Stand Alone Control of MICROWIRE™ Peripherals Using the NMC87C257

National Semiconductor
Application Note 791
Charlie Mitchell
October 1991

INTENT

This note describes the implementation and use of a standard memory element in the realization of state machine control for the purpose of generating serial data. The applications shown here employ serial data streams to control and program peripheral devices which would otherwise require CPU support for use.

The benefit to the user of the demonstrated techniques is the low cost, low effort, implementation of tasks normally allocated to more sophisticated and more engineering intensive methods. These solutions expand the range of systems and applications in which a variety of National Semiconductor's MICROWIRE devices may be used.

MICROWIRE

The MICROWIRE standard is an interface technique first developed at National in the 1970's in an effort to reduce the component pin count (and hence package size and cost) required for the interfacing of microcontrollers to peripheral components. Over the ensuing years a wide variety of devices employing this interface technique have been introduced to the market. They include display drivers, analog to digital converters, phase lock loop frequency synthesizers, memories and complex analog devices. A full list of all but the most recent devices using the MICROWIRE interface can be found in the *Master Selection Guide*.

A MICROWIRE connection is a straight forward serial hookup consisting of data and clock. Generally, input and output data are presented on separate lines. The clock to data relationship resembles that of a TTL or CMOS 7400 series shift register with the positive edge of the clock performing the active transfer of data into and out of the device. Care must be taken to examine the data sheet for a device under consideration as there may be deviations from this general description. A more complete description of this interface method is available in National Semiconductor Application Note 452 by Abdul Aleaf.

STATE SEQUENCERS

State machines or sequencers in their simplest form consist of a current state memory element and a next state determination network. Upon a clock edge the next state information is converted and held as a new current state while a fresh next state logical determination takes place. One way of implementing this state sequencer is by utilizing a register or latch as the memory element and a Read Only Memory to supply the logic function for the next state. Because a ROM is a ''rectangular'' or complete logic array (i.e., for every input combination there exists a unique output), this next state logic is a lookup table.

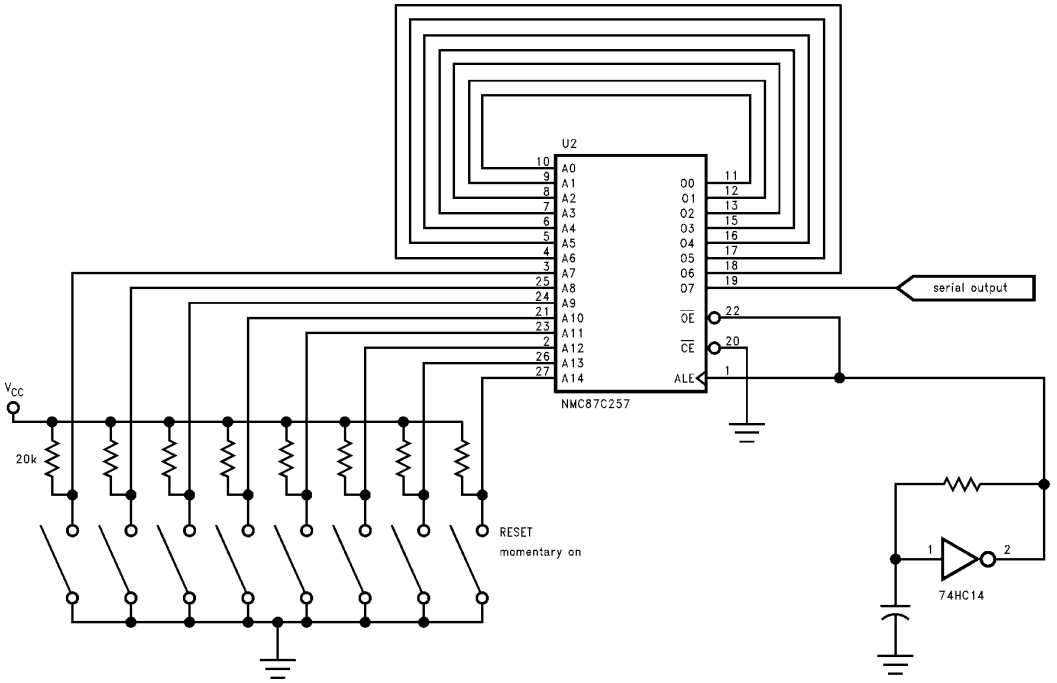
THE NMC87C257 UV ERASABLE CMOS PROM WITH LATCHES

The NMC87C257 is a device first conceived to reduce the chip count in microprocessor systems which had a multiplexed address/data bus. As such, the latches required to capture the address while return data occupied the bus were put on board the device. Intended for this microprocessor application, the NMC87C257 does not have the speed of some of the bipolar ''logic'' PROMs (nor the power dissipation), but it's large memory array would be exceedingly expensive in a bipolar device. The 32k x 8 memory means that in a state machine application fifteen inputs can define over 32,000 states, represented in eight output pins.

GENERATING A SERIAL OUTPUT WORD FROM THE NMC87C257 BASED STATE MACHINE

Figure 1 depicts a state machine capable of generating 128 different 128-bit serial data streams. DIP switches 0–7 select the specific data stream program. Seven bits of output data are fed back to inputs to define the next state in the serial data sequence. Bit 08 is the serial data output. A CMOS oscillator generates the clock. It is important to note that the clock drives both the ALE (Address Latch Enable) and $\overline{OE}$ (Output Enable) inputs. ALE is the signal which activates the ''open'' state of the input latch, as such, unlike an edge triggered register, the outputs follow the inputs until its (ALE's) fall. To avoid a high speed feedback phenomenon while the latches are open it is necessary to break the feedback loop and ''freeze'' the data at the desired output/input state. This is accomplished by disabling the TRI-STATE® outputs. As long as the outputs are loaded only by the high impedance inputs of the CMOS device, the next state information will be transferred into the latches. Resistive or bipolar logic loads should not be attached to lines operating in this manner.

RRD-B30M75/Printed in U. S. A.



Note: Refer to AN 140 for R & C values.

