

DataHolster project package

Dear Fluke 80 user:

Thank you for your interest in the DataHolster project as seen in the October 1993 issue of Circuit Cellar Ink. By now, you probably understand the power and versatility of the Fluke 80 series meters and want to further extend their capabilities into the realm of data acquisition. I am offering this information packet as a courtesy to Circuit Cellar readers in hopes that enthusiasts like you will find new and creative uses for the Fluke 80 series' ultrasonic output capabilities.

This packet should provide you with enough information to thoroughly understand the Fluke 80 series data format and how to use it. I have divided the information in this packet into five sections:

- Main section: Fluke 80 series data format explanation (you're reading page 1 of this section)
- Appendix A: Segment layouts and data format charts
- Appendix B: Sample data strings and display worksheets
- Appendix C: Software listings and memory map
- Appendix D: Schematics and parts list

After reading the Circuit Cellar article and this data packet, you should be able to construct and debug your own DataHolster and even extend the data logging capabilities beyond what appears in the article. This data package was written under the assumption that readers will have a basic understanding of microprocessors, communications systems, electronics, and the use of the Fluke 80 series meters.

I'm interested in hearing about your applications for this project. Please send correspondence, questions, and feedback to the E-mail address below. Good luck!

Derek Matsunaga

Disclaimers

- (1) The John Fluke Mfg. Co., Inc. does not support the ultrasonic output on the Fluke 80 series meters. Please do not contact them with questions concerning this feature.
- (2) The data presented in this package was obtained by reverse engineering and experimental methods. Although the author has made a considerable effort to assure its accuracy, the data is in no way guaranteed to be error free.
- (3) This package contains copyrighted material. It may not be duplicated in any form, including by electronic means.
- (4) The author offers no warranties, expressed or implied, as to the applicability or accuracy of this information for any purpose.

The basics of the “Ultrasonic Output” mode on the Fluke 80 series meters

The “Ultrasonic Output” mode is enable by pressing the meter’s HOLD button for about two seconds while turning the meter on. A periodic “buzzing” noise will be heard when the meter is in this mode.

With the ultrasonic output enabled, the meter transmits the displayed data via its internal speaker. Naturally, all other audio indicators are non-functional when the meter is in this mode. The data output is heard as a “buzzing” noise which occurs about twice every second, depending on the measurement mode of the meter. When in Ultrasonic Output mode, the measurement rate of the meter is reduced by about half. Each occurrence of the “buzz” sound represents a string of 36 four bit words (nybbles) that contain information about which segments and annunciators (indicators) are lit-up on the meter’s display.

The output signal is modulated on a carrier of about 16.6KHz. The bit time is approximately 1.3ms. Each four bit word is lead by a start bit (logical 1) and terminated by a stop bit (logical 0). This allows fairly accurate asynchronous detection because a word counter can be reset every time a start bit is received, thus eliminating timing error stack-up. A sample timing diagram is shown below:

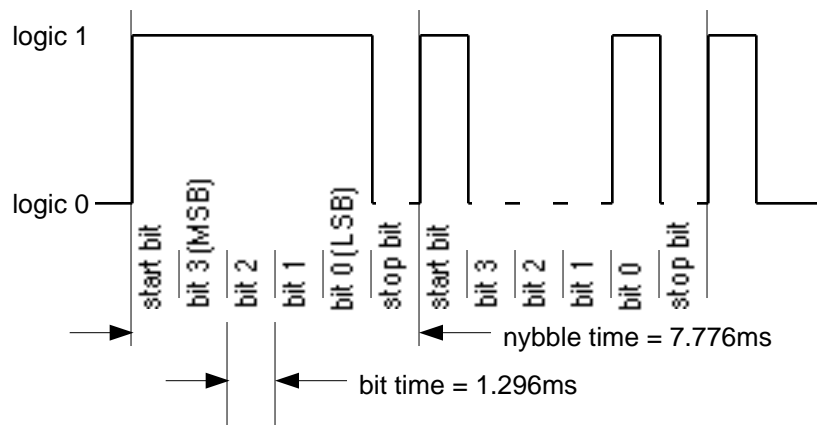


Figure 1: Nybble timing diagram

With the start bit, the stop bit, and four data bits each taking about 1.3ms, the total time to transmit a single word is 7.8ms. Since there are 36 words per data string, each string will require a total of about 280ms of transmission time. The time between data strings varies with the measurement mode of the meter, but it is always at least one word length (7.8ms).

Each string of data contains a three word header and a one word trailer, which leaves 32 words for actual measurement information. I'm not sure what the function of the header is, but the trailer word is a checksum for all previously transmitted words in the string.

Each bit of the 32 data words (128 bits total) indicates the status of a specific segment or annunciator on the meter’s display. The bar graph segments are included in the transmission. An example of an entire record transmission is shown below:

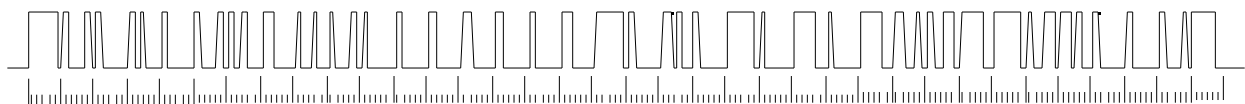


Figure 2: Example record

The large ticks on the scale represent the start bit for each nybble. The signal is representative of what would appear on U3B pin 7 (see the schematic in Appendix D) while acquiring a data record from the meter. It is then fed into the microprocessor for decoding, where the the microprocessor removes the start bits and stop bits and stores the data bits for later processing.

The data format

Once the microprocessor has acquired the data and stripped-off the start and stop bits, it must decode the data. Each data bit in the string represents the status of a specific segment on the meter's display. A logic 1 indicates that the segment is on while a logic zero indicates that the segment is off. So, the software must look for seven-segment "signatures" to decode the data.

The segment layouts and data formats are shown in Appendix A. Note that the data format for the 87 is different than that of the 83 and 85. Appendix B contains sample data strings and display worksheets so you can test your understanding of the data format. Using the data format charts to determine which segments are lit, fill-in the segments on the worksheet. The answers are on the last page of the Appendix B.

There are probably several ways in which to decode the numerical data, but one method is to combine the low and high segment nibbles into one 8 bit word and do comparisons with known "signatures". The software listings in Appendix C illustrate this method.

Using the checksum

The last nybble transmitted by the meter is a checksum derived from all previously transmitted nybbles. This number can be used to test the integrity of the received data.

The meter determines the checksum by adding-up all of the previously transmitted nybbles, masking-off everything but the lower four bits, and subtracting 1. For example, if the checksum nybble is 1100 (decimal 12), then the sum of all previously transmitted words would be 13. If the lower four bits of the sum is 0000 (decimal 0), then subtracting 1 should yield 1111 (decimal 15). The software makes this calculation (see code listings) and determines if the data is valid.

An actual example of this process is shown below. This example uses the Fluke 87: Sample #1 string from Appendix B.

<u>nybble #</u>	<u>Segment register name</u>	<u>Binary value</u>	<u>Decimal value</u>	<u>Hex value</u>
1	header 1	1111	15	\$f
2	header 2	0001	1	\$1
3	header 3	0000	0	\$0
4	S0	1000	8	\$8
5	S16	0000	0	\$0
6	S1	0001	1	\$1
7	S17	0100	4	\$4
8	S2	1000	8	\$8
9	S18	0000	0	\$0
10	S3	0001	1	\$1
11	S19	0000	0	\$0
12	S4	0000	0	\$0
13	S20	0000	0	\$0
14	S5	1000	8	\$8
15	S21	0000	0	\$0
16	S6	0000	0	\$0
17	S22	0000	0	\$0
18	S7	1111	15	\$f
19	S23	0000	0	\$0
20	S8	1010	10	\$a
21	S24	1000	8	\$8
22	S9	1110	14	\$e
23	S25	0000	0	\$0
24	S10	1111	15	\$f
25	S26	0000	0	\$0

26	S11	1011	11	\$b
27	S27	0000	0	\$0
28	S12	0000	0	\$0
29	S28	1100	12	\$c
30	S13	1001	9	\$9
31	S29	1101	13	\$d
32	S14	0000	0	\$0
33	S30	0000	0	\$0
34	S15	0000	0	\$0
35	S31	0001	1	\$1
35	checksum	1001	9	\$9

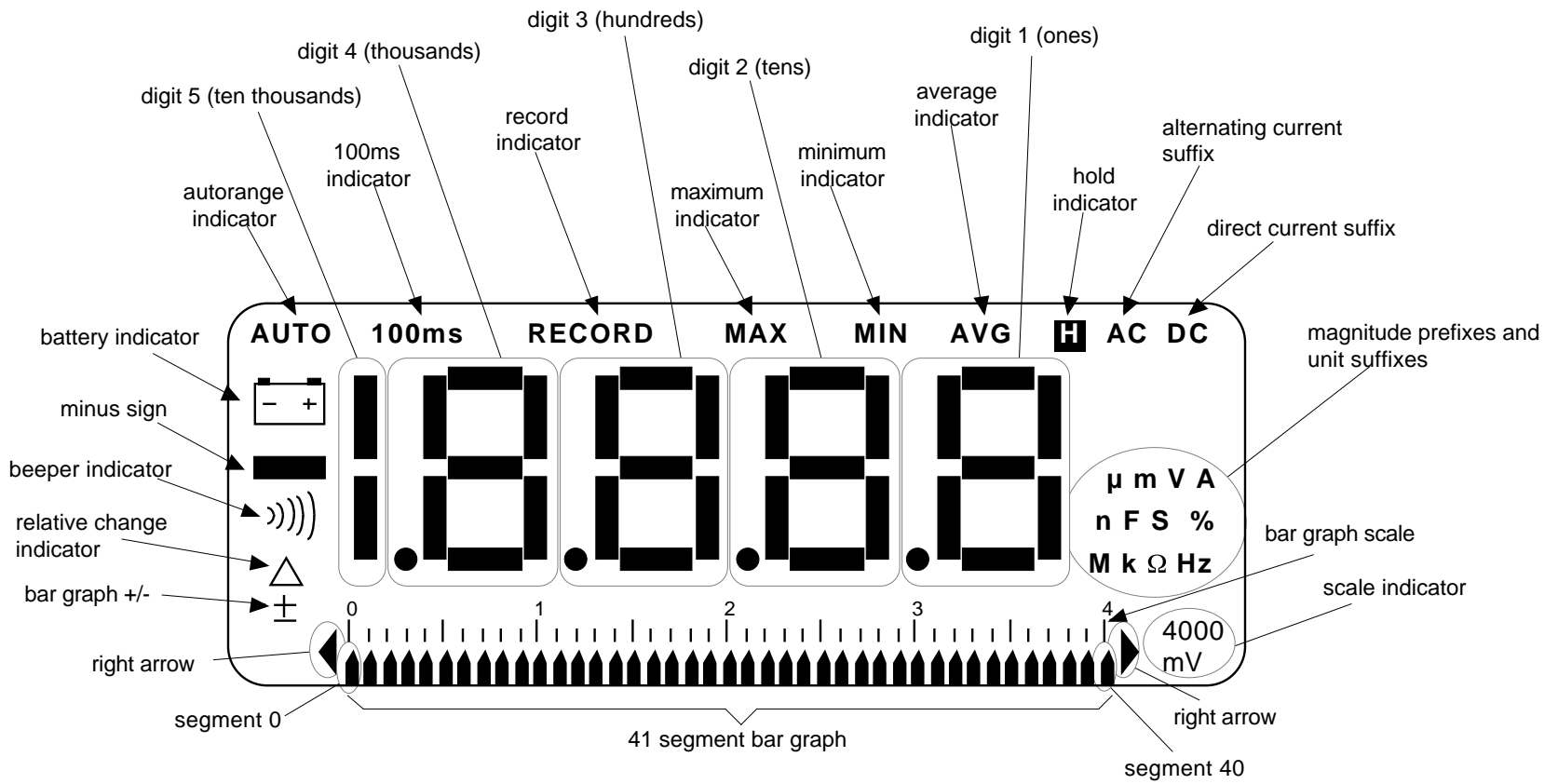
	total (minus checksum)	10011010	154	\$9a

The total of all data words is 10011010 in 8 bit binary. To test the validity of the data, the upper four bits of the sum are masked (i.e. $10011010 \cdot 00001111$) to obtain the lower four bits of the sum. The four bit value of the sum is then 1010 (hex \$0a, decimal 10). Now 1 is subtracted from this value to yield 1001 (hex \$09, decimal 9). This value matches the checksum value (nybble 35) so the data is valid.

