**Blick Propeller Clock**

**“noblink” Main Program Overview**

**Note:** The following is my current understanding of the program.
I guarantee the previous sentence and nothing else contained herein!
0x in front of a number simply means that number is in hex format
i.e. 0x19 is 19 hex (but 25 decimal).

Starts clock at 12:00:00.
Initializes all the variables used
in the program to appropriate
values.

Sets which ports will be used
and what direction they are
(input or output).

Enables interrupts.
Sets how crystal clock pulses
are counted by the internal timer
(1:1 or every clock pulse in this case).
Clears timers.

Checks if “blinker” has just cleared shade. If so
it resets digit and dot (column) indexes to initiate
a new display sequence and then checks how long it
took for a revolution (period_count & period_use)
in order to keep a stable display.

Each CALL displays a single vertical LED column of a single digit only.
The width of the display is based on the value of “period_use”.
The display sequence (successive calls) is initiated immediately by the blinker
clearing the shade (CHECK_INDEX). The sequence is run until digit &
dot (column) indexes have incremented to completion. After that
each call basically just returns having displayed nothing until the next
reset in CHECK_INDEX.

Checks if someone is trying to set the time with the key switches.

Increments the seconds, minutes and hours based on variables controlled by the
INTERRUPT SERVICE ROUTINE or as directed by CHECK_KEYS above.
This interrupt routine is run every 256 instruction cycles (256us or 256/1,000,000) when timer overflows from FF --> 00.

Here period_count is incremented. It is not allowed to “reset” if it gets to “FF” i.e. it is held at “FF”. Hence, 256us/interrupt x 256 (FF max count) = 65.5msec maximum (or 915.5 RPM is the minimum clock speed). Slower speeds will hit max period_count FF while faster speeds will have a lower period_count (but with progressively lower “resolution”).

Here the number of interrupts is counted to keep track of minutes. Every 234375 times this interrupt routine is run the minute flag (bit 1) is set.

i.e. 256us x 234375 = 60 sec (1 min)
234375 = 0x039387
0x000000 - 0x039387 = 0xFC6C79 (the number to count up from). Therefore bigtick_dbly is set to FC, bigtick_hi is set to 6C while bigtick_lo is set to 79 and the interrupt routine increments from this value (up to 0).

Here the number of interrupts is counted to keep track of seconds. Every 3907 times this interrupt routine is run the seconds flag (bit 5) is set.

i.e. 256us x 3907 = 1.000192 sec
3907 = 0x0F43
0x000000 - 0x0F43 = 0xF0BD (the number to count up from). Therefore sectick_hi is set to F0 while sectick_lo is set to BD and the interrupt routine increments from this value (up to 0).
Start

RAM_INIT Subroutine

Start

KEYS=07 (111)
Reset the time setting keys

HOURS=12
Start the clock at 12:00:00

MIN=0

SEC=0

DOT_INDEX=0
Reset the digit and column index

DIGIT_INDEX=0

BIGTICK_DBL=FC
Reset the 3 values that are used to count out a minute

BIGTICK_HI=6C

BIGTICK_LO=79

SECTICK_HI=F0
Reset the 2 values that are used to count out a second

SECTICK_LO=BD

PERIOD_CALC=40
Reset the rotation time value

Return
PORT_INIT
Subroutine

Set all "B" ports to outputs for LEDs (0x00 --&gt; TRIS PORTB)

Set "A" ports as follows;

b '00010111'

b0 = min
b1=10min
b2 = hrs
b3 = unused
b4 = rotation index

Return

A “0” sets a port to an output while a “1” sets it to an input.

b0-b2 for time-setting key switch input.

b4 for “blinker” input
**TIMER_INIT Subroutine**

Start

- **CLR Timer flag**
- **Enable global Interrupts**
- **Enable timer Interrupt**
- **Clear timer**
- **Clear Watchdog timer**

Set up timer as follows;

```
b '11011000'
```

- b7 = port B pull-ups disabled
- b6 = interruption on rising edge of INT pin
- b5 = Internal instruction clock cycle (CLKOUT)
- b4 increment on the H/L transistion on TOCK1 pin
- b3 = prescaler assigned to WDT (bypassed)
- b2, b1, b0 = prescaler 0,0,0 = 1:1

Return
Subroutine

Get state of port A

Check for change in rotation index

Edge?

