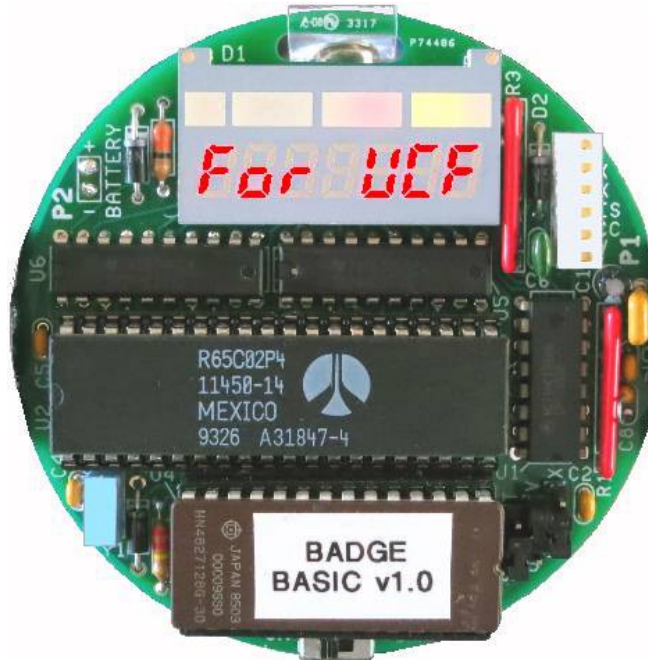# A 6502 BADGE

for the 2019 Vintage Computer Festival

by Lee Hart, Daryl Rictor, and Josh Bensadon – 11 May 2020

What is it?

A cool retro nametag that displays a scrolling message of your choice

Celebrates the 40th anniversary of the Apple][, Commodore PET, and Atari VCS

Classic user interface that is functional and educational

Minimum size, cost, and parts count

Maximum fun!

Description: The Badge is a complete working 6502 computer, built entirely with vintage parts and technology. Powered by batteries or a USB port, its LEDs display up to a 32-character message. A serial port provides user interaction with any computer. A BASIC interpreter and monitor program in ROM make it a great way to show how simple programming can be. It has:

- A 65C02 microprocessor, running at 2 MHz
- 2K RAM (expandable to 16K or 32K), with battery backup
- 16K EPROM (expandable to 32K), with floating-point BASIC and 6502 machine-level monitor
- Two 8-bit output latches
- 7-digit 7-segment LED display, plus annunciators
- Software-driven 9600 baud TTL serial I/O port
- And, just a few components to tie it all together

For the complete manual, software, and more information, go to **http://www.sunrise-ev.com/6502.htm**

Table of Contents

6502 Badge Parts List

QTY	ID#	Description	Source
1	C1	22uF 16vdc electrolytic capacitor	mouser.com 667-ECE-A1CKA220
4	C2,4,5,7	0.1uF 50v X7R ceramic capacitor	jameco.com 332672
1	C3	0.56uF (560nF) 50v ceramic capacitor	mouser 80-C330C564K5R
1	C6	3900pF 50v ceramic capacitor	mouser.com 80-C320C392K5R
1	C8	0.01uF 50v ceramic capacitor	mouser.com 581-SR155C103KARTR1
1	D1	7-segment 7-digit LED display	Rohm LS-2074M2G (get it from us)
3	D2,3,4	1N5818 or 1N5819 Schottky diode	jameco.com 177957
1	P1	6-pin connector Molex KK	mouser.com 538-22-18-2061
1	P2	wires to battery holder	
1	R1	22K x 4 8-pin SIP resistor network, isolated	mouser.com 268-22K-RC
1	R2	200K 1/4w resistor	jameco.com 691614
1	R3	2K x 4 8-pin SIP resistor network, isolated	mouser.com 268-2.0K-RC
1	R4	39 ohm 5% 1/4w resistor	jameco.com 690507
1	S1	switch SPDT micro slide	mpja.com 18453-SW
1	U1	74HC139 dual 1-of-4 decoder	mouser.com 595-SN74HC139N
1	U2	R65C02 microprocessor	jameco.com 43166
1	U3	27C256 EPROM with 32K.ROM file, labeled "BADGE 32K BASIC v1.0"	jameco.com 39731
		(or 27C128 EPROM with 16K.ROM file)	(jameco.com 101101)
1	U3a	28-pin IC socket, 0.6" wide	jameco.com 40328
1	U4	RAM 0.3" wide, 2K CXK5814 (basic kit),	jameco.com 242683
		(or 32K CY7C199 etc. with deluxe kit)	(jameco.com 242376)
2	U5,U6	74HC273 octal latch	jameco.com 45743
1	Y1	2 MHz ceramic resonator with capacitors	mouser.com 520-ZTT200MG
1	PCB	Badge printed circuit board	(it's from us again)

Extra parts with Deluxe Kit:

QTY	ID#	Description	Source
1	HW-597	USB to TTL serial adapter	ebay
1	3-AAA	3-cell AAA battery holder	jameco.com 216303
3	AAA	Nimh cells	jameco.com 231095
1	clip	to hang the Badge on your shirt etc.	scrounge from an old convention badge

Notes:

An IC socket is only supplied for EPROM U3. You can socket the others ICs if you like. For the lowest height, use socket pins for RAM U4 (digikey.com ED5037-ND or mouser.com 575-055210).

Rev.A – Original release; it only went to developers, and had a slightly different reset circuit.
Rev.B – R1 was 10K, now 22K. R2 was 220K, now 200K. C3 was 22uF, now 0.56uF. Added wire from S1 "off" position to U2 pin 37 to stop the clock and disable memory when off/standby.
Rev.C – Add jumper "R". Short R for a Rockwell R65C02 at U2. Leave R open for WDC W65C02 at U2. Text messages and PCB changed from "VCFMW" to simply "VCF" to support other shows.
Rev.D – A new batch of rev.C boards for VCFMW-14. Scrolling message says "6502 badge for VCF".
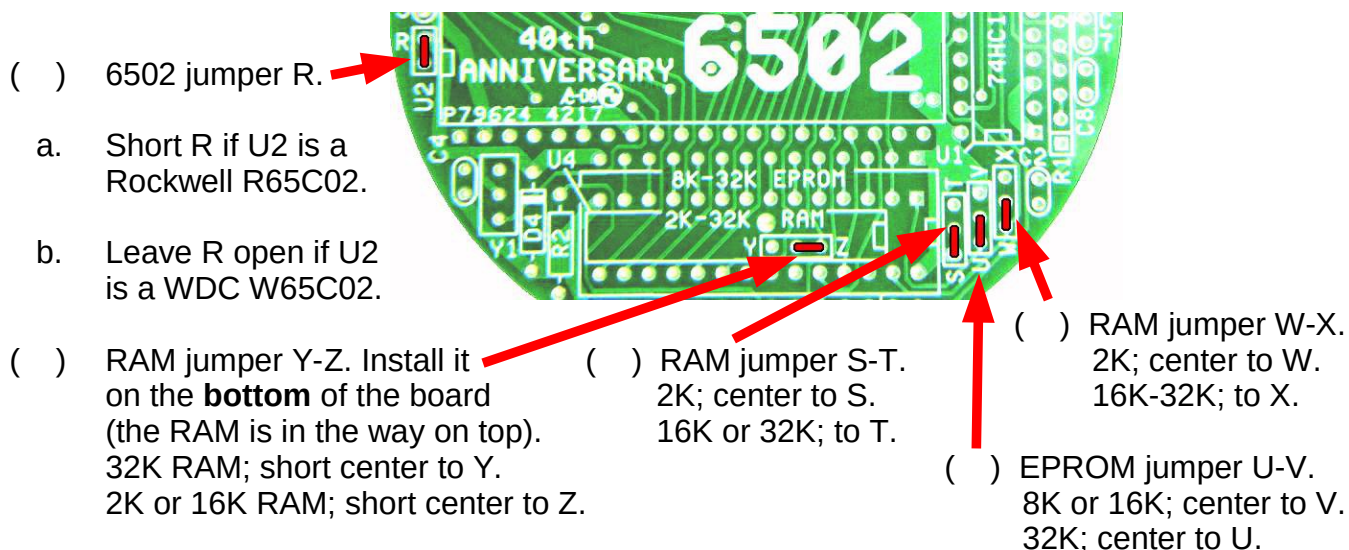
Assembly

Check the parts list to be sure you have all the parts. Mount all parts on the TOP (printed) side and solder them on the bottom side. Put a check mark in the box as you install each part.

( ) R4 39 ohm resistor (orange-white-black-gold). Only install R4 if using rechargeable batteries!

( ) D3 1N5818 diode. Banded end top (↑)

( ) R3 2K x 4 SIP (red, marked L83S202). Pin 1 down (↓)

( ) P2 battery holder. red wire on top (+), black on bottom (-)

( ) D1 LED display. Bumps on top (↑)

( ) U5 74HC273. Pin 1 notch is on left end (←)

( ) U6 74HC273. Pin 1 left (←)

( ) C5 0.1uF (yellow, marked 104).

( ) U2 R65C02P4 Pin 1 left (←)

( ) C4 0.1uF (yellow, marked 104).

( ) Y1 2MHz resonator (blue, marked 2000A). Not polarized.

( ) D4 1N5818 diode. Banded end on top (↑)

( ) R2 200K resistor (red-(black-black-orange-brown)

( ) U4 CXK5814 2K RAM (Note 1). Pin 1 notch on the **RIGHT** (→) but with the 4 holes at the right end EMPTY. The lettering on U4 will be upside down.

( ) D2 1N5818 diode. Banded end on top (↑)

( ) C6 3900pF (green, marked 2A392K).

( ) P1 6-pin connector. Top holes right (→) Cut off "ears" on left

( ) C1 22uF capacitor. (black) White stripe (- wire) down (↓)

( ) C3 0.56uF (yellow, marked RMC 56K).

( ) C7 0.1uF (yellow, marked 104).

( ) U1 74HC139. Pin 1 down (↓)

( ) C8 0.01uF (yellow, marked 103).

( ) R1 22K x 4 SIP (red, marked L83S223). Pin 1 end down (↓)

( ) S1 switch.

( ) U3 EPROM socket.
  a. Cut the three bars to make it into two 14-pin socket strips.
  b. Install the strips at U3, one on each side of U4.
  c. U3 EPROM (labeled BADGE 32K BASIC v1.0). Plug it into socket U3 with the pin 1 notch on the **RIGHT** (→) and the label right side up.

( ) C2 0.1uF (yellow, marked 104).

Note 1: The Deluxe kit U4 has a 32K RAM, which uses all 28 holes. See "jumper options" on page 5.

