Digital Odometer, Fox Cougar T-Bird

Quick Answers

A) How to make the mileage of one cluster become the mileage of another cluster?

You would want to do this if you swapped your current cluster from one pulled from a junkyard. You would want the mileage on your current cluster transferred over to the new cluster. There are two ways to do this:

1) Remove the memory chip from your current cluster and put it into the new cluster. The memory chip is the 14 pin DIP integrated circuit. The chip can be neatly unsoldered from the board, or you can cheat by delicately clipping the leads at the top of the chip and resolder it onto the replacement board.

2) A second much more involved way to do this is to reprogram the device. The memory integrated circuit will have to be removed to accomplish this, as reprogramming the device in-circuit is not really possible. One advantage is any mileage could actully be entered.

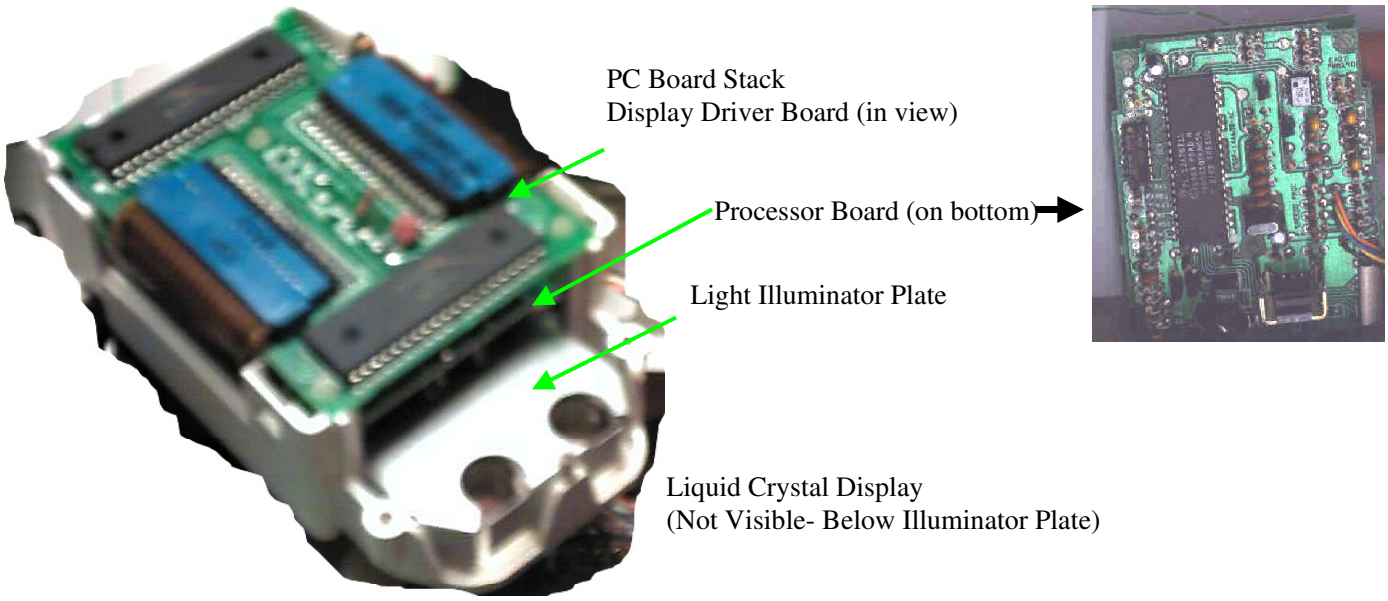
B) How to zero the odometer, or reset the mileage to xxxxx miles?

There is no jumper reset or in-cluster method of resetting the odometer value. The nonvolatile memory integrated circuit must be removed and put in some form of test jig where the programming lines must be stepped through to set the memory to new values. This requires a method of reading(1 line) and writing (5 lines) digital TTL level signal lines to the integrated circuit. The value stored is the mileage contained over two 16 bit words. The upper three bits of each 16 bit words are parity values, which are still not known how to calculate precisely. But for the 0 value odometer case, all 3 parity bits of each word are 1's.

Functional Description

The digital odometer is actually a submodule inside the electronic cluster that provides the speedometer, trip, and odometer functions.

