

THE PRESSURE SENSITIVE TOUCH-PAD

James P. Williams
Gregory F. Welch

April 30, 1985

E.E.T. 454
Prof. Tom Shultz

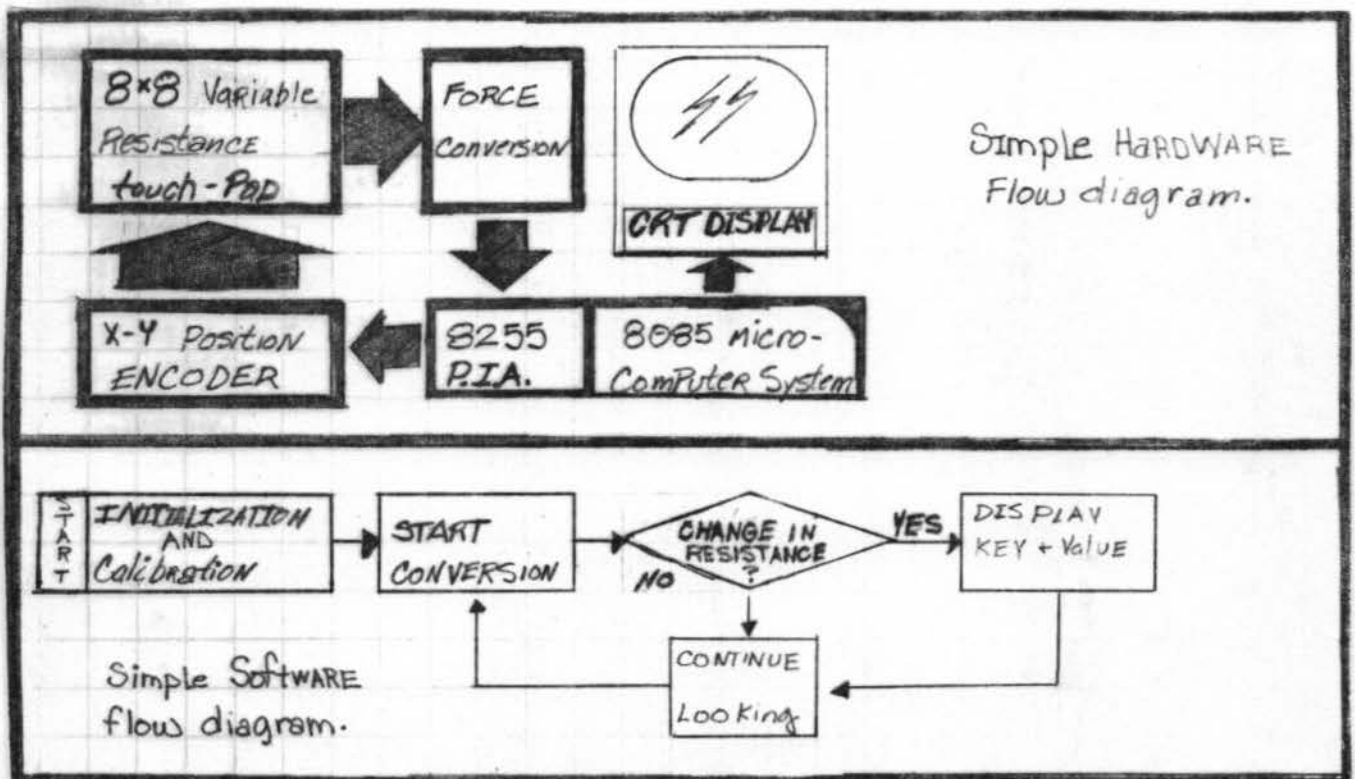
System Plan:

This project, which was designed by James P. Williams and Gregory F. Welch, will use an original design for a pressure sensitive touch-pad used to allow input to a micro-processor system.

The touch-pad will use a matrix of conductive foam squares which behave as variable resistors. Using some special scan circuitry, the software will continue to scan the touch-pad for a large change in resistance from one scan to another.

The problem faced is that of designing the hardware and the software required to scan the pressure sensitive touch-pad, and to determine (by software) whether or not a specific area of the touch-pad was being pressed.

The touch-pad should be such that a small child could use it to control a small vehicle such as a wheelchair. The positional data could represent the direction in which to move, and eventually, the amount of pressure could determine the speed at which the wheelchair moves.



Introduction:

The pressure sensitive touch-pad system was designed to eliminate the need for a positive (definite) depression of a specific key to perform a task. However, the touch-pad is still set-up in a matrix fashion, so that the area of the most pressure could be determined. The purpose of this is to facilitate control of a small vehicle by children with underdeveloped fine motor skills. Such children might have problems with pressing one specific key among many, and they might not be able to apply direct force on a key.