VINTAGE COMPUTER FESTIVAL MIDWEST WWW.VCFMW.ORG

74HC273   74HC273

40th ANNIVERSARY **6502**

8K-32K EPROM
2K-32K RAM

S1 ON   OFF

Jumper Options

There is one jumper to select the brand of 6502, three to select the RAM size, and one for the EPROM memory size. These jumpers must be set to match the chips used with your Badge. The photo shows the jumper positions for a **Rockwell R65C02** CPU, a **2K RAM** (CXK5814), and a **32K EPROM** (27C256). These are the standard parts supplied with the basic kit. If you use different parts, the jumpers must be changed! (see below). To install a jumper, solder a piece of scrap wire from the CENTER hole to ONE of the two outer holes as shown.

( )   6502 jumper R.

   a.   Short R if U2 is a Rockwell R65C02.

   b.   Leave R open if U2 is a WDC W65C02.

( )   RAM jumper Y-Z. Install it on the **bottom** of the board (the RAM is in the way on top). 32K RAM; short center to Y. 2K or 16K RAM; short center to Z.

( )   RAM jumper S-T. 2K; center to S. 16K or 32K; to T.

( )   RAM jumper W-X. 2K; center to W. 16K-32K; to X.

( )   EPROM jumper U-V. 8K or 16K; center to V. 32K; center to U.

EPROM: Three different size EPROMs can be used. The kit comes with a programmed 27C256 EPROM marked "BADGE 32K BASIC 1.0". If you're using your own parts, you can use any of the following EPROMs, programmed with your own program. Ignore any letters or numbers at the beginning or end; only use the number in the middle. For example, if it is marked "NMC27C256Q-25" then it is a 32K EPROM.

EPROM	Part #	Jumper
8K	2764, 27C64	V
16K	27128, 27C128	V
32K (supplied)	27256, 27C256	U

RAM: Look at the part number of the RAM supplied with your kit. Again, ignore any letters or numbers at the beginning or end. The standard part is a 2K 5814 marked "CXK5814P-45L". Two different size RAMs can be used, and jumpered for three different amounts of memory.

| RAM	Part #	Jumpers
2K (supplied)	5814, 6116, 7C128	S, W, Z
16K	58256, 6206, 7C199	T, X, Z
32K	58256, 6206, 7C199	T, X, Y

Let's See It Work!

Now for the big moment. Connect a source of 4-5v DC to P1 or P2. If you have the Deluxe kit, this can be from the USB adapter (but read Appendix B first!), or three AAA cells in the battery holder. Switch S1 on, and you should be rewarded with the default scrolling message "6502 badge for VCF-14"!

Note: You many have to use some other batteries for your first test, as the rechargeable AAA cells sent with the kit are shipped DEAD. The Chinese send them that way. The 1800 mAH claim on them is also a lie; they are more like 300 mAH (still, 8-10 hours of running time).

The PCB has holes for two #2 self-tapping screws to mount the battery holder on the back. Screws are not included, because I haven't found a source for them (I stole mine from a broken toy). Also, there is a bare wire on the back of the battery holder: **Insulate** it with a few layers of tape so it won't short to anything on the board!

Block Diagram
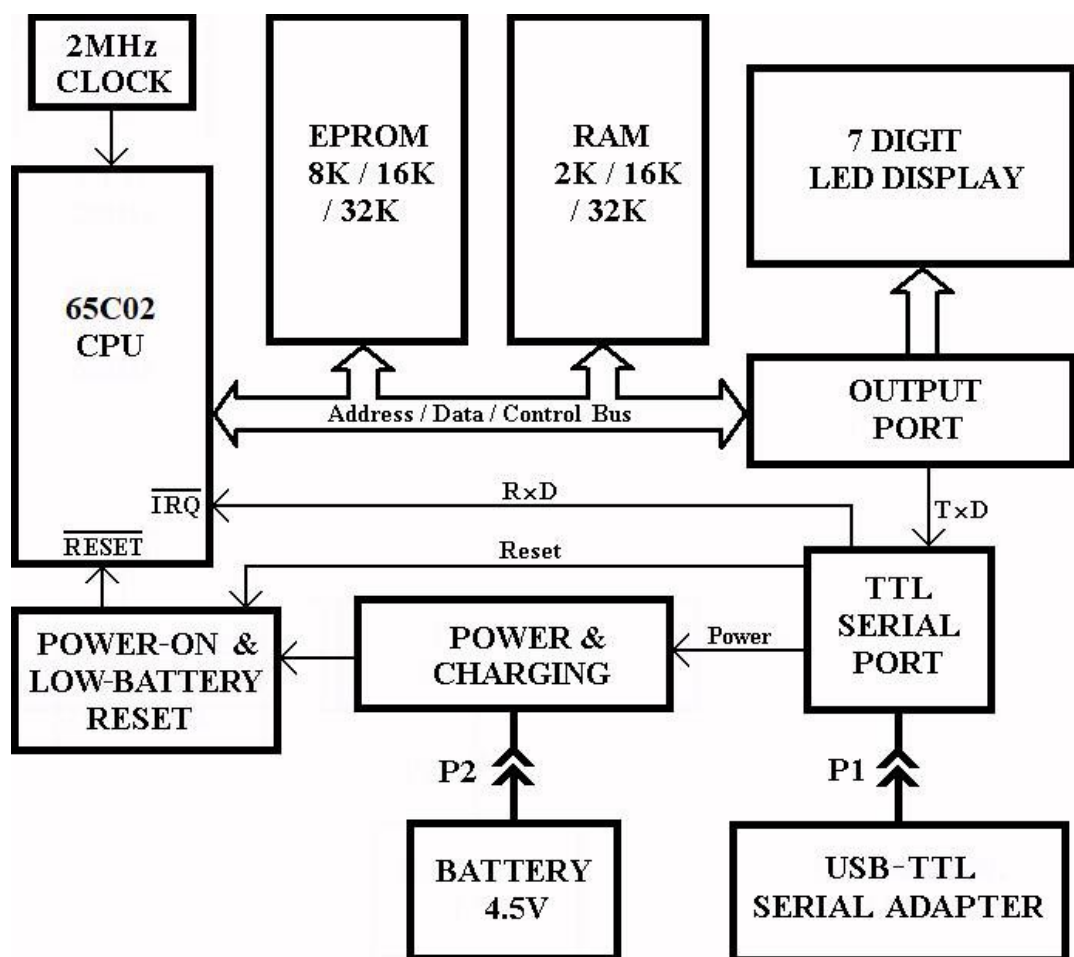
Logically, the Badge is a complete computer with all the standard building blocks. Electrically, it has been simplified to minimize the size and number of parts used. See Appendix A for the full schematic.

Memory Maps

The Badge has 3 RAM size options: 2K, 16K, or 32K. RAM always starts at address $0000, and fills the space up to $3FFF regardless of the RAM chip size (partial address decoding).

With a 2K RAM chip, its contents are repeated 8 times from 0-16K ($0000-$3FFF). In other words, $0000, $0800, $1000, $1800, etc., are all the same address. 16K-32K ($4000-$7FFF) is free, in case the user wants to interface additional memory or I/O.

With a 32K chip, the Badge can be jumpered to address it from 0-16K (leaving room for additional memory or I/O expansion), or from 0-32K (to maximize the amount of RAM, but leave no addresses free for expansion). When jumpered for 16K, only the lower half of a 32K RAM chip is accessible.

RAM Memory Map

| Address Start	End			
7800	7FFF			
7000	77FF			
6800	6FFF	16K	16K	
6000	67FF	undecoded	undecoded	
5800	5FFF	free space	free space	
5000	57FF			
4800	4FFF			
				32K
4000	47FF			RAM
3800	3FFF	2K mirror		
3000	37FF	2K mirror		
2800	2FFF	2K mirror		
2000	27FF	2K mirror		
			16K	
1800	1FFF	2K mirror	RAM	
1000	17FF	2K mirror		
0800	0FFF	2K mirror		
0000	07FF	2K RAM		

The Badge has 3 ROM options: 8K, 16K, or 32K. ROM always starts at $8000 and fills all the space to $FFFF. The same repeating goes for ROM (i.e. addresses are partially decoded). Thus an 8K ROM will be addressed at $8000-$9FFF and will be repeated at $A000-$BFFF, $C000-$DFFF, and $E000-$FFFF.   A 16K ROM will be addressed at $8000-$BFFF, and repeated at $C000-$FFFF.

ROM Memory Map

Address				
Start	End			
E000	FFFF	8K ROM	16K ROM	
C000	DFFF	8K mirror		32K ROM
A000	BFFF	8K mirror	16K mirror	
8000	9FFF	8K mirror		

The repeating is a byproduct of the reduced address decoding that comes with the minimized parts design goal. The 65C02 requires RAM at $0000-$01FF for zero page and stack usage. It also needs ROM to be located at the top of the memory map for the Reset and interrupt vectors to be functional.

Input / Output
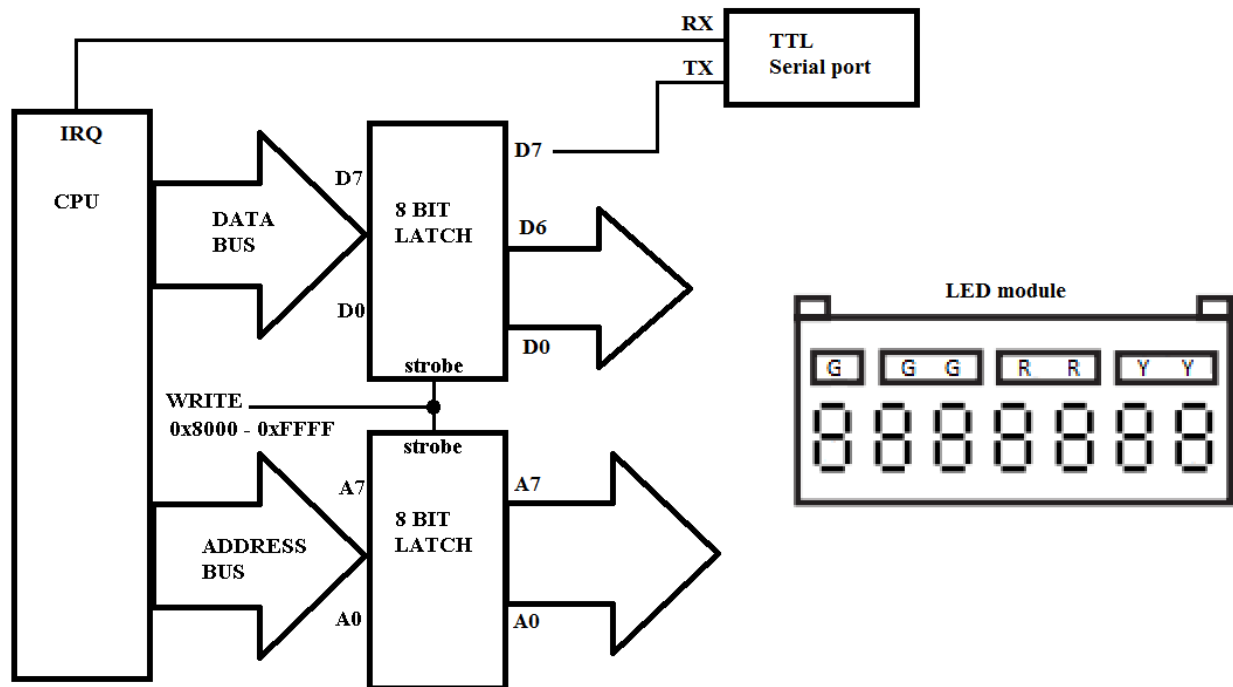
The two 74HC273 8-bit latches are configured in a unique way. They share a common clock which is derived from a memory write pulse to the ROM address space. All 8 data bus lines and the low 8 address bus lines get stored in the latches any time a write to addresses $8000-$FFFF occur. These latches drive the 8 rows and 7 columns of the multiplexed LED module. The block diagram shows the logical data bus and address bus connections. In reality, the address and data pins are intermixed between the two latches to simplify board layout. The important thing to see is that each input is connected to the proper output. D0 to D0, D5 to D5, A4 to A4, etc.