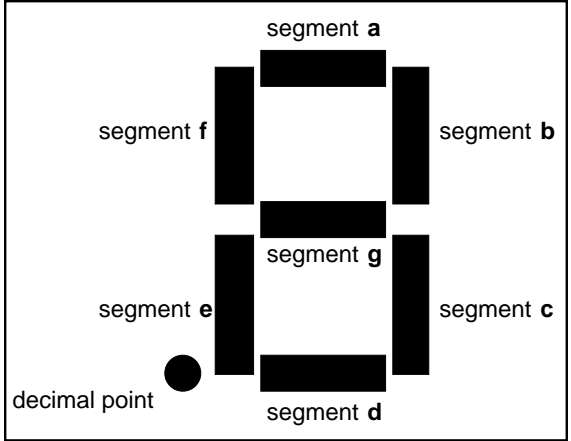
It should be noted that this checksum method is not bullet proof. It can only be guaranteed to detect a single fault in the data. However, other clever methods may be incorporated in the decoding software to increase the fault tolerance of the error detection. For instance, the software may check to determine if more than one units indicator is lit. If so, then the data is faulty. Similarly, the software may check for the presence of more than one magnitude indicator (such as kilo and nano at the same time). The software could also check for the presence of more than one decimal point, illegal seven segment signatures, etc.

Appendix A

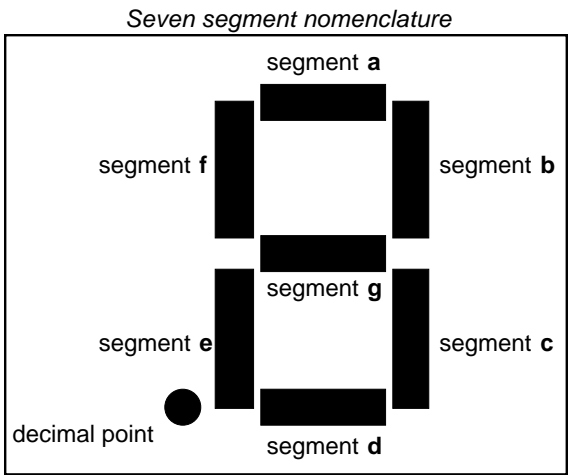
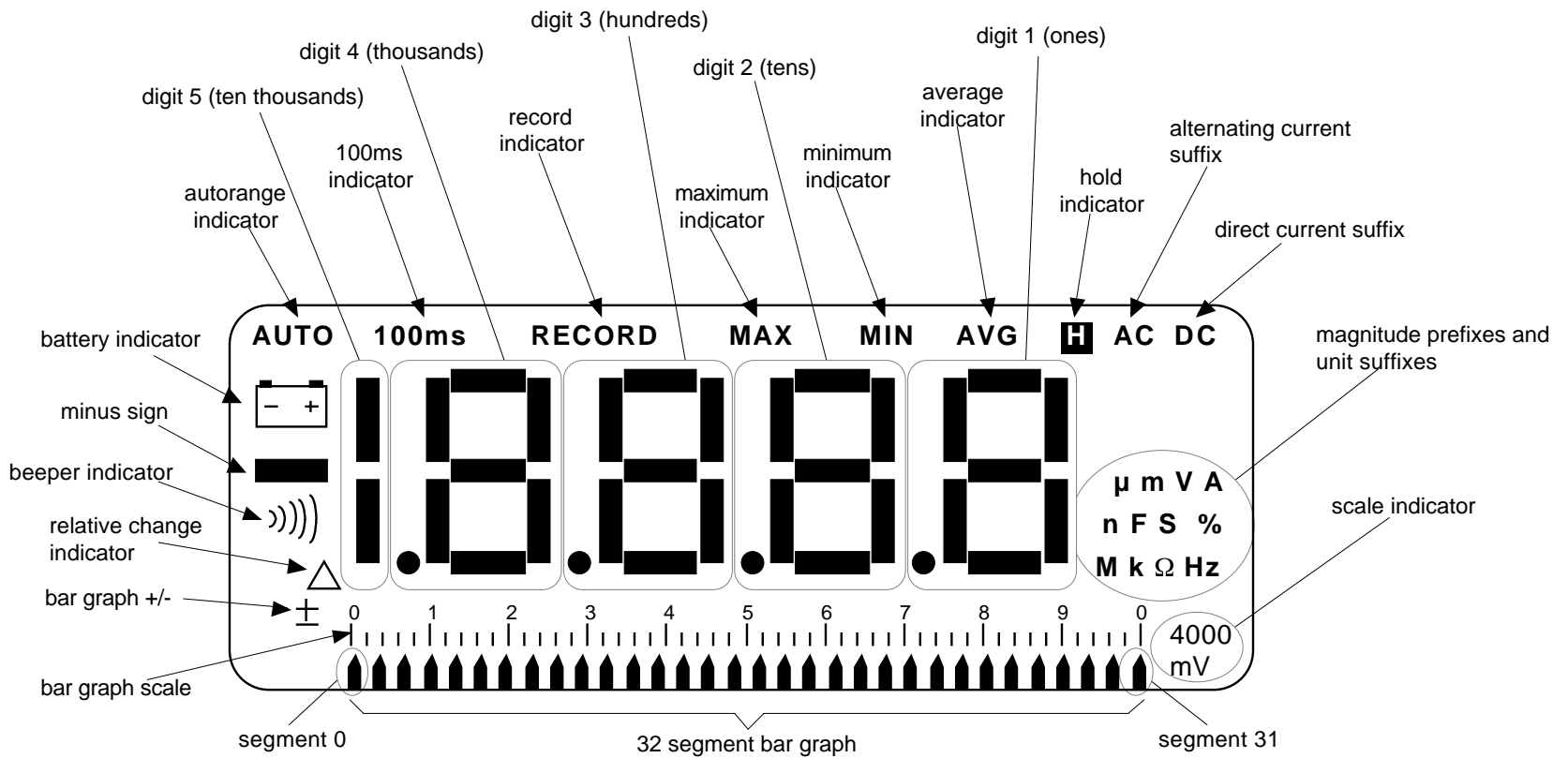
Segment layouts and data format charts



Seven segment nomenclature



Nybble	Segment register name	Bit	Function	Nybble	Segment register name	Bit	Function	Nybble	Segment register name	Bit	Function
1	header 1	1	header (function unknown)	13	S20	49	unused	25	S26	97	bar graph, segment 25
		2	header (function unknown)			50	bar graph, segment 7			98	bar graph, segment 28
		3	header (function unknown)			51	bar graph, segment 8			99	bar graph, segment 27
		4	header (function unknown)			52	bar graph, segment 9			100	bar graph, segment 26
2	header 2	5	header (function unknown)	14	S5	53	unused	26	S11	101	digit 3, decimal point
		6	header (function unknown)			54	unused			102	digit 3, segment e
		7	header (function unknown)			55	MIN (minimum indicator)			103	digit 3, segment g
		8	header (function unknown)			56	MAX (maximum indicator)			104	digit 3, segment f
3	header 3	9	header (function unknown)	15	S21	57	unused	27	S27	105	unused
		10	header (function unknown)			58	bar graph, segment 12			106	bar graph, segment 29
		11	header (function unknown)			59	bar graph, segment 11			107	bar graph, segment 30
		12	header (function unknown)			60	bar graph, segment 10			108	unused
4	S0	13	mV (scale millivolts indicator)	16	S6	61	digit 1, segment d	28	S12	109	digit 4, segment d
		14	4000 (scale indicator)			62	digit 1, segment c			110	digit 4, segment c
		15	DC (direct current suffix)			63	digit 1, segment b			111	digit 4, segment b
		16	AC (alternating current suffix)			64	digit 1, segment a			112	digit 4, segment a
5	S16	17	bar graph -	17	S22	65	unused	29	S28	113	bar graph, segment 34
		18	(relative change indicator)			66	bar graph, segment 13			114	bar graph, segment 31
		19	minus sign			67	bar graph, segment 14			115	bar graph, segment 32
		20	AUTO (autorange indicator)			68	bar graph, segment 15			116	bar graph, segment 33
6	S1	21	Hz (Hertz unit)	18	S7	69	digit 1, decimal point	30	S13	117	digit 4, decimal point
		22	% (percent indicator)			70	digit 1, segment e			118	digit 4, segment e
		23	A (Ampere unit)			71	digit 1, segment g			119	digit 4, segment g
		24	unused			72	digit 1, segment f			120	digit 4, segment f
7	S17	25	unused	19	S23	73	bar graph, segment 16	31	S29	121	unused
		26	left arrow			74	bar graph, segment 19			122	bar graph, segment 37
		27	bar graph, segment 0			75	bar graph, segment 18			123	bar graph, segment 36
		28	unused			76	bar graph, segment 17			124	bar graph, segment 35
8	S2	29	(Ohms unit)	20	S8	77	digit 2, segment d	32	S14	125	digit 5, segments b and c
		30	S (seconds unit)			78	digit 2, segment c			126	100ms indicator digits "00"
		31	V (Volts unit)			79	digit 2, segment b			127	100ms indicator digits "ms"
		32	unused			80	digit 2, segment a			128	RECORD indicator
9	S18	33	bar graph scale labels on/off	21	S24	81	unused	33	S30	129	bar graph, segment 38
		34	bar graph, segment 1			82	bar graph, segment 21			130	bar graph, segment 39
		35	bar graph, segment 2			83	bar graph, segment 20			131	bar graph, segment 40
		36	bar graph, segment 3			84	unused			132	unused
10	S3	37	k (kilo prefix)	22	S9	85	digit 2, decimal point	34	S15	133	bar graph +
		38	F (Farads suffix)			86	digit 2, segment e			134	beeper indicator
		39	m (milli prefix)			87	digit 2, segment g			135	battery indicator
		40	H (hold indicator)			88	digit 2, segment f			136	100ms indicator digit "1"
11	S19	41	bar graph scale on/off	23	S25	89	unused	35	S31	137	4 (scale indicator)
		42	bar graph, segment 6			90	bar graph, segment 22			138	right arrow
		43	bar graph, segment 5			91	bar graph, segment 23			139	400 (scale indicator)
		44	bar graph, segment 4			92	bar graph, segment 24			140	40 (scale indicator)
12	S4	45	M (mega prefix)	24	S10	93	digit 3, segment d	36	checksum	141	checksum bit 3 (MSB)
		46	n (nano prefix)			94	digit 3, segment c			142	checksum bit 2
		47	μ (micro prefix)			95	digit 3, segment b			143	checksum bit 1
		48	AVG (average indicator)			96	digit 3, segment a			144	checksum bit 0 (LSB)



