Time Block



TimeBlock Overview 3

Usage 3

Battery Management 3

A Note on Lithium batteries and battery safety 4

Other Functionality 4

Circuit Description 5

Power 5

Display operation 5

Timing 5

Button Input 5

Voltage Measurement 6

Firmware Operation 6

Fuse Settings 6

Main.c description 7

GPIO configuration 7

Other initialization 7

Timer/Display Drive operation 8

Variable Description 8

ISR/Display Flow 9

ADC Operation 12

Shutdown 13

# TimeBlock Overview

TimeBlock is a super-simple timer intended to help keep track of where the time has gone. Its single-button interface and power-optimized (battery-conserving) design make it a great little helper when cooking, exercising, procrastinating or doing anything else where it’s good to know how long much time has passed, but it might not be convenient to access a phone’s stopwatch app. In its default configuration, TimeBlock works as a stopwatch with one-second resolution. When turned on it will count up to 99 minutes and 59 seconds and then shut itself off.

# Usage

When turned off, TimeBlock is in a deep sleep mode to minimize draw on the battery. A quick press of the button will turn on the display and start the timer counting up from zero. At any time the clock can be reset to zero by quickly pressing and releasing the button.

The button can also be used to turn the clock off or change the display intensity, allowing the user to find an optimal trade-off between battery life and display visibility. Shutting it off when not in use is the best way to maximize battery life. To shut TimeBlock off, press and hold the button for two seconds. After two seconds the display will change to ‘batt’ for one second, then display the current battery voltage. When the button is released it will go into a deep sleep mode until the button is pressed again.

To change the TimeBlock’s intensity, press and hold the button for 4 seconds. TimeBlock will go through the battery display routine but if the button isn’t released it will start cycling through different intensity options from 1-9. When the desired intensity setting is reached, releasing the button will cause that setting to be stored in the device and remembered for subsequent power-ups. Note that a higher intensity setting will significantly affect battery life. For inside usage away from direct sunlight, an intensity of 1 is usually sufficient. Outside or in highly lit environments it may be necessary to use a higher value. Selecting the lowest intensity value that still provides good display visibility is ideal to maximize battery life.

# Battery Management

TimeBlock is powered from an integrated 3.7V, 1000mAh lipo battery and has an on-board battery management system, which allows the battery to be charged from the micro USB connector. A small LED will turn on when the battery is charging, automatically turning off when the battery is fully charged. The battery has a nominal voltage of 3.7V, a full-charge voltage of 4.2V and a minimum safe discharge voltage of 3.0V. When running on batteries, TimeBlock will check the battery voltage once a second and will not run with a battery voltage below 3V. If the display briefly flashes all zeroes then turns off, the battery is in need of a charge. TimeBlock can be used when the battery is charging.

## A Note on Lithium batteries and battery safety

**ANY lithium battery can get angry if it’s mistreated**, and so TimeBlock has been designed to do its part to keep the battery happy. The on-board charge management IC will limit charging current to a safe level, and the TimeBlock will check battery voltage once/second and turn itself off if the voltage is approaching an unsafe discharge level. The TimeBlock enclosure is designed with room for rubber padding to protect the battery from shock if dropped and protect the battery cable insulation from becoming damaged. Keeping TimeBlock dry, away from extreme heat, water and avoiding hard shocks will ensure a happy and safe battery.

# Other Functionality

In addition to being configured as a Timer, the TimeBlock board can be configured as a simple voltmeter. In this configuration, TimeBlock board will be powered from the USB connector and the battery measurement circuit can be used to measure a DC voltage and display it with .01V resolution and high accuracy. In order to configure TimeBlock as a voltmeter, some jumpers must be configured. SJ1 determines whether VCC is provided by the battery or the USB connector. To use the device as a voltmeter, VCC must be sourced from the USB connector. SJ3 is a logic input that pulls PB1 to VCC, letting the microcontroller know that the device is functioning in voltmeter mode. SJ2 is used to short out Q1, essentially ensuring that Q1 is always on. The purpose of Q1 is described in the **Circuit Description** section below.