This "screwy" setup is a consequence of several conflicting requirements:

- It allows a single instruction to simultaneously update all 15 LED output control bits (so the display doesn't flicker).

- It minimizes the hardware needed (no UART).

- It made the PC board layout practical (without multilayer or tiny traces and spaces),

Bit 7 of the Data latch is not needed for the LEDs, so it is used for the serial output. The IRQ pin on the processor is used for the serial input. Together with software, these provide simple, minimalist interrupt-driven serial port to provide user input and output. A system monitor and BASIC interpreter are accessed via this port.

LED Display

The data bus latches drive the LED Anodes (segments), and the address bus latches drives the LED cathodes (digits).

Data bus latch bits 0 - 6 select the LED segments (A - G). Since the digits have only 7 segments, data bit 7 is used for the Transmit Data (TxD) output for the serial IO port.

Address bus latch bits 1 – 7 select the common cathodes of each of the 7 digits. Since there are only 7 digits, address bus latch bit 0 selects the cathodes of the 5 discrete LED annunciators. (There are 7 annunciator LEDs; but two of them are used as diodes in the reset circuit).

Here is a graphical representation of how the LED module maps to the address and data bus latches.
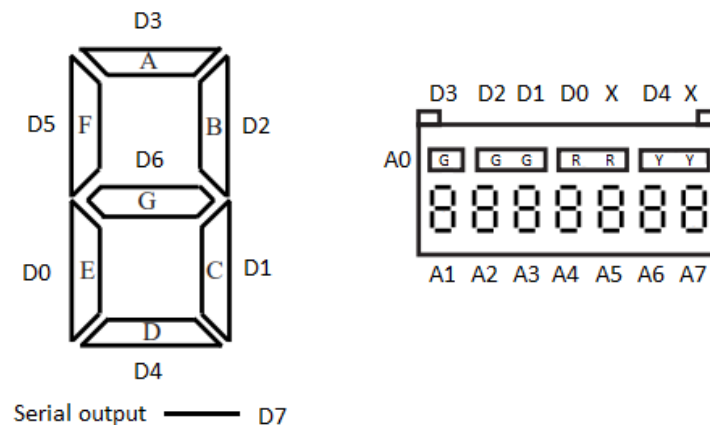
The Anode signals are active when high, and the cathode signals are active when low. For example, to turn on segment A of the left-most digit, data bit 3 must be set high and address bit 1 set low. This is achieved by writing the value $08 to address $80FD.

Let's break that down:

```
        Data bits              Address bits
        76543210               76543210
        00001000 = $08         11111101 = $FD
```

To write the latches, do a Write to the ROM space from $8000 to $FFFF. So add $8000 to address $FD to get $80FD. Note that writing to $95FD or $FEFD would produce the same result. Any address page at or above $8000 will work.

Serial Port

The serial port is a TTL asynchronous data transfer type using Transmit Data, Receive Data and a common ground. It is similar to the RS-232 standard, but uses 5V and 0V logic levels and non-inverted data (Idle = 5V = logic 1). You can use a TTL to RS-232 adapter to talk to any traditional terminal or computer with an RS-232 serial port. Or, you can use a TTL-to-USB serial adapter which are supported by most modern computers and operating systems. As a bonus, the 5V power from the USB port is used to power the board and charge the batteries (if you have rechargeable batteries and R4 is installed).

Connector P1 on the board is for connecting the USB adapter. Along with Power and serial In and Out, there is one additional pin on P1. The RTS line is an input from the USB adapter that is normally low when the serial terminal on the PC is Ready To Send. The Badge uses RTS to provide an active-high reset signal. If for any reason the Badge stops responding, manually command the terminal software to raise the RTS line, then set it back low. This will reset the 65C02 and allow it to start running again.

The serial port uses the following configuration: 9600,N,8,1 (9600 baud, No parity, 8 data bits, and 1 Stop bit).

Writing to the serial port TX bit is as simple as writing data $00 (to set TX=0) or $80 (to set TX=1) to any address at or above $8000. To leave all LED segments off, write to any address above $8000 with a low byte of $FF (eg. $80FF). That will ensure all 8 cathode outputs are high (inactive). To maintain the current LED status, the LED data and address values will need to be stored in RAM and the serial routine would read and adjust the high data bit while writing it to the address stored in RAM.

To read from the serial port, the 65C02 IRQ pin is used. The software contains an Interrupt handler to capture the incoming logic level changes. Since the IRQ pin is active low, it can only directly detect the low level. Precise timing loops are used to detect a high, which is detected by a lack of interrupt condition.

You will need some kind of "terminal" program to run on your computer (to make it behave like a good old serial terminal). These usually come with the operating system, or can be downloaded for free. Hyperterm is common for Windows, but there are many others.

See Appendix B for information on the USB adapter supplied with the Deluxe kit.

Software Description

It all starts with the power-on reset. When the 65C02 resets, it performs a reset vector jump from the 16-bit value at location $FFFC and $FFFD. The reset routine does initialization of the LED display and software serial port and then enters a machine language monitor. The monitor is accessed over the serial port. It uses the standard 9600, N, 8, 1 configuration.

The monitor will also produce a scrolling display on the 7 LED digits with the default message being "6502 badge for VCF." The 5 discrete LEDs will also scroll back and forth in a "Knight Rider" pattern. The display is refreshed while waiting for serial data input, so as long as no data is coming in, the display will cycle continuously.

Monitor commands

The monitor uses a combination of Hexadecimal digits and non-numeric command characters. The basic format of a command line is like this:

SSSS . **EEEE C** *<return>*

SSSS and EEEE represent the start and end address (with a period to separate them).
C represents the Command Symbol (a non-alphanumeric character)
<return> is the Carriage Return or Enter Key on most terminals.
Note: No spaces are permitted between these elements (i.e.) 1234.5678L*<return>*.

SSSS and EEEE can be either 1 or 2 bytes in length (2 or 4 Hex digits). If more than 4 hex digits are entered, then only the last 4 digits will be interpreted as the address. For example, if 1A2B3C.4D5E6C is entered, then the effective start and end addresses will be the bold characters 1A**2B3C**.4D**5E6C** or simply 2B3C and 5E6C. Entering less than 4 digits will simply pad the upper digits with 0. So entering 0.3FF is valid and will cover the first 1K of memory (i.e. from $0000 to $03FF).

With some commands, entering an address is optional. The monitor keeps track of the last hexa-decimal value entered, so if none is entered, it assumes the previous address. It will also auto increment that address so repeated commands will increment as well. For example, the command **1000L<Return>** will LIST (disassemble) 20 lines starting at address $1000. If the next command is simply **L<Return>** with no address, the next 20 lines will be disassembled from where it left off. By entering **1000LLL<Return>**, 60 lines of disassembly starting at address $1000 will be displayed. This style helps reduce the keypresses needed when working in the Monitor.

Entering "**?<Return>**" provides this simple help screen:

```
 Commands are :
 Syntax = {} required, [] optional, HHHH hex address, DD hex data

 [HHHH][ HHHH]{Return} - Hex dump address(s)(up to 16 if no address entered)
 [HHHH]{.HHHH}{Return} - Hex dump address range (16 per line)
 [HHHH]{:DD}[ DD]{Return} - Change data bytes
 [HHHH]{G}{Return} - Execute a program (use RTS to return to monitor)
 {HHHH.HHHH>HHHHI}{Return} - move range at 2nd HHHH down to 1st to 3rd HHHH
 [HHHH]{L}{Return} - List (disassemble) 20 program lines
 [HHHH]{.HHHH}{L}{Return} - Dissassemble range
 {HHHH.HHHH>HHHHM}{Return} - Move range at 1st HHHH thru 2nd to 3rd HHHH
 {HHHH}[ HHHH]{Q}{Return} - Text dump address(s)
 [HHHH]{.HHHH}{Q}{Return} - Text dump address range (16 per line)
 {S}[up to 32 text characters]{Return} - Set 32 byte LED message
 [HHHH]{U}{Return} - Upload File from PC to SBC (Xmodem/CRC)
 [HHHH.HHHH]{X}{Return} - Download File from SBC to PC (Xmodem/CRC)
 {V}{Return} - Monitor Version
 {P*}{Return} - Protected Power Down
 {!}{Return} - Enter Assembler
 {@}{Return} - Cold-Start EhBASIC
 {#}{Return} - Warm_Start EhBASIC
 {?}{Return} - Print commands
```

HEX Dump

```
[HHHH][ HHHH]<Return> - Hex dump address(s)(up to 16 if no address entered)
[HHHH]{.HHHH}<Return> - Hex dump address range (16 per line)
```

The Hex Dump command will provide a raw display of the memory contents selected in a Hexadecimal format. There are three basic method of entering this command:

1) Entering a start address

2) Entering a start and end address

3) Just pressing *<Return>*

If just a start address is entered, that single address will be displayed. Multiple addresses can also be entered, separated by a *<space>*.

If a start and end address is entered, the entire range will be displayed. The Monitor likes to frame the hex dump in 16 byte lines with $xxx0 being the first byte. So depending upon on the start address, up to 16 bytes will fill the first line. The next lines will be a full 16 bytes until the final address is reached. The last line will stop at the ending address entered.