Nybble	Segment register name	Bit	Function	Nybble	Segment register name	Bit	Function	Nybble	Segment register name	Bit	Function
1	header 1	1	header (function unknown)	13	S20	49	bar graph, segment 7	25	S26	97	bar graph, segment 31
		2	header (function unknown)			50	bar graph, segment 6			98	bar graph, segment 30
		3	header (function unknown)			51	bar graph, segment 5			99	bar graph, segment 29
		4	header (function unknown)			52	bar graph, segment 4			100	bar graph, segment 28
2	header 2	5	header (function unknown)	14	S5	53	H (hold indicator)	26	S11	101	digit 3, segment c
		6	header (function unknown)			54	unused			102	digit 3, segment d
		7	header (function unknown)			55	unused			103	digit 3, segment a
		8	header (function unknown)			56	unused			104	digit 3, segment b
3	header 3	9	header (function unknown)	15	S21	57	bar graph, segment 8	27	S27	105	unused
		10	header (function unknown)			58	bar graph, segment 9			106	unused
		11	header (function unknown)			59	bar graph, segment 10			107	unused
		12	header (function unknown)			60	bar graph, segment 11			108	unused
4	S0	13	AC (alternating current suffix)	16	S6	61	AVG (average indicator)	28	S12	109	digit 3, segment e
		14	DC (direct current suffix)			62	MIN (minimum indicator)			110	digit 3, decimal point
		15	unused			63	MAX (maximum indicator)			111	digit 3, segment f
		16	unused			64	RECORD (record indicator)			112	digit 3, segment g
5	S16	17	battery indicator	17	S22	65	bar graph, segment 15	29	S28	113	unused
		18	minus sign			66	bar graph, segment 14			114	unused
		19	beeper indicator			67	bar graph, segment 13			115	unused
		20	digit 5, segments b and c			68	bar graph, segment 12			116	unused
6	S1	21	A (Ampere unit)	18	S7	69	digit 1, segment c	30	S13	117	digit 4, segment c
		22	% (percent indicator)			70	digit 1, segment d			118	digit 4, segment d
		23	Hz (Hertz unit)			71	digit 1, segment a			119	digit 4, segment a
		24	4 (scale indicator)			72	digit 1, segment b			120	digit 4, segment b
7	S17	25	unused	19	S23	73	bar graph, segment 16	31	S29	121	unused
		26	AUTO (autorange indicator)			74	bar graph, segment 17			122	unused
		27	function unknown			75	bar graph, segment 18			123	unused
		28	(relative change indicator)			76	bar graph, segment 19			124	unused
8	S2	29	V (Volts unit)	20	S8	77	digit 1, segment e	32	S14	125	digit 4, segment e
		30	S (seconds unit)			78	digit 1, decimal point			126	digit 4, decimal point
		31	(Ohms unit)			79	digit 1, segment f			127	digit 4, segment f
		32	100ms record indicator			80	digit 1, segment g			128	digit 4, segment g
9	S18	33	unused	21	S24	81	bar graph, segment 23	33	S30	129	unused
		34	bar graph -			82	bar graph, segment 22			130	unused
		35	bar graph +			83	bar graph, segment 21			131	unused
		36	unused			84	bar graph, segment 20			132	unused
10	S3	37	m (milli prefix)	22	S9	85	digit 2, segment c	34	S15	133	100ms indicator digit "1"
		38	F (Farads suffix)			86	digit 2, segment d			134	100ms indicator digit "00"
		39	k (kilo prefix)			87	digit 2, segment a			135	100ms indicator digit "m"
		40	bar graph scale on/off			88	digit 2, segment b			136	100ms indicator digit "s"
11	S19	41	bar graph, segment 0	23	S25	89	bar graph, segment 24	35	S31	137	mV (scale millivolts indicator)
		42	bar graph, segment 1			90	bar graph, segment 25			138	4000 (scale indicator)
		43	bar graph, segment 2			91	bar graph, segment 26			139	400 (scale indicator)
		44	bar graph, segment 3			92	bar graph, segment 27			140	40 (scale indicator)
12	S4	45	μ (micro prefix)	24	S10	93	digit 2, segment e	36	checksum	141	checksum bit 3 (MSB)
		46	n (nano prefix)			94	digit 2, decimal point			142	checksum bit 2
		47	M (mega prefix)			95	digit 2, segment f			143	checksum bit 1
		48	DUTY (unused)			96	digit 2, segment g			144	checksum bit 0 (LSB)

Appendix B

Sample data strings and display worksheets

Sample data strings

Fluke 87: Sample #1

1111 0001 0000 1000 0000 0001 0100 1000 0000 0001 0000 0000 0000 1000 0000 0000 0000 1111 0000 1010 1000 1110 0000 1111 0000 1011 0000 0000 1100 1001 1101 0000 0000 0000 0001 1001

Fluke 87: Sample #2

1111 0001 0000 0100 0100 0001 0100 1000 0100 0001 0100 0000 0000 0000 0000 0000 1110 0000 1011 0000 1110 0000 0111 0000 1111 0000 1010 1100 1001 0111 0000 1111 0000 0001 0000

Fluke 87: Sample #3

1111 0001 0000 0100 0000 0001 0000 1000 0010 1001 0000 0000 0000 0000 0000 0000 1111 0000 0101 0000 1110 0001 0011 0000 1111 0010 0011 1110 1111 1110 0001 0001 0000 1011 1001

Fluke 87: Sample #4

1111 0001 0000 0000 0000 0001 0100 0010 0010 0011 0001 0000 0000 1000 0000 0000 0000 1001 0000 0000 0000 1110 0000 1011 0000 1001 0010 0100 1110 0111 0000 1001 0001 0000 0000 0100

Fluke 87: Sample #5

1111 0001 0000 0000 0000 0000 0100 0000 0010 0100 0000 0100 0000 0000 0000 0000 1110 0000 0011 0000 1111 0000 0101 0000 1111 1101 1010 0101 1111 0000 1010 0000 0000 0000 0110

Fluke 87: Sample #6

1111 0001 0000 0100 0000 1001 0100 0000 0010 1001 0100 0000 0000 1000 0000 0000 0000 0111 0000 1001 0000 1110 0000 0111 0000 1111 1000 1010 0110 1111 0000 1010 0000 0000 0001 1101

Fluke 87: Sample #7

1111 0001 0000 0100 0000 1001 0100 0000 0010 0001 0000 1000 0000 0000 0000 0000 0000 1001 1000 0011 0000 1001 0000 0000 0000 1110 0100 0011 0000 1111 0100 1010 0000 0000 0111 0001

Fluke 83/85: Sample #1

1111 0001 0000 0001 0001 0000 0010 0010 1111 0001 1111 0000 0111 0000 0111 1111 0111 0101 1001 1101 0000 1111 0000 0111 0000 0000 0000 0110 0000 0000 0000 0000 0000 1001 1000

Fluke 83/85: Sample #2

1111 0001 0000 0010 1011 0000 0010 0010 1111 0000 1111 0000 0111 0000 0001 1101 0000 0111 0000 1101 0000 1011 0000 1111 0000 0101 0000 0110 0000 0000 0000 0000 0000 1001 0101

Fluke 83/85: Sample #3

1111 0001 0000 1010 0000 0000 0010 0010 1111 0010 1111 0000 0111 0000 0111 1111 0111 1010 1111 1101 0110 0011 0111 1111 1111 0011 0110 1111 1111 0010 0111 0000 1100 1000 1011 0100

Fluke 83/85: Sample #4

1111 0001 0000 0000 0001 0000 0010 1000 1111 1001 1111 0000 0111 0000 0111 0110 0111 0000 1111 1101 0110 0111 0000 0110 0000 1000 0000 1011 0000 0110 0000 0000 0000 1000 1000 0100

Fluke 83/85: Sample #5

1111 0001 0000 0000 0001 0000 0000 0000 0000 0100 0000 0100 0000 0000 0000 1101 0000 0011 0000 1111 0000 1010 0000 1111 0000 0101 0000 1111 0000 0101 0000 0000 0000 1000 0000 0001

Fluke 83/85: Sample #6

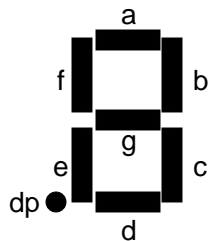
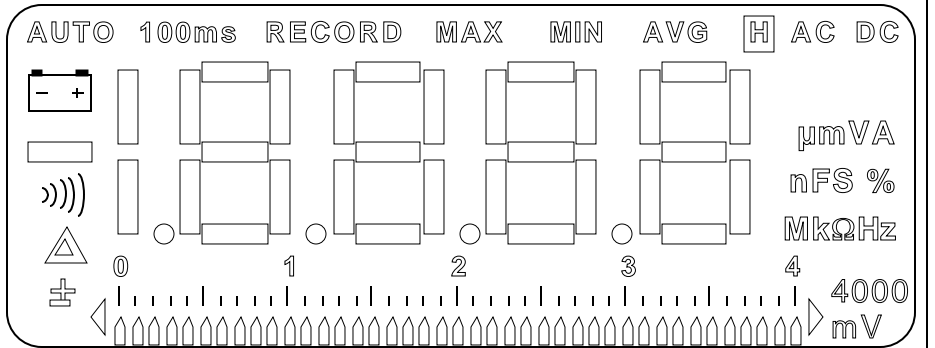
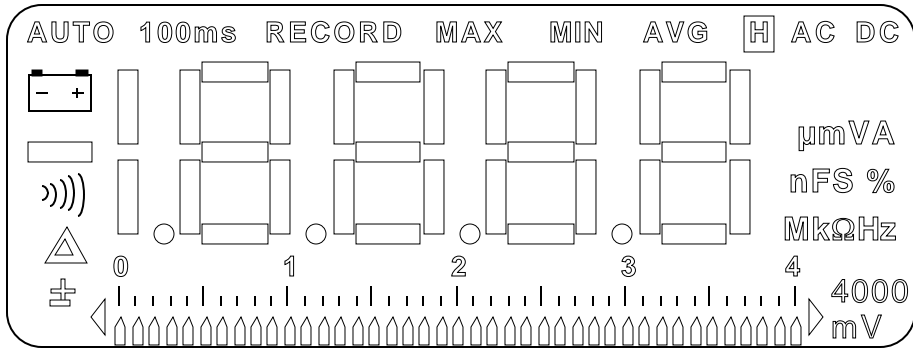
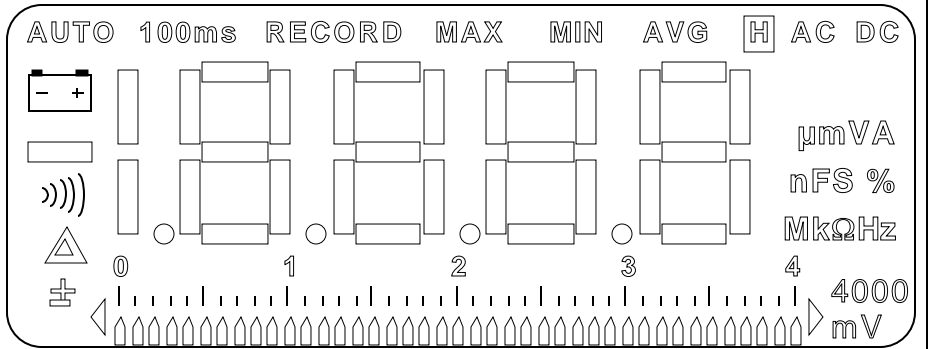
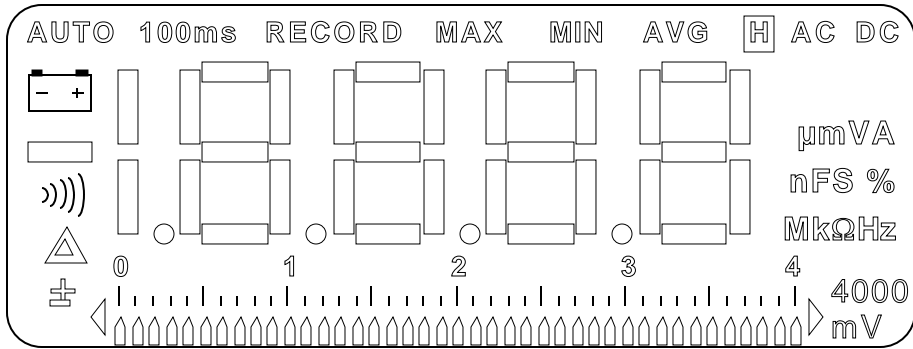
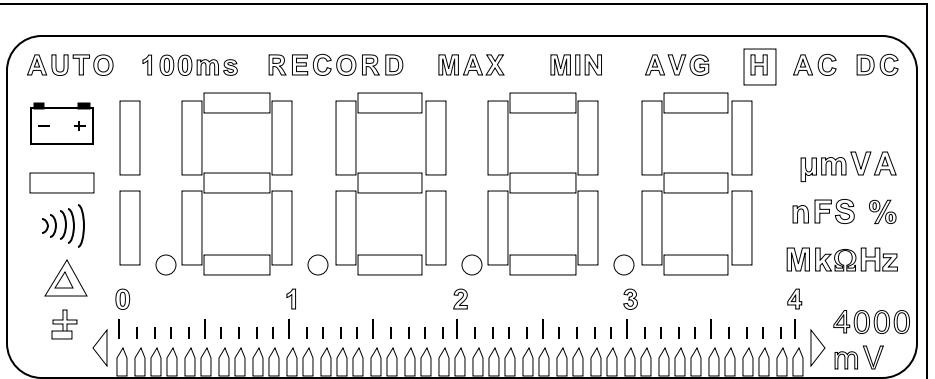
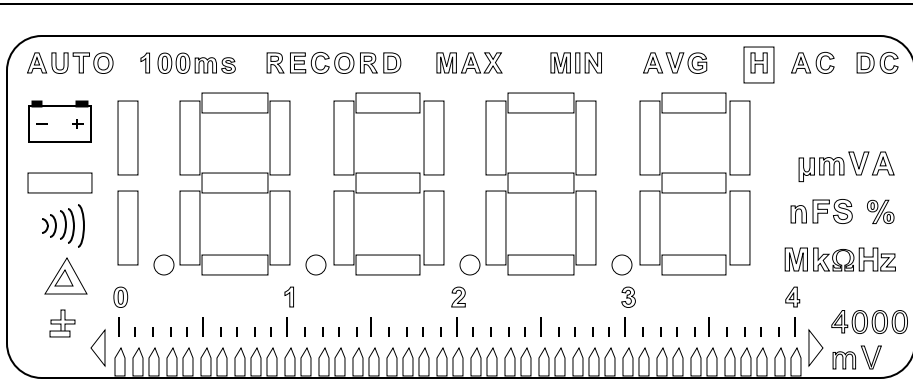
1111 0001 0000 0010 0001 0010 0010 0000 1000 0011 1000 0000 0000 0000 0000 1011 0000 0110 0000 1101 0000 1011 0000 1111 0000 0101 0000 1111 0000 0101 0000 0000 0000 1000 1001 1011