TL/D/11274–1

FIGURE 1. State Machine for Serial Word Generation

TABLE I. PROM Addresses and Data

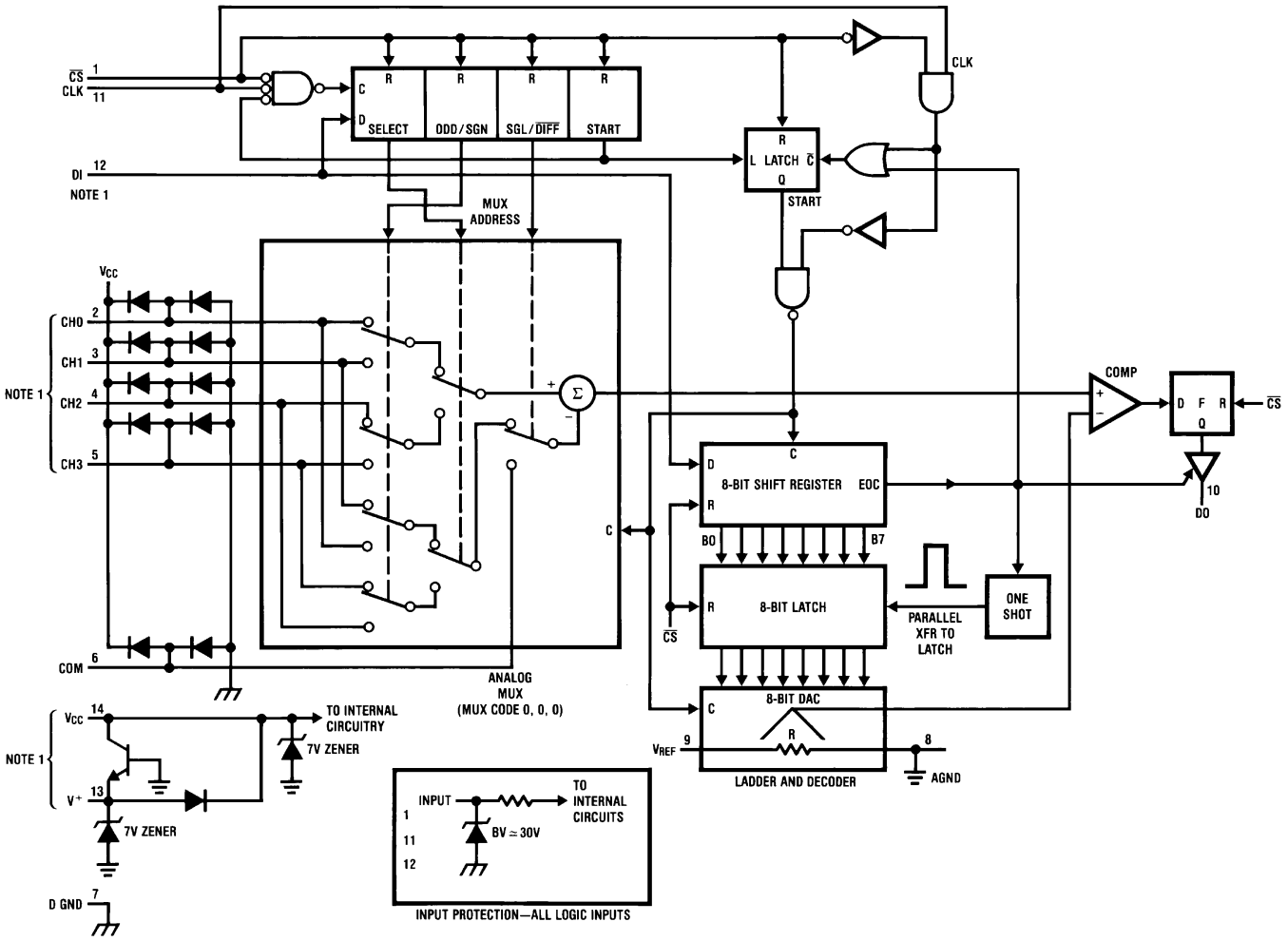
Address	Data (7 Bits)	Serial Output	Comments
nn 7F	00	0	Startup Address Generates 1st Word
nn 00	01	1	
nn 01	02	1	
nn 02	B4	0	
nn B4	03	1	
•			
•			
•			
•			
nn 7E	7E	0	Loop to Self - Stop

Table I shows an example of the PROM code which generates the serial output. The address includes a leading byte ''nn'' which will determine which of the bit streams will be selected. Notice that the 7-bit data is in fact the next state information reflected in the next address.

APPLICATIONS FOR A SERIAL WORD GENERATOR USING THE NMC87C257

A Power Supply Sequencer

The ADC0854 is a comparator circuit with a MICROWIRE controlled four input multiplexer and a settable 8-bit reference divider which drives the second compare input. A block diagram is depicted in *Figure 2*.



TL/D/11274–2

Note 1: For ADC0852; D1 is input directly to the D input of ODD/SIGN, select: is forced to a "1", A_{GND} and COM are internally tied to D_{GND}, only V_{CC} is brought out, V_{REF} is internally tied to V_{CC}, only CH2 and CH3 are brought out.

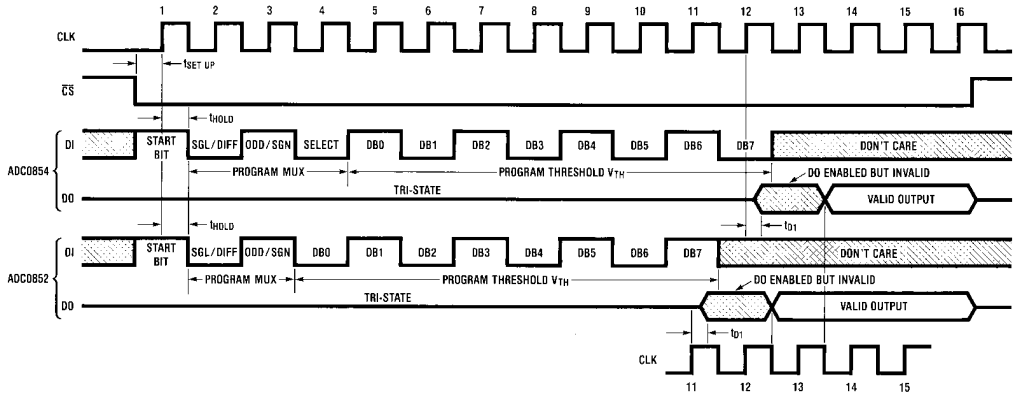
FIGURE 2. ADC0854 Detailed Block Diagram

Many voltage regulators feature an on-off input. Couple these linear components along with the serial word generator and a sophisticated power supply sequencer can be built.

The ADC0854 requires a 12-bit serial word to provide setup information. A start bit is required, which is followed by one bit to select four single ended or two differential inputs. A two bit channel selection and the eight bit reference data byte complete the serial word. It is depicted in *Figure 3*.

