



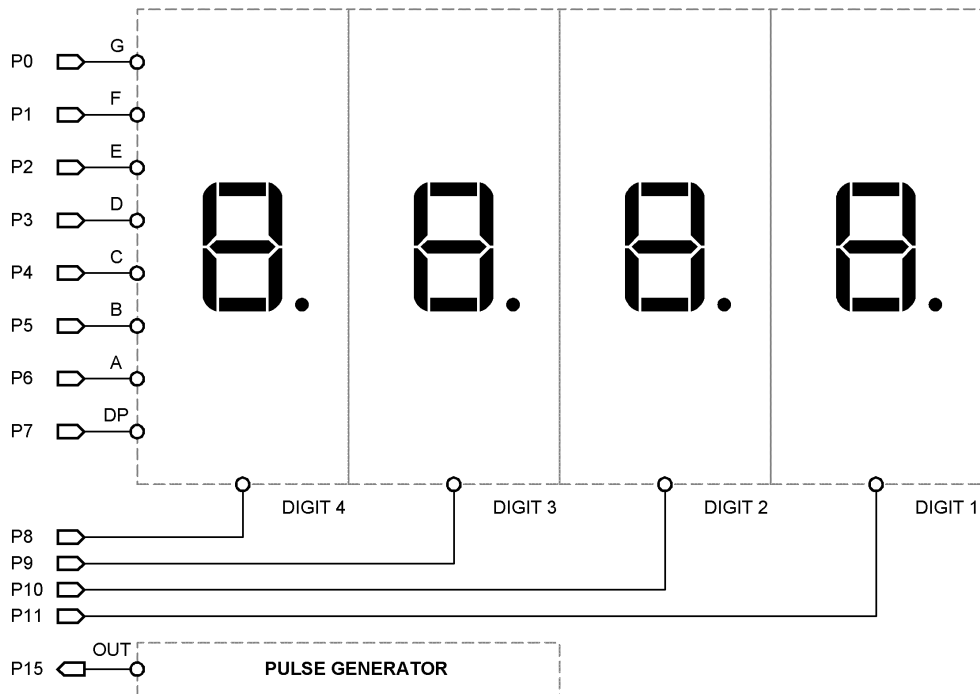
Experiment #10: LED Clock Display

The purpose of this experiment is create a simple clock display using four, seven-segment LED modules.

New PBASIC elements/commands to know:

- OutA, OutB, OutC, OutD
- DirA, DirB, DirC, DirD
- In0 - In15
- DIG

Building The Circuit



Experiment #10: LED Clock Display

```
' =====
'
' File..... Ex10 - Clock.BS2
' Purpose... Simple software clock
' Author.... Parallax
' E-mail.... stamptech@parallaxinc.com
' Started...
' Updated... 01 MAY 2002
'
' {$STAMP BS2}
'
' =====
'
' -----
' Program Description
' -----
'
' This program monitors a 1 Hz input signal and uses it as the timebase for
' a software clock.
'
' -----
' I/O Definitions
' -----
'
Segs          VAR      OutL          ' segments
DigSel        VAR      OutC          ' digit select
Tic           VAR      In15          ' 1 Hz Pulse Generator input
'
' -----
' Constants
' -----
'
DecPoint      CON      %10000000    ' decimal point bit
Blank         CON      %00000000    ' all segments off
'
Dig0          CON      %1111        ' digit select control
Dig1          CON      %1110
Dig2          CON      %1101
Dig3          CON      %1011
Dig4          CON      %0111
'
IsLow         CON      0             ' Tic input is low
IsHigh        CON      1             ' Tic input is high
```

Experiment #10: LED Clock Display

```
' -----  
' Variables  
' -----  
  
secs          VAR      Word      ' seconds  
time          VAR      Word      ' formatted time  
digit        VAR      Nib       ' current display digit  
  
' -----  
' EEPROM Data  
' -----  
  
'          .abcdefg  
'          -----  
DecDig       DATA    %01111110    ' 0  
              DATA    %00110000    ' 1  
              DATA    %01101101    ' 2  
              DATA    %01111001    ' 3  
              DATA    %00110011    ' 4  
              DATA    %01011011    ' 5  
              DATA    %01011111    ' 6  
              DATA    %01110000    ' 7  
              DATA    %01111111    ' 8  
              DATA    %01111011    ' 9  
  
' -----  
' Initialization  
' -----  
  
Initialize:  
  DirL = %11111111    ' make segments outputs  
  DirC = %1111        ' make digit selects outputs  
  DigSel = Dig0      ' all digits off  
  
' -----  
' Program Code  
' -----  
  
Main:  
  GOSUB Show_Time    ' show current digit  
  IF (Tic = IsHigh) THEN Inc_Sec    ' new second?  
  GOTO Main          ' do it again
```