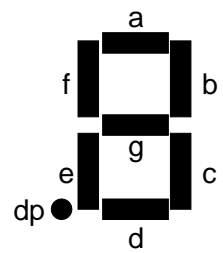
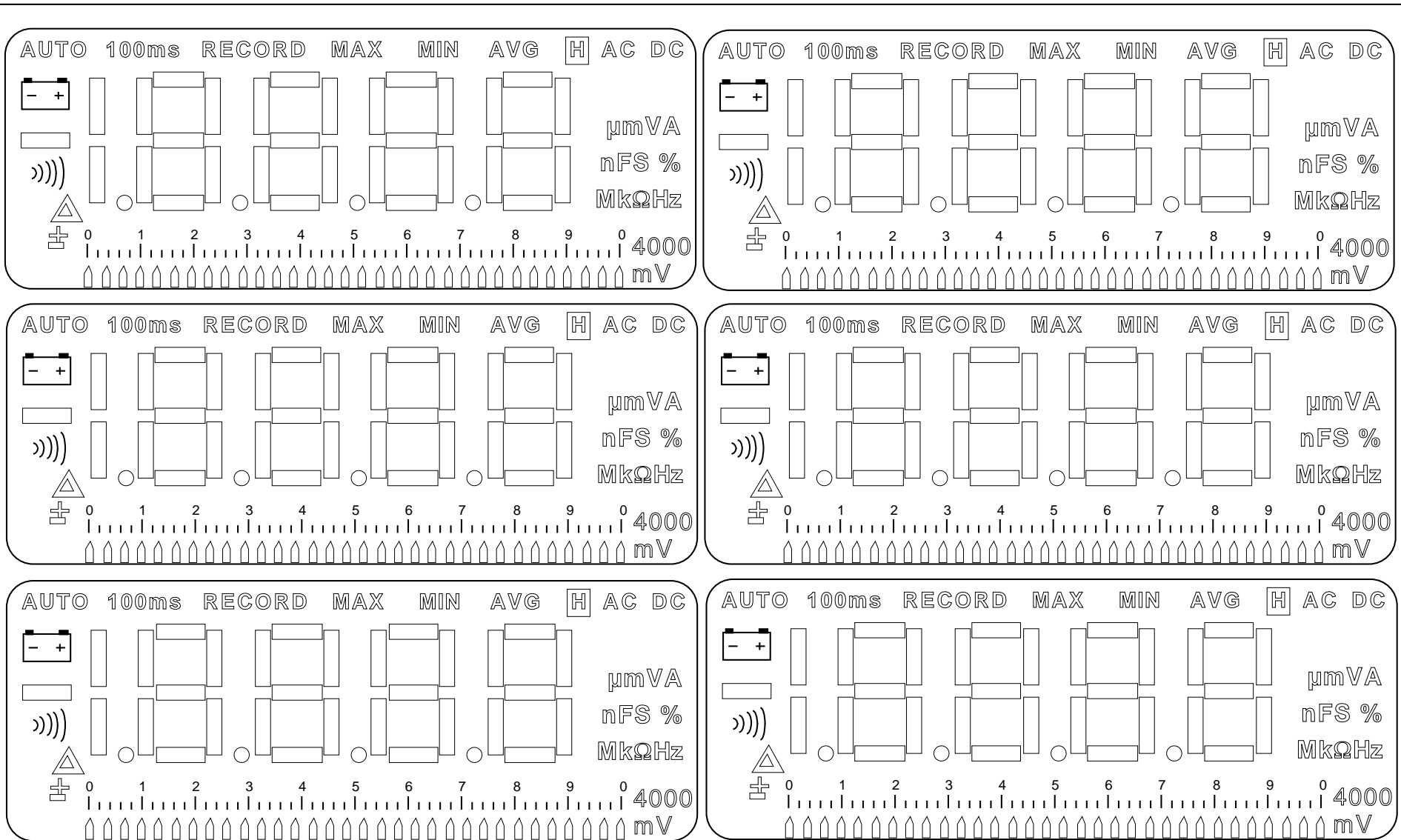
Fluke 83/85: Sample #7

1111 0001 0000 0110 0001 0010 0010 0000 1111 0000 1011 0010 0000 0000 0000 0110 0000 0011 0000 0110 0000 0000 0000 1101 0000 0011 0000 1111 0000 0101 0000 0000 0000 1000 1011 1100

Fluke 83/85: Sample #8

1111 0001 0000 0000 0001 0000 0010 1000 1111 0000 1111 1000 0111 0000 0111 0000 0111 0000 1111 1000 0110 1101 0111 1111 1111 0101 0110 0000 1111 0000 0111 0000 1110 0000 1101 0000





Answers to sample data strings

Fluke 87: Sample #1

17.60VAC, HOLD, AUTO, bg 7.4, range 40

Fluke 87: Sample #2

-10.56VDC, AUTO, bg -.4, range 40

Fluke 87: Sample #3

395.3mvDC, AUTO, bg +8.8, range 400mV

Fluke 87: Sample #4

2.161k , HOLD, AUTO, bg 1.0, range 4

Fluke 87: Sample #5

00.35nF, AUTO, bg+, no bar graph scale

Fluke 87: Sample #6

00.52mADC, HOLD, AUTO, bg+.4, range 40

Fluke 87: Sample #7

0514 μ ADC, AUTO, bg +5.2, range 4000

Fluke 83/85: Sample #1

17.60VAC, HOLD, AUTO, bg 1.7, range 40

Fluke 83/85: Sample #2

-10.56VDC, AUTO, bg -1.0, range 40

Fluke 83/85: Sample #3

395.3mvDC, AUTO, bg +3.9, range 400mV

Fluke 83/85: Sample #4

2.161k , AUTO, HOLD, bg +2.1, range 4

Fluke 83/85: Sample #5

00.35nF, AUTO, bg+, no bar graph scale

Fluke 83/85: Sample #6

00.52mADC, HOLD, AUTO, bg+0, range 40

Fluke 83/85: Sample #7

0514 μ ADC, AUTO, bg+.5, range 4000

Fluke 83/85: Sample #8

OL, M , AUTO, bg 4.0, range 40, right arrow

Appendix C

Software listings and memory map

DataHolster sourcecode RAM map

(see 68HC705K1 technical data for a complete system memory map)

Note: Some addresses are named more than once for code clarity (readability). This was done so that different functional sections of the sourcecode can refer to the same location by different (more meaningful) names than other sections. This memory map applies to both the Fluke 87 and Fluke 83/85 versions of the code.

<u>Address</u>	<u>Name</u>	<u>Function during acquisition</u>	<u>Function during decoding</u>	<u>Function during serial output</u>
\$00e0	templ	"high" counting byte	digit information	(not used)
\$00e1	lcount	number of low cycles detected	(variable name not used)	(not used)
	lowS	(variable name not used)	the low S register	(not used)
\$00e2	hcount	number of high cycles detected	(variable name not used)	(not used)
	highS	(variable name not used)	the high S register	(not used)
\$00e3	scount	(variable name not used)	offset of next S register	(variable name not used)
	bitcount	number of bits acquired	(variable name not used)	the number of SERIAL bits sent
\$00e4	bytecount	number of bytes acquired	offset of next ASCII string byte	the number of ASCII bytes sent
\$00e5	firstbyte	Fluke header	the base address of S register RAM	(variable name not used)
	string	(variable name not used)	the base address of string RAM	the base address of the output string
\$00e6		S0	(not used)	(not used)
\$00e7		S16,S1	(not used)	(not used)
\$00e8		S17,S2	(not used)	(not used)
\$00e9		S18,S3	(not used)	(not used)
\$00ea		S19,S4	(not used)	(not used)
\$00eb		S20,S5	(not used)	(not used)
\$00ec		S21,S6	(not used)	(not used)
\$00ed		S22,S7	(not used)	(not used)
\$00ee		S23,S8	(not used)	(not used)
\$00ef		S24,S9	(not used)	(not used)
\$00f0		S25,S10	(not used)	(not used)
\$00f1		S26,S11	(not used)	(not used)
\$00f2		S27,S12	ASCII output byte 11	ASCII output byte 11
\$00f3		S28,S13	ASCII output byte 10	ASCII output byte 10
\$00f4		S29,S14	ASCII output byte 9	ASCII output byte 9
\$00f5		S30,S15	ASCII output byte 8	ASCII output byte 8
\$00f6		S31,Fluke checksum	ASCII output byte 7	ASCII output byte 7
\$00f7		data checksum	ASCII output byte 6	ASCII output byte 6
\$00f8		(not used)	ASCII output byte 5	ASCII output byte 5
\$00f9		(not used)	ASCII output byte 4	ASCII output byte 4
\$00fa		(not used)	ASCII output byte 3	ASCII output byte 3
\$00fb		(not used)	ASCII output byte 2	ASCII output byte 2
\$00fc		(not used)	ASCII output byte 1	ASCII output byte 1
\$00fd		(not used)	ASCII output byte 0	ASCII output byte 0
\$00fe		(not used)	program counter	(not used)
\$00ff		(not used)	program counter	(not used)


```

*****
* Fluke 83/85 Data Acquisition sourcecode *
* *
* Author: Derek Matsunaga *
* Revision: 2.283 *
* Revision date: October 2, 1993 *
* Processor: 68HC705K1 *
* *
* @1993 Derek Matsunaga *
* *
* Revision 2.2 corrects serial timing. *
*****

```

```
* Set-up 68HC05K1 equate table.
```

```

porta      equ      $0000      ;Port a address.
portb      equ      $0001      ;Port b address.
ddra       equ      $0004      ;Data direction register a.
ddrb       equ      $0005      ;Data direction register b.