This is what the command **5.2D<Return>** looks like:

```
0005 - 2D FF 06 - 42 55 7A 68 AE F5 5B FF
0010 - 7C FF 37 FF FF FC DF EF - 8F CB D7 FF FF 5F 19 76
0020 - 36 DA D4 5D EF F3 EA FF - EA E1 E3 65 4B FF
```

If no address is entered, up to 16 bytes will be displayed continuing from the last address stored.

Edit Memory

```
[HHHH]{:DD}[ DD] <Return> - Change data bytes
```

This command allows RAM contents to be changed. Enter the address and new data. More than one data byte can be entered and will fill sequentially up from the start address. After the address, enter a colon ":" to let then monitor know the next value is the data. If more than 1 byte is entered, separate them with a *<space>*.

To change the memory at location $1000 to $55, enter **1000:55<Return>**.

Attempts to write ROM will not cause an error, they will simply not work. In the case of this badge, the write will affect the I/O latches as described previously.

Move Memory

```
{HHHH.HHHH>HHHHM}<Return> - move range at 2nd HHHH down to 1st to 3rd HHHH
```

This command will move a block of memory. If the syntax is rewritten this way: {SSSS.EEEE>DDDDM}*<Return>*, then memory is copied from SSSS to DDDD, until EEEE is reached. The memory is read from SSSS to EEEE. A useful function of this command is to fill memory. For instance, to zero a block of RAM, first use the EDIT MEMORY command to write $00 to the first location. Next, use the MOVE MEMORY command to fill the rest of the block. For example to fill $1000-$1FFF with $00, these are the commands to use:

```
1000:00<Return>
1000.1FFE>1001M<Return>
```

This will move $1000 to $1001, $1001 to $1002, etc; until reaching the last byte, $1FFE to $1FFF. The $00 in $1000 gets copied to all locations.

Note: If location DDDD falls within the range of SSSS-EEEE, then the original data from DDDD-EEEE will be overwritten and not copied. The INSERT MEMORY command can be used to prevent this.

Insert Memory

```
{HHHH.HHHH>HHHHI}<Return> - move range at 2nd HHHH down to 1st to 3rd HHHH
```

This command will move a block of memory but it does it in the reverse direction that the MOVE MEMORY command does. Using this syntax: {SSSS.EEEE>DDDDI}{Return}, DDDD will get adjusted to DDDD + (EEEE-SSSS) and then memory will get copied starting at EEEE to DDDD, then EEEE-1 to DDDD-1, all the way down to SSSS to the original DDDD. This way, any sized gap can be created inside of a range. If DDDD falls outside of SSSS-EEEE, then either the INSERT MEMORY or MOVE MEMORY command will work.

Execute

```
[HHHH]{G}<Return> - Execute a program (use RTS to return to monitor)
```

This command is used to start a machine language program. An RTS (opcode $60) instruction will return execution to the Monitor. If no address is entered, the last stored address is used, but entering a start address is recommended. So, if a program has been entered at $1000, typing **1000G**<Return> will cause that program to execute until it encounters the RTS instruction.

List

```
[HHHH]{L}}<Return>  - List (disassemble) 20 program lines
[HHHH]{.HHHH}{L}}<Return>  - Dissassemble range
```

The LIST command will produce lines of disassembled code. It can be used by either specifying a starting address, a range of addresses, or no address. When only a start address is specified, 20 lines of disassembled code is listed (starting with the address entered). When a range is specified, the lines listed will cover the entire range entered. If no address is specified, then 20 lines are listed, starting from the last used address stored in memory.

This is a sample disassembly:

```
    >FF00L

    FF00-  x    78          SEI
    FF01-  X    D8          CLD
    FF02-  ".   A2 FF       LDX  #$FF
    FF04-  .    9A          TXS
    FF05-  ).   A9 7F       LDA  #$7F
    FF07-  ..   85 01       STA  $01
    FF09-  ).   A9 00       LDA  #$00
    FF0B-  ..   85 00       STA  $00
    FF0D-  ..   92 00       STA  ($00)
    FF0F-  F.   C6 00       DEC  $00
    FF11-  Pz   D0 FA       BNE  $FF0D
    FF13-  F.   C6 01       DEC  $01
    FF15-  .v   10 F6       BPL  $FF0D
    FF17-  ..   85 01       STA  $01
    FF19-  -x.  AD F8 03    LDA  $03F8
    FF1C-  I%   49 A5       EOR  #$A5
    FF1E-  My.  CD F9 03    CMP  $03F9
    FF21-  p.   F0 10       BEQ  $FF33
    FF23-  ".   A2 00       LDX  #$00
    FF25-  )h   A9 E8       LDA  #$E8
    >
```

It shows the address of the opcode, the 1 to 3 ASCII characters of the bytes in memory (non-printable characters are replaces with a period), hex value of the byte(s) in memory, the opcode and any operands.

Text Dump

```
{HHHH}[ HHHH]{Q}<Return> - Text dump address(s)
[HHHH]{.HHHH}{Q}<Return> - Text dump address range (16 per line)
```

The TEXT DUMP command works similarly to the HEX DUMP command, but displays ASCII characters instead of hexadecimal characters. This command requires a starting address, or can dump a block of memory. This can be useful in finding a block of text stored in memory. Non-printable characters are replaced with a period ".".

Set LED Text

```
{S}[up to 32 text characters] <Return> - Set 32 byte LED message
```

The S command changes the scrolling message on the LED. There is a 32 byte buffer that will be padded with blanks. If more than 32 characters are entered, then only the first 32 characters will be loaded. Format is Stext to display*<Return>*. Typing "S6502 badge*<Return>* will produce this text on the LED:

```
    6502 badge
```

See the section later on titled "LED Operations" for more information on changing the way the display is refreshed and for help with custom display programming.

Upload (XMODEM)

```
[HHHH]{U}<Return> - Upload File from PC to SBC (Xmodem/CRC)
```

This command will perform and XMODEM/CRC file transfer from the PC terminal to the Badge. The file being transferred will be stored starting at the address provided, or the last saved address in memory. Since XMODEM uses a 128 byte buffer, up to 127 extra bytes could be stored at the end of the file, so care must be taken not to overwrite data. XMODEM CRC is the only format supported. Checksum mode or 1k mode will not work.

Download (XMODEM)

```
[HHHH.HHHH]{X}<Return> - Download File from SBC to PC (Xmodem/CRC)
```

This command will perform an XMODEM/CRC file transfer from the Badge to the PC terminal. The file being transferred will come from the memory range entered, or the last saved address in memory. It is recommended that an address is always entered. Since XMODEM uses a 128 byte buffer, up to 127 extra bytes could be stored at the end of the file, so care must be taken to not overwrite data when this file is sent back to the Badge. XMODEM CRC is the only format supported. Checksum mode or 1k mode will not work.

Version

```
{V}<Return> - Monitor Version
```

This command prints the Monitor version. It will look similar to this:

```
>V

65C02 Monitor v5.2 (5-27-17) Ready
with Enhanced Basic Interpreter (c) Lee Davison
(Press ? for help)
>
```

Power Down

```
{P*}<Return> - Protected Power Down
```

This command will turn off the LEDs, and then enter an endless loop in ROM. Use this to ensure RAM integrity before powering down the Badge.

Mini Assembler

```
{!}<Return> - Enter Assembler
```

The mini-assembler can do basic assembly in RAM. It uses Hexadecimal for all operands. It does not support labels or math functions in the operands. When in the assembler, the input prompt changes from ">" to "!"

This is the HELP available in the mini-assembler.

```
    !?

    Current commands are :
    Syntax = {} required, [] optional
    HHHH=hex address, OPC=Opcode, DD=hex data, '_'=Space Bar or Tab
    '$' Symbols are optional, all values are HEX.
    Any input after a 'semi-colon' is ignored.

    {HHHH}{Return} - Set input address
    [HHHH][_]{OPC}[_][#($DD_HHHH,X),Y]{Return} - Assemble line
    [HHHH]{L}{Return} - List (disassemble) 20 lines of program
    {Return} - Exit Assembler back to Monitor
    {?}{Return} - Print menu of commands

    ADC AND ASL BCC BCS BEQ BIT BMI BNE BPL BRA BRK BVC BVS CLC CLD
    CLI CLV CMP CPX CPY DEC DEX DEY EOR INC INX INY JMP JSR LDA LDX
    LDY LSR NOP ORA PHA PHP PHX PHY PLA PLP PLX PLY ROL ROR RTI RTS
    SBC SEC SED SEI STA STX STY STZ TAX TAY TRB TSB TSX TXA TXS TYA
    WAI STP BBRx BBSx RMBx SMBx .DB .DW .DS

    !
```

Entering an address will move the address pointer to that address. If no address is entered, the last accessed address is used. Note, a space is required before entering an opcode, even if no address is entered. Another space is required after the opcode if an operand is used. Any operand used has to be entered in the same order as in the help. WDC opcodes WAI, STP, BBRx, BBSx, RMBx, and SMBX are supported. Please refer to the WDC manual for their use. The Badge processor is non-WDC so these are not available to use. Three additional pseudo-opcodes are provided:

 .DS HH will store a 1 byte hexadecimal value in RAM at the given address.
 .DW HHHH will store a 2 byte hexadecimal value in RAM at the given address, low byte first.
 .DS 'text to enter' will store an ASCII string in RAM. Single quotes are used to delimit the text.

Here is an example assembler session:

```
        !1000 LDA #FF
        1000-  ).   A9 FF      LDA   #$FF
        ! STA 00
        1002-  ..   85 00      STA   $00
        ! LDA (00),Y
        1004-  1.   B1 00      LDA   ($00),Y
        ! STA (02,X)
        1006-  ..   81 02      STA   ($02,X)
        ! INX
        1008-  h    E8         INX
        ! INY
        1009-  H    C8         INY
        ! BNE 1000
        100A-  Pt   D0 F4      BNE   $1000
        ! BNE 2000
                      ^
        ! RTS
        100C-  `    60         RTS
        ! .DB 45
        ! .DB 45
        ! .DW 3456
        ! .DS ' THIS IS A TEST'

        !1000L
        1000-  ).   A9 FF      LDA   #$FF
        1002-  ..   85 00      STA   $00
        1004-  1.   B1 00      LDA   ($00),Y
        1006-  ..   81 02      STA   ($02,X)
        1008-  h    E8         INX
        1009-  H    C8         INY
        100A-  Pt   D0 F4      BNE   $1000
        100C-  `    60         RTS
        100D-  EE   45 45      EOR   $45
        100F-  V4   56 34      LSR   $34,X
        1011-   TH  20 54 48   JSR   $4854
        1014-  IS   49 53      EOR   #$53
        1016-   IS  20 49 53   JSR   $5349
        1019-   A   20 41 20   JSR   $2041
        101C-  T    54         ???
        101D-  ES   45 53      EOR   $53
        101F-  T    54         ???
        1020-  .    00         BRK
        1021-  .    00         BRK
        1022-  .    00         BRK
        !
```

Notice the BNE 2000 entry. It printed a "^" to the right of the address 2000. Since a branch to address $2000 is out of range, it tries to point out the error location to the user. Any error will result in a similar "^" near the error. At the end is a LIST command starting at $1000, showing the code and data entered.