With this current system, the user would apply pressure to a certain area of the touch-pad, and the system would recognize the key (or area), and display a number on the video display terminal which corresponds to that area.

Software:

There are several special features in the touch-pad software which require further explanation.

The first area of interest is a typed subroutine called Value. The purpose of this subroutine is to allow the software to determine the amount of resistance (relatively) that is characteristic of a certain key or area. It begins by placing the count of the key OR'd with data to initialize the hardware on an output port. This selects the key to be read, and places a high on both the address latch enable (ALE) and the start conversion (SC). The values are OR'd together because the control is all processed through the same port.

Next, the routine places the count value on the output port twice again. Both times, the count is OR'd with the data required to control the hardware. The first time, a low is placed on the ALE line, and then a low on the SC line. This begins the process of conversion.

Finally, the routine loops, checking for an end of conversion (EOC) signal. When the data is ready, it is returned through the typed subroutine.

The next area of software interest is the main section of code. The main section of code is comprised of two loops, one inside the other. The outer loop repeats until either a break is encountered, or some other means such as a reset is used to halt the microprocessor. It begins by zeroing out (so to speak) several variables to be used. It then scans the keyboard and looks for the largest difference in resistance between the present and the last scan. Then, if this difference is greater than a threshold value, the number corresponding to the area pressed is displayed on the vdt.

The inner loop repeats sixty-four times, every time it is encountered. It begins by calling the Value subroutine which returns a value between zero and 128, representing the current value of resistance for that key. It then determines if the current scan value is less than the permanent value, then the permanent value is updated to the current value. Next, it determines the difference between the permanent value and the current value. Throughout the looping, it checks to see if that difference is the greatest found yet. If so, it stores the key number and the value, otherwise, it continues on to the next key.

Hardware:

The hardware consists mainly of two parts; the touch-pad, and the scan circuitry.

The touch-pad consists of 64 conductive foam squares which are sandwiched between eight conductive strips positioned in one direction below, and eight others in a perpendicular direction above. This forms an eight by eight matrix which allows which allows the scan circuitry to determine the current resistance of any square of foam.

The scan circuitry uses an eight channel digital demultiplexer, and an eight channel analog multiplexer. The software delivers a six bit word to this circuitry, three of which are used to place a high on one of eight lines using the demultiplexer, the other three which are used to select one of the eight channels of the analog multiplexer A/D

converter. This chip then converts the analog signal on the selected line to an eight bit digital word which is then made available to the microprocessor. This, although it is an actual voltage reading, is used to represent the current resistance of the selected square. The larger the number, the greater the force applied to the square, the less the resistance.

Analysis:

There were two problems encountered during the development of the touch-pad which were considered to be major.

The first problem was that of selectivity. Originally, the touch-pad consisted of one large sheet of conductive foam with a matrix of wires above and below it. The problem was that pressure in one spot might appear as pressure in several spots. This problem was solved almost completely by separating the sheet into 64 separate squares of foam which did not come in contact with each other. Then to further eliminate the problem, used flexible copper strips instead of the stiff wire originally used.

The second problem was that of determining with software, what actually represented a key being pressed. The first attempt kept the values read during each scan in an array which was updated with each scan. The problem was that because the array was being updated so frequently, there was never really time to determine if the change was due to an actual depression of the pad, or some other outside event. This problem was solved by updating the array only once at the beginning of the program. Later scans are compared to this scan only, and the array is only updated if the movement (resistance change) seems to be opposite to that of a depression, in other words, settling of the foam.