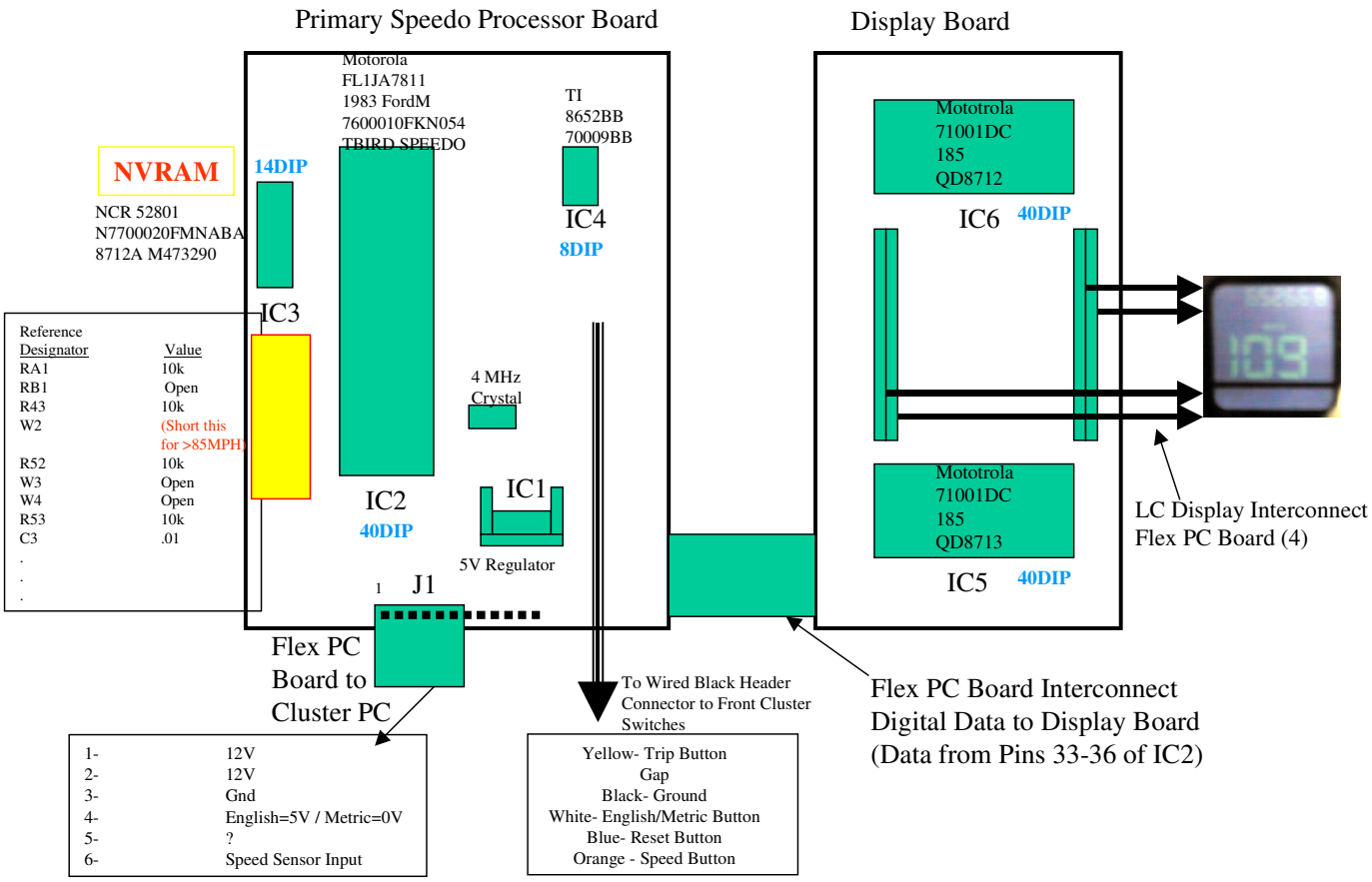
A detailed view of the speedometer/odometer is shown below:



The odometer submodule shown above consists of:

1) a light illuminator plate which contains green film to provide the green light coloring of the display and to disperse the light from two illuminator bulbs to provide backlighting for the liquid crystal display (LCD)
2) a liquid crystal display with clear brown flex interconnect PC board
3) a display driver board, containing two large Motorola integrated circuits that serve to drive the LC display.
4) a processor board, containing a Motorola the processor running at 4 MHz which counts the pulses coming from the speedometer gear mounted in the transmission housing, and stores the data in a Nitron/NCR nonvolatile 14 pin DIP RAM chip that has 16 memory locations that are each 16 bits wide.

A detailed view of the board devices are shown below:



Memory was a rarer commodity back in the 1980's (remember the IBM XT was introduced with a whopping 64 kbytes of RAM in 1983). Even more rare is memory that can hold its values when power is removed, making it a special class of memory known as nonvolatile random access memory, or NVRAM. Even today, nonvolatile memory (such as that used with USB thumb drives) does have a limited life. When a certain age and number of writes is reached, the memory will begin wearing out, no longer retaining the stored values. This was even more the case for the early memory used in the Cougar/T-Bird odometers, as memory semiconductors in the 80's was more crude than todays manufacturing standards. The specification of the odometer memory is typically with 5000 writes, the memory will last 8-12 years, and with 12000 writes the expected lifetime drops to 4-6 years.

Because of concerns of the memory lifetime for a vehicle application, Ford had engineering try to make the odometer as robust as possible. This is done by not continually writing the data to the same location, but rather rotate among several locations. The odometer has 8 location pairs that it steps thru, updating the data every 10 miles when moving or whenever the ignition key is turned off. So if you have a car with over 200,000 miles, the data is spread over 8 rotating locations that most likely have been written to over 2500 times, still within the typical memory specification. The details of this data location rotation in order to ensure odometer integrity is explained in two Ford Motor Company patents, # 4,710,888 and 4,803,646.

