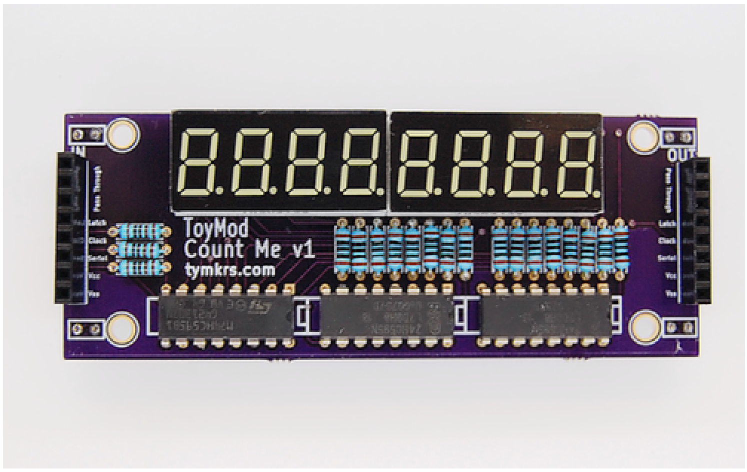
The Toymakers @ tymkrs.com
Questions? Please contact us:
feedback@tymkrs.com

DATASHEET



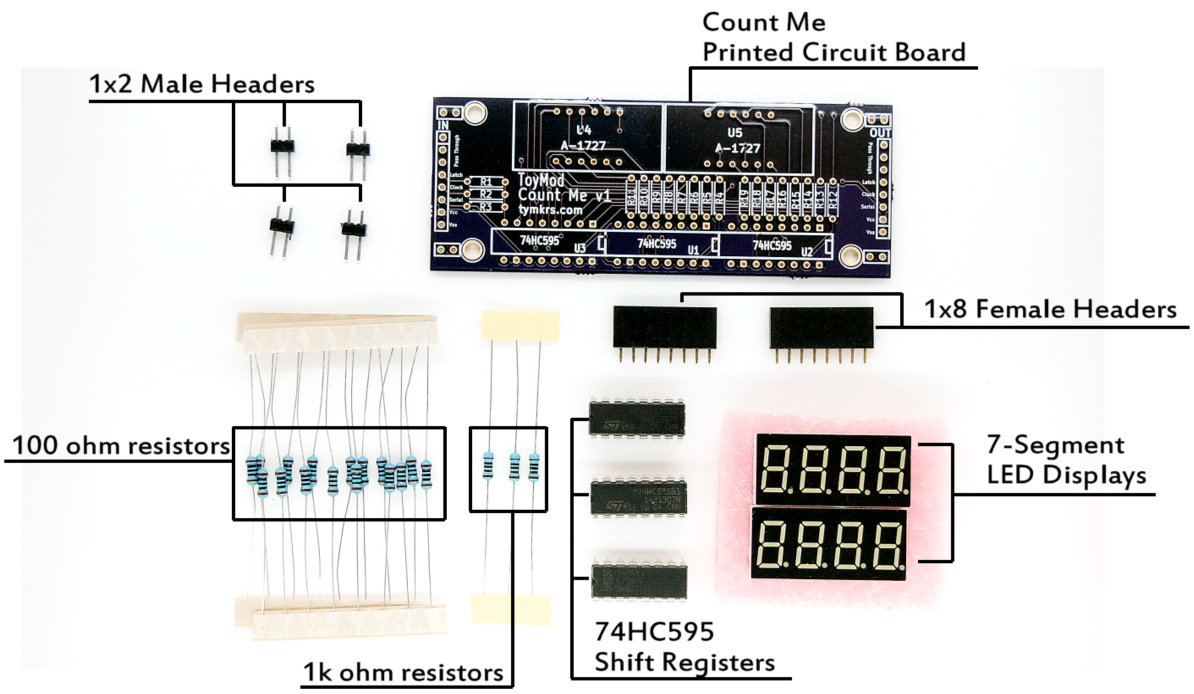
Count Me

7 Segment Display Kit

The Count Me kit is a 8 digit 7-segment display kit controlled by only 5 wires!