# Circuit Description

## Power

Depending on the TimeBlock configuration, power can be sourced from either the on-board USB connector or from the battery connector. Power input is determined by SJ1, which can tie the VCC trace to one of the input options. VCC will provide power to U1 to run the board. If power is sourced from the battery connector, IC2 can be used as a battery management IC to charge the battery from 5VDC from the USB connector. The ‘charge’ LED will be on when the battery is charging, and automatically turn off when full charge (roughly 4.2V) has been reached. If the power will be sourced from the USB connector, it is not necessary to populate IC2, C8, R26-R27 and the ‘charge’ LED.

## Display operation

The HDSP-B04E display has 4 individual digits each with a common anode. The 4 anodes are driven by dual P-Channel FET’s Q2 and Q3, which cycle through the four digits, updating every 2 milliseconds. The cathodes for the digits are each tied to the collectors of dual NPN BJT’s Q4-Q7. At the start of each 2mS cycle, the NPN transistors are set to a value corresponding to the digit to be displayed and then the appropriate P-Ch FET for that digit is turned on. The 2mS cycle is divided into 10 x 200uS intervals, and the P-Ch FET will remain on for 1-9 200uS intervals, depending on the intensity setting. When the number of 200uS intervals reaches the intensity set point, the FET will turn off. At the start of the next interval, the BJT’s will be configured for that digit

## Timing

The clock of microcontroller U1 is sourced from 4Mhz resonator U8. All on board timing is derived from this resonator, which has 0.07% accuracy. Over several 10s of minutes this will result in the timer’s value drifting by a couple of seconds as compared to a more accurate timer. This choice of 0.07% accuracy is a tradeoff of accuracy vs. complexity/cost.

## Button Input

Button S1 is pulled-up by resistor R25 and monitored by U1. When the device is in Deep Sleep mode, S1 is configured as an interrupt that can wake the device back up when pressed.

## Voltage Measurement

U1 has an on-board Analog to Digital Converter that can be used to measure a voltage level. This can be used to measure display voltage when running the panel in voltmeter mode, or to read the battery voltage when running on battery. The 10-bit ADC’s reference voltage is provided by 2.5V shunt regulator D1. In order to conserve battery voltage, D1 is normally not biased. When taking a voltage reading, U1 will turn on PC5 (REF\_ON) which will bias D1 via R28 to establish the 2.5V reference for the ADC. R22 and R23 are used to scale the voltage on the BAT\_IN line to a maximum of 2.5V. When configured for battery use, R22 and R23 will be 24.3k and 33k respectively. This will result in a voltage on DIV\_IN that scales from BAT\_IN by a factor of: (33/(33+24.3)) = .576. Therefore the maximum voltage that can be read is (2.5/.576) = 4.34V, which is designed with the maximum 4.2V battery voltage in mind. In the name of battery conservation, the resistor divider of R22/R23 is turned off when not measuring voltage. This is accomplished via transistor Q1. When it’s time to measure battery voltage, Q1 is turned on at the same time as D1 is biased and then several mS are allowed to pass so the voltages can settle out. At that point, an ADC reading is initiated, then Q1/D1 are turned off to go back to a power-saving state.

If running in Voltmeter mode when the voltage to be measured is going to be greater than ~5V, R22/R23 and SJ2 must be configured to ensure that the voltage on DIV\_IN does not get high enough to damage U1. By shorting SJ2, transistor Q1 is effectively shorted out/always on and then the R22/R23 divider can be selected based on the input voltage to keep DIV\_IN to a maximum voltage of 2.5V.