Help

```
 {?}{Return} - Print commands
```

This simply prints the Help menu. It can be used within the Monitor or Mini-assembler.

EhBASIC Cold Start

```
{@}<Return> - Cold Start EhBASIC
```

Use this command the start EhBASIC for the first time. It will initialize RAM, and then set the top of memory. It will prompt you to enter a valid top of memory address in decimal format. Enter **2048** if your badge has a 2K RAM; or **32768** for a 32K RAM chip. Or if your monitor is dated 2-24-18, just press *<Return>* to automatically size memory. It will look like this (with a 2K RAM):

```
>@
Memory size ? 2048
1023 Bytes free
Enhanced BASIC 2.22

Ready
```

The first 1K of RAM is used by the Monitor and LED display, so entering 2048 as the top of RAM returns 1K of free space. See the EhBASIC manual for instructions on how to use it and the commands available. A new SYS command has been added; type SYS*<Return>* from the EhBASIC input prompt to return to the monitor.

EhBASIC Warm Start

```
{#}<Return> - Warm Start EhBASIC
```

Use this to return to EhBASIC after a SYS command. This will not initialize the EhBASIC system, so any program in RAM should still be available.

EhBASIC Manual

EhBASIC was written by Lee Davison, and adapted to use here with his kind permission. Please refer to the manual at http://www.sunrise-ev.com/photos/6502/EhBASIC-manual.pdf for any questions on EhBASIC commands and syntax. The source code for EhBASIC is well documented, so reading it might help with more technical questions. Here is a list of keywords to get you started:

ABS	AND	ASC	ATN	BIN$	BITCLR	BITSET	BITTST	CALL	CHR$
CLEAR	CONT	COS	DATA	DEC	DEEK	DEF	DIM	DO	DOKE
ELSE	END	EOR	EXP	FN	FOR	FRE	GET	GOSUB	GOTO
HEX$	IF	INC	INPUT	INT	IRQ	LCASE$	LEFT$	LEN	LET
LIST	LOAD	LOG	LOOP	MAX	MID$	MIN	NEW	NEXT	NMI
NOT	NULL	OFF	ON	OR	PEEK	PI	POKE	POS	PRINT
READ	REM	RESTORE		RETIRQ	RETNMI	RETURN	RIGHT$	RND	RUN
SADD	SAVE	SGN	SIN	SPC(	SQR	STEP	STOP	STR$	SWAP
SYS	TAB(	TAN	THEN	TO	TWOPI	UCASE$	UNTIL	USR	VAL
VARPTR	WAIT	WHILE	WIDTH	+	-	*	/	^	<<
>>	>	=	<						

Keywords must be UPPER CASE with no spaces. Numbers can be integer, decimal, or floating point. Prefix integers with $ for hex, or % for binary. Examples: $0A   1   -142   96.3   2.718E-3 Variables can be numeric, strings$, or arrays(n). Strings are delimited by quotes, i.e. "Hello world".

LED Operations

New routines in the Monitor were added to refresh the scrolling display of both LED digits and discrete LED's. The basic premise for the display to work is as follows. Set the address latch to point to the first digit. Get the bit pattern of that digit from RAM, write it to the buffers. Keep that LED on for a brief period. Advance to the second digit and repeat. Do this for all 7 digits and the discrete LED row. Repeat this refresh process. That would create a static display. If done at the right rate, the display (through the use of persistence of vision) won't flicker and will be bright enough to see easily.

To scroll the display, an adjustment is made to the starting address of the buffer to be displayed. There is a control for how many refresh cycles are completed before adjusting the buffer pointers. The fewer number of refreshes, the faster it will scroll. The refresh routine gets called anytime the monitor or EhBASIC is waiting for a character from the serial port. So as long as the program is waiting, the display runs smoothly and consistently. When a character is received, the display is briefly interrupted. Whenever the input routine is not getting called, the display will be either be frozen or randomly written to. Once the program is back in the wait loop, the display will return to normal.

A study of the source files named LEDdrive.asm and font.asm will provide a better understanding of how they work in detail. For simple control, a few locations in RAM are used to control the display.

```
Lbuff     =     $2A0        ; text storage for LED 32 bytes max
LDbuff    =     $2C0        ; discrete LED buffer 32 bits max
Lptr      =     $E2         ; LED pointer
Ldig      =     $E3         ; digit counter
Lscn      =     $E4         ; 8 bit scan speed delay
Lscl      =     $E5         ; scroll speed
Lscnc     =     $E6         ; scan counter
Lsclc     =     $E7         ; scroll counter
LEDchk1   =     $E8         ; config checksum 1
LEDchk2   =     $E9         ; config checksum 2
```

Lbuff holds the bit-mapped patterns for the LED digits, at location $02A0 - $02BF.

LDbuff holds the bit-mapped patterns for the discrete LED's, at location $02C0 - $02DF.

The 32 digits and 32 discrete LED patterns in the buffers are synchronized, meaning as the digits scroll from position 0 to 31 within the buffer, so do the discrete LED's.

Lptr ($E2) is used to point to the left-most byte in Lbuff and LDbuff that is being displayed. When Lptr = 0, it points to the beginning of the buffer. As it advances, the display scrolls to the left. The range is from $00 - $1F (32 bytes).

Ldig ($E3) is used to point to the current digit being refreshed. $00 is the discrete LED's, $01 is left-most digit and $07 is the right most digit. The range is from $00 - $07.

Lscn (`$E4`) is used to store the scan speed delay. This is the amount of time each digit is on for. The longer the time, the brighter it will be with less flicker. Lower values result in flicker and dimmer LEDs. The range is $00 - $FF. Setting Lscn to $00 will disable the refresh of the display. Do this to write your own routines to control the LEDs.

Lscl (`$E5`) is the scroll speed delay. This counts the number of scan cycles before the Lptr is incremented. Lower values result in a faster scrolling display. The range is from $00 - $FF. Setting Lscl to $00 with stop the scrolling and provide a static display. Set Lptr to point to the leftmost digit to be displayed statically.

Lscnc (`$E6`) is the actual counter used for the scan speed delay. It counts from value in Lscn down to $00 and then resets. Ldig also gets incremented when this counter reaches $00.

Lsclc (`$E7`) is the actual counter used for the scroll speed delay. It counts from the value in Lscl down to $00 and then resets. Lptr also gets incremented when this counter reaches $00.

LEDchk1 & **LEDchk2** (`$E8 & $E9`) hold a simple flag that controls whether the contents of Lbuff and LDbuff get overwritten during a system Reset. If LEDchk contains $A5 AND LEDchk2 contains $5A, then the contents of Lbuf and LDbuff are not overwritten. If either contains another value, the buffers get initialized with the default "6502 badge for VCF" message and Knightrider effect. This was designed to allow users to set a custom display and then keep that set while the system is in standby on battery power. Use the EDIT MEMORY command to alter either of these flags to cause the initialization to happen on the next system Reset.

Serial Operations

There are 3 subroutines in the Monitor for serial I/O that the user can use. Use the JSR instruction to call them. Note: Two different monitor ROMs have been shipped. Use the addresses that correspond to the date of your ROM (viewed with the V command, 5-27-17 or 2-24-18)

Serial_Output ($EAE8 or $EAEC) – This routine will send 1 byte from the badge to the PC. The byte to send is loaded in the accumulator prior to calling the routine. The X & Y registers are unchanged.

Example:

```
5-17-17 ROM       2-24-18 ROM
LDA   #$41        LDA   #$41          ; ASCII "A"
JSR   $EAE8       JSR   $EAEC         ; sent to terminal
```

Serial_Input ($EB35 or $EB39) – This routine will get the next received byte from the serial port. It will wait indefinitely for a byte to be received. The returned byte is in the accumulator. The X & Y register are unchanged. The Z and N flags reflect the value in the accumulator.

Example:

```
5-17-17 ROM       2-24-18 ROM
JSR   $EB35       JSR   $EB39         ; get character from terminal
STA   $400        STA   $400          ; save in RAM
JSR   $EAE8       JSR   $EAEC         ; echo it back to terminal
```

Scan_Input ($EB4C or $EB50) – This routine checks for a received byte. If none is present, it returns immediately with the Carry flag (C) cleared. The accumulator will be changed and the X & Y register unchanged. If a byte is present, it will be stored in the accumulator with the Z and N flags reflecting the received byte. The Carry (C) flag will be set.

Example:

```
        5-17-17 ROM       2-24-18 ROM
Loop    JSR   $EB4C       JSR   $EB50         ; check for character from terminal
        BCC   Loop        BCC   Loop          ; no input, check again
        BMI   Loop        BMI   Loop          ; if input is > $7F (N flag = 1),
                                              ; check again
        STA   $400        STA   $400          ; save in RAM
```

System Memory Usage

The following locations are used by the system. It is up to the user to decide if they can overwrite these locations with their code. Studying the source files will help the user understand the risks.

Zero Page:

$00 - $13 – EhBASIC
$14 - $31 – free
$32 - $3F – Monitor variables
$40 - $5A – free
$5B - $DF – EhBASIC
$E2 - $E9 – LED driver
$EA - $ED – Serial port driver
$EF - $FF – EhBASIC

RAM:

$100 - $1FF – System Stack
$200 - $27F – Serial Receive Ring buffer
$2A0 - $2DF – LED buffers
$300 - $37F – Monitor input buffer
$390 - $3FF – EhBASIC input buffer
$400 - $7FF – User RAM, EhBASIC program space (2K RAM)
$400- $1FFF – User RAM, EhBASIC program space (8K RAM)
$400 - $7FFF – User RAM, EhBASIC program space (32K RAM)

ROM:

5-17-17 ROM	2-24-18 ROM	
$C000 - $C1FF	$C000 - $C1FF	– CRC lookup table for XMODEM
$C200 - $EA9B	$C200 - $EA9F	– EhBASIC
$EA9C - $EB64	$EAA0 - $EB68	– Serial IO routines
$EB65 - $FC39	$EB69 - $FC3D	– Monitor
$FC3A - $FD93	$FC3E - $FD97	– LED support
$FD94 - $FFBE	$FD98 - $FFC2	– Xmodem
$FFBF - $FFFF	$FFC3 - $FFFF	– Reset code and vectors

Source Code Organization and Modification

With the exception of EhBASIC, all of the software for this badge was written by Daryl Rictor. You are free to use and modify this software for non-commercial use. This is how the source code file are organized:

Sbc.asm – This is basically the make file. It will load all of the other source files in the proper locations and order so the assembler can build the system correctly.