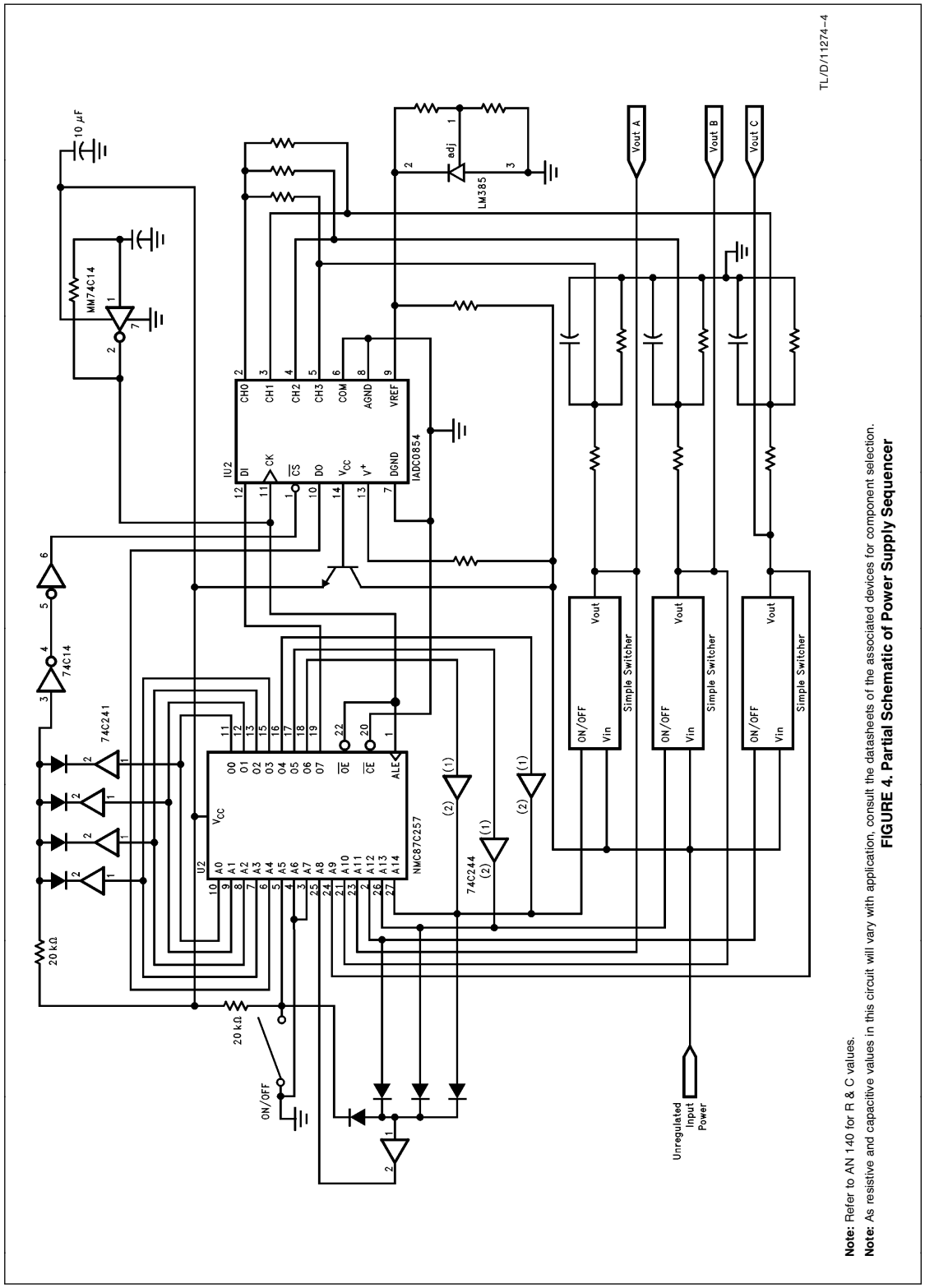
Referencing the simplified schematic in *Figure 4,* analog input signals are presented to the multiple inputs of the comparator from the voltage sources, in this case Simple Switcher™ Regulators are used.

The serial word is presented to the ADC0854 is generated from the sequence shown in *Figure 4A.* The Chip Select input which acts to latch the data word into the comparator is generated by a diode AND gate from the four output/input lines controlling the count. All diodes depicted in the schematic are in a single FSA2619P 16-pin dual-in-line package. An MM74C14 hex Schmitt trigger circuit provides the necessary clock wave form and an MM74C244 contributes buffering for the diode gates.



Note: Valid Output can change only on Falling Edge of CLK. TL/D/11274–3

FIGURE 3. ADC0854 Clock, Control Data and Output Data Relationships



Note: Refer to AN 140 for R & C values.

Note: As resistive and capacitive values in this circuit will vary with application, consult the datasheets of the associated devices for component selection.

FIGURE 4. Partial Schematic of Power Supply Sequencer

Each regulator output is sequentially selected by the PROM generated MICROWIRE and tested for compliance to a voltage level also set via the MICROWIRE. When Voltage A reaches its terminal value, the sequencer delays for a defined period while voltage settles as determined here by the RC network at the comparator input and then raises the control voltage to the regulator B ON/OFF input and, after monitoring that regulator's voltage rise, continues to regulator C. A stable and fixed reference is supplied for the comparisons by an LM385.

A similar control procedure allows an orderly shutdown. During operation the controller monitors the sum of the three supply voltages. A drop from the proper sum will commence the controlled shutdown. These procedures are delineated by the state diagram in *Figure 5b.* In order to differentiate the state defining the condition of the voltages when powering up and the state produced when shutting down a ''history circuit'' consisting of four diodes and a capacitor records the ''all supplies on'' condition. The ON/OFF switch must be recycled for the system to power up once again.

A Zener diode regulated output is provided from the ADC0854 permitting the PROM and the Schmitt Trigger oscillator to be powered from the primary source.

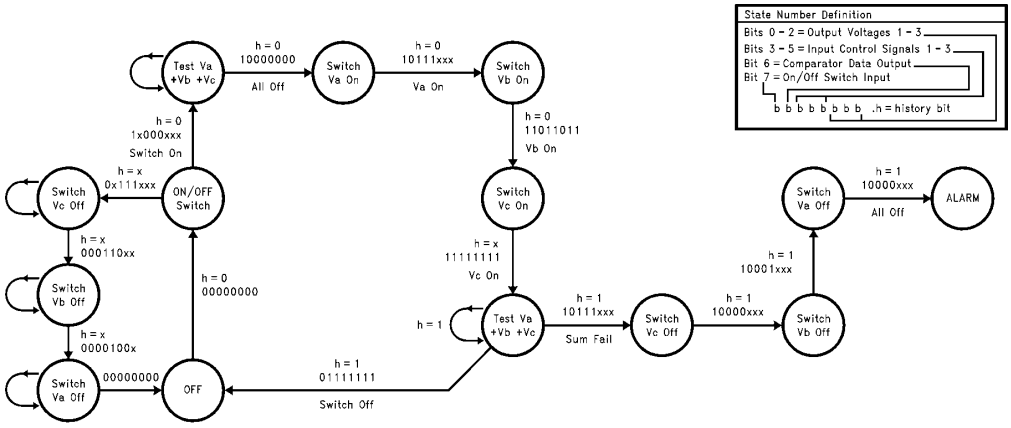
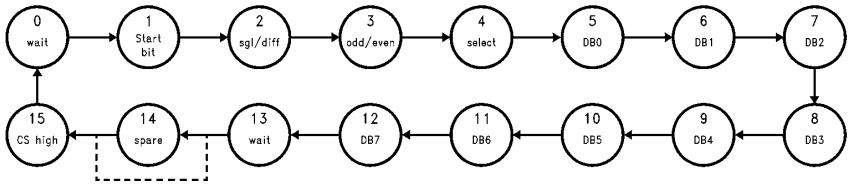


