MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

Order this document
by AN1122/D

AN1122

Running the MC44802A PLL Circuit

Prepared by
Paul Brownlee/Linear Applications
Bipolar Analog IC Division

INTRODUCTION

The MC44802A is the PLL portion of a tuning circuit intended for applications involving television, FM radio, and Set-Top converters up to 1.3 GHz. Coupled with a VCO and mixer, a complete tuning circuit can be formed. The tuning frequency is controlled through an MCU serial interface (I^2C).

As noted in the MC44802A data sheet, an MCU is recommended for sending the serial control bytes. This application note describes combining an MC68HC11E9 with an MC44802A in a tuner design. The information is sufficiently general however, that most any MCU could be used for this function. Those with a limited background in the use and programming of MCUs will find the information adequately detailed to permit a thorough understanding.

A Look at the MC44802A

The MC44802A is manufactured using Motorola's high density bipolar MOSAIC process. It features:

- I^2C interface for MCU control.
- Selectable ÷8 prescaler and a 15-bit divider accept frequencies up to 1.3 GHz.
- Programmable reference divider.
- Phase/frequency comparator output can be set to high impedance for disabling.
- Op amp provides direct tuning voltage output (0.3 V to 30 V).
- Seven programmable output buffers (10 mA, 12 V) for band switching, etc.
- Output options for 62.5 kHz, reference frequency and the programmable divider which are useful for system debugging.

Figure 1 shows a simplified block diagram of the MC44802A. The I^2C Bus receiver is a central block that controls the HF prescaler, 15-bit divider, the oscillator (typ. 4.0 MHz crystal) reference divider, and the output buffers.

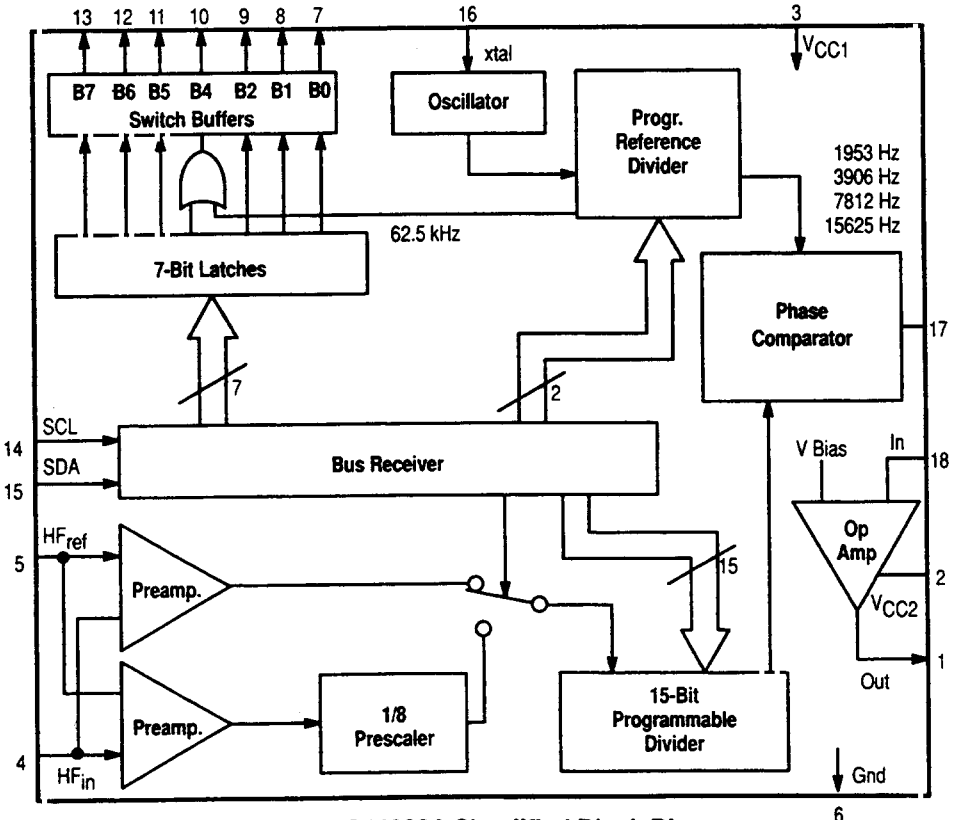

Figure 1. MC44802A Simplified Block Diagram

MOSAIC is a registered trademark of Motorola Inc.