Basic.asm – This is the EhBASIC source code.

Basldsv.asm – This is the EhBASIC Load and Save patch that allows file load and save via xmodem.

SbcOS.asm – This is the Monitor program which included the disassembler and mini-assembler.

Serial.asm – This is the driver for the software-driven serial port

LEDdrive.asm – This is the LED driver. It has the refresh, scroll, and set text functions.

Font.asm – This is the font for the LED's ASCII text. It maps the proper segments to the matching data port latch pins.

Xmodem.asm – This provides the Xmodem file transfer protocol

CRCtable.asm – This is the Cyclic Redundancy Check data table. It is used to speed up the CRC calculations by using a look-up table.

Reset.asm – This contains the system reset code and initialization code along with the ROM-based vectors at locations $FFFA - $FFFF. It also has the software BRK (opcode $00) handler which will cause the system to do a reset and enter the monitor.

To assembler these source files:

1) Run the 6502 Macroassembler & Simulator (6502.exe)

2) Open the sbc.asm file   File –> Open -> sbc.asm

3) Set the following Simulator options:

    a. Assembler -> uncheck Extra byte after BRK instruction

    b. General -> select 65C02, 6501

4) Assemble (F7)

To save the object file, select File -> Save Code:

    There are file types for intel Hex, Motorola S records, and binary.
    Select the type and then select "Options"
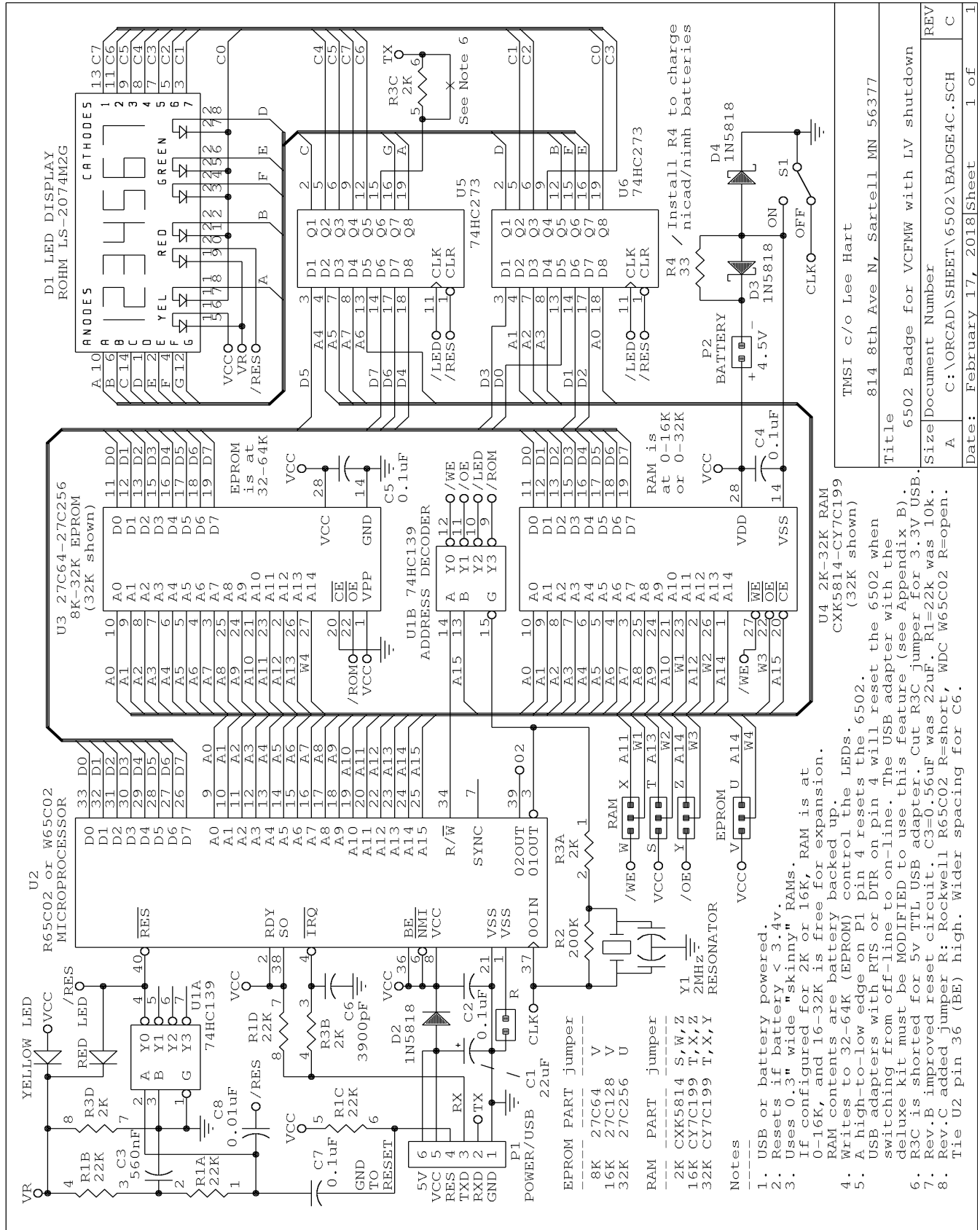   Set the start address to 0xC000 for a 16k ROM or 0x8000 for a 32k ROM.
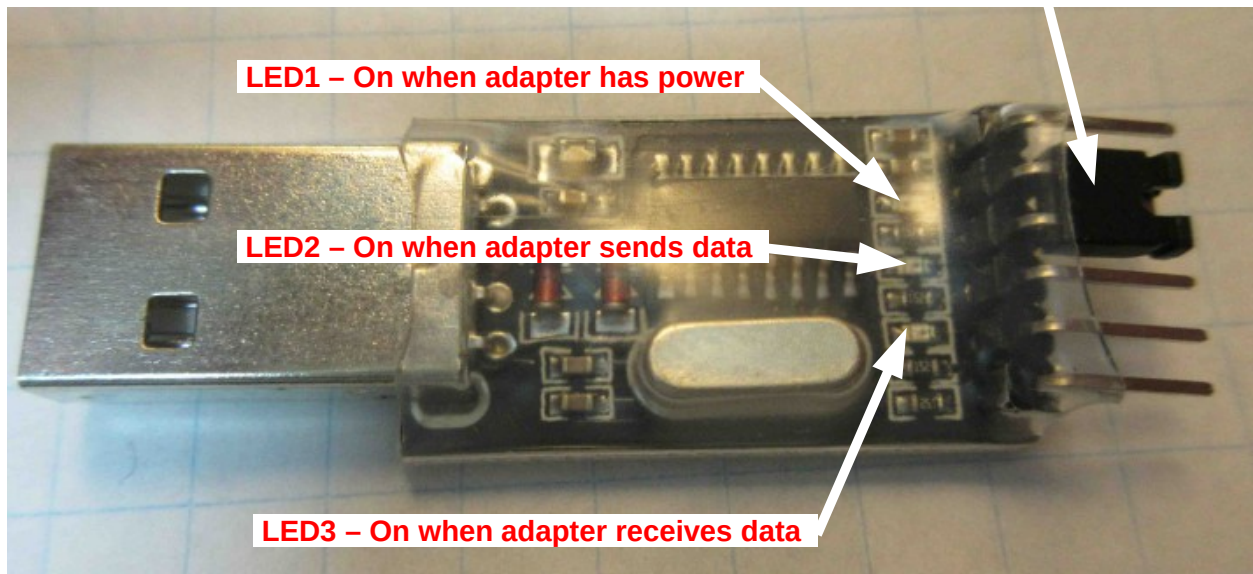   Ensure the End address is set to 0xFFFF.
        Hit OK then
    Hit SAVE to save the file.

Appendix A -- Badge Schematic

Appendix B - CH340G USB-TTL Adapter

This device converts a modern computer's USB port into a vintage serial port. The (included) jumper configures it for 3.3V (short VCC to 3V3) or 5V (short VCC to 5V) output levels. The 6502 Badge uses 5V levels; the VCC-5V connection is already made on the Badge PC board, so **REMOVE** this jumper.

LED1 – On when adapter has power

LED2 – On when adapter sends data

LED3 – On when adapter receives data

It's a typical modern "no documentation or support" gadget. We do not support it for them; so use these notes at your own risk. This is just an unofficial and independent description of how we got it to work with Windows 7. It **should** work with other version of Windows and other operating systems; but you'll have to find the drivers, and the key pokes and mouse strokes to install it will be a little different.

A driver **must** be downloaded and installed manually **BEFORE** you plug in the module! For Windows 7, one source is http://www.arduined.eu/ch340g-converter-windows-7-driver-download/. This is an Arduino site, and believed to be legitimate and less likely to contain malware or viruses. If this URL is no longer valid, or you need one for a different operating system, search for "CH340G driver". The manufacturer's own driver is at http://www.wch.cn/download/CH341SER_ZIP (Google can translate it for you).

As an example, here are step-by-step instructions for installing the driver for Windows 7:

1. Find a website that lists a driver for your operating system and version. For Win7, we'll use the one on the Arduino page http://www.arduined.eu/ch340g-converter-windows-7-driver-download/.

2. Click on the link to download the driver. This downloads **CH341SER.zip** which is a ZIP file. It must be "unzipped" to extract all the files inside it.

3. Open your **Downloads** folder (or wherever you or your computer puts downloaded files).

4. Right-click on the **CH341SER.zip** file, then click "Extract All..." This creates a new folder named **CH341SER** in your Downloads folder with all the new files and subfolders unzipped inside it.

5. Open (double-click) the new **CH341SER** folder. Inside it, you will see *another* "CH341SER" folder, and an "INSTALL" folder.

6. Open (double-click) this inside "CH341SER" folder. Inside it are a bunch of files including a **SETUP.EXE** program. Aha! Run (double-click) the SETUP program to display this dialog box...
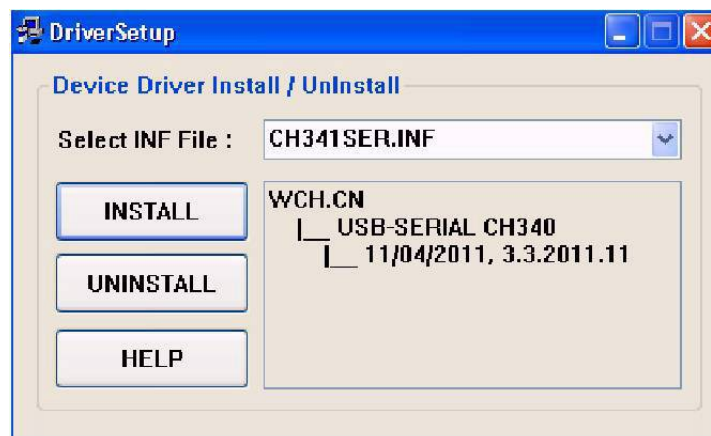
7. Click **INSTALL**. You should get the following message box. (Please forgive poor Engrish.)