Figure 4A



TL/D/11274–5

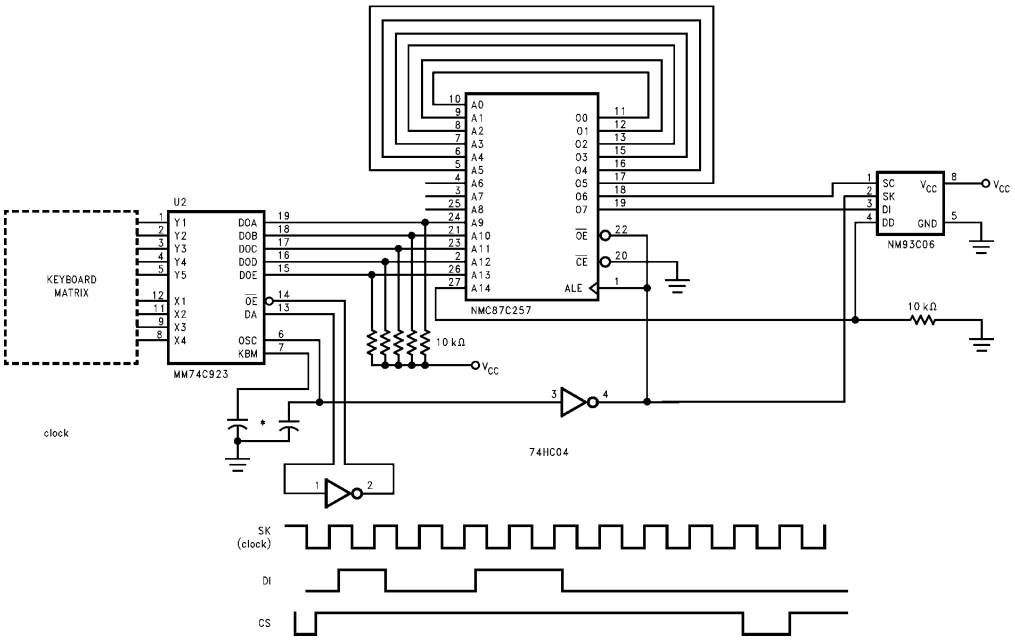
FIGURE 5. State Diagram for Power Supply Sequencer

PRODUCTION LINE PROGRAMMING OF MICROWIRE EEPROMS

EEPROMs are frequently programmed prior to board insertion on the production line. This programming may reflect the revision level of the system software and the engineering change level of the printed circuit board.

In this case an NM93C06 256-bit device has been selected. In a production line situation it is desirable to write a segment of memory with the parameters described above. In order to perform this task the memory must receive a series of MICROWIRE commands. Each of the commands is itself a state ordered sequence. The command sequences first enable the write capability, write 16 bits of information to the specified address and then turn off the write enable. Table II lists the inputs and outputs of the NMC87C257 for a sequence to write a sixteen bit word.

The logic diagram in *Figure 6* depicts a circuit which includes keypad entry capability such that an operator can select up to 20 different such commands.



TL/D/11274–6

*Consult MM74C923 datasheet for component values appropriate to the specific application.

FIGURE 6. Logic Diagram for EEPROM Programmer

TABLE II. PROM Listing for EEPROM Programming

Address (Hex)	00−04 (Hex)	EN	74HC173 (Hex)	CS	DI	DO	Comments
000	01	0	0	0	0	X	
001	02	0	0	1	0	X	Start Erase/Write Enable
002	03	0	0	1	1	X	Start Bit
003	04	0	0	1	0	X	Opcode
004	05	0	0	1	0	X	
005	06	0	0	1	1	X	
006	07	0	0	1	1	X	
007	08	0	0	1	0	X	
008	09	0	0	1	0	X	
009	0A	0	0	1	0	X	
00A	0B	0	0	1	0	X	
00B	0C	0	0	1	0	X	
00C	0D	0	0	1	0	X	
00D	0E	0	0	0	0	X	End Erase/Write Enable
00E	0F	0	0	0	0	X	
00F	10	0	0	1	0	X	Start Write of Data
010	11	0	0	1	1	X	
011	12	0	0	1	0	X	
012	13	0	0	1	1	X	
013	14	0	0	1	a	X	Address Entry
014	15	0	0	1	a	X	
015	16	0	0	1	a	X	
016	17	0	0	1	a	X	
017	18	0	0	1	a	X	
018	19	0	0	1	a	X	
019	1A	0	0	1	a	X	
01A	1A	0	0	1	a	X	
01A	1A	0	0	1	d	X	Data Input
01A	1B	0	0	1	d	X	
01B	1F	1	0	1	d	X	
01F	00	0	F	1	d	X	
020	01	0	F	1	d	X	
021	02	0	F	1	d	X	
022	03	0	F	1	d	X	
023	04	0	F	1	d	X	
024	05	0	F	1	d	X	
025	06	0	F	1	d	X	
026	07	0	F	1	d	X	
027	08	0	F	1	d	X	
028	09	0	F	1	d	X	
029	0A	0	F	1	d	X	
02A	0B	0	F	1	d	X	
02B	0C	0	F	1	d	0	End of Cycle
02C	0D	0	F	0	0	0	Wait for Data Out to
02D	0E	0	F	1	0	0	Go High
02E	0F	0	F	1	1	1	
02F	10	0	F	0	0	X	
030	11	0	F	1	0	X	Start of Erase/Write
031	12	0	F	1	1	X	Disable Cycle
032	13	0	F	1	0	X	
033	14	0	F	1	0	X	
034	15	0	F	1	0	X	
035	16	0	F	1	0	X	
036	17	0	F	1	0	X	
037	18	0	F	0	0	X	End of EWDS Command
038	19	0	F	0	0	X	