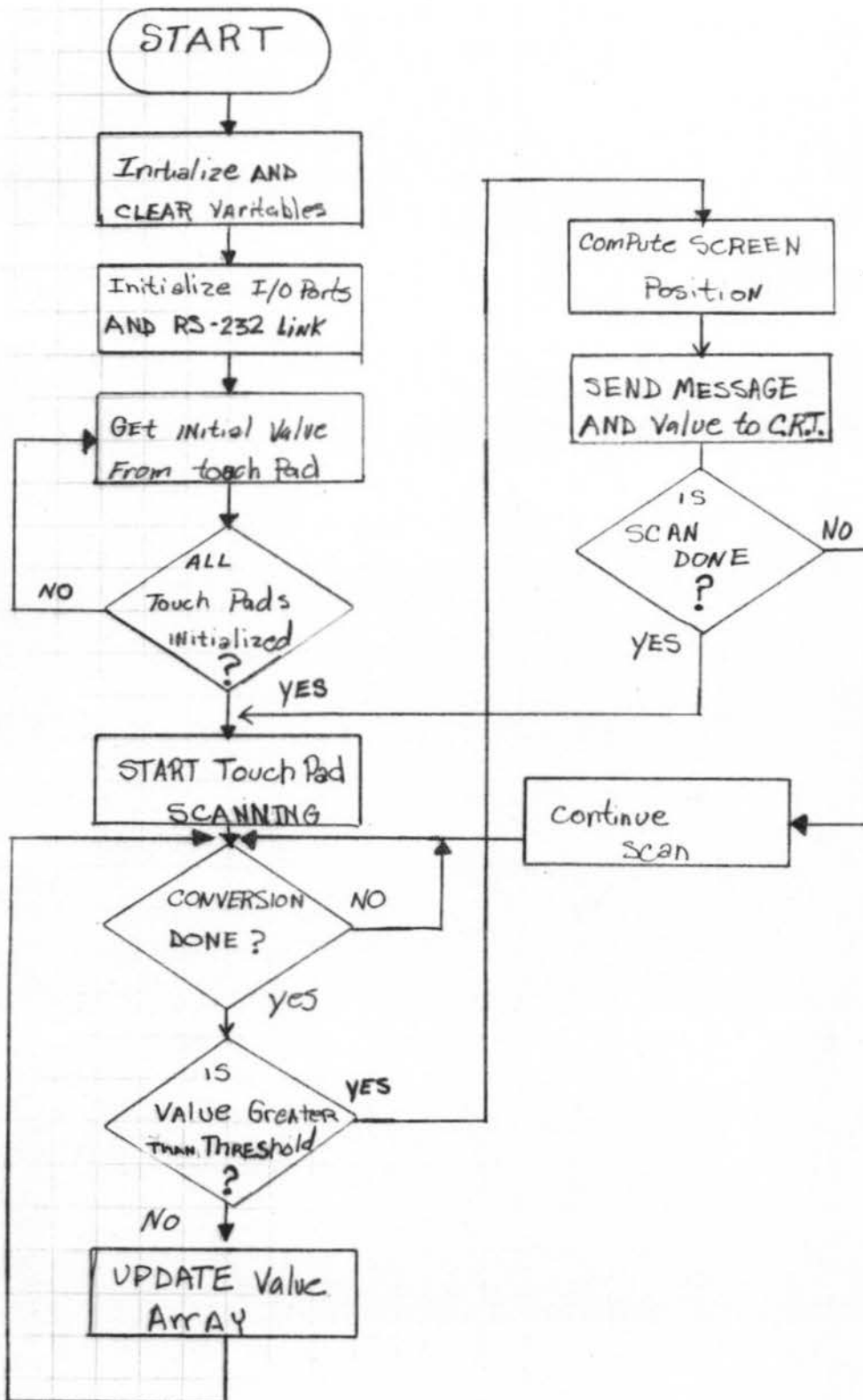
In the future, one might separate the conductive foam further with strips of insulating foam. This would tend to physically support the keys surrounding the one being pressed, but still still allow the flexibility of the original design.

Also, another nice improvement to be made in the future would be to allow the amount of pressure applied to control a device such as a motor. This would not be a very tough project enhancement as far as the current software is concerned. The byte value returned from the Value subroutine could be used directly to regulate the pulse width of a signal sent to a motor, thus varying the speed.

James P. Williams

Gregory F. Welch

Software flow of Program KYBD.P80



SIS-11 PL/M-80 V3.1 COMPILATION OF MODULE KYBD
 OBJECT MODULE PLACED IN :F9:KYBD.OBJ
 COMPILER INVOKED BY: :F1:PLM80 :F9:KYBD.P80 WORKFILES(:F0:,:F0:) DEBUG PAGEWIDTH
 -H(80) TITLE('F9:KYBD COMPILATION') DATE(04/30/85)

```

/* THIS PROGRAM WILL PROCESS THE I/O FROM A
  KEYBOARD WHICH USES CONDUCTIVE FOAM TO
  VARY THE RESISTANCE BETWEEN +V AND THE
  LINE GOING TO THE KEY. THE KEYBOARD
  CONSISTS OF 64 KEYS (8X8 MATRIX) WHICH
  ARE MULTIPLEXED AND CONTROLLED BY THE
  PROGRAM. THE PROGRAM WILL OUTPUT AN
  ADDRESS TO THE KEYBOARD WHICH WILL SELECT
  A CERTAIN KEY. THEN IT WILL (USING AN
  A/D CONVERTER) INPUT THE VOLTAGE LEVEL
  OF THAT KEY AND COMPARE IT TO THE LAST
  VALUE FOR THAT KEY (OBTAINED EARLIER). */