- Kit Type: Through-hole soldering
- Assembly instructions: In datasheet
- Function: 8 digit 7-segment display module
- This 7-segment display module allows easy shift register control of 8 digits.

KIT CONTENTS



Contents of the Count Me Kit:

- Count Me printed circuit board (93.01 x 35.68 x 1.60mm)
- 2 1x8 Female Headers
- 4 1x2 Male Headers

Electrical Components:

Reference	Quantity	Type	Value
R1 - R3	3	Resistor, 1/4W	1k ohm
R4 - R19	16	Resistor, 1/4W	100 ohm
U1 - U3	3	Serial to Parallel Shift Register	74HC595
U4 - U5	2	4 Digit 7-Segment Displays	Common Cathode

Absolute Maximum Ratings

75HC595 Shift Register Maximal Operating Conditions
Datasheet: http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

Parameter	Maximal Ratings	Unit
Supply Voltage	-0.5 – +7.0	V
Operating Temperature	-40 to +125	ºC
Output Current (Qn)	+/- 35 per pin	mA
Supply Current	70	mA

Note: Absolute maximum ratings are stress ratings only and functional device operation is not implied. The device could be damaged beyond Absolute maximum ratings.

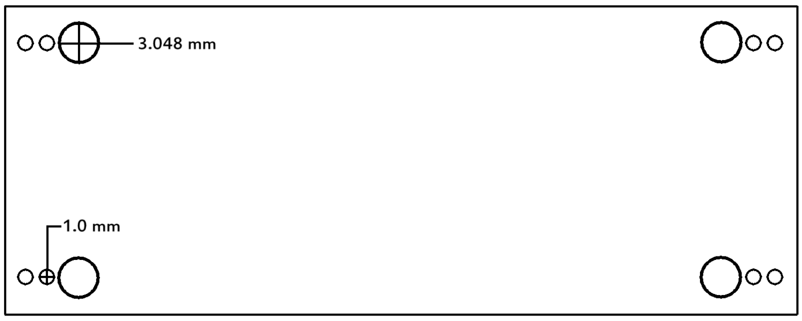
Tools and material required for assembly (not included with the kit):

- Soldering iron
- Solder
- Wire clippers

User provided items required for function:

- Microcontroller sending serial data to the shift register to control information displayed by the 7-segment displays.

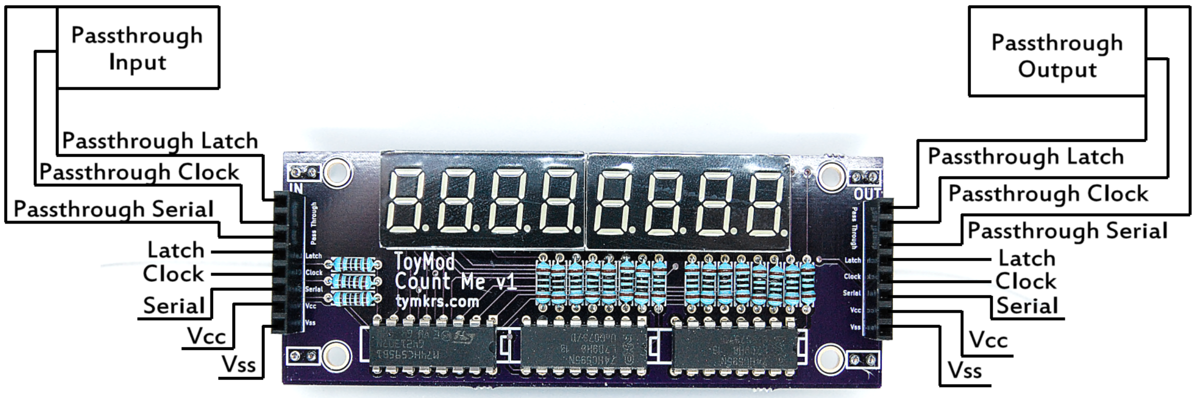
Mounting Holes:



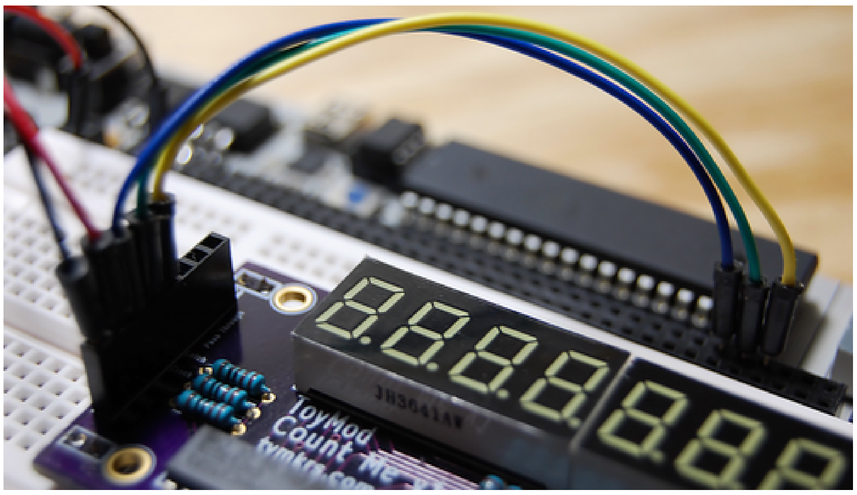
Additional physical/electrical specifications:

- Printed Circuit Board size: 3.66 x 1.41 x 0.063" (93.01 x 35.86 x 1.60mm)
- PCB thickness: 0.063" (1.60mm), not including any components
- PCB thickness: 0.472" (12mm), max height with all components
- Mounting holes: 4 holes provided. 1X2 header holes in all four corners are available as well.

Board Connections:



Additional Photo:



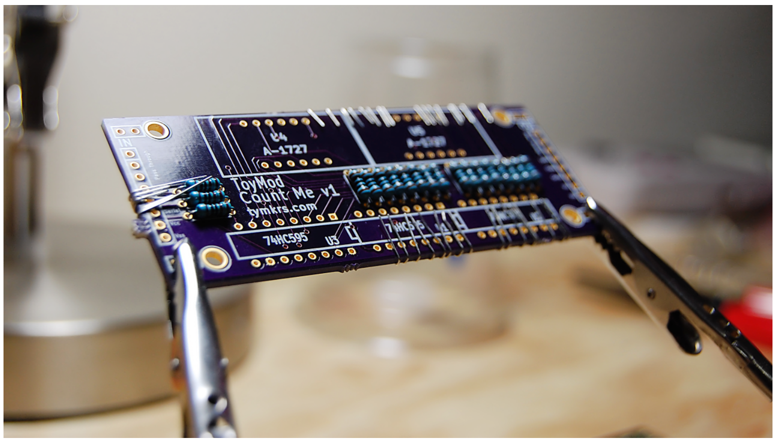
PCB on Professional Propeller Development Board

Assembly Instructions

Build Notes:

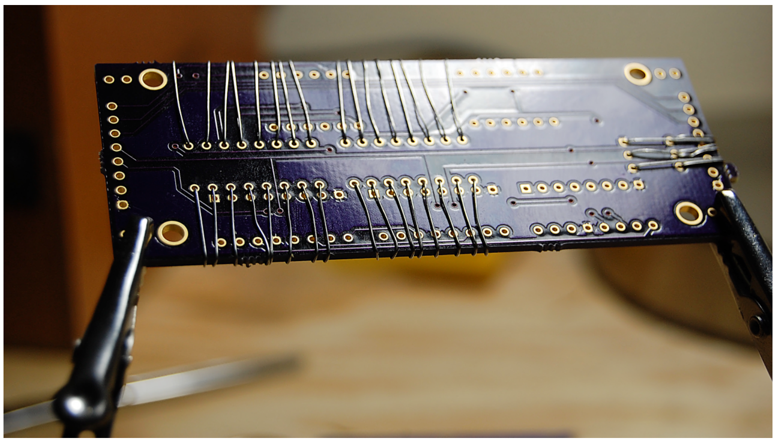
Note, the following instructions can be done in pretty much any order. I personally place all of the components on before soldering, but you are welcome to put in a component, solder it, then repeat with the rest of the components.

Step 1: Put in the components!