```

```
* Define RAM locations.
```

```

org        $00e0      ;The beginning of RAM.
templ     equ      $00e0      ;A general purpose temporary register.
lcount    equ      $00e1      ;The register which keeps track of low time.
lowS      equ      $00e1      ;The low "S" register from the meter.
hcount    equ      $00e2      ;The register which keeps track of high time.
highS     equ      $00e2      ;The high "S" register from the meter.
scount    equ      $00e3      ;The "S" register counter.
bitcount  equ      $00e3      ;The number of bits per byte received.
bytecount equ      $00e4      ;The number of 8 bit bytes received (#18 max).
firstbyte equ      $00e5      ;The location of the first byte in the record.
string    equ      $00e5      ;The start of the output string.
checksum  equ      $00f8      ;The checksum register.

```

```
* Define vector table.
```

```

org        vectors
fdb        rom
fdb        rom
fdb        rom
fdb        reset

```

```
* Initialize ports and registers.
```

```

reset      org      $0200      ;The beginning of program ROM.
           lda      #$ff
           sta      ddra      ;Configure all porta pins as outputs.
           sta      porta     ;Write all ones to porta.
           clr      ddrb      ;Configure both portb pins as inputs.
           clr      bytecount  ;Initialize counting registers.
           clr      bitcount
           clr      checksum   ;Clear the checksum register.

```

```

* Since each nibble begins with a "1" as a start bit, the deadtime during
* valid data transmission will not exceed the 7.8ms nibble time
* (1.3ms/bit * 6 bits). So, a complete record of data has just been sent if deadtime
* exceeds the 7.8ms nibble time. Use about 11ms of dead time to prevent false record triggering
* (for noise immunity).

```

```

* Look for about 11ms of deadtime before reading data. Due to the length of the
* deadtime, a 16 bit counter is needed. The "templ" location is used to store the
* "high" byte (upper 8 bits) of the 16 bit counter. The X register is used to store the "low"
* byte.

```

```

start      clr      X          ;Clear the X register.
           stx      templ     ;Clear the "high" byte of X.

loop       brset    0,portb,start ;Loop & clear X until portb0 goes low.

           decx    X          ;Decrement the X register.
           bne     loop       ;If X>0, go to top of loop.

           inc     templ     ;Directly increment the high byte of the counter.
           lda     templ     ;Load the new high byte into the accumulator.
           cmp     #$08      ;Templ will be 8 when about 11.3ms of
           ;         deadtime has occurred.
           bne     loop

```

```

* About 11.3ms of deadtime has been detected at portb0. We are now ready
* to start reading the data from the meter. The beginning of the actual
* data occurs with a low to high transition following the deadtime.

```

```

* Wait until deadtime is over (a high is detected at portb0).
* Each nibble is started with a 1.3ms high (a startbit).

```

```
getstart   brclr    0,portb,getstart ;Read portb0 until it goes high.
```

```
* When the high at portb0 is detected, initialize the timer locations.
```

```

nextbit    ldx      #$91      ;Initialize the X register with the bit time.
           clr      lcount    ;Initialize the low count register.
           clr      hcount    ;Initialize the high count register.

timer      decx     X          ;Decrement X register to keep track of time.

```

```

        beq          timeout          ;Check portb until X=0 (the timer has expired).
chkport  brset      0,portb,high     ;Check for a high at portb0.
low      inc        lcount          ;Increment the low count location.
        bra        timer            ;Go back to timer.
high     inc        hcount          ;Increment the high count location.
        bra        timer            ;Go back to timer.
timeout  lda        bitcount        ;Load the bit counter.
        cmp        #$00            ;Determine if it is the start bit of the 1st nibble.
        beq        housekeep       ;Ignore this bit if it is the start bit.
        cmp        #$05            ;Determine if it is the start bit of the 2nd nibble.
        beq        housekeep       ;Ignore this bit if it is the start bit.
        lda        lcount          ;Load the number of low counts.
        sub        hcount          ;Subtract the number of high counts.
        ;C will be set if there were more highs than lows.
        ldx        bytecount       ;Load the record pointer into X.
        rol        firstbyte,x     ;Roll the carry bit onto the current record byte.
housekeep inc        bitcount        ;Increment the number of bits read.
        lda        bitcount        ;Load the incremented value into A.
        cmp        #$05            ;If bitcount is #5, get ready for next nibble.
        beq        add_chksum      ;If bitcount is #10, get ready for next byte.
        cmp        #$0a
        bne        nextbit
add_chksum lda        firstbyte,x
        and        #$0f
        add        checksum
        sta        checksum
        lda        bitcount
        cmp        #$05
        beq        getstart
nextbyte inc        bytecount        ;Increment the number of bytes read.
        lda        bytecount        ;Load the incremented number into A.
        cmp        #$12            ;Determine if #18 bytes have been read.
        beq        cs_test         ;Start processing data if #18 bytes have been read.
        clr        bitcount        ;Reset bitcount to zero.
        bra        getstart        ;Wait for the next start bit.
* The integrity of the received data is determined by cs_test. The received data is considered
* good if the received checksum correlates with the calculated checksum.
* The checksum location contains the least significant nibble of the sum of all all nibbles received,
* including the meter's checksum nibble. Because of this, the meter's checksum nibble must be
* subtracted from the calculated checksum prior to comparison.
* The meter's checksum is 1 less than the calculated checksum. (i.e. a calculated checksum
* of $0f corresponds to a received checksum of $0e).
cs_test  lda        firstbyte+$11    ;Load S31 and the meter's checksum into A.
        and        #$0f            ;Remove S31.
        sta        templ           ;Store the meter's checksum in templ.
        lda        checksum        ;Load the calculated checksum into A.
        sub        templ           ;Subtract the meter's checksum nibble.
        decx       #1              ;Subtract 1.
        and        #$0f            ;Knock-off the high nibble.
        ldx        #$18            ;Load the output string offset into X.
        stx        bytecount       ;Bytecount now points to the output string character.
checksum. cmp        templ           ;See if the calculated checksum matches the meter's
        beq        ol_test         ;If so, procede with decoding.
        ;If the checksums do not match, send an exclamation point
to       ; indicate a checksum error.
        lda        #$21            ;Load the ASCII value for an exclamation point (!).
        sta        string,x        ;Store it as the first character in the output string.
        decx       #1              ;Decrement the string pointer.
        jmp        sendstring      ;Send the exclamation point and restart the program.
* The following code segment, ol_test, determines if the overload message is on the
* display. To accomplish this, the tens digit is checked for an "L". If the "L" exists,
* then it is not necessary to process anything else. If not, then proceed with the
* numeric decoding. Note: the bottom leg of the "L" character is unique in S9 so it is used
* to check for the entire "L".
ol_test  lda        firstbyte+$09    ;Load S8.
        and        #$0f            ;Knock-off the high nibble.
        cmp        #$08            ;Check for the bottom leg of the "L" character.
        bne        sign            ;Go to the sign routine if the "L" does not exist.
        jmp        overload        ;Go to the overload routine if the "L" exists.
* No overload has been detected so now we can decode the display digits.
* The sign routine checks the acquired data for a minus sign. If the minus sign is present, the routine
* stores the ASCII value of a minus sign as the first byte of the output string. If not, the program proceeds
* to decode the ten-thousands digit.
sign     brcclr     5,firstbyte+$02,ten_k ;See if the sign bit is clear.
        lda        #$2d            ;Load the ASCII value for a minus sign.

```

```

        jsr          outbyte          ;Store a minus sign as the first byte in the output.

* The only digit that can appear in the ten-thousands place is a "1". The ten_k routine checks for the presence
* of a "1". Note that this digit will only be filled when the meter is in four digit mode. If a "1" exists, it is
* stored as the next character in the output string. If not, the program proceeds to decode the remaining four
* digits.
ten_k      brclr     3,firstbyte+$0f,start_thou    ;See if there is a 1 in the ten-thousands place.
          lda       #$31                          ;Load the ASCII value for a "1".
          jsr       outbyte                       ;Store the "1" in the output string.

* Start_thou is the beginning of the decoding routine for the remaining four display digits.

start_thou stx       bytecount                    ;Store the output string pointer.
          ldx       #$0e                          ;Load X with the offset of the first S register.
          stx       scout                         ;Store it in scout.

thousands ldx       scout                        ;Load the current scout value into X.
          cpx       #$06                          ;See if it is the last S register.
          bne       continue                     ;If it is the last S register, decode the magnitude.
          bra       magnitude

continue   lda       firstbyte,x                 ;Load S13 the 1st time, S11, S9, S7.
          sta       highS                        ;Store it in highS.
          decx      ;Decrement scout.
          lda       firstbyte,x                 ;Load S12 the 1st time, S10, S8, S6.
          sta       lowS                        ;Store it in lowS.
          decx      ;Decrement scout.
          stx       scout                       ;Store scout.
          ldx       bytecount                   ;Load the current output string offset into X.

display    lda       highS                      ;Load the high S register.
          asla      ;Move the 4 LSBs to the MSBs.
          asla
          asla
          asla
          sta       templ                       ;Store it in templ.
          lda       lowS                        ;Load the low S register.
          and       #$0f                       ;Knock-off the 4MSBs.
          ora       templ                      ;Now we have a byte which represents the digit.
          sta       templ                      ;Store this byte in templ.

* The dptest routine checks the digit to see if the decimal point is lit. If it is, dptest stores the
* ASCII value of a decimal point at the current output string location. If not, it proceeds to decode the
* numerals.
dptest     brclr     7,templ,numerals            ;Test for a decimal point.
          lda       #$2e                        ;The ASCII value for a period (decimal point).
          jsr       outbyte                     ;Store it in the current output string location.

numerals   lda       templ                      ;Load the digit information into A.
          and       #$7f                       ;Get rid of the decimal point bit.
          stx       bytecount                   ;Store the output string offset because the X register
          ; is needed for numeric decoding.

compare    clrx
          cmp       #$5f                       ;Test for a "0".
          beq       storechar                  ;If a "0" is present, go to the storechar routine.
          incx      ;Increment X if a zero is not present.
          cmp       #$06                       ;Test for a "1".
          beq       storechar                  ;If a "1" is present, go to the storechar routine.
          incx      ;Increment X if a one is not present.
          cmp       #$6b                       ;Test for a "2".
          beq       storechar                  ;If a "2" is present, go to the storechar routine.
          incx      ;Increment X if a two is not present.
          cmp       #$2f                       ;Test for a "3".
          beq       storechar                  ;If a "3" is present, go to the storechar routine.
          incx      ;Increment X if a three is not present.
          cmp       #$36                       ;Test for a "4".
          beq       storechar                  ;If a "4" is present, go to the storechar routine.
          incx      ;Increment X if a four is not present.
          cmp       #$3d                       ;Test for a "5".
          beq       storechar                  ;If a "5" is present, go to the storechar routine.
          incx      ;Increment X if a five is not present.
          cmp       #$7d                       ;Test for a "6".
          beq       storechar                  ;If a "6" is present, go to the storechar routine.
          incx      ;Increment X if a six is not present.
          cmp       #$07                       ;Test for a "7".
          beq       storechar                  ;If a "7" is present, go to the storechar routine.
          incx      ;Increment X if a seven is not present.
          cmp       #$7f                       ;Test for an "8".
          beq       storechar                  ;If an "8" is present, go to the storechar routine.
          incx      ;Increment X if an eight is not present.
          cmp       #$3f                       ;Test for a "9".
          bne       thousands                  ;If an "9" is present, go to the storechar routine.
          ;All numeric possibilities have been tested for. If none
          ; exist, proceed to decode the next digit.

of them
storechar  txa
          ;Transfer the X register to A. The X register contains
          ;A value indicative of the numeral on the display.
          add       #$30                       ;Add X to the ASCII value for a "0".
          ldx       bytecount                   ;Load X with a current output string offset.
          sta       string,x                   ;Store the ASCII value of the displayed numeral at the
current

```

```

                                ; output string position.
                                ;Decrement the output string pointer.
                                ;Repeat the numeric decoding process for the remaining
digits.      dec      bytecount
              bra      thousands

```

* The overload routine stores the ASCII values for "OL" at the current output string location.

```

overload      lda      #$4f      ;The ASCII value of an "O"
              jsr      outbyte   ;Store it in the current output string location.
              lda      #$4c      ;The ASCII value of an "L"
              jsr      outbyte   ;Store it in the current output string location.
              lda      #$20      ;The ASCII value of a space
              jsr      outbyte   ;Store it in the current output string location.
              stx      bytecount ;Store the current output string offset.