The I²C Bus

The I²C (Inter-Integrated Circuit) Bus required by the MC44802 is a serial transfer process using two wires for data and clock (SDA — serial data, SCL — serial clock). Each transfer is initiated by a master and acknowledged by a slave device. Each slave is assigned a unique address, allowing multiple I²C devices to be connected to a single bus. An example of a data transfer is shown in Figure 2.

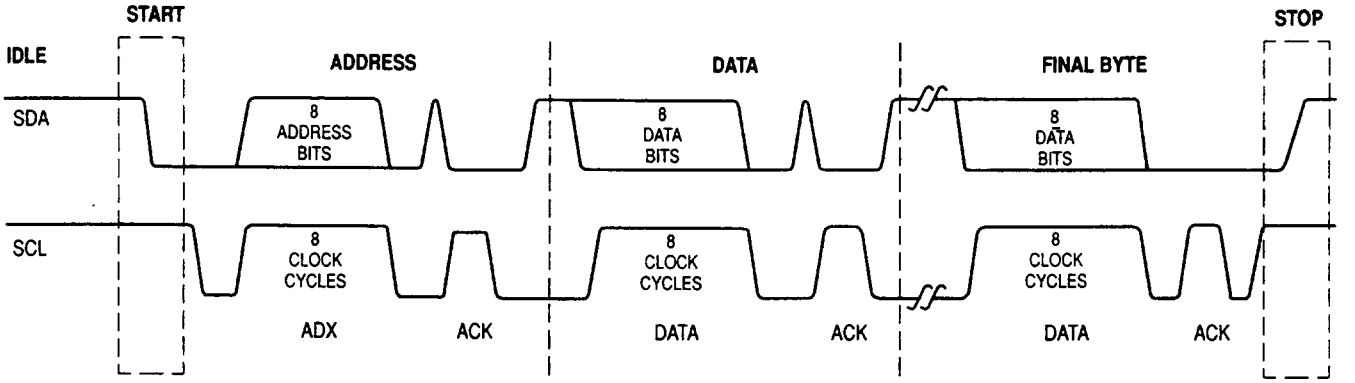

Figure 2. Complete Data Transfer Process

Referring to Figure 2:

Idle — When there are no transfers taking place on the bus, SDA and SCL idle high.

Start — A master initiates a data transfer by pulling SDA low while maintaining SCL in the high state. At this time all slave devices on the bus are listening for their address.

Address — The first byte is sent to select a slave device(s). Slaves that have read and write capabilities have a unique address for each. Upon completion of an address transmission, the master must leave the data line high and create the ACK clock pulse. The slave device is to acknowledge by pulling the data line to a stable low state before the end of the ACK pulse. From this point until a Stop Condition is generated, only the selected slave(s) device is active.

Data — The transfer continues with data bytes sent in the same manner as the address byte. An acknowledge is required at the end of each byte (except the last one). The master indicates the last data byte by sending the acknowledge (low) bit rather than leaving SDA high for slave acknowledge.

Stop — The master creates a Stop Condition by sending SCL high followed by a low-to-high SDA transition. This leaves the bus back in the idle state.

If a required acknowledge bit is not received for any reason, the master terminates the transfer and generates a Stop Condition.

The Microcontroller

The MCU chosen for an I²C data transfer must have a serial port with the following characteristics:

— Two-lines, clock and data, with open drain (collector) outputs
— 8-bit transfer buffer
— An I²C interface or I/O serial lines capable of emulating I²C protocol (Idle, Start/Stop conditions and ACK pulse).

