to get-day :n
    if (:n = 1)
        [output "Sunday stop]
    if (:n = 2)
        [output "Monday stop]
    if (:n = 3)
        [output "Tuesday stop]
    if (:n = 4)
        [output "Wednesday stop]
    if (:n = 5)
        [output "Thursday stop]
    if (:n = 6)
        [output "Friday stop]
    if (:n = 7)
        [output "Saturday stop]
    output "Unknown
end
```

The **get-month** function is the look-up function for the name of the month. The function takes a single argument, the number corresponding to the month, which ranges from 1-12. A series of if-statements are used to check the argument value and when a match is found, the text string is output.

```
to get-month :n
    if (:n = 1)
        [output "January stop]
    if (:n = 2)
        [output "February stop]
    if (:n = 3)
        [output "March stop]
    if (:n = 4)
        [output "April stop]
    if (:n = 5)
        [output "May stop]
    if (:n = 6)
        [output "June stop]
    if (:n = 7)
        [output "July stop]
    if (:n = 8)
        [output "August stop]
    if (:n = 9)
        [output "September stop]
    if (:n = 10)
        [output "October stop]
    if (:n = 11)
        [output "November stop]
    if (:n = 12)
        [output "December stop]
    output "Unknown
end
```

The following two functions are used to set and read values from a long-term timer. While the built-in timer on the foundation is great for timing things on the sub-second level, this timer implementation uses the Clock layer to provide timer values anywhere from seconds to hours of elapsed time.

To use the timer, you must first reset it, by calling the **start-timer** function. This function takes no arguments, and simply sets the values of three global variables to the current clock values to use as a reference point.

```
global [timer-hour timer-min timer-sec]

to start-timer
    settimer-hour clock-read 2
    settimer-min clock-read 1
    settimer-sec clock-read 0
end
```

The current timer value can be obtained by calling the **get-timer** function. This function takes no arguments, and returns the elapsed time (in seconds).

```
to get-timer
    output (((clock-read 2) - timer-hour) * 3600)
           + (((clock-read 1) - timer-min) * 60)
           + ((clock-read 0) - timer-sec)
end
```

Examples of Use

To use the Clock layer, a user would first want to check what time is currently stored on the chip. If the layer has been used before, a value may already be saved in the clock registers, which would still be present due to the on-board battery backups. Calling **get-time** will print out the current time data.

```
get-time
> Today is Sunday, January 1 1970, 12:07:43 AM
```

Obviously, this time data is probably not accurate, and the clock was probably reset recently. To set the clock, we use the set-time function as shown below:

```
set-time 2002 6 7 23 19 43 15
get-time
> Today is Friday, July 23 2002, 7:43:15 PM
```

This code will set the time registers to correspond to the desired date and time, and then read it back out to ensure that it was stored properly. The order of arguments is important, and is explained in detail in the function definition earlier in this document.

To use the timer functions, we must first reset it by calling **start-timer**.

```
start-timer
```

At any point in time after the timer has been started, **get-timer** can be used to see how much time has passed:

```
print get-timer
> 153
```

In this case, 153 seconds (that is, 2 minutes and 33 seconds) has passed since the time at which we started the timer.

The timer can easily be used to wait a fixed amount of time, as shown in the following example:

```
start-timer waituntil [get-timer = 92] print-string "|It's been 93 seconds.|
> It's been 93 seconds.
```

In this code, we reset the timer and wait for it to become the desired value. As soon as it reaches that point, the program continues. In the sample above, a string is then printed which tells us that the desired time interval is over.