# Firmware Operation

## Fuse Settings

The fuses for the AVR must be set properly to ensure proper operation and clock source. They must be set for an external crystal oscillator where 4MHz is within the valid frequency range and the clock/8 bit must not be enabled. Valid fuse settings are: L = 0xDE H= 0xD9 E = 0xFF

## Main.c description

### GPIO configuration

* The four LSB of PORTC are outputs to drive the gates of the 4 (2x dual-package) P-FETs that enable the common anodes for each of the four display digits. Only one of these bits will be enabled at any given time.
* PC4 and PC5 are DIV\_ON and REF\_ON respectively. These are outputs where DIV\_ON drives the gate of Q1, terminating the resistor divider that’s used to measure the battery voltage. REF\_ON biases D1 through R28, providing a 2.5V reference for the ADC. When running on battery, both DIV\_ON and REF\_ON are disabled (LOW) when not in use to minimize battery consumption.
* All of PORTD is dedicated to driving the BJT’s that control the individual cathodes for the 7-segments of the display. The 8-bits drive the 7 segments plus the colon. The colon is used when displaying time, it flashes on and off every second. When displaying voltage, the lower dot of the colon is used as a decimal place.
* The two LSB of PORTB are configured as inputs, where PB0 is tied to the button input and PB1 is tied to jumper SJ3. When SJ3 is closed, the board will come up into ‘voltmeter mode’
* PB3-PB5 are used only for ISP via the Tag-Connect connector TC1. The Reset Pin is also routed to TC1

### Other initialization

After the GPIO has been configured, main.c will unmask Pin Change Interrupt 0, which is tied to the button. The interrupt is not enabled at this point, it is only enabled when the device goes to Deep Sleep mode, so that the button can wake it up. The ADC is initialized by setting the prescaler and pointing the MUX to ADC7, which is the divider input, but at this point the ADC is not enabled. Timer0 is initialized by setting it up to rollover/generate an interrupt every 200uS. This 200uS interrupt is the basis for all timing on the board. Global interrupts are enabled and the Sleep Mode is set to IDLE, so that the device can still wake up from timer interrupts but the CPU will not be clocking between interrupts, minimizing battery consumption.

After this initialization, PB1 is checked to see if it’s high (SJ3 is shorted) or low (SJ3 is open). The ADC coefficient will be set depending on this check, and the ADC coefficients must match the resistor divider installed in the positions of R22/R23. If it is hi, a function call will let the Timer and ADC files know that the device is in voltmeter mode. If it is low, the device will check if the battery is above 3V and, if not, will initiate a shutdown.

Within the main loop, the device will go to sleep (where sleep mode has previously been set to ‘IDLE’, and will be ‘woken up’ every 200uS by the Timer0 interrupt. Upon wakeup, it will check the ‘gotosleep’ flag and if this has been set it will initiate a shutdown into deep sleep. The shutdown routine includes enabling the Pin Change Interrupt 0, which will allow the device to wake up on a button press.

# Timer/Display Drive operation

The timing for the entire device is based on a 200uS interrupt generated by Timer0. Timer0 is configured in CTC mode and an interrupt will be generated each time it rolls over.

### Variable Description

Inside the timer operation there are a number of flags/variables described below:

* butstat: every 200uS interrupt, the ISR looks at the button input. If the button is pressed, butstat will be low. If the button isn’t pressed, butstat will be high.
* laststat: after checking the button status and whether or not it has changed, laststat is updated with the value of butstat. This gives a snapshot of what the butstat value was on the last ISR, enabling the ISR to see if the button’s status has changed between ISR’s.
* cyclesinstat: this variable is incremented if butstat and laststat are the same i.e. the button’s status hasn’t changed since the last cycle. If butstat is not the same as laststat, the button’s status has changed and so cyclesinstat is reset to zero.
* fallarm: This is used as a debouncer to determine whether or not a low-level signal on the button should be acted upon. If ‘butstat’ has been high consistently for the last 50 cycles (50 x 200uS = 10mS) then fallarm will be set to 1. If butstat has recently been detected low, fallarm will be 0. This means that the firmware won’t take action on a falling edge of the button unless the button has been consistently high for the last 10mS. It prevents falsely detecting a button press if the button is noisy/bouncing from the previous press.
* turnoff\_arm: This will determine whether or not the board should initiate the shutoff procedure. If the board is in timer mode and the button has been pressed for more than 2 seconds, the shutdown will be initiated. When initiated, turnoff\_arm will be set to zero to indicate that the shutdown is already in progress and shouldn’t be started again. When the board is restarted (or, alternatively, when the intensity set procedure is complete) turnoff\_arm will be re-armed so that the next 2-second button press will re-initiate the shutdown.
* clockstat: clockstat is used to direct a couple of state machines in various stages of the ISR. It has several possible values:
  + clockrun: in clockrun, the clock is incrementing once/second
  + clockstop: the clock is stopped
  + intenset: This is while the intensity is being set
  + batdisp: When in timer mode, batdisp is selected right after ‘batt’ is put on the display. On the subsequent 1-second increment, the actual voltage is displayed. When in voltmeter mode, clockstat is consistently in batdisp, so that once every second the function is called to display the voltage. After the function to display voltage is called, clockstat is changed to ‘voltdisp’.
  + voltdisp: Clockstat is set to voltdisp right after the function is called to display the voltage. When in Timer mode, the board will stay in ‘voltdisp’ until either the button is released (and so shutdown is initiated) or the board goes into Intensity Set mode. When in voltmeter mode, the clockstat is reset to ‘batdisp’ so that the function to display voltage is called again at the next 1-second rollover.
* Intensity: This variable will initially be populated from a value stored in EEPROM. Its valid range is 1-9, and this determines for how many 200uS rollovers the P-CH FET driving the anode of the current digit will be left on.
* Voltmeter: This is used as a Boolean to determine whether or not the board is working in voltmeter mode.
* Digits[]: This array of 4 bytes stores the current value of the 4 digits. For example at 10 minutes and 15 seconds, digits will contain the following values: digits[] = {1,0,1,5}
* Numbers[]: this array contains the values that will drive the 7-segment display for their corresponding index. For example, numbers[0] = 0xD7. When D7 is applied to PortD, the digit whose P-FET is currently enabled will show 0.

### ISR/Display Flow

Every 200uS an interrupt is generated, and the ISR handles most of the heavy lifting for driving the display. The first thing the ISR does is to read the button into variable butstat. If butstat is the same as laststat, it will increment the cyclesinstat counter. If they are not the same, it will reset cyclesinstat to zero.

If the board is not in voltmeter mode, it will then check to see if the button has been off for greater than 10mS. If it has been, ‘fallarm’ will be set to 1 so that the firmware will take action on the next falling edge of the button. Here it also re-iterates that the board should be in ‘clockrun’ mode, which is what restarts the clock following a quick button press.

Next it checks if the button has been pressed after ‘fallarm’ has been re-armed. If it has been, the time will be set to zero and the clock stopped. If the button remains pressed, the firmware will take further action as described below. If it is released, the code described in the previous paragraph will handle restarting the clock.

Following this, firmware will check if the button has been held down for the previous 0x2710 (10000d) cycles, which equal 2 seconds. If it has been down for that period, a function is called to display ‘batt’ on the display and the shutdown sequence is started. Clockstat is changed to ‘batdisp’, which will ensure that one second later a function will be called to calculate and display the current battery voltage.

The preceding 3 paragraphs describe what the firmware will do if it’s not in voltmeter mode. If it is in voltmeter mode, at this stage it only verifies that it’s not in ‘intensity set’ and if not, is changes clockstat to ‘batdisp’. This way, once every second the function is called to display the voltage.

Next the firmware checks if the board is in intenset mode and the button has been released. This indicates that the desired intensity was reached and the button was released in order to store this intensity in EEPROM. The set\_intensity function is called to store the current intensity value in EEPROM for future use, and the time is reset to zero and the device put into clockrun mode.

After this the ‘laststat’ variable is updated with the current button status, so that they can be compared on the next ISR in order to detect a button status change.