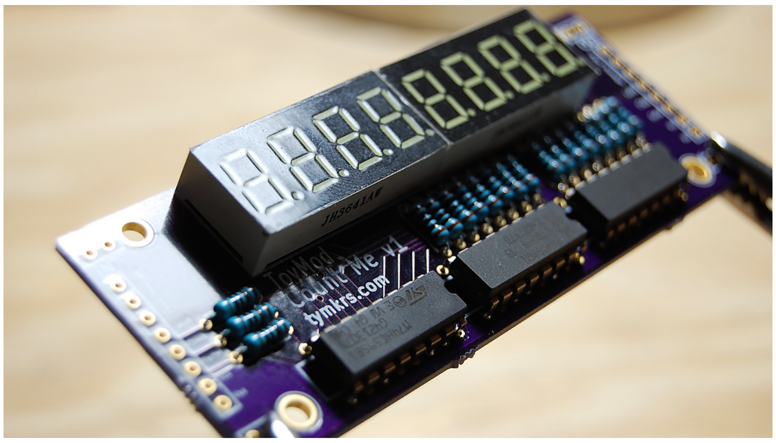
R1 - R3: 1k ohm Resistor

These 3 resistors go into R1 - R3 slots – polarity does not matter. (Brown-Black-Black-Brown-Brown)



R4 - R19: 100 ohm capacitor

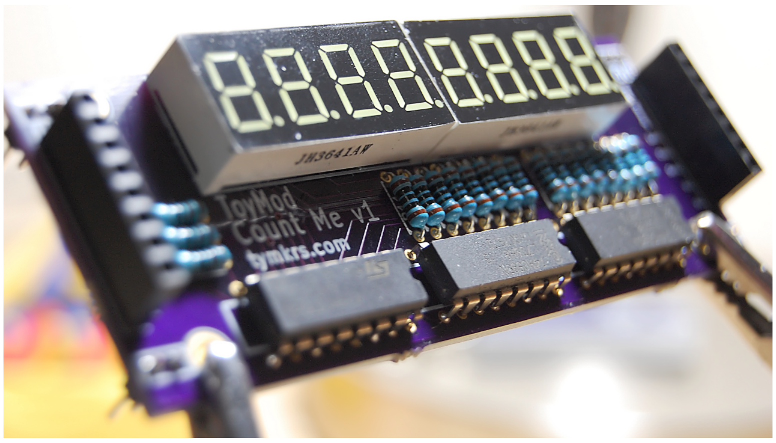
These resistors go into R4 – R19 slots – polarity does not matter (Brown – Black – Black – Black - Brown)



U1 – U3: 74HC595

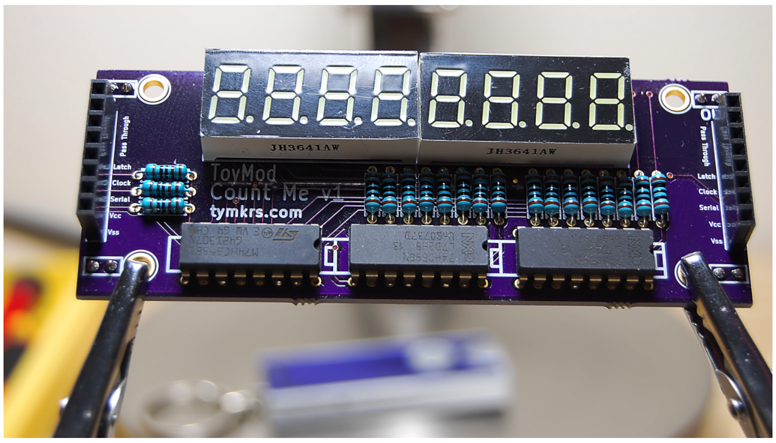
Be sure to line the divot in the chip to the graphic on the board!

DIP sockets can be soldered into the PCB so that the shift registers can be changed if necessary.