Suitable microcontroller examples are the MC68HC11 or MC68HC05 families.

A SAMPLE SYSTEM

Overview

The remainder of this application note is devoted to describing a sample MC44802A system. From a high level view this system is simple (see Figure 3). Whenever the push button is pressed the circuit responds by changing the tuning frequency, and provides a display indicating the frequency. The following paragraphs describe this system which was built and tested to demonstrate the functionality of the MC44802A. Included are descriptions of each segment of this system — PLL tuning circuit, MCU control, user interface and LED displays.

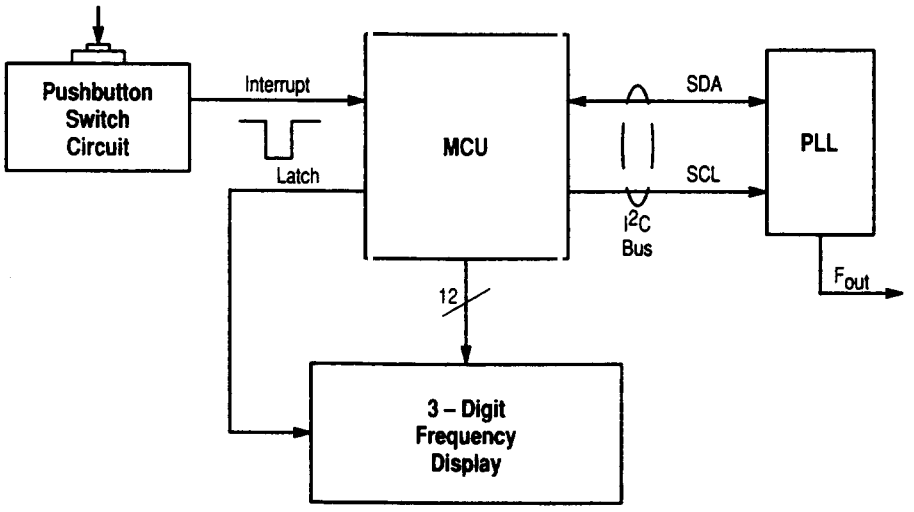

Figure 3. Simplified Block Diagram of the Video Frequency Controller

PLL Tuning Circuit Implementation

The MC44802A works with an MC1648 voltage controlled oscillator (VCO) to form a Phase-Locked Loop (see Figure 4). The MC1648 requires an external parallel tank circuit consisting of an inductor (L) and capacitors (Cv and Cx). Varactor diodes (Cv) are used in this case to provide a voltage variable capacitance for the VCO. The MC1648 may be operated from a +5.0 or –5.2 Vdc supply, depending upon system requirements (+5.0 V in this case). Its maximum frequency is typically 225 MHz.

The VCO output is connected through a capacitor to the phase detector input of the MC44802A. With the feedback network (G(s)) the MC44802A produces a stable voltage input to the tank circuit. A general purpose open collector output buffer (B2, Pin 9) is used in this application to switch a capacitor (Cx) in and out of the tank circuit. When that output buffer is switched low (by writing a "1" to it), the pin diode (D1) conducts making Cx part of the tank circuit (Cx//(Cv/2)). When the output buffer is open D1 does not conduct, thereby presenting a high impedance to Cx, making it ineffective. The tank circuit's capacitance is then Cv/2.