8. **Now** you can plug the USB adapter into your computer. Windows should detect the new USB device, and look for the driver for it. It **might** find the new driver all by itself, in which case you should now have a new COMn port.

9. If this doesn't work, you have to manually guide Windows to new driver. When you plugged in the USB adapter, Windows may ask you where find the driver for it. Try telling it to look in the new CH341SER folder you created.

10. If you plugged in the USB adapter <u>before</u> you installed the driver, or if Windows can't find the downloaded driver, you'll get an error message like...
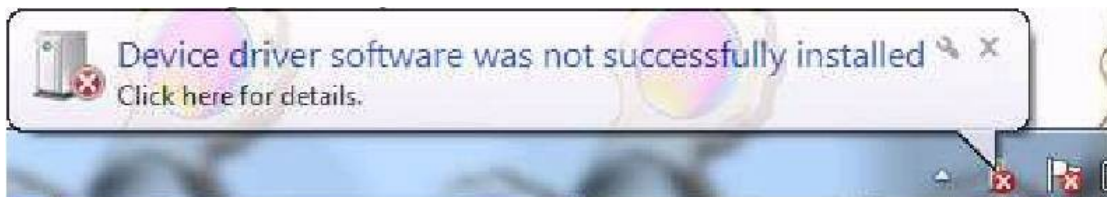
In this case, you have to manually guide Windows to find and install the driver. Briefly, you go to the **Device Manager**, right-click to update the driver, then select "Choose my own path", and point that path to the CH341SER folder you created above. Here is the procedure in detail...
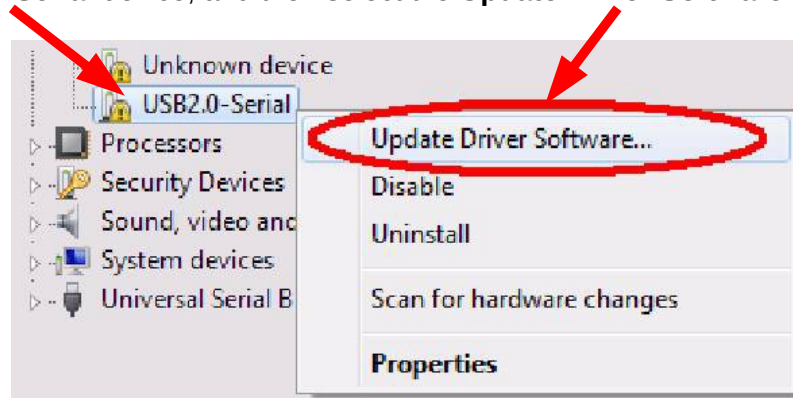
11. Click on the **START** menu, then right-click **Computer**, and select **Manage**.
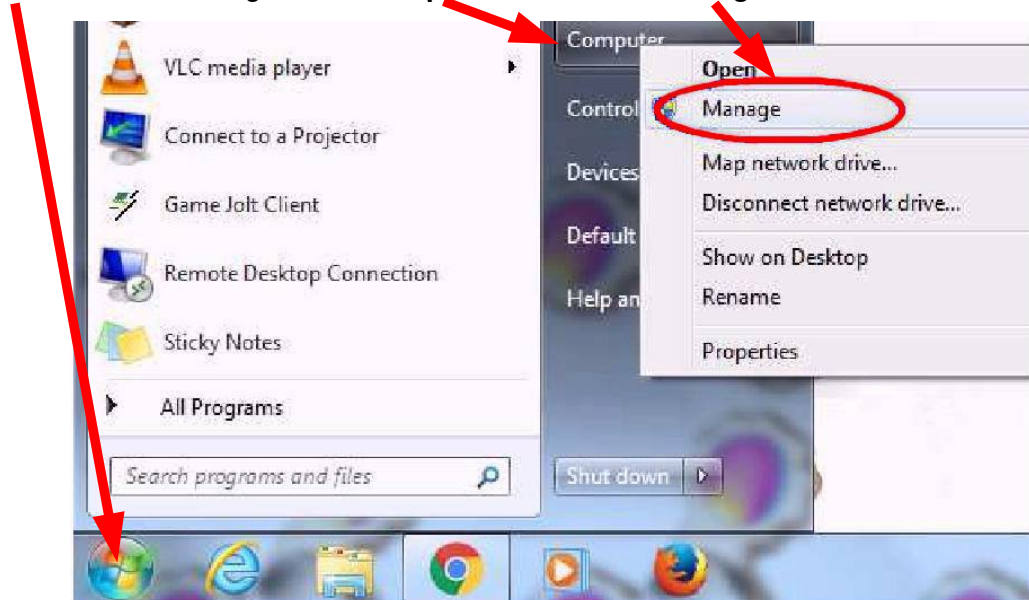
12. This opens the Computer Management menu as shown below. Under "System Tools", click the **Device Manager**, then under "Other Devices" look for the **USB2.0-Serial** device.
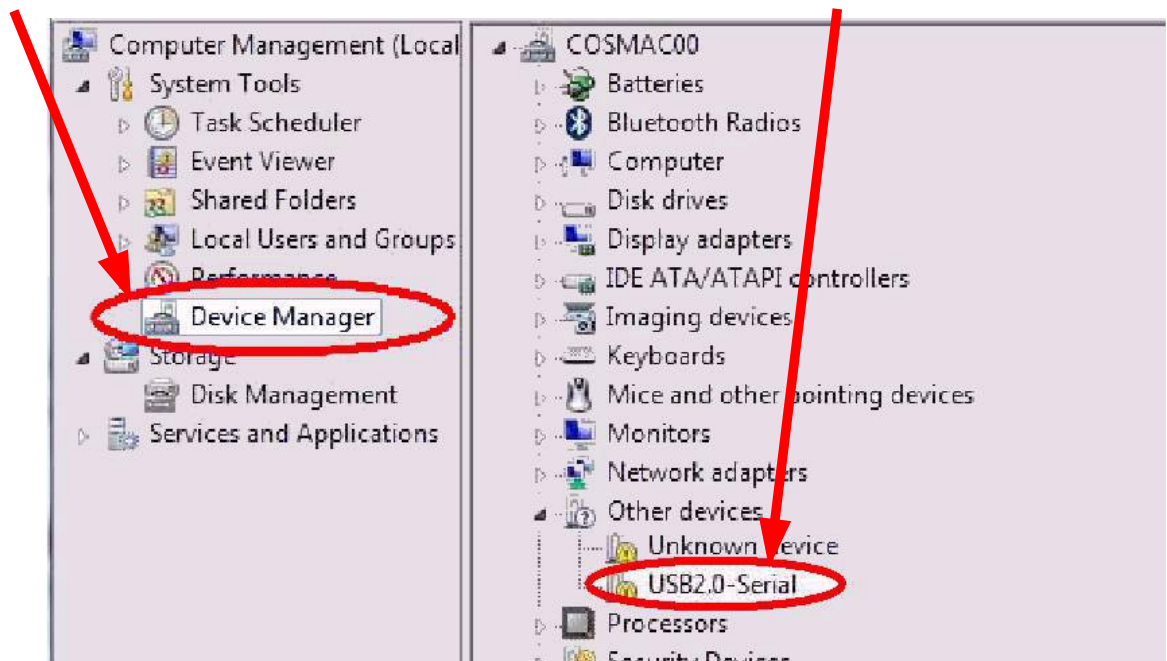

13. Right-click the **USB2.0-Serial** device, and then select the **Update Driver Software...** option.

14. Click the **Browse my computer for driver software** option.
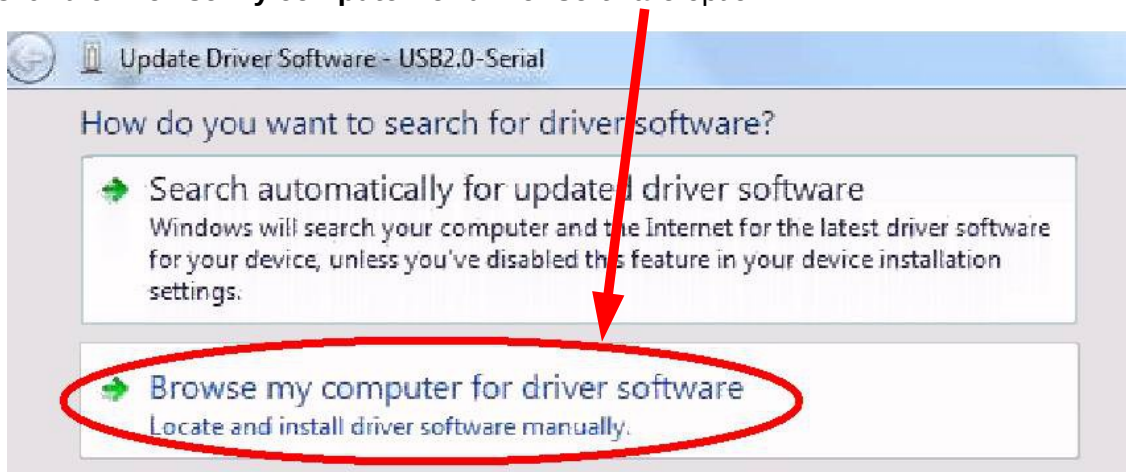
15. (Installing the driver... continued...) The following dialog box (or its equivalent) should pop up. Click the **Browse...** button, find the **CH341SER** folder with the driver files, and click on it so it appears in the "Search for driver software in this location:" box. Be sure the "Include subfolders" box is checked. Then click the **Next** button to install it.
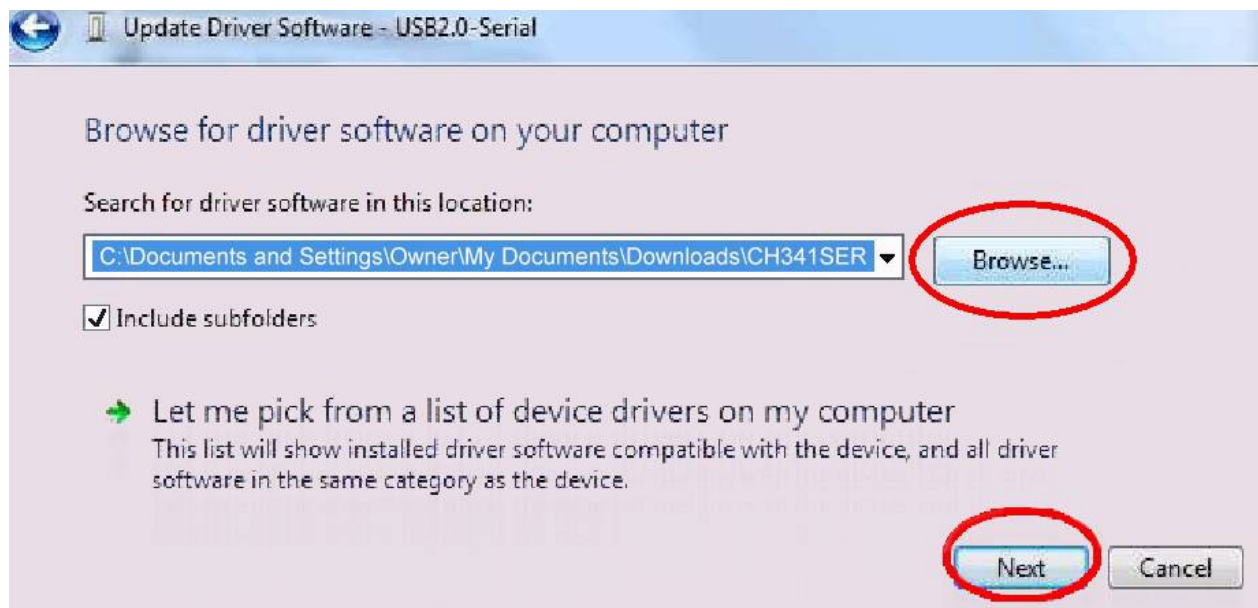
16. Remove, and then re-insert your CH340G USB-serial adapter. Now Windows should find it, and use the correct driver. (You should no longer get the error message in step 10.)