```

* The magnitude routine checks the appropriate S registers for magnitude indicators such as mega, kilo, milli, etc.

* Each magnitude designator has its own ASCII storage routine similar to that of the overload routine.

```

magnitude     ldx      bytecount
              brset   1,firstbyte+$05,micro ;Check S4 for a "micro" sign.
              brset   2,firstbyte+$05,nano  ;Check S4 for a "nano" sign.
              brset   3,firstbyte+$05,mega  ;Check S4 for a "mega" sign.
              brset   1,firstbyte+$04,milli ;Check S3 for a "milli" sign.
              brset   3,firstbyte+$04,kilo  ;Check S3 for a "kilo" sign
              bra      units                ;Go to the units routine if no magnitude designators
                                                ; are detected.

```

* The outbyte subroutine stores the contents of the accumulator at the current output string location.

* On entry, the X register must contain the offset relative to the string location (see equate table).

* The X register is decremented to prepare for the next output string byte.

```

outbyte       sta      string,x      ;Store the accumulator at string+x.
              decx      ;Decrement X.
              rts      ;Return to caller.

```

* The following routines store the ASCII values of the magnitude designators. These routines are entered from the magnitude routine.

```

micro         lda      #$75      ;Load the ASCII values for a "u".
              bra      storemag   ;Go to the magnitude storing routine.
nano          lda      #$6e      ;Load the ASCII values for an "n".
              bra      storemag   ;Go to the magnitude storing routine.
mega         lda      #$4d      ;Load the ASCII values for an "M".
              bra      storemag   ;Go to the magnitude storing routine.
milli        lda      #$6d      ;Load the ASCII values for an "m".
              bra      storemag   ;Go to the magnitude storing routine.
kilo         lda      #$4b      ;Load the ASCII values for a "K".
storemag      jsr      outbyte     ;Go to the ASCII byte storing subroutine.

```

* The units routine checks the appropriate S registers for unit indicators such as farads, volts, ohms, etc.

* Each unit designator has its own ASCII storage routine similar to that of the overload routine.

```

units         brset   2,firstbyte+$04,farads ;Check S3 for a farads indicator.
              brset   1,firstbyte+$03,volts ;Check S2 for a volts indicator.
              brset   2,firstbyte+$03,seconds ;Check S2 for a seconds indicator.
              brset   3,firstbyte+$03,ohms  ;Check S2 for a ohms indicator.
              brset   1,firstbyte+$02,amps  ;Check S1 for a amps indicator.
              brset   2,firstbyte+$02,percent ;Check S1 for a percent indicator.
              brset   3,firstbyte+$02,hertz ;Check S1 for a hertz indicator.
              bra      comments        ;Jump to the comments routine if no units indicators were
                                                ; detected.

```

* The following routines store the ASCII values of the units designators. These routines are entered from the unit routine.

```

farads        lda      #$46      ;The ASCII value of "F".
              bra      storeunit
volts         lda      #$56      ;The ASCII value of "V".
              bra      storeunit
seconds       lda      #$73      ;The ASCII value of "s".
              bra      storeunit
ohms          lda      #$6f      ;The ASCII value of "o".
              jsr      outbyte
              lda      #$68      ;The ASCII value of "h".
              jsr      outbyte
              lda      #$6d      ;The ASCII value of "m".
              jsr      outbyte
              bra      seconds      ;Pluralize "ohm" using the "s" from the seconds routine to
              ; save ROM space.
amps          lda      #$41      ;The ASCII value of "A".
              bra      storeunit
percent       lda      #$25      ;The ASCII value of "%".
              bra      storeunit
hertz        lda      #$48      ;The ASCII value of "H".
              jsr      outbyte
              lda      #$7a      ;The ASCII value of "z".
storeunit     jsr      outbyte

```

* The comments routine detects the "AC" and "DC" annunciators.

```

comments      brset   0,firstbyte+$01,alternate ;Check S0 for the alternating current annunciator.
              brset   1,firstbyte+$01,direct   ;Check S0 for the direct current annunciator.
              bra      sendstring              ;Jump to the serial output routine if neither of these
exist.

```

```

alternate      lda      #$41          ;The ASCII value of "A".
               jsr      outbyte      ;Store the "A" at the current output string location.
               bra      current      ;Jump to the routine to store a "C".
direct         lda      #$44          ;The ASCII value of "D".
               jsr      outbyte      ;Store the "D" at the current output string location.
current        lda      #$43          ;The ASCII value of "C".
               jsr      outbyte      ;Store the "D" at the current output string location.

* The sendstring routine terminates the output string with a <CR> and sends the output string, in serial format at
* 9600bps 8N1, to porta bit 0 (where the MAX 232A serial driver is connected).

sendstring     lda      #$0d          ;The ASCII value of a <CR>.
               sta      string,x     ;Store the <CR> at the current output string location. The
<CR>
string         ; provides a carriage return and serves as the output
               ; terminator.

string.        lda      #$18          ;Load the offset of the first character in the output
               sta      bytecount    ;Store it in bytecount.

serstart       ldx      #$08          ;The number of bits in a byte.
               stx      bitcount     ;Store it in bitcount.

startbit       clr      porta        ;Send a start bit.
               ldx      #$1e          ;Load X as a countdown register with the serial bit time.
startdly       decx     startdly     ;Loop until the serial bit time has expired.

               ldx      bytecount    ;Load X with the current output string offset.
               lda      string,x     ;Load A with the current output string character.

sendbit        sta      porta        ;The bit of interest is stored at porta bit 0.
bitdelay       ldx      #$1e          ;Load X as a countdown register with the serial bit time.
               bne      bitdelay     ;Loop until the serial bit time has expired.
               rora                ;Rotate the data byte to get the next bit of interest into
               ; the LSB of A.
               dec      bitcount     ;Decrement bitcount.
               bne      sendbit      ;Repeat the bit shifting if eight data bits have not yet
been sent.

stopbit        bset     0,porta      ;Send a stop bit.

bytedelay      decx     bytedelay     ;Delay before next character. (X is $00 on entry)
               bne      bytedelay   ;Loop until the interbyte delay has expired.
               ldx      bytecount    ;Load X with the current output string offset.
               lda      string,x     ;Load the current output string character into A.
               cmp      #$0d         ;See if it is the <CR> termination character.
               beq      end          ;If it is the termination character, exit the sendserial
routine.       dec      bytecount    ;Update the current output string offset.
               bra      serstart     ;Send the next byte in the output string.

end            jmp      reset        ;Go to the beginning of the program and start the entire
               ; process again.

```

```

*****
* Fluke 87 Data Acquisition sourcecode *
* *
* Author: Derek Matsunaga *
* Revision: 2.2 *
* Revision date: October 2, 1993 *
* Processor: 68HC705K1 *
* *
* @1993 Derek Matsunaga *
* *
* Revision 2.2 corrects serial timing. *
*****

```

```
* Set-up 68HC05K1 equate table.
```

```

porta      equ      $0000      ;Port a address.
portb      equ      $0001      ;Port b address.
ddra       equ      $0004      ;Data direction register a.
ddrb       equ      $0005      ;Data direction register b.

```

```
* Define RAM locations.
```

```

temp1      org      $00e0      ;The beginning of RAM.
           equ      $00e0      ;A general purpose temporary register.
lcount     equ      $00e1      ;The register which keeps track of low time.
lowS       equ      $00e1      ;The low "S" register from the meter.
hcount     equ      $00e2      ;The register which keeps track of high time.
highS      equ      $00e2      ;The high "S" register from the meter.
scount     equ      $00e3      ;The "S" register counter.
bitcount   equ      $00e3      ;The number of bits per byte received.
bytecount  equ      $00e4      ;The number of 8 bit bytes received (#18 max).
firstbyte  equ      $00e5      ;The location of the first byte in the record.
string     equ      $00e5      ;The start of the output string.
checksum   equ      $00f8      ;The checksum register.

```

```
* Define vector table.
```

```

           org      vectors
           fdb      rom
           fdb      rom
           fdb      rom
           fdb      reset

```

```
* Initialize ports and registers.
```

```

reset      org      $0200      ;The beginning of program ROM.
           lda      #$ff
           sta      ddra      ;Configure all porta pins as outputs.
           sta      porta     ;Write all ones to porta.
           clr      ddrb      ;Configure both portb pins as inputs.
           clr      bytecount  ;Initialize counting registers.
           clr      bitcount
           clr      checksum   ;Clear the checksum register.

```

```

* Since each nibble begins with a "1" as a start bit, the deadtime during
* valid data transmission will not exceed the 7.8ms nibble time
* (1.3ms/bit * 6 bits, see explanation diagram). So, a complete record of
* data has just been sent if deadtime exceeds the 7.8ms nibble time. Use
* about 11ms of dead time to prevent false record triggering (for noise immunity).

```

```

* Look for about 11ms of deadtime before reading data. Due to the length of the
* deadtime, a 16 bit counter is needed. The "temp1" location is used to store the
* "high" byte (upper 8 bits) of the 16 bit counter. The X register is used to store the "low"
* byte. See sourcecode validation for detailed timing information.

```

```

start      clr      X          ;Clear the X register.
           stx      temp1     ;Clear the "high" byte of X.

loop       brset   0,portb,start ;Loop & clear X until portb0 goes low.

           decx      X        ;Decrement the X register.
           bne      loop     ;If X>0, go to top of loop.

           inc      temp1    ;Directly increment the high byte of the counter.
           lda      temp1    ;Load the new high byte into the accumulator.
           cmp      #$08     ;Temp1 will be 8 when about 11.3ms of
           ; deadtime has occurred.
           bne      loop

```

```

* About 11.3ms of deadtime has been detected at portb0. We are now ready
* to start reading the data from the meter. The beginning of the actual
* data occurs with a low to high transition following the deadtime.

```

```

* Wait until deadtime is over (a high is detected at portb0).
* Each nibble is started with a 1.3ms high (a startbit).

```

```
getstart   brclr   0,portb,getstart ;Read portb0 until it goes high.
```

```
* When the high at portb0 is detected, initialize the timer locations.
```

```

nextbit    ldx      #$91      ;Initialize the X register with the bit time.
           clr      lcount    ;Initialize the low count register.
           clr      hcount    ;Initialize the high count register.

timer      decx      X        ;Decrement X register to keep track of time.
           beq      timeout   ;Check portb until X=0 (the timer has expired).