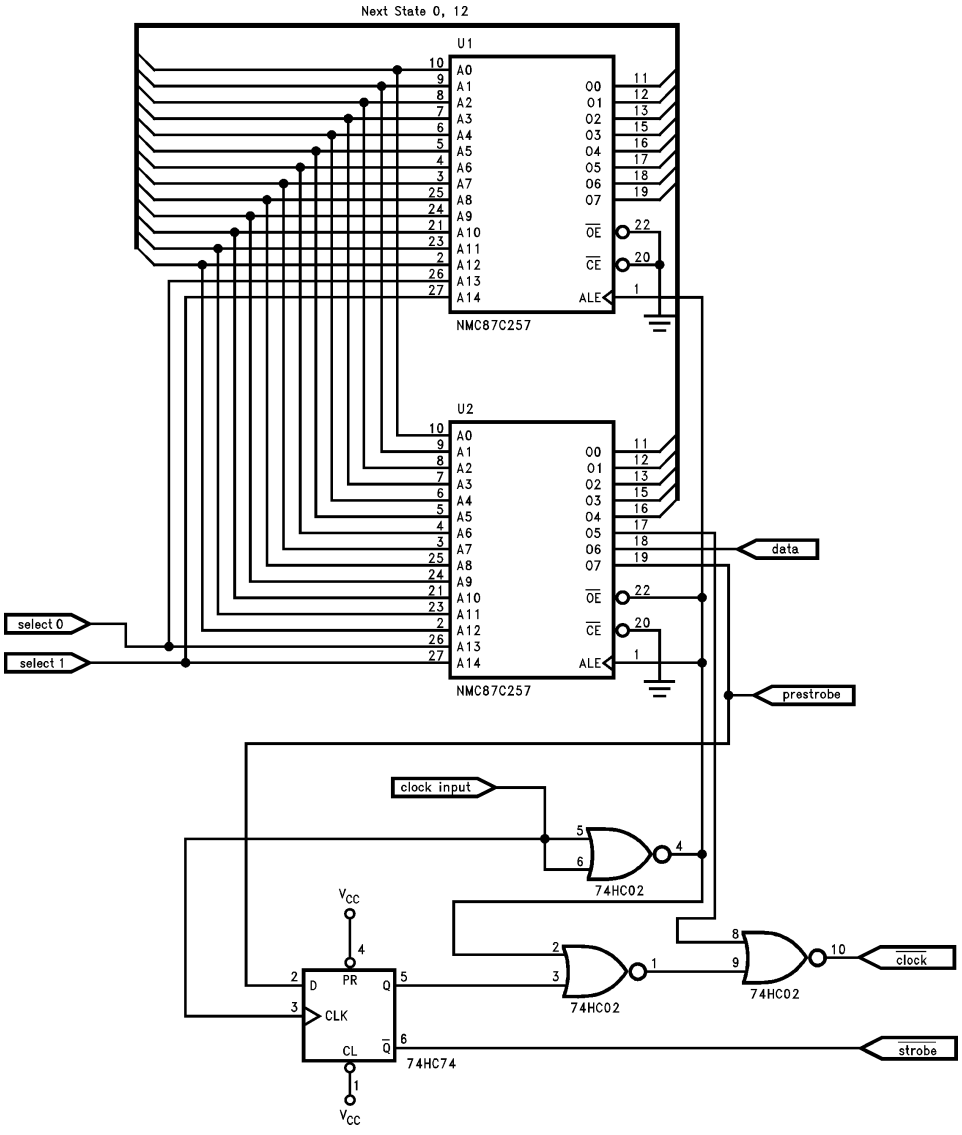
FILTER PARAMETER CONTROL

The LMC835 provides the complete resistor and switch set to implement a stereo 7 band equalizer or a mono 14 band system. While control of this device is usually provided from a microcontroller there are instances where that expense and effort are not necessary to achieve a complex filtering function.

To program the LMC835 one of fourteen frequency bands must be selected and the gain for that band entered. This band gain setting requires a minimum of 18 states. All fourteen bands must be preset and various control states must be implemented. Because of the large number of states involved in setting each of the fourteen bands a wider word is needed than can be implemented in a single PROM. There are several methods of dealing with this, however the most straight forward (for purposes of illustration) is to employ two PROMs.

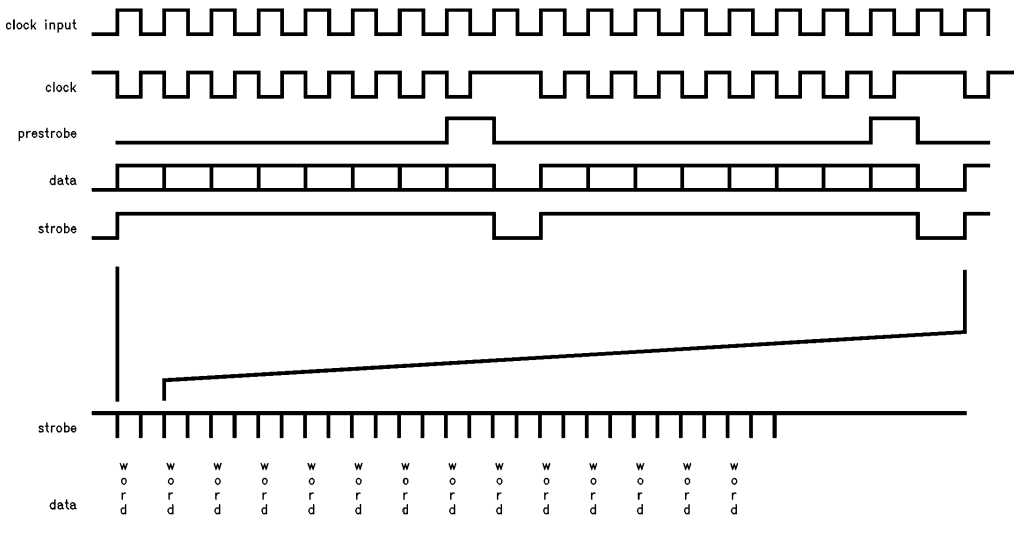
The MICROWIRE interface used with the LMC835 differs slightly from those implemented above in that it uses a strobe to transfer data from the internal shift register to the latch for the addressed switch matrix. This permits the reprogramming of individual bands without the necessity of rewriting the entire machine state.

A complete logic diagram for the word generator is shown in *Figure 7*. The schematics for the implementation of the linear portions of the circuit can be obtained by referencing the LMC835 datasheet. A template for the PROM listing is also shown in *Figure 8*.



TL/D/11274–7

FIGURE 7. Logic Diagram for Equalizer Controller



Data / Prestrobe / Address																			
00	1	b	24	1	b	48	1	b	6C	1	b	90	1	b	B4	1	b	D8	1	b	band selection bits
01	1	b	25	1	b	49	1	b	6D	1	b	91	1	b	B5	1	b	D9	1	b	
02	1	b	26	1	b	4A	1	b	6E	1	b	92	1	b	B6	1	b	DA	1	b	
03	1	b	27	1	b	4B	1	b	6F	1	b	93	1	b	B7	1	b	DB	1	b	
04	1	R	28	1	R	4C	1	R	70	1	R	94	1	R	B8	1	R	DC	1	R	range, Ch, B
05	1	R	29	1	R	4D	1	R	71	1	R	95	1	R	B9	1	R	DD	1	R	range, Ch, A
06	1	X	2A	1	X	4E	1	X	72	1	X	96	1	X	BA	1	X	DE	1	X	don't care
07	0	H	2B	0	H	4F	0	H	73	0	H	97	0	H	BB	0	H	DF	0	H	DATA I
08	1	0	2C	1	0	50	1	0	74	1	0	98	1	0	BC	1	0	E0	1	0	
09	1	g	2D	1	g	51	1	g	75	1	g	99	1	g	BD	1	g	E1	1	g	
0A	1	g	2E	1	g	52	1	g	76	1	g	9A	1	g	BE	1	g	E2	1	g	
0B	1	g	2F	1	g	53	1	g	77	1	g	9B	1	g	BF	1	g	E3	1	g	
0C	1	g	30	1	g	54	1	g	78	1	g	9C	1	g	C0	1	g	E4	1	g	gain selection bits
0D	1	g	31	1	g	55	1	g	79	1	g	9D	1	g	C1	1	g	E5	1	g	
0E	1	g	32	1	g	56	1	g	7A	1	g	9E	1	g	C2	1	g	E6	1	g	
0F	1	B	33	1	B	57	1	B	7B	1	B	9F	1	B	C3	1	B	E7	1	B	boost/cut
10	0	L	34	0	L	58	0	L	7C	0	L	A0	0	L	C4	0	L	E8	0	L	DATA II
11	1	0	35	1	0	59	1	0	7D	1	0	A1	1	0	C5	1	0	E9	1	0	
12	1	b	36	1	b	5A	1	b	7E	1	b	A2	1	b	C6	1	b	EA	1	b	
13	1	b	37	1	b	5B	1	b	7F	1	b	A3	1	b	C7	1	b	EB	1	b	
14	1	b	38	1	b	5C	1	b	80	1	b	A4	1	b	C8	1	b	EC	1	b	
15	1	b	39	1	b	5D	1	b	81	1	b	A5	1	b	C9	1	b	ED	1	b	
16	1	R	3A	1	R	5E	1	R	82	1	R	A6	1	R	CA	1	R	EE	1	R	
17	1	R	3B	1	R	5F	1	R	83	1	R	A7	1	R	CB	1	R	EF	1	R	
18	1	X	3C	1	X	60	1	X	84	1	X	A8	1	X	CC	1	X	F0	1	X	
19	0	H	3D	0	H	61	0	H	85	0	H	A9	0	H	CD	0	H	F1	0	H	
1A	1	0	3E	1	0	62	1	0	86	1	0	AA	1	0	CE	1	0	F2	1	0	
1B	1	g	3F	1	g	63	1	g	87	1	g	AB	1	g	CF	1	g	F3	1	g	
1C	1	g	40	1	g	64	1	g	88	1	g	AC	1	g	D0	1	g	F4	1	g	
1D	1	g	41	1	g	65	1	g	89	1	g	AD	1	g	D1	1	g	F5	1	g	
1E	1	g	42	1	g	66	1	g	8A	1	g	AE	1	g	D2	1	g	F6	1	g	
1F	1	g	43	1	g	67	1	g	8B	1	g	AF	1	g	D3	1	g	F7	1	g	
20	1	g	44	1	g	68	1	g	8C	1	g	B0	1	g	D4	1	g	F8	1	g	
21	1	B	45	1	B	69	1	B	8D	1	B	B1	1	B	D5	1	B	F9	1	B	
22	0	L	46	0	L	6A	0	L	8E	0	L	B2	0	L	D6	0	L	FA	0	L	
23	1	0	47	1	0	6B	1	0	8F	1	0	B3	1	0	D7	1	0	FB	0	0	NEXT ADDR = 00