Testing the USB-TTL Adapter

TEST the Adapter before you plug it into your 6502 Badge. Use the jumper (on the adapter) to short TXD to RXD, and plug it into your PC. Run your Terminal program (Hyperterm, RealTerm, TeraTerm etc.). Configure it for 9600N8 (9600 baud, No parity, 8 Data bits), and No hardware or software handshaking. When you tell it to "connect", any keys you type on the keyboard should be echoed and appear on the screen. TXD and RXD are normally high, and go low on Start bit and zero data bits.

You'll need some kind of Terminal program. There are dozens of free Terminal programs, for every computer and operating system. They usually come as part of the operating system. For decades, Windows came with "Hyperterm", but they've stopped providing it. It's not very good; but it works and is very common and free to download.

Once this works, remove the jumper and plug the Adapter into connector P1 on the 6502 Badge. The "GND" pin of the adapter goes in the Badge socket at the top marked "**–**" and is the one closest to the edge of the board. **BE SURE NOT TO PLUG IT IN BACKWARDS !** If you do, it reverses the +5v and GND connections, and can destroy the Badge!

A good plan is to "key" the adapter so it can't be plugged in backwards. To do this, **CUT OFF** the adapter pin labeled "3V3". Plug the mating hole on the 6502 Badge (P1 pin 4, labeled "RES") with a piece of a toothpick etc.

If you want to use the adapter with some other gadget, you won't want to cut off the 3V3 pin. It is used to set the adapter's serial output logic level to either 5v or 3.3v. The 5V pin provides +5v power from the USB port to power things (like the 6502 Badge). The 3V3 pin provides a small amount of 3.3v power from the CH340G chip, and is not able to drive any significant load.
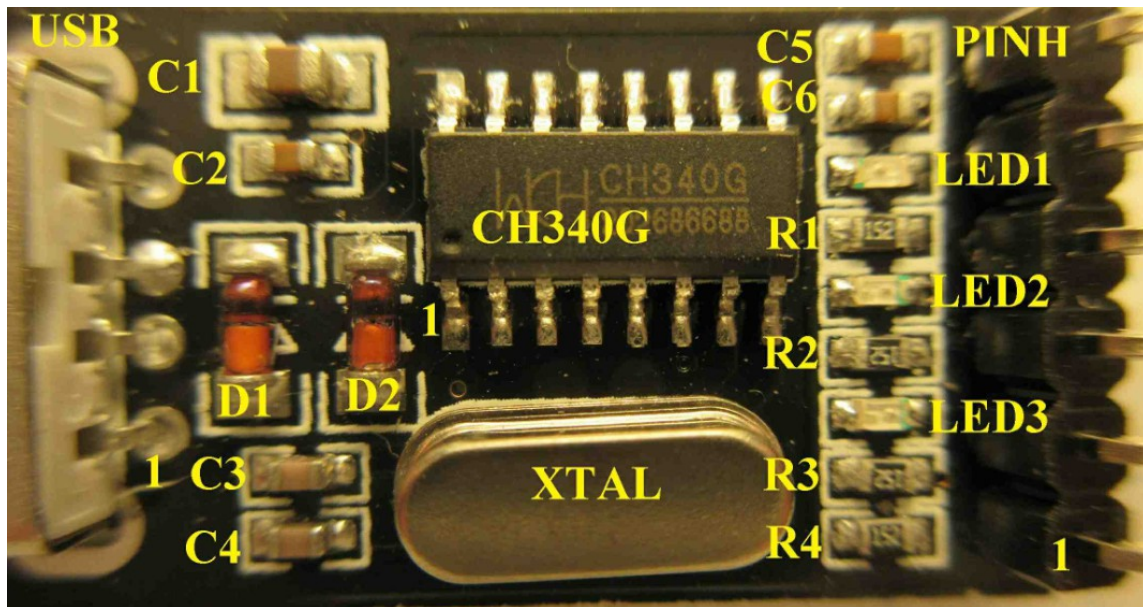
**Top of the Adapter board, with shrink-wrap removed and parts labeled.**
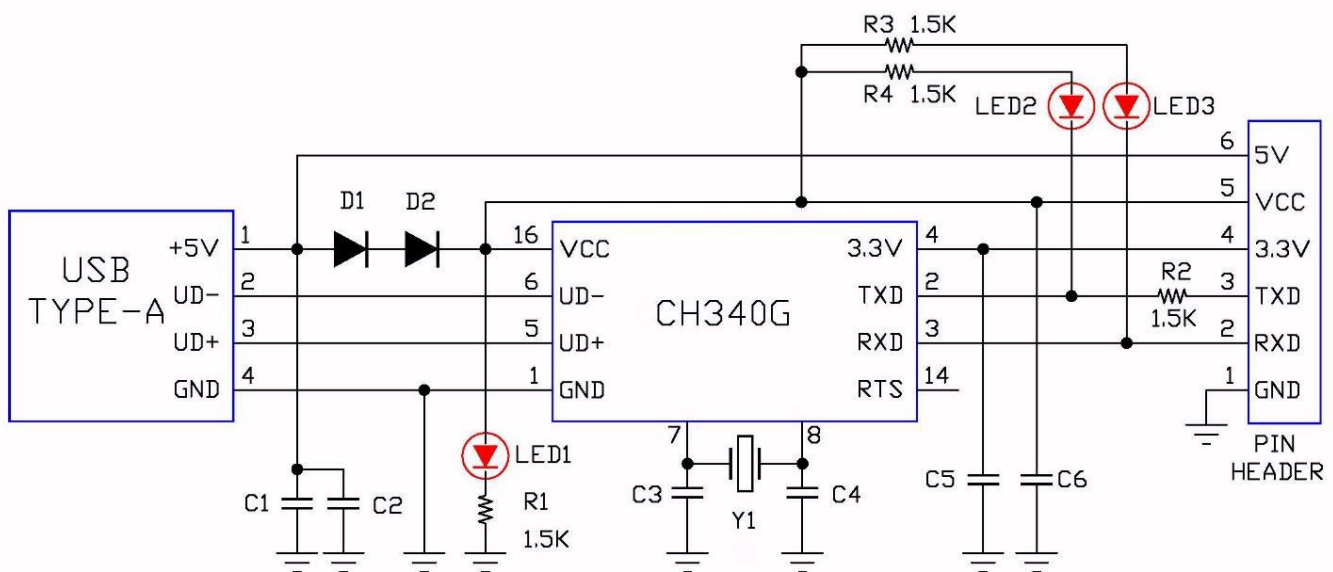

Schematic

Modification to add RTS (Optional – Not for mere mortals!)

Most terminal programs set RTS high initially, or when you use their "hang up" or "disconnect" command; and set RTS low when you use their "on-line" or "connect" command. RTS can thus be used to reset the 6502 on the Badge (and other devices). RTS is available on pin 14 of the CH340G chip, but is not brought out to the 6-pin header. This modification replaces the 3V3 pin with the RTS signal.
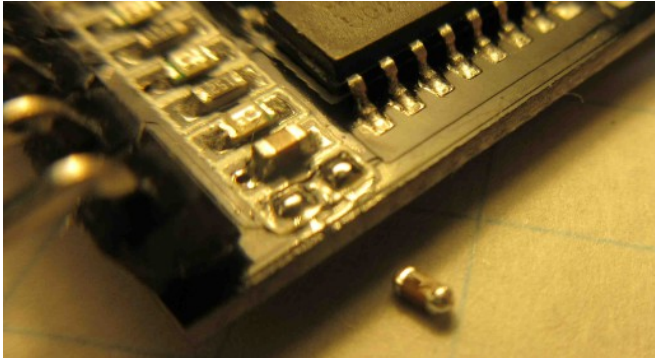
1. Remove C5.   2. Cut Trace to 3V3 pin.
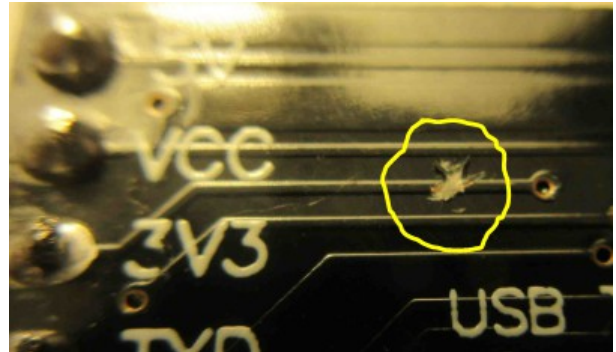
3. Add a jumper wire from CH340G pin 14 to the outer pad of C5.

4. Add a new 0.1uF decoupling capacitor between pin 1 and pin 4. (This replaces the one removed in step 1.)

Your terminal program's connect/disconnect commands should now control the level on the 3V3 pin of the USB adapter. When plugged into the 6502 Badge, switching from "off-line" to "connect" will reset the 6502, and then let the 6502 program run.