With the above-described housekeeping out of the way, the ISR goes on to drive the display. This is done by incrementing static variable two\_hundred\_mics which is a counter of how many 200uS rollovers have occurred. This variable counts from 0-10 then rolls over, so it has a 2mS period. Every time the 200uS rollover occurs, the value of two\_hundred\_mics is compared to the value of the Intensity. If it’s greater or equal to intensity, the current digit is turned off at its common anode. This is how the display’s intensity is controlled. When two\_hundred\_mics has been incremented 10 times, the ‘next\_digit()’ function is called. ‘next\_digit’ is what actually cycles through the 4 digits on the display. Its flow is explained below:

next\_digit() has a static variable named ‘placeholder’ that represents which of the 4 digits is currently active. Every time this function is called (every 2mS), placeholder is incremented. It counts from 0-3 and then rolls back over to zero. Array dig\_array[] represents which bit of PORTC must be made low in order to turn on the P-ch FET for the appropriate digit’s Anode. For example dig\_array[0] = 0x04. If PORTC-2 is cleared, this will activate the anode of the Least Significant Digit of DISP1.

Within next\_digit(), after placeholder has been updated, PORTD will be set to the appropriate value. This is done by looking at the value of numbers[] that corresponds to the value currently in digits[] for the current digit determined by placeholder. This will set the bases of the appropriate BJT’s to set up the cathodes of DISP1. Following this, a switch-case of clockstat will determine whether or not to set PORTD-3 which is the colon of the display. If the display is in clockrun mode (displaying time) then the colon will be enabled any time that an odd number of seconds is displayed. If the display is in voltmeter mode and displaying voltage, or if it’s in timer mode and displaying the battery, the colon cathode will be connected to ground when digit 2 is active, which will only turn on the bottom dot of the colon. This enables it to work as a decimal point.

After the appropriate value for PORTD has been established, the appropriate P-ch FET is turned on from PORTC and that concludes the next\_digit() function.

Going back to where next\_digit was called from, at this point the ‘two\_hundred\_mics’ is cleared so that the next sequence will start counting from zero, and the two\_milliseconds variable is incremented. Following this, two\_milliseconds is checked to see if it’s equal to 10 and if clockstat is in clockrun mode. If these conditions are met, an ADC reading will be initialized. This functionality is part of the minimum battery voltage check. Every 1 second (the point at which the timer actually increments, as described below), the ADC reference and divider are turned on. These values need some time to settle down, and 20mS (10 cycles of two\_milliseconds) is an appropriate settling period. After the settling time has passed, an ADC conversion will be initialized. The ADC will in turn generate an interrupt, and within the ISR the battery voltage will be checked to see if it’s below its minimum threshold (709LSB which equates to ~3.05V). If it is below the minimum value, a shut-down will be initiated.

When two\_milliseconds is incremented to a value of 500 (1 second!), a switch-case determines where to go based on the value of clockstat. The actions of the switch-case are outlined below for the various states that clockstat may be in:

* Clockrun: If clockstat is in ‘clockrun’ a function is called to increment the seconds. Within ‘inc\_seconds’, the first thing that’s done is to call ‘check3V’ which will turn on the ADC reference and the resistor divider to facilitate checking the battery level 20 milliseconds later. After that it will go through incrementing each of the ‘digits[ ]’ values. If digits[3] arrives at 10, this indicates that we’ve already gone through 99minutes and 59 seconds, and the setsleepstat(1) function is called so that the main loop will initiate shutdown.
* Batdisp: If clockstat is in ‘batdisp’, it’s time to display the ADC voltage and so the ‘display\_volts’ function is called, with the voltage to be displayed coming from the ‘Read\_ADC’ function, which is described below. Display\_volts will take an ADC reading which the ADC measures and scales to millivolts. A voltage reading of 2.555V will initially come to the ‘display\_volts’ function as 2555. First, this value is divided by 10 to give a resolution of tens of millivolts. For each of the digits, a modulus of 10 function is performed and the remaineder is assigned to digits[n], then the ADC value is divided by 10 to get to the next digit. For digits[3] if the value is zero, a value of 10 is assigned. Numbers[10] = 0x00 so this will result in the 3rd (most significant) digit being turned off. After the display\_volts function is called, clockstat is changed to ‘voltdisp’ which is described below. If the board is operating in voltmeter mode, this is where the button is checked to see if it’s been held down for >4 seconds, and if so clockstat will be changed to ‘intenset’.
* Intenset: If clockrun is in ‘intenset’, the ‘display\_intensity’ function is called. This will result in the value of intensity being incremented 1/second, and when it reaches 10 it will roll back over to 1. Also here, each of the values of digits[] is set to the current intensity value, so the current intensity setting will be displayed as the values are cycled through.
* Voltdisp: If clockrun is in ‘voltdisp’, it will stay here until either the button is released, at which point it will initiate shutdown by calling the ‘setsleepstat(1)’ function, or until the button has been maintained down for 4 seconds, at which point it will changed clockstat to ‘intenset’.