The memory in the odometer is manufactured by NCR (National Cash Register). This company was located in Dayton, Ohio, and was a major player in the early days of computers (http://www.thecorememory.com/) going all the way back to World War II participating in encryption technology, and even earlier with its mechanical cash register machines. Its since has gone through multiple splits and changes, and from what I can tell, the memory group became Symbios, which eventually was purchased by Hyundai and likely disbanded.

The data sheet of the memory device was obtained from a lucky person who actually still had a 1985 NCR data book, and was very kind enough to provide a data sheet on the NCR 52801 256 Bit (16x16) SNOS (Silicon-Nitride-Oxide-Silicon) 125Khz memory. Each memory location is 16 bits wide, and there are 16 different locations available in the memory in the odometer. Since it takes 32 bits to hold one odometer value, there are 8 possible write location pairs. Each 16 bit word uses the upper three bits for parity. Parity is a function that uses the 13 data bits of each word and derives a bit to be used as a check value to ensure the data bits have not been corrupted. One method of parity is two count the number of one values occurring in the 13 data bits, and if the sum is even, your parity bit is then a one, if it is odd, then the parity bit is a zero. The problem with having just one parity bit is if there are two errors that say a one went to a zero, and a second errored bit which a zero became a one, then the parity would be the same and an error would not be discovered. For this reason, Ford engineers uses a method where 3 bits are used for 13 bits, likely more ensuring an error be discovered.

I have not figured out how the parity bits are calculated, and this would be a nice puzzle for someone who is looking for a problem to solve.

The data is stored in miles, with 0.0625 mile resolution. This was determined by taking snapshots of the memory values at given mileages and looking how the bits varied with mileage.

The following is the format:

The first MSB 16 bit word has data bits 15 to 0. Bits 12 to 0 represent the mileage with bit 12 being the 2^21 MSB value and bit 0 being the 2^9 value.

16 Bit Word #1															
Parity			Upper Data Bits												
			21	20	19	18	17	16	15	14	13	12	11	10	9
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The remaining mileage (bits 2^8 and below) are represented in a second 16 bit word:

16 Bit Word #2															
Parity			Lower Data Bits												
			8	7	6	5	4	3	2	1	0	-1	-2	-3	-4
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

In this second word, bits 12 to 0 represent the mileage with bit 12 being the 2^8 value and bit 0 being the 2^(-4) value. The upper 3 bits of these two sixteen bit words are parity bits for the word. It does not appear that there is parity between the two words, rather the parity bits in a single 16 bit word are just for the 16 bits and are not interrelated to the other 16 bit word. The mileage stored is actually the mileage times 10, probably to get the tenth mile display readout without doing a lot of math.

So in other words, be example, the value stored for 12.75 km is given by a bit sequence of:

Power of 2	Miles	Bit	Subtotal
21	2097152	0	0
20	1048576	0	0
19	524288	0	0
18	262144	0	0
17	131072	0	0
16	65536	0	0
15	32768	0	0
14	16384	0	0
13	8192	0	0
12	4096	0	0
11	2048	0	0
10	1024	0	0
9	512	0	0
8	256	0	0
7	128	0	0
6	64	1	64
5	32	0	0
4	16	0	0
3	8	1	8
2	4	1	4
1	2	1	2
0	1	1	1
-1	0.5	1	0.5
-2	0.25	0	0
-3	0.125	1	0.125
-4	0.0625	1	0.0625
Stored Value	79.6875		
Divide by 10	7.96875	Miles	
	12.75	Km	

This would be stored as:
Word #1: ppp0000000000000
Word #2: ppp0010011111011
Where ppp are the parity values for each word, which at this point I don't know how to calculate.

Some typical values are shown below, where the true parity values read from memory are shown (the yellow columns) for a given observed mileage read-out on the display:

16 Bit Word #1																16 Bit Word #2														Observed	Calculated
Parity			Upper Data Bits													Parity			Lower Data Bits												
15	14	13	21	20	19	18	17	16	15	14	13	12	11	10	9	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Kilometers	Kilometers
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0	0.0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	12.7	12.8
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	0	1	1	0	0	1	0	1	1	0	1	0	1	0	1	0	104462.5	104462.5
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	1	104463.5	104463.5
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	104466.0	104466.1
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	104466.1	104466.1
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	104466.2	104466.3
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	0	0	0	1	1	1	1	0	104466.2	104466.2
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	104466.7	104467.0
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	1	0	0	0	1	0	1	1	0	0	0	1	1	1	1	104466.9	104466.9
0	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	1	0	104480.0	104480.0
0	1	1	0	0	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1	1	0	0	1	0	1	0	1	0	109899.9	109899.9
0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	1	1	0	1	1	1	162623.9	162624.0
0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	162631.0	162631.1

I guessed the parity values to be 111, when the data values are all 0 and this was correct, producing a 0 miles value on the display. Note if the parity bits are wrong, the odometer will read "Error" and the odometer will not function.