TL/D/11274–8

FIGURE 8. Template for PROM Program for Filter Controller

SUMMARY

The use of the NMC87C257 CMOS PROM with latches has been shown to be an effective element in the implementation of several MICROWIRE interfaces. This use allows the designer to implement systems with devices necessitating a MICROWIRE interface without the use of a microcontroller or microprocessor.

In constructing or adapting any of the circuits described in this note the reader is advised to obtain copies of the National Semiconductor data sheets for the components included and to review their operation for applicability to their system requirements.

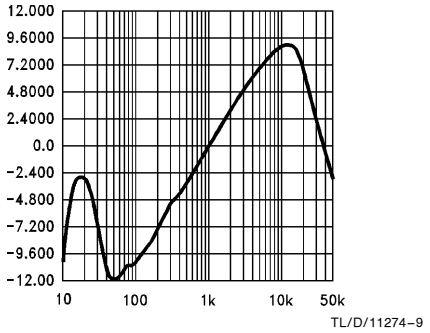


FIGURE A1. A Ramp Transfer Characteristic. Refer to Table A1 for program inputs.

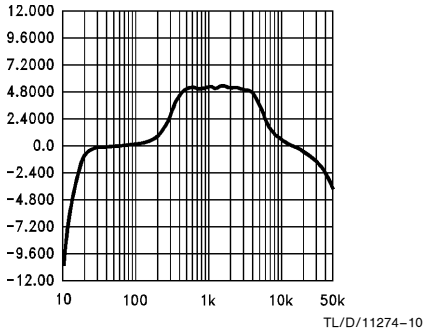


FIGURE A2. Vocal Presence Filter. Refer to Table A2 for program inputs.

TABLE A1. Program Inputs for Ramp Transfer Characteristic

Frequency (Hz)	Level (dB)
40	−7
63	−6
100	−5
160	−4
250	−3
400	−2
630	−1
1k	0
1.6k	+1
2.5k	+2
4k	+3
6.3k	+4
10k	+5
16k	+6

TABLE A2. Program Inputs for Vocal Presence Filter

Frequency (Hz)	Level (dB)
40	0 Subsonic Filter
63	0
100	0
160	0
250	0
400	+3
630	+3
1k	+3
1.6k	+3
2.5k	+3
4k	+3
6.3k	0 Supersonic Filter
10k	0
16k	0

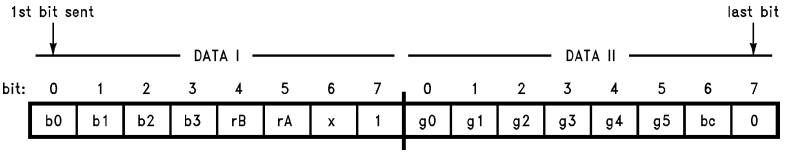
Figures A1 and *A2* are gain vs. frequency plots of specimen filters realized using the logic of *Figure 7* and PROM code generated with SM835.c.

```
/*************************************************************
*                                                            *
* File: SM835.C                                              *
* Author: Bob Moses, Rane Corporation, Mukilteo, WA          *
* Revision: 18 June 1991                                     *
* Compiler: Borland TurboC                                   *
*                                                            *
* Description: Generates Intel Hex file for NMC87C257 based  *
*              LMC835 state machine loader.                  *
*                                                            *
* File Input:  void                                          *
* File Output: SML835.HEX                                    *
*                                                            *
*************************************************************/

#include "stdio.h"

/*------------*/
/* Data Types */
/*------------*/

struct LMC835_RECORD
{
    int chAbands[7];   /* gains for chan A bands */
    int chArng;        /* chan A range */
    int chBbands[7];   /* gains for chan B bands */
    int chBrng;        /* chan B range */
};

/*--------------------*/
/* Function Prototypes*/
/*--------------------*/

void say_howdy(void);
void get_parameters(struct LMC835_RECORD *eq);
void compile_state_mach(struct LMC835_RECORD *eq, unsigned int states[]);
void output_data(unsigned int states[]);
void wr_ihex_data_rec(FILE *outfile, unsigned int addr, unsigned char recsize,
                      unsigned char data[]);

/*--------------*/
/* Main Program */
/*--------------*/

main()
{
    /* declare one LMC835 equalizer */
    struct LMC835_RECORD eql;
```

```
    /* 14 bands * 2 blocks/band * 9 states/block + final state = 253 states */
    unsigned int states[253];

    say_howdy();
    get_parameters(&eql);
    compile_state_mach(&eql,states);
    output_data(states);
}

/*------------*/
/* Functions  */
/*------------*/

void say_howdy(void)
{
    clrscr();
    fprintf(stdout,"\nSM835 - NMC87C257 LM835 State Machine Loader.");
    fprintf(stdout,"\n\nThis Program accepts parameters for an LMC835-based");
    fprintf(stdout,"\nequalizer and generates an Intel Hex file ( SML835.HEX )");
    fprintf(stdout,"\nfor the NMC87C257 State Machine Loader. This file can be");
    fprintf(stdout,"\nloaded into most EPROM programmers and split-programmed");
    fprintf(stdout,"\n(even and odd bytes) into two EPROMs.");
    fprintf(stdout,"\n\nThe LMC835 graphic equalizer consists of two channels");
    fprintf(stdout,"\n(chan A & chan B), each channel has 7 bands. The range of");
    fprintf(stdout,"\neach band is selectable for ± 12 dB in 1 dB steps, or");
    fprintf(stdout,"\n± 6 dB in 1/2 dB steps.\n");
}

void get_parameters(struct LMC835_RECORD *eq)
{
    unsigned int i,currng;
    int tempint;
    char chan;
    float tempfloat;

    /* get range for chan A */
    fprintf(stdout,"\nPlease enter range of chan A (0 = ±12dB, 1 = ±6dB): ");
    fscanf(stdin,"%d",&tempint);
    eq->chArng = tempint&0x0001;

    /* get range for chan B */
    fprintf(stdout,"Please enter range of chan B (0 = ±12dB, 1 = ±6dB): ");
    fscanf(stdin,"%d",&tempint);
    eq->chBrng = tempint&0x0001;
```

```
    /* get EQ bands */
    for(i = 0,chan = 'A',currng = 12-6*(eq->chArng);i < 14;)
    {
        /* if start of chan B, modify chan flag and chan range variables */
        if(i == 7)
        {
            chan = 'B';
            currng = 12-6*(eq->chBrng);
        }
        /* get a band value */
        fprintf(stdout,"Gain of chan %c band #%d? ",chan,(i%7)+1);
        fscanf(stdin,"%g",&tempfloat);

        /* scale for range */
        if(currng == 6) tempint = (int)(tempfloat*2.0);
        else tempint = (int)tempfloat;

        /* check limits */
        if((tempint > 12) || (tempint < -12))
        {
            fprintf(stderr,"...err! value must be between -%d and +%d\n",currng,currng);
            continue;
        }
        /* save band */
        if(chan == 'A') eq->chAbands[i] = tempint;
        else eq->chBbands[i-7] = tempint;
        i++; /* band counter */
    }
}
void compile_state_mach(struct LMC835_RECORD *eq, unsigned int states[])
{
```