After that switch-case is complete, two\_milliseconds is reset to 0 so that the one-second count starts fresh, and after counting to 500 again the switch-case will be repeated. This concludes the ISR.

# ADC Operation

The ADC is used to measure battery voltage when running in Timer mode, or to measure some DC level when running in voltmeter mode. There are two ways that the ADC is used in this firmware: Interrupt-driven checks that the battery voltage is above a safe level when running in Timer mode, and via calls to the ‘read\_ADC()’ function.

* Interrupt-driven checks: When running in Timer mode, the ADC is activated once every second to check if the battery is above its safe minimum voltage of 3V. This is done by first turning on the GPIO’s to bias the ADC reference and terminate the resistor divider for the ADC measurement. This is accomplished by the ‘check3V()’ function that is called every time the ‘inc\_seconds’ function is called. ‘check3V()’ is also called at the very start of main.c just after the initialization. 20 milliseconds later, the Timer0 ISR will check if clockstat is in clockrun, and if so it will enable the ADC conversion-complete interrupt and then start a conversion. When the conversion is complete, the ISR will check the value of the ADC and if it’s below the safe threshold, will call ‘setsleepstat(1)’ which will initiate shutdown. Performing this task in the ‘background’ via ISR calls means there is no glitch on the display or the timing.
* Read\_ADC() calls: The read\_ADC function will first bias the ADC reference and terminate the resistor divider for the ADC measurement level. Note that in voltmeter mode, battery conservation isn’t a concern so these won’t have been turned off after the previous measurement. Following that, if the device isn’t in voltmeter mode (so it’s in timer mode) a 20mS delay will happen to allow the ADC reference and the input voltage to stabilize, then a conversion will be initiated. After starting conversion, the loop will wait until the conversion complete flag is set. Once the flag is set, if not in voltmeter mode the loop will then disable the ADC and the reference/divider to conserve battery, then copy the ADC results to the ‘voltval’ variable. Voltval is then multiplied by a coefficient that’s determined by the resistor divider feeding the ADC, which will result in a value that is equal to 100x the number of mV. The hi coefficient enables keeping a relatively hi resolution without using floats. The result of (ADC \* coefficient) is then divided by 100 to return the number of millivolts, and this value is returned from the function call to be displayed. The coefficient is calculated with the following equation:

Coefficient =

2.5 is the voltage reference and 1024 is the max value of the ADC. The x1000 coefficient is necessary to keep the result in the range of integers to avoid rounding errors.

# Shutdown

To preserve battery, a deep-sleep shutdown is entered into when the TimeBlock isn’t in use. To initiate shutdown, the firmware will call the ‘setsleepstat(1)’ function. This will result in the ‘gotosleep’ variable in main.c being set to 1. The main loop checks this variable and, if it is set the ‘shut\_r\_down’ function is called. The ‘shut\_r\_down’ function will stop Timer0, turn off the display and enable the interrupt tied to the button input. It then changes the sleep mode from ‘Idle’ to ‘Power-Down’ and enters sleep mode. In this sleep mode the device draws well under 1uA.

When a button press triggers the interrupt and wakes the device back up, it will restart the timer, call ‘sleepstat(0) to clear the ‘gotosleep’ variable, and change the sleep mode back to ‘Idle’