```

```

1      KYBD:DO; /* BEGIN KYBD PROGRAM */

2      1      DECLARE /* VARIABLE DECLARATION SECTION */
          (COUNT,MAX$VALUE,DIFFERENCE,KEY$PUSHED,
           THRESHOLD,VOLTAGE) BYTE,
          LAST$VALUE (64) BYTE,
          EOM LITERALLY '03H',
          COMMAND LITERALLY 'OUTPUT(83H)',
          KEY$SELECT LITERALLY 'OUTPUT(81H)',
          KEY$VALUE LITERALLY 'INPUT(80H)',
          DATA$READY LITERALLY '(INPUT(80H) AND 10000000B)=0',
          CLEARSCREEN(*) BYTE DATA(1BH,1CH,EOM),
          BLANKS(*) BYTE DATA(' ',EOM),
          FOREVER LITERALLY 'WHILE 1';

3      1      VALUE:PROCEDURE BYTE;

          /* THIS ROUTINE RETURNS THE RELATIVE
            VOLTAGE VALUE READ FROM THE VOLTAGE DIVIDER
            FORMED BY A PERMANENT RESISTOR AND THE
            VARIABLE RESISTOR (CONDUCTIVE FOAM) */

4      2      KEY$SELECT=(COUNT OR 11000000B);
          /* LOAD ADDRESS AND SETS ALE AND SC HIGH */
5      2      KEY$SELECT=(COUNT OR 01000000B);
          /* ALE STROBED LOW: LATCH ADDRESS */
6      2      KEY$SELECT=COUNT;
          /* SC STROBED: START CONVERSION */
7      2      DO WHILE NOT(DATA$READY);
8      3          END;
9      2      RETURN (KEY$VALUE AND 01111111B);
          /* MASK RESULT AND RETURN TO CALL LOCATION */
10     2      END VALUE;

```

```
11 1      CHOUT:PROCEDURE (CH$SEND); /*SENDS ONE CHARACTER TO CRT SCREEN*/
12 2      DECLARE CH$SEND BYTE; /*ADDRESS AND MASKING SPECIFIC TO S100*/
13 2      DO WHILE (INPUT(0) AND 00000001B)=0; /*AWAIT TXRDY*/
14 3      END;
15 2      OUTPUT(1)=CH$SEND;
16 2      END CHOUT;

17 1      DISPLAY:PROCEDURE(NUM); /* DISPLAYS A BYTE VALUE ON THE CRT */
18 2      DECLARE (NUM,J) BYTE; J=100;
20 2      DO WHILE J>0;
21 3          CALL CHOUT(NUM/J+'0'); /* DISPLAY AN ASCII CHARACTER */
22 3          NUM=NUM MOD J;
23 3          J=J/10;
24 3      END;
25 2      END DISPLAY;

26 1      MESSAGE:PROCEDURE(STARTADDR); /*SENDS STRING TO CRT-UP TO EOM*/
27 2      DECLARE STARTADDR ADDRESS, (CH$MESS BASED STARTADDR) BYTE;
28 2      DO WHILE CH$MESS<>EOM;
29 3          CALL CHOUT(CH$MESS);
30 3          STARTADDR=STARTADDR+1;
31 3      END;
32 2      END MESSAGE;

33 1      LOCATE:PROCEDURE(Y,X); /* LOCATES CURSOR TO POSITION ON CRT */
34 2      DECLARE (Y,X,I) BYTE,
          HOME(*)BYTE DATA(1BH,12H,EOM),
          RIGHT(*)BYTE DATA(10H,EOM),
          DOWN(*)BYTE DATA(0AH,EOM);
35 2          CALL MESSAGE(.HOME); /* HOME CURSOR */
36 2          DO I=1 TO Y;
37 3              CALL MESSAGE(.DOWN); /* MOVE CURSOR DOWN */
38 3          END;
39 2          DO I=1 TO X;
40 3              CALL MESSAGE(.RIGHT); /* MOVE CURSOR RIGHT */
41 3          END;
42 2      END LOCATE;

43 1      THRESHOLD=40; /* SET THE THRESHOLD TO DECIDE
                       IF A KEY WAS PRESSED */
44 1      COMMAND=10011000B; /* INITIALIZE PORTS */
                       /* A-INPUT, B-OUTPUT, C-CONTROL */
45 1      CALL MESSAGE(.CLEARSCREEN); /* CLEAR THE SCREEN */

46 1      DO COUNT=0 TO 63; /* GETS INITIAL VALUES FOR ARRAY */
47 2          LAST$VALUE(COUNT)=VALUE;
48 2      END;

49 1      DO FOREVER;
50 2          MAX$VALUE=0; /* ZERO OUT INITIAL VALUES */
```

```
51 2      KEY$PUSHED=0;
52 2      DO COUNT=0 TO 63;
53 3          VOLTAGE=VALUE; /* GET INITIAL VOLTAGE READING */
                    /* IF VOLTAGE VALUE IS LESS THAN LAST VALUE, THEN
                    UPDATE THE ARRAY VALUE */
54 3          IF LAST$VALUE(COUNT)>VOLTAGE THEN
55 3              LAST$VALUE(COUNT)=VOLTAGE;
56 3          DIFFERENCE=VOLTAGE-LAST$VALUE(COUNT); /* DETERMINE THE
                                                    DIFFERENCE */
57 3          IF DIFFERENCE>MAX$VALUE THEN DO;
59 4              MAX$VALUE=DIFFERENCE; /* EXCHANGE VALUES IF LARGEST YET */
60 4              KEY$PUSHED=COUNT;
61 4              END;
62 3          END; /* END 0-63 LOOP */
63 2      IF MAX$VALUE>THRESHOLD THEN DO; /* IF A VALID PUSH */
65 3          CALL LOCATE(1,1); /* POSITION THE CURSOR */
66 3          CALL DISPLAY(KEY$PUSHED); /* DISPLAY THE VALUE */
67 3          END;
68 2      END; /* END DO FOREVER LOOP */
69 1      END KYBD; /* END KYBD PROGRAM */
```