```

```

chkport      brset      0,portb,high      ;Check for a high at portb0.
low          inc        lcount            ;Increment the low count location.
            bra        timer            ;Go back to timer.
high         inc        hcount            ;Increment the high count location.
            bra        timer            ;Go back to timer.
timeout      lda        bitcount          ;Load the bit counter.
            cmp        #$00             ;Determine if it is the start bit of the 1st nibble.
            beq        housekeep        ;Ignore this bit if it is the start bit.
            cmp        #$05             ;Determine if it is the start bit of the 2nd nibble.
            beq        housekeep        ;Ignore this bit if it is the start bit.

            lda        lcount            ;Load the number of low counts.
            sub        hcount            ;Subtract the number of high counts.
            ;C will be set if there were more highs than lows.
            ldx        bytecount         ;Load the record pointer into X.
            rol        firstbyte,x       ;Roll the carry bit onto the current record byte.

housekeep    inc        bitcount          ;Increment the number of bits read.
            lda        bitcount          ;Load the incremented value into A.
            cmp        #$05             ;If bitcount is #5, get ready for next nibble.
            beq        add_chksum        ;If bitcount is #10, get ready for next byte.
            cmp        #$0a             ;If bitcount is #10, get ready for next byte.
            bne        nextbit

add_chksum   lda        firstbyte,x       ;Load the incremented value into A.
            and        #$0f             ;Mask off the high nibble.
            add        checksum          ;Add the received nibble to the checksum.
            sta        checksum          ;Store the checksum.

            lda        bitcount          ;Load the incremented value into A.
            cmp        #$05             ;If bitcount is #5, get ready for next nibble.
            beq        getstart

nextbyte     inc        bytecount         ;Increment the number of bytes read.
            lda        bytecount         ;Load the incremented number into A.
            cmp        #$12             ;Determine if #18 bytes have been read.
            beq        cs_test          ;Start processing data if #18 bytes have been read.
            clr        bitcount         ;Reset bitcount to zero.
            bra        getstart          ;Wait for the next start bit.

* The integrity of the received data is determined by cs_test. The received data is considered
* good if the received checksum correlates with the calculated checksum.
* The checksum location contains the least significant nibble of the sum of all all nibbles received,
* including the meter's checksum nibble. Because of this, the meter's checksum nibble must be
* subtracted from the calculated checksum prior to comparison.
* The meter's checksum is 1 less than the calculated checksum. (i.e. a calculated checksum
* of $0f corresponds to a received checksum of $0e).

cs_test      lda        firstbyte+$11     ;Load S31 and the meter's checksum into A.
            and        #$0f             ;Remove S31.
            sta        templ            ;Store the meter's checksum in templ.

            lda        checksum          ;Load the calculated checksum into A.
            sub        templ            ;Subtract the meter's checksum nibble.
            dec        #1                ;Subtract 1.
            and        #$0f             ;Knock-off the high nibble.

            ldx        #$18             ;Load the output string offset into X.
            stx        bytecount        ;Bytecount now points to the output string character.

checksum.    cmp        templ            ;See if the calculated checksum matches the meter's
            beq        ol_test          ;If so, procede with decoding.
            ;If the checksums do not match, send an exclamation point

to           ; indicate a checksum error.
            lda        #$21             ;Load the ASCII value for an exclamation point (!).
            sta        string,x         ;Store it as the first character in the output string.
            decx        #1                ;Decrement the string pointer.
            jmp        sendstring        ;Send the exclamation point and restart the program.

* The following code segment, ol_test, determines if the overload message is on the
* display. To accomplish this, the tens digit is checked for an "L". If the "L" exists,
* then it is not necessary to process anything else. If not, then proceed with the
* numeric decoding. Note: the bottom leg of the "L" character is unique in S9 so it is used
* to check for the entire "L".

ol_test      lda        firstbyte+$0a     ;Load S9.
            and        #$0f             ;Knock-off the high nibble.
            cmp        #$04             ;Check for the bottom leg of the "L" character.
            bne        sign             ;Go to the sign routine if the "L" does not exist.
            jmp        overload          ;Go to the overload routine if the "L" exists.

* No overload has been detected so now we can decode the display digits.

* The sign routine checks the acquired data for a minus sign. If the minus sign is present, the routine
* stores the ASCII value of a minus sign as the first byte of the output string. If not, the program proceeds
* to decode the ten-thousands digit.

sign         brclr      6,firstbyte+$02,ten_k ;See if the sign bit is clear.
            lda        #$2d             ;Load the ASCII value for a minus sign.
            jsr        outbyte          ;Store a minus sign as the first byte in the output.

```

* The only digit that can appear in the ten-thousands place is a "1". The ten_k routine checks for the presence of a "1". Note that this digit will only be filled when the meter is in four digit mode. If a "1" exists, it is stored as the next character in the output string. If not, the program proceeds to decode the remaining four digits.

```
ten_k      brclr      4,firstbyte+$02,start_thou    ;See if there is a 1 in the ten-thousands place.
          lda        #$31                          ;Load the ASCII value for a "1".
          jsr        outbyte                        ;Store the "1" in the output string.
```

* Start_thou is the beginning of the decoding routine for the remaining four display digits.

```
start_thou stx        bytecount                    ;Store the output string pointer.
          ldx        #$0f                          ;Load X with the offset of the first S register.
          stx        scount                        ;Store it in scount.

thousands ldx        scount                        ;Load the current scount value into X.
          cpx        #$07                          ;See if it is the last S register.
          bne        continue                      ;If it is the last S register, decode the magnitude.
          bra        magnitude

continue   lda        firstbyte,x                 ;Load S14 the 1st time, S12, S10, S8.
          sta        highS                         ;Store it in highS.
          decx       scount                        ;Decrement scount.
          lda        firstbyte,x                 ;Load S13 the 1st time, S11, S9, S7.
          sta        lowS                          ;Store it in lowS.
          decx       scount                        ;Decrement scount.
          stx        scount                        ;Store scount.
          ldx        bytecount                    ;Load the current output string offset into X.

display   lda        highS                        ;Load the high S register.
          asla       asla                          ;Move the 4 LSBs to the MSBs.
          asla       asla
          asla       asla
          sta        templ                         ;Store it in templ.
          lda        lowS                          ;Load the low S register.
          and        #$0f                          ;Knock-off the 4MSBs.
          ora        templ                         ;Now we have a byte which represents the digit.
          sta        templ                         ;Store this byte in templ.
```

* The dptest routine checks the digit to see if the decimal point is lit. If it is, dptest stores the ASCII value of a decimal point at the current output string location. If not, it proceeds to decode the numerals.

```
dptest    brclr      6,templ,numerals             ;Test for a decimal point.
          lda        #$2e                          ;The ASCII value for a period (decimal point).
          jsr        outbyte                        ;Store it in the current output string location.

numerals  lda        templ                         ;Load the digit information into A.
          and        #$bf                          ;Get rid of the decimal point bit.
          stx        bytecount                    ;Store the output string offset because the X register
          ; is needed for numeric decoding.

compare   clr       clrx                          ;Test for a "0".
          cmp        #$af                          ;If a "0" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a zero is not present.
          incx       X                             ;Test for a "1".
          cmp        #$09                          ;If a "1" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a one is not present.
          incx       X                             ;Test for a "2".
          cmp        #$97                          ;If a "2" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a two is not present.
          incx       X                             ;Test for a "3".
          cmp        #$1f                          ;If a "3" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a three is not present.
          incx       X                             ;Test for a "4".
          cmp        #$39                          ;If a "4" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a four is not present.
          incx       X                             ;Test for a "5".
          cmp        #$3e                          ;If a "5" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a five is not present.
          incx       X                             ;Test for a "6".
          cmp        #$be                          ;If a "6" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a six is not present.
          incx       X                             ;Test for a "7".
          cmp        #$0b                          ;If a "7" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if a seven is not present.
          incx       X                             ;Test for an "8".
          cmp        #$bf                          ;If an "8" is present, go to the storechar routine.
          beq        storechar                    ;Increment X if an eight is not present.
          incx       X                             ;Test for a "9".
          cmp        #$3f                          ;If an "9" is present, go to the storechar routine.
          bne        thousands                    ;All numeric possibilities have been tested for. If none
of them   ; exist, proceed to decode the next digit.

storechar txa                                     ;Transfer the X register to A. The X register contains
          add        #$30                          ;A value indicative of the numeral on the display.
          ldx        bytecount                    ;Add X to the ASCII value for a "0".
          sta        string,x                      ;Load X with a current output string offset.
          ;Store the ASCII value of the displayed numeral at the
current   ; output string position.
```



```

        dec          bytecount          ;Decrement the output string pointer.
        bra          thousands          ;Repeat the numeric decoding process for the remaining
digits.

* The overload routine stores the ASCII values for "OL" at the current output string location.
overload      lda          #$4f          ;The ASCII value of an "O"
              jsr          outbyte      ;Store it in the current output string location.
              lda          #$4c          ;The ASCII value of an "L"
              jsr          outbyte      ;Store it in the current output string location.
              lda          #$20          ;The ASCII value of a space
              jsr          outbyte      ;Store it in the current output string location.
              stx          bytecount     ;Store the current output string offset.

* The magnitude routine checks the appropriate S registers for magnitude indicators such as mega, kilo, milli, etc.
* Each magnitude designator has its own ASCII storage routine similar to that of the overload routine.
magnitude     ldx          bytecount
              brset       3,firstbyte+$05,micro ;Check S4 for a "micro" sign.
              brset       2,firstbyte+$05,nano  ;Check S4 for a "nano" sign.
              brset       1,firstbyte+$05,mega  ;Check S4 for a "mega" sign.
              brset       3,firstbyte+$04,milli ;Check S3 for a "milli" sign.
              brset       1,firstbyte+$04,kilo  ;Check S3 for a "kilo" sign
              bra          units          ;Go to the units routine if no magnitude designators
              ; are detected.

* The outbyte subroutine stores the contents of the accumulator at the current output string location.
* On entry, the X register must contain the offset relative to the string location (see equate table).
* The X register is decremented to prepare for the next output string byte.
outbyte       sta          string,x      ;Store the accumulator at string+x.
              decx          ;Decrement X.
              rts          ;Return to caller.

* The following routines store the ASCII values of the magnitude designators. These routines are entered
* from the magnitude routine.
micro         lda          #$75          ;Load the ASCII values for a "u".
              bra          storemag      ;Go to the magnitude storing routine.
nano          lda          #$6e          ;Load the ASCII values for an "n".
              bra          storemag      ;Go to the magnitude storing routine.
mega         lda          #$4d          ;Load the ASCII values for an "M".
              bra          storemag      ;Go to the magnitude storing routine.
milli        lda          #$6d          ;Load the ASCII values for an "m".
              bra          storemag      ;Go to the magnitude storing routine.
kilo         lda          #$4b          ;Load the ASCII values for a "K".

storemag      jsr          outbyte      ;Go to the ASCII byte storing subroutine.

* The units routine checks the appropriate S registers for unit indicators such as farads, volts, ohms, etc.
* Each unit designator has its own ASCII storage routine similar to that of the overload routine.
units         brset       2,firstbyte+$04,farads ;Check S3 for a farads indicator.
              brset       3,firstbyte+$03,volts ;Check S2 for a volts indicator.
              brset       2,firstbyte+$03,seconds ;Check S2 for a seconds indicator.
              brset       1,firstbyte+$03,ohms  ;Check S2 for a ohms indicator.
              brset       3,firstbyte+$02,amps  ;Check S1 for a amps indicator.
              brset       2,firstbyte+$02,percent ;Check S1 for a percent indicator.
              brset       1,firstbyte+$02,hertz ;Check S1 for a hertz indicator.
              bra          comments      ;Jump to the comments routine if no units indicators were
              ; detected.

* The following routines store the ASCII values of the units designators. These routines are entered
* from the unit routine.
farads        lda          #$46          ;The ASCII value of "F".
              bra          storeunit
volts         lda          #$56          ;The ASCII value of "V".
              bra          storeunit
seconds       lda          #$73          ;The ASCII value of "s".
              bra          storeunit
ohms          lda          #$6f          ;The ASCII value of "o".
              jsr          outbyte
              lda          #$68          ;The ASCII value of "h".
              jsr          outbyte
              lda          #$6d          ;The ASCII value of "m".
              jsr          outbyte
              bra          seconds      ;Pluralize "ohm" using the "s" from the seconds routine to
              ; save ROM space.
amps          lda          #$41          ;The ASCII value of "A".
              bra          storeunit
percent       lda          #$25          ;The ASCII value of "%".
              bra          storeunit
hertz        lda          #$48          ;The ASCII value of "H".
              jsr          outbyte
              lda          #$7a          ;The ASCII value of "z".

storeunit     jsr          outbyte

* The comments routine detects the "AC" and "DC" annunciators.
comments      brset       3,firstbyte+$01,alternate ;Check S0 for the alternating current annunciator.
              brset       2,firstbyte+$01,direct  ;Check S0 for the direct current annunciator.
              bra          sendstring    ;Jump to the serial output routine if neither of these
exist.