Experiment #10: LED Clock Display

```
Inc_Sec:
  secs = (secs + 1) // 3600           ' update seconds counter

Waiting:
  GOSUB Show_Time                     ' show current digit
  IF (Tic = IsLow) THEN Main         ' if last tic gone, go back

  ' additional code could go here

  GOTO Waiting                       ' do tic check again

END

' -----
' Subroutines
' -----

Show_Time:
  time = (secs / 60) * 100           ' get minutes, put in hundreds
  time = time + (secs // 60)        ' get seconds, put in 10s & 1s
  Segs = Blank                       ' clear display
  ' enable digit
  LOOKUP digit, [Dig1, Dig2, Dig3, Dig4], digSel
  READ (DecDig + (time DIG digit)), Segs ' put segment pattern in digit
  IF (digit <> 2) THEN Skip_DP
  Segs = Segs + DecPoint             ' illuminate decimal point

Skip_DP:
  PAUSE 1                            ' show it
  digit = (digit + 1) // 4           ' get next digit
  RETURN
```

Behind The Scenes

The first two projects with seven-segment displays used only one digit. This project uses all four. A new problem arises; since the segment (anode) lines of the four displays are tied together, we can only show one at a time. This is accomplished by outputting the segment pattern then enabling the desired digit (by making its cathode low).

The goal of this program though, is to create a clock display, which means we want to see all four digits at the same time. While we can't actually have all four running at once, we can trick the human eye into thinking so.

Experiment #10: LED Clock Display

The human eye has a property known as Persistence Of Vision (POV), which causes it to hold an image briefly. The brighter the image, the longer it holds in our eyes. POV is what causes us to see a bright spot in our vision after a friend snaps a flash photo. We can use POV to our advantage by rapidly cycling through each of the four digits, displaying the proper segments for that digit for a short period. If the cycle is fast enough, the POV of our eyes will cause the all four digits to appear to be lit at the same time. This process is called multiplexing.

Multiplexing is the process of sharing data lines; in this case, the segment lines to the displays are being shared. If we didn't multiplex, 28 output lines would be required to control four seven-segment displays. That's 12 more lines than are available on the BASIC Stamp.

The real work in this program happens in the subroutine called `show_time`. Its purpose is to time-format (MMSS) the seconds counter and update the current digit. Since the routine can only show one digit at a time, it must be called frequently, otherwise display strobing will occur. This program will update the display while waiting for other things to happen.

The clock display is created by moving the minutes value (`secs / 60`) into the thousands and hundreds columns of the variable `time`. The remaining seconds (`secs // 60`) are added to `time`, placing them in the tens and ones columns. Here's how the conversion math works:

Example: 754 seconds

```
754 / 60 = 12
12 x 100 = 1200      (time = 1200)
754 // 60 = 34
1200 + 34 = 1234 (time = 1234; 12 minutes and 34 seconds)
```

Now that the `time` display value is ready, the segments are cleared for the next update. Clearing the current segments value keeps the display sharp. If this isn't done, the old segments value will cause "ghosting" in the display. A `LOOKUP` table is used to enable the current digit and the segments for that digit are `READ` from an EEPROM `DATA` table.

The StampWorks display does not have the colon (:) normally found on a digital clock, so we'll enable the decimal point behind the second digit. If the current digit is not a second, the decimal point illumination is skipped. The final steps are a short delay so the digit illuminates and the current digit variable is updated.

The main loop of this program watches an incoming square-wave signal, produced by the StampWorks signal generator. When set at 1 Hz, this signal goes from LOW to HIGH once each

Experiment #10: LED Clock Display

second. When this low-to-high transition occurs, the seconds counter is updated. The modulus operator (//) is used to keep seconds in the range of 0 to 3599 (the range of seconds in one hour).

When the seconds counter is updated, the display is refreshed and then the program waits for the incoming signal to go low, updating the display during the wait. If the program went right back to the top and the incoming signal was still high, the seconds counter would be prematurely updated, causing the clock to run fast. Once the incoming signal does go low, the program loops back to the top where it waits for the next low-to-high transition from the pulse generator.

Challenge

If the decimal point illumination is modified as follows, what will happen? Modify and download the program to check your answer.

```
secs = secs + (DPoint * time.Bit0)      ' illuminate decimal point
```