MODULE INFORMATION:

```
CODE AREA SIZE      = 01CBH      459D
VARIABLE AREA SIZE  = 004EH      73D
MAXIMUM STACK SIZE  = 0006H      6D
127 LINES READ
0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

SIS-11 OBJECT LOCATER V3.0 INVOKED BY:
 LOCATE :F9:KYBD.LNK TO :F9:KYBD.LOC COLUMNS(2) RESTARTO CODE(100H)&
 * SYMBOLS LINES PURGE PRINT(:F9:KYBD.MAP)

YMBOL TABLE OF MODULE KYBD
 READ FROM FILE :F9:KYBD.LNK
 WRITTEN TO FILE :F9:KYBD.LOC

VALUE TYPE SYMBOL

MOD	KYBD
02BH SYM	MEMORY
00EH SYM	MAXVALUE
0E0H SYM	KEYPUSHED
0F2H SYM	VOLTAGE
100H SYM	CLEARSCREEN
101H SYM	VALUE
023H SYM	CHSEND
024H SYM	NUM
046H SYM	MESSAGE
0267H SYM	LOCATE
0329H SYM	X
006H SYM	HOME
10BH SYM	DOWN
101H LIN	3
108H LIN	5
104H LIN	7
1E0H LIN	9
1E5H LIN	11
1F2H LIN	14
1FAH LIN	16
1FFH LIN	19
10DH LIN	21
136H LIN	23
145H LIN	25
14CH LIN	28
15CH LIN	30
166H LIN	32
16DH LIN	35
182H LIN	37
18FH LIN	39
1A4H LIN	41
10DH LIN	43
119H LIN	45
12DH LIN	47
141H LIN	49
146H LIN	51
159H LIN	53
16EH LIN	55
189H LIN	57
197H LIN	60
19DH LIN	62
1AEH LIN	65
1BCH LIN	67
1BFH LIN	69

VALUE TYPE SYMBOL

02DDH SYM	COUNT
02DFH SYM	DIFFERENCE
02E1H SYM	THRESHOLD
02E3H SYM	LASTVALUE
0103H SYM	BLANKS
01E5H SYM	CHOUT
01FBH SYM	DISPLAY
0325H SYM	J
0326H SYM	STARTADDR
0328H SYM	Y
032AH SYM	I
0109H SYM	RIGHT
01C1H LIN	4
01CFH LIN	6
01DDH LIN	8
01E5H LIN	10
01E9H LIN	13
01F5H LIN	15
01FBH LIN	17
0204H LIN	20
0223H LIN	22
0242H LIN	24
0246H LIN	26
0255H LIN	29
0263H LIN	31
0267H LIN	33
0273H LIN	36
0288H LIN	38
029EH LIN	40
02ABH LIN	42
0115H LIN	44
011FH LIN	46
013AH LIN	48
0141H LIN	50
014BH LIN	52
015FH LIN	54
017AH LIN	56
0191H LIN	59
019DH LIN	61
01A4H LIN	63
01B5H LIN	66
01BCH LIN	68