U4 - U5: 7-segment displays

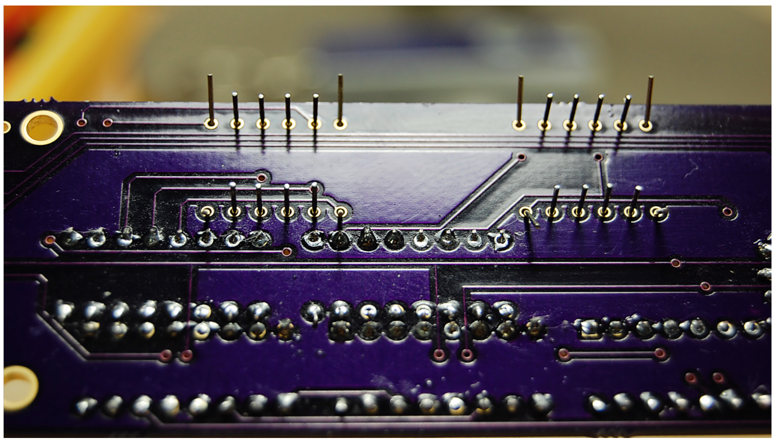
Make sure to solder these with the decimal points facing the bottom of the board.



1x8 Female Headers

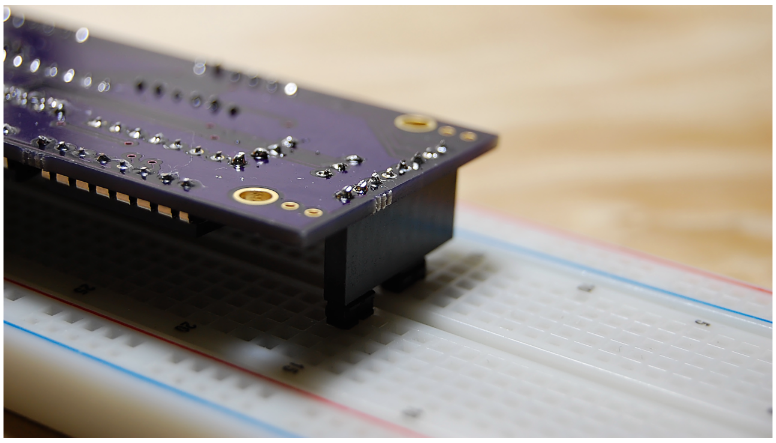
These are to be soldered on the ends! We suggest soldering them facing upwards.

Step 2: Solder the electrical components in!



I use 60/40 0.38mm gauge solder for these pads. But also have 1.3mm gauge solder for the larger solder pads.

Step 3: Trim the extra leads off of the electrical components!



Once you're done soldering the components, it's a good idea to clean up all of the extra leads from the electrical components. We use a spare pair of nail clippers – it works quite easily!

Helpful Links

- **Driver OBJ:** http://obex.parallax.com/object/102
- **Assembly Video**: https://www.youtube.com/watch?v=ih0g9t6YMRc
- http://tymkrs.tumblr.com/post/54105338290/tymkrs-count-me-7-segment-led-display-kit