```

```

alternate      lda      $$41                ;The ASCII value of "A".
               jsr      outbyte       ;Store the "A" at the current output string location.
               bra      current       ;Jump to the routine to store a "C".
direct         lda      $$44                ;The ASCII value of "D".
               jsr      outbyte       ;Store the "D" at the current output string location.
current        lda      $$43                ;The ASCII value of "C".
               jsr      outbyte       ;Store the "D" at the current output string location.

* The sendstring routine terminates the output string with a <CR> and sends the output string, in serial format at
* 9600bps 8N1, to porta bit 0 (where the MAX 232A serial driver is connected).

sendstring     lda      $$0d                ;The ASCII value of a <CR>.
               sta      string,x         ;Store the <CR> at the current output string location. The
<CR>
string         ; provides a carriage return and serves as the output
               ; terminator.

string.        lda      $$18                ;Load the offset of the first character in the output
               sta      bytecount       ;Store it in bytecount.

serstart       ldx      $$08                ;The number of bits in a byte.
               stx      bitcount        ;Store it in bitcount.

startbit       clr      porta             ;Send a start bit.
               ldx      $$20            ;Load X as a countdown register with the serial bit time.
startdly       decx
               bne      startdly        ;Loop until the serial bit time has expired.

               ldx      bytecount       ;Load X with the current output string offset.
               lda      string,x        ;Load A with the current output string character.

sendbit        sta      porta             ;The bit of interest is stored at porta bit 0.
               ldx      $$20            ;Load X as a countdown register with the serial bit time.
bitdelay       bne      bitdelay        ;Loop until the serial bit time has expired.
               rora                    ;Rotate the data byte to get the next bit of interest into
               ; the LSB of A.
               dec      bitcount        ;Decrement bitcount.
               bne      sendbit        ;Repeat the bit shifting if eight data bits have not yet
been sent.

stopbit        bset     0,porta           ;Send a stop bit.

bytedelay      decx
               bne      bytedelay      ;Delay before next character. (X is $00 on entry)
               ldx      bytecount       ;Loop until the interbyte delay has expired.
               lda      string,x        ;Load X with the current output string offset.
               cmp      $$0d            ;Load the current output string character into A.
               beq      end             ;See if it is the <CR> termination character.
               ;If it is the termination character, exit the sendserial

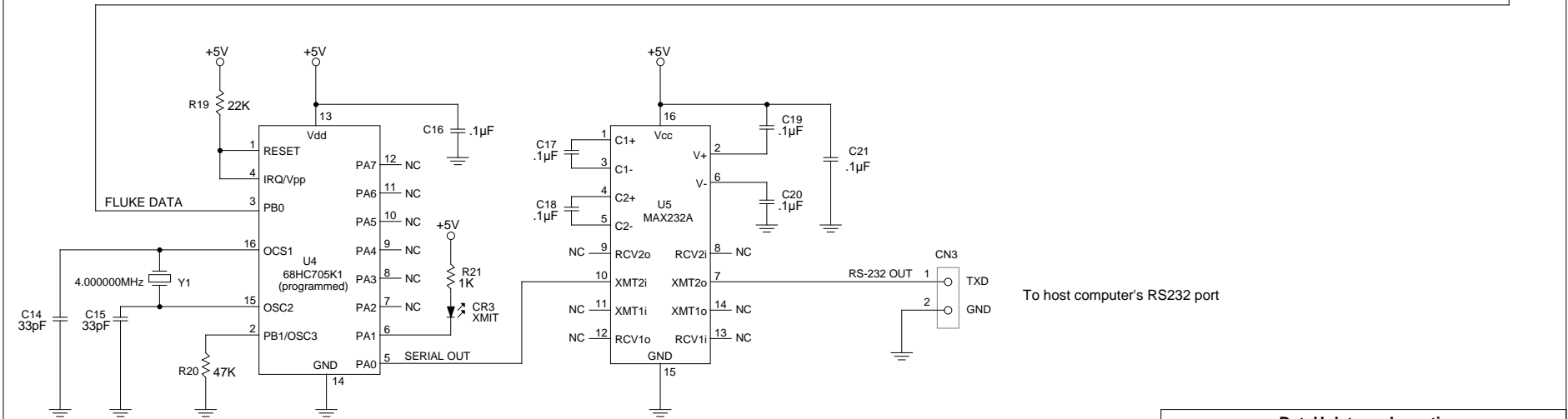
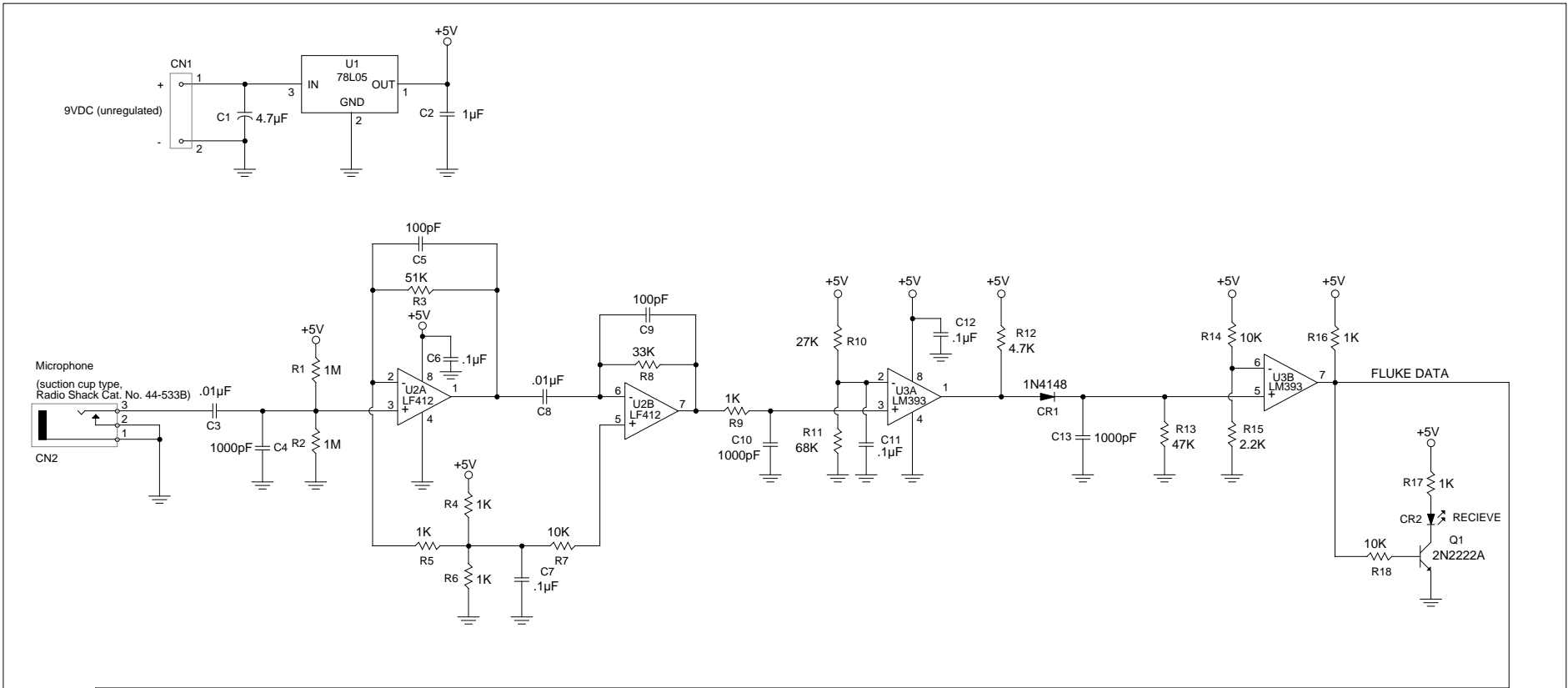
routine.       dec      bytecount       ;Update the current output string offset.
               bra      serstart       ;Send the next byte in the output string.

end            jmp      reset            ;Go to the beginning of the program and start the entire
               ; process again.

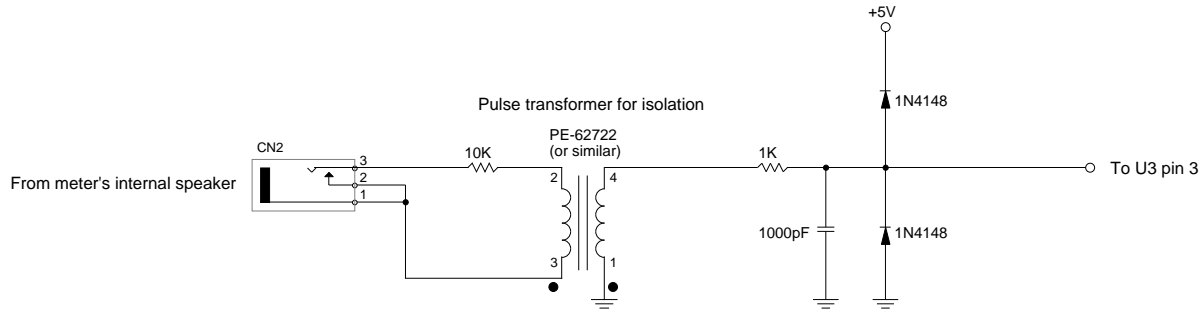
```

Appendix D

Schematics and parts list



Alternate construction for hardwire interface. Eliminate amplifier front-end from page 1 and replace it with the circuit below. R10 and R11 may need to be adjusted depending on the turns ratio of the transformer.



Component list

Designator	Component	Type	Value	Designator	Component	Type	Value
C1	capacitor	tantalum	4.7 μ F	R1	resistor	1/4 watt, 5%	1M
C2	capacitor	ceramic	1 μ F	R2	resistor	1/4 watt, 5%	1M
C3	capacitor	ceramic	.01 μ F	R3	resistor	1/4 watt, 5%	51k
C4	capacitor	ceramic	1000pF	R4	resistor	1/4 watt, 5%	1k
C5	capacitor	ceramic	100pF	R5	resistor	1/4 watt, 5%	1k
C6	capacitor	ceramic	.1 μ F	R6	resistor	1/4 watt, 5%	1k
C7	capacitor	ceramic	.1 μ F	R7	resistor	1/4 watt, 5%	10k
C8	capacitor	ceramic	.01 μ F	R8	resistor	1/4 watt, 5%	33k
C9	capacitor	ceramic	100pF	R9	resistor	1/4 watt, 5%	1k
C10	capacitor	ceramic	1000pF	R10	resistor	1/4 watt, 5%	27k
C11	capacitor	ceramic	.1 μ F	R11	resistor	1/4 watt, 5%	68k
C12	capacitor	ceramic	.1 μ F	R12	resistor	1/4 watt, 5%	4.7k
C13	capacitor	ceramic	1000pF	R13	resistor	1/4 watt, 5%	47k
C14	capacitor	ceramic	33pF	R14	resistor	1/4 watt, 5%	10k
C15	capacitor	ceramic	33pF	R15	resistor	1/4 watt, 5%	2.2k
C16	capacitor	ceramic	.1 μ F	R16	resistor	1/4 watt, 5%	1k
C17	capacitor	ceramic	.1 μ F	R17	resistor	1/4 watt, 5%	1k
C18	capacitor	ceramic	.1 μ F	R18	resistor	1/4 watt, 5%	10k
C19	capacitor	ceramic	.1 μ F	R19	resistor	1/4 watt, 5%	22k
C20	capacitor	ceramic	.1 μ F	R20	resistor	1/4 watt, 5%	47k
C21	capacitor	ceramic	.1 μ F	R21	resistor	1/4 watt, 5%	1k
CN1	connector	DC adapter	2 pin	U1	voltage regulator	100mA	78L05
CN2	connector	1/8" phone	2 pin	U2	op amp	dual JFET	LF412
CN3	connector	3/32" phone	2 pin	U3	comparator	dual open collector	LM393
CR1	diode	general purpose	1N4148	U4	microcontroller	16 pin, programmed	68HC705K1
CR2	LED	T1 3/4	red	U5	RS-232 driver	16 pin, 5V only	MAX232A
CR3	LED	T1 3/4	red	Y1	crystal		4.000000MHz
Q1	transistor	NPN	2N2222A				

DataHolster schematic

Rev.: 1.5

Page 2 of 2

©1993 Derek Matsunaga