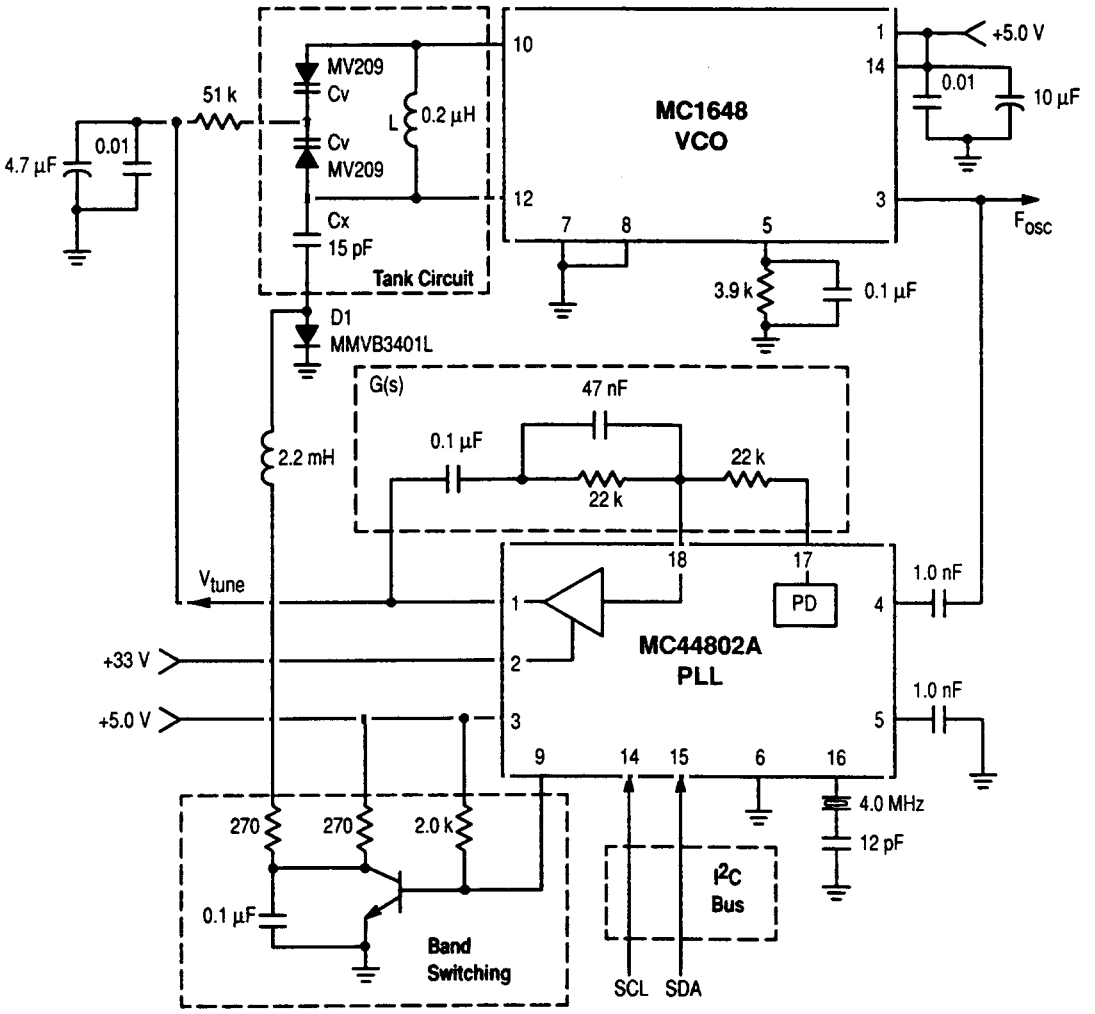

Figure 4. Sample PLL Tuning Circuit

I²C Data

Configuration data is sent by the MCU to the MC44802A I²C Bus Interface in five bytes as shown in Figure 5. Communication of the data is covered in the section describing MCU Implementation.

CA — Chip Address	1	1	0	0	0	0	1	0
CO — Control Info.	1	R6	T	P	R3	R2	R1	R0
BA — Band Info.	B7	B6	B5	B4	X	B2	B1	B0
FM — Frequency Info.	0	N14	N13	N12	N11	N10	N9	N8
FL — Frequency Info.	N7	N6	N5	N4	N3	N2	N1	N0

Figure 5. MC44802A I²C Byte Definitions

Referring to Figure 5:

CA — I^2C chip address for the MC44802A, $C2 (fixed internally).