Example Code

```
File: CountMe_Simple_74HC595_Demo.spin
Version: 1.0
Copyright (c) 2015 Tymkrs
See end of file for terms of use.

Author: Whisker 'http://tymkrs.com/code/CountMe_Simple_74HC595_Demo.spin
```

```
{
HISTORY:
  This object is made as an example for using the CountMe ToyMod kit from
http://tymkrs.com/

USAGE:

   • Connect CountMe pins Latch, Clock, and Serial to
     Propeller Pins Latch_Pin, Clock_Pin, and Serial_Pin

   • Connect CountMe pins Vcc to +3vdc and Vss to GND
     (Vcc to +5vdc will result in brighter displays. Maybe too bright? Your
call!)
}

Con

  'Initialize the system clock speed

  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

  Latch_Pin  = 3
  Clock_Pin  = 4
  Serial_Pin = 5

OBJ

  'Include the driver for the 595 shift registers

  Shift :       "Simple_74HC595"

Var

  'Allocate hub memory space in which to run the Display refresh loop
  long  DisplayStack[32]

  'Allocate space for the eight character offsets
  'Used to select which characters to Print on the display
  byte  ScreenA[4]
  byte  ScreenB[4]

Pub Main | Index, OnesPlace, TensPlace, SixteensPlace

  'Start the Simple 74HC595 driver object in a new cog
  Shift.init(Clock_Pin, Latch_Pin, Serial_Pin)

  'Start the Display refresh loop in a new cog, allowing you to print characters
to the screen from any routine
  cognew(Display, @DisplayStack)
  repeat

    repeat Index from 0 to 7

      'Here is an easy way to print strings onto the display
      'Supports these characters: 0123456789AbcdEFghijKlnopqrStuyZ _-.[]=|
CILNORUV
      case Index
        0:
          Print(String("count   "))
        1:
          Print(String("pLAy    "))
        2:
          Print(String("StOp    "))
        3:
          Print(String("run     "))
        4:
          Print(String("SIgNAL  "))
        5:
          Print(String("dEAd    "))
        6:
          Print(String("bEEF    "))
        7:
          Print(String("intro   "))

      'Here is an easy way to count to 99 on the display
      repeat TensPlace from 0 to 9
       repeat OnesPlace from 0 to 9
        ScreenB[0] := OnesPlace
        ScreenB[1] := TensPlace
        waitcnt((clkfreq / 32) + cnt)

      'Here is an easy way to count to 255 in hexadecimal on the display
      repeat SixteensPlace from 0 to 15
       repeat OnesPlace from 0 to 15
        ScreenB[0] := OnesPlace
        ScreenB[1] := SixteensPlace
        waitcnt((clkfreq / 4) + cnt)

Pub Print(Message) | CharIndex, MapIndex

  'This function compares each character in the Message to characters in
CharacterMapKey
  'If it a match is found, the offset of the match is stored into ScreenA or
ScreenB (depending on which half of the Message the character is in)

  repeat CharIndex from 0 to 4
    repeat MapIndex from 0 to 47
      if BYTE[Message][CharIndex] == CharacterMapKey[MapIndex]
        ScreenA[3 - CharIndex] := MapIndex
        quit
  repeat CharIndex from 0 to 4
    repeat MapIndex from 0 to 47
      if BYTE[Message][CharIndex + 4] == CharacterMapKey[MapIndex]
        ScreenB[3 - CharIndex] := MapIndex
        quit

Pub Display | Index

  'This function runs in its own cog, constantly updating the shift registers
  'The display is a common cathode matrix
  'Only a single digit should be enabled on each screen at any time

  'Using the offsets stored in ScreenA and ScreenB, the CharacterMap table, and
```

```
one of the four DigitEnableMasks it builds a 24 bit number which is shifted out
to the 595s

  'The format of this number is:
  'Bits  0-7   Digit Enable Bits                           (low is on)
  'Bits  8-15  Screen B Segment Enable Bits                (high is on)
  'Bits 16-23  Screen A Segment Enable Bits                (high is on)

  repeat
    repeat Index from 0 to 3
      Shift.Out(CharacterMap[ScreenA[Index]] << 16 |
CharacterMap[ScreenB[Index]] << 8 | DigitEnableMask[Index])

Dat

  'Used by Print to find the offsets for each supported character
  CharacterMapKey       byte    "0123456789AbcdEFghijKlnopqrStuyZ _-.[]=|
CILNORUV"

  'Segment enable bit masks for each of the 48 supported characters
  'The format for this pattern is:
  'Dot, Middle, Top Left, Bottom Left, Bottom, Bottom Right, Top Right, Top
  'High is enabled

  CharacterMap          byte    %00111111, %00000110, %01011011, %01001111,
%01100110, %01101101, %01111101, %00000111, %01111111, %01101111, %01110111,
%01111100, %01011000, %01011110, %01111001, %01110001, %01101111, %01110100,
%00000100, %00001110, %01110110, %00110000, %01010100, %01011100, %01110011,
%01100111, %01010000, %01101101, %01111000, %00011100, %01101110, %01011011,
%00000000, %00001000, %01000000, %10000000, %00111001, %00001111, %01001000,
%00110110, %01100001, %00000010, %00111000, %00100011, %01100011, %00100001,
%01100010, %00111110

  'Digit enable bit masks for each of the four digits. Enables the digit on both
Screens.
  'The format for this pattern is:
  'Digit 0 Screen A, Digit 1 Screen A, Digit 2 Screen A, Digit 3 Screen A, Digit
0 Screen B, Digit 1 Screen B, Digit 2 Screen B, Digit 3 Screen B
  'Low is enabled

  DigitEnableMask       byte    %01110111, %10111011, %11011101, %11101110

{{
```

```
}}
```