LMC835 CONTROL CODES



TL/D/11274–11

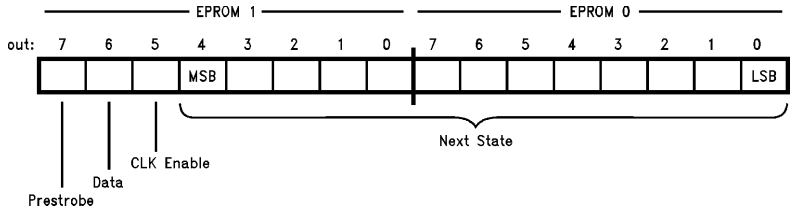
/*

State Machine:

--- State ---		Outputs		
Current	Next	Data	Prestrobe	CLK Enable
0	1	Chan A, band 1, DATA I b0	0	0
1	2	Chan A, band 1, DATA I b1	0	0
2	3	Chan A, band 1, DATA I b2	0	0
3	4	Chan A, band 1, DATA I b3	0	0
4	5	Chan A, band 1, DATA I rB	0	0
5	6	Chan A, band 1, DATA I rA	0	0
6	7	Chan A, band 1, DATA I x	0	0
7	8	Chan A, band 1, DATA I 1	1	0
8	9	rest for strobe	0	0
9	9	Chan A, band 1, DATA II g0	0	0
10	11	Chan A, band 1, DATA II g1	0	0
11	12	Chan A, band 1, DATA II g2	0	0
12	13	Chan A, band 1, DATA II g3	0	0
13	14	Chan A, band 1, DATA II g4	0	0
14	15	Chan A, band 1, DATA II g5	0	0
15	16	Chan A, band 1, DATA II bc	0	0
16	17	Chan A, band 1, DATA II 0	1	0
17	18	rest for strobe	0	0
18	19	Chan A, band 2, DATA I b0	0	0
:	:	:	:	:
34	35	Chan A, band 2, DATA II 0	1	0
35	36	rest for strobe	0	0
:	:	:	:	:
124	125	Chan A, band 7, DATA II 0	1	0
125	126	rest for strobe	0	0
126	127	Chan B, band 1, DATA I b0	0	0
:	:	:	:	:
250	251	Chan B, band 7, DATA II 0	1	0
251	252	rest for strobe	0	0
252	252	0	0	1

NMC87C257 pin connections:

TL/D/11274–13



TL/D/11274–12

/*

```
    unsigned int band,substate,stateimg,curstate;
    unsigned char LMC835GainCodeTable[] = {0x2F,0x2D,0x29,0x01,0x16,0x2A,
                                           0x12,0x02,0x04,0x08,0x10,0x20,
                                           0x00,
                                           0x20,0x10,0x08,0x04,0x02,0x12,
                                           0x2A,0x16,0x01,0x29,0x2D,0x2F};

    /*--------*/
    /* chan A */
    /*--------*/

    for(band = 0,curstate = 0;band < 7;band++)
    {
        for(substate = 0;substate < 18;substate++)
        {
            /* next state = current state +1 */
            stateimg = (curstate+1)&0x1FFF;

            /* CLK Enable */
            stateimg &= 0xDFFF;

            /* Prestrobe */
            if((substate == 7)||(substate == 16)) stateimg += 0x8000;

            /* Data */
            switch(substate)
            {
                case 0: /* DATA I b0 */
                    stateimg += (((band+1)&0x0001)<<14);
                    break;
                case 1: /* DATA I b1 */
                    stateimg += (((band+1)&0x0002)<<13);
                    break;
                case 2: /* DATA I b2 */
                    stateimg += (((band+1)%0x0004)<<12);
                    break;
                case 3: /* DATA I b3 */
                    stateimg += (((band+1)%0x0008)<<11);
                    break;
                case 4: /* DATA I rB */
                    stateimg += (((eq->chBrng)&0x0001)<<14);
                    break;
                case 5: /* DATA I rA */
                    stateimg += (((eq->chArng)&0x0001)<<14);
                    break;
                case 6: /* DATA I don't care */
                    break;
                case 7: /* DATA I 1 */
                    stateimg += 0x4000;
                    break;
*/
```

```
                case 8: /* rest for strobe */
                    break;
                case 9: /* DATA II g0 */
                    stateimg += ((LMC835GainCodeTable[eq->chAbands[band]+12]&0x0001)<<14);
                    break;
                case 10: /* DATA II g1 */
                    stateimg += ((LMC835GainCodeTable[eq->chAbands[band]+12]&0x0002)<<13);
                    break;
                case 11: /* DATA II g0 */
                    stateimg += ((LMC835GainCodeTable[eq->chAbands[band]+12]&0x0004)<<12);
                    break;
                case 12: /* DATA II g3 */
                    stateimg += ((LMC835GainCodeTable[eq->chAbands[band]+12]&0x0008)<<11);
                    break;
                case 13: /* DATA II g4 */
                    stateimg += ((LMC835GainCodeTable[eq->chAbands[band]+12]&0x0010)<<10);
                    break;
                case 14: /* DATA II g5 */
                    stateimg += ((LMC835GainCodeTable[eq->chAbands[band]+12]&0x0020)<<9);
                    break;
                case 15: /* DATA II bc ( 0 = cut, 1 = boost) */
                    if(eq->chAbands[band] < 0) stateimg &= 0xBFFF;
                    else stateimg += 0x4000;
                    break;
                case 16: /* DATA II 0 */
                    stateimg &= 0xBFFF;
                    break;
                case 17: /* rest for strobe */
                    break;
            } /* switch... */

            /* write this state to states array */
            states[curstate++] = stateimg;

        } /* for(substate... */
    } /* for(band... */

    /*--------*/
    /* chan B */
    /*--------*/

    for(band = 7;band < 14;band++)
    {
        for(substate = 0;substate < 18;substate++)
        {
            /* next state = current state +1 */
            stateimg = (curstate+1)&0x1FFF;

            /* CLK Enable */
            stateimg &= 0xDFFF;
```

```
/* Prestrobe */
if((substate == 7)||(substate == 16)) stateimg += 0x8000;

/* Data */
switch(substate)
{
    case 0: /* DATA I b0 */
        stateimg += (((band+1)&0x0001)<<14);
        break;
    case 1: /* DATA I b1 */
        stateimg += (((band+1)&0x0002)<<13);
        break;
    case 2: /* DATA I b2 */
        stateimg += (((band+1)&0x0004)<<12);
        break;
    case 3: /* DATA I b3 */
        stateimg += (((band+1)&0x0008)<<11);
        break;
    case 4: /* DATA I rB */
        stateimg += (((eq->chBrng)&0x0001)<<14);
        break;
    case 5: /* DATA I rA */
        stateimg += (((eq->chArng)&0x0001)<<14);
        break;
    case 6: /* DATA I don't care */
        break;
    case 7: /* DATA I 1 */
        stateimg += 0x4000;
        break;
    case 8: /* rest for strobe */
        break;
    case 9: /* DATA II g0 */
        stateimg += ((LMC835GainCodeTable[eq->chBbands[band-7]+12]&0x0001)<<14);
        break;
    case 10: /* DATA II g1 */
        stateimg += ((LMC835GainCodeTable[eq->chBbands[band-7]+12]&0x0002)<<13);
        break;
    case 11: /* DATA II g2 */
        stateimg += ((LMC835GainCodeTable[eq->chBbands[band-7]+12]&0x0004)<<12);
        break;
    case 12: /* DATA II g3 */
        stateimg += ((LMC835GainCodeTable[eq->chBbands[band-7]+12]&0x0008)<<11);
        break;
    case 13: /* DATA II g4 */
        stateimg += ((LMC835GainCodeTable[eq->chBbands[band-7]+12]&0x0010)<<10);
        break;
```

```
                case 14: /* DATA II g5 */
stateimg +=((LMC835GainCodeTable[eq->chBbands[band-7]+12]&0x0020)<<9);
                    break;
                case 15: /* DATA II bc ( 0 = cut, 1 = boost) */
                    if(eq->chBbands[band-7] < 0) stateimg &= 0xBFFF;
                    else stateimg += 0x4000;
                    break;
                case 16: /* DATA II 0 */
                    stateimg &= 0xBFFF;
                    break;
                case 17: /* rest for strobe */
                    break;
            } /* switch... */

            /* write this state to states array */
            states[curstate++] = stateimg;

        } /* for(substate... */
    } /* for(band... */

    /* final state: "jump 0" with clock disabled */
    states[252] = 252+0x2000;
}

void output_data(unsigned int states[])
{
    unsigned int i,addr,bitmask;
    unsigned char csum,data[16];
    FILE *outfile;

    /* open output file */
    if((outfile = fopen("sml835.hex","w")) == NULL )
    {
        fprintf(stderr,"can't open file SML835.HEX");
        exit(0);
    }

    /* write states to stdout */
    /*------------------------*/
    for(i = 0;i < 253;i++)
{
        if(i == 252) fprintf(stdout,"\n\nFinal state...");
        else if (!(i&18)) fprintf(stdout,"\n\nBand %d...",(i/18)+1);
        fprintf(stdout,"\nState %d: ",i);

        /* write each state as a binary image  */
        for(bitmask = 0;bitmask < 16;bitmask++)
        {
            if((states[i]<<bitmask)&0x8000) fprintf(stdout,"1");
            else fprintf(stdout,"0");
        }
    }
```

```
    /* write states to Intel Hex file */
    /*------------------------------*/
    fprintf(stdout,"\n\nWriting Intex Hex file: SML835.HEX...\n");

    /* write first 252 states */
    for(addr = 0;addr < 252;addr += 8)
    {
        /* copy 8 states (16 bytes) to temp data buffer */
        for(i = 0;i < 16;i += 2)
        {
            data[i] = (char)(states[addr+(i/2)]&0x00FF);
            data[i+1] = (char)((states[addr+(i/2)]>>8)*0x00FF);
        }

        /* write data to Intex Hex record */
        wr_ihex_data_rec(outfile,addr*2,16,data);
    }

    /* write last state */
    data[0] = (char)(states[252]&0x00FF);
    data[1] = (char)((states[252]>>8)&0x00FF);
    wr_ihex_data_rec(outfile,252*2,2,data);

    /* EOF record */
    fprintf(outfile,"\n:00000001FF");

    /* close file */
    fclose(outfile);
}

void wr_ihex_data_rec(FILE *outfile, unsigned int addr, unsigned char recsize, unsigned
char data[])
{
    unsigned int i;
    unsigned char csum;

    /* record mark, record length, record address, and record type fields */
    fprintf(outfile,"\n:%2.2X%4.4X00",recsize,addr);
    csum = recsize + (char))addr&0x00FF) + (char)((addr>>8)&0x00FF);

    /* data field */
    for(i = 0;i < recsize;i++)
    {
        fprintf(outfile,"%2.2X",data[i]);
        csum += data[i];
    }

    /* checksum field */
    csum &= 0x00FF;
    csum *= -1;
    fprintf(outfile,"%2.2X",csum);
}
```

BIBLIOGRAPHY

NMC87C257 — CMOS PROM with Address Latches

ADC0854 — Multiplexed Comparator with 8-bit Reference Divider

FSA2619 — Monolithic Diode Array

MM74C14 — Hex CMOS Schmitt Trigger

NM93C06 — 256-bit Electrically Erasable Programmable Memory

MM74C923 — 20 Key Encoder

LMC35 — Digital Controlled Graphic Equalizer

AN-452 — MICROWIRE Serial Interface

AN-140 — CMOS Schmift Trigger

Appendix

ACKNOWLEDGEMENTS

The author would like to express his gratitude to the engineering staff at Rane Corporation in Mukilteo, Washington for providing expert assistance and breadboarding of the LMC835 application and to Bob Moses for contributing a C program (see Appendix) to generate the ROM map.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998