Falling Edge?

PERIOD_DUP = PERIOD_COUNT

Clear PERIOD_COUNT, DIGIT_INDEX, DOT_INDEX

PERIOD_CALC = PERIOD_DUP?

Difference >2?

Difference >2?

Decrement PERIOD_CALC

PERIOD_USE = PERIOD_CALC

Return

Check the “blinker”

Clock has now swept completely through the “blinker”. The period_count is immediately duplicated to protect the value from an interrupt.

Clock is now set to update display and period_count has been reset for interrupt service routine to start new count.

To minimize jitter, period_calc is only recalculated if it has changed +/- from the old value by more than 3.

The result is saved as period_use which is used in the DISPLAY_NOW subroutine for setting display width.
The "to be displayed" hours, min or sec value here is a hex value, not a decimal value. See also the note in KEEP_TIME subroutine. i.e. 59sec = 0x59 = 89 decimal

Below is how the digit 0 (zero) is displayed. 1 = LED off, 0 = LED on, BIT 7 = no LED

One can see the 0's form a zero. The top bit 7 position is not displayed.

The box shows what one DISPLAY_NOW call might display. "BE" is returned from the character table and sent to PORTB. The TICK delay determines the width of the column (the box).

Size (or direction) of this jump is dependant on the program counter (PCL). 2 x is used as each branch requires 2 instruction cycles.

Tick = 0?
No
Delay
Tick = W
Tick = Tick - 1
Tick = 0?
No
Direction of display

PERIOD_USE or TICK is used to set the display width of a single column of LEDs within a digit. See also above right. By default that will also determine the overall display width.

Note: The “to be displayed” hours, min or sec value here is a hex value, not a decimal value. See also the note in KEEP_TIME subroutine.
CHECK_KEYS Subroutine

Start

Get state of port A

Any keys pressed?

YES

Store keys

Scratch = 64

Tick = FF

Decrement Tick

Tick = 0?

YES

Decrement Scratch

Scratch = 0?

YES

MINUTE key pressed?

NO

TENS key pressed?

NO

Scratch = 10 (A)

YES

Go to INC_MINUTES in KEEP_TIME Subroutine

NO

Go to INC_MINUTES in KEEP_TIME Subroutine

INC_HOURS

Hour key pressed?

NO

YES

hours = 12?

YES

NO

hours = hours + 1

hours = hours + 7

hours = 1

NO

IF test if hours + 7 causes a carry i.e. hours = 9?

YES

RETURN

From KEEP_TIME Subroutine
Note:
Program is working in Hex but display is interpreted as decimal hence jumps of 7.

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>08</td>
<td>Jump of 7</td>
</tr>
<tr>
<td>09</td>
<td>09</td>
<td>Jump of 7</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>25</td>
<td>Jump of 7</td>
</tr>
<tr>
<td>20</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

**KEEP_TIME Subroutine**

Start

Change in sec flag? (flag bit 5)

No

Clear seconds flag (flag bit 5)

Seconds = seconds + 7

Carry bit set?

YES

NO

Seconds = seconds - 6

Keep_time2

Change in min flag? (flag bit 1)

No

Clear minute flag (flag bit 1)

Clear seconds

Sectick_hi = F0

Sectick_lo = BD

From CHECK_KEYS subroutine

INC_MINUTES

In W test minutes = minutes + 7

Carry?

NO

Save minutes

Minutes = 00?

YES

Clear minutes

Increment minutes

Go to INC_HOURS subroutine (in CHECK_KEYS)

Return