CO — Sets up the 4.0 MHz oscillator divider ratio (R1, R0), prescaler (P), test outputs (R2, R3) and phase comparator output state (R2, R6, T) according to Figure 6.

BA — Each band buffer (Pins 7–13) can be set to active low by writing a 1 to it.

FM, FL — These two bytes set the tuning frequency. Their relationship with frequency (at Pin 4) depends on whether or not the prescaler is enabled, and the setting of the reference division ratio:

$$N = \frac{F_{out} \times \text{Divider ratio}}{F_{crystal}} \quad \text{(prescaler disabled)}$$

$$\text{or: } N = \frac{F_{out} \times \text{Divider ratio}}{F_{crystal} \times 8} \quad \text{(prescaler enabled)}$$

A hexadecimal representation of N at FM and FL sets the tuning frequency (F_{out}).

Per Figure 5, the address is sent and followed by CO, BA and/or FM, FL. Control and frequency byte pairs are distinguished in the first bit (1 for control, 0 for frequency). Therefore, it is not necessary to always send 5 bytes. A data transfer could consist of CA-CO-BA, or CA-FM-FL. The following example describes the five hex control bytes required to instruct the circuit to tune to VHF Channel 2 (101 MHz):

1) *$C2* (1100 0010) — This is the MC44802A address. The first byte of all MC44802A transmissions must be $C2.
2) *$88* (1000 1000) — R2, R6 and T are set to 000 to indicate normal operation. P=0 enables the internal prescaler. R1, R0=00 sets the divider ratio to 2048 which gives the greatest frequency resolution in the < 512 MHz region. R3 is optionally set high to output a 62.5 kHz test signal at Pin 10 (B4).
3) *$04* (0000 0100) — Sets band buffer B2 (Pin 9) high thereby disabling Cx.
4) and 5) *$19 40* (0001 1001, 0100 0000) — With the given prescaler and divider values, the frequency is defined by $N = F_{out}/15{,}625$ Hz. For 101 MHz:

$$N = \frac{101 \text{ MHz}}{15{,}625 \text{ Hz}} = 6464$$

which is represented in hex by $19 40.

Note that this is not a unique solution to getting 101 MHz out of the circuit since a different combination of prescaler setting, divider ratio and N could be used. Figure 7 shows a table of frequency control bytes (FM, FL) used in this application note. In all cases the internal prescaler is enabled, and the divider ratio is 2048.

MCU Implementation

The Motorola MC68HC11E9 has the required characteristics for generating I^2C transfers. It is equipped with parallel and serial I/O ports, timers, a pulse accumulator, an A/D converter system and expansion capability for multiple MPU systems. Each of these functions must be set-up and activated in user-programmed software to be part of the system. This allows the user to be concerned with only applicable functions. What follows are hardware and software descriptions for the

R1	R0	Divider Ratio
0	0	2048
0	1	1024
1	0	512
1	1	256

P	Prescaler
0	Enabled
1	Bypassed

R2	R3	Pin 10	Pin 11
0	0	—	—
0	1	62.5 kHz	—
1	0	F (ref)	FBY2
1	1	—	—

R2	R6	T	Phase Comparator Output State
0	0	0	Normal Operation
0	0	1	Off (High Impedance)
0	1	0	High
0	1	1	Low
1	0	0	Normal Operation
1	0	1	Off (High Impedance)
1	1	0	Normal Operation
1	1	1	Off (High Impedance)

Figure 6. CO Bit Specifications

FM	FL	F_{out} (MHz)	Display	FM	FL	F_{out} (MHz)	Display
$02	$80	10	None	$1A	$C0	107 (ch3)	C03
$06	$40	25	None	$1C	$40	113 (ch4)	C04
$08	$C0	35	None	$1E	$C0	123 (ch5)	C05
$0C	$80	50	None	$20	$40	129 (ch6)	C06
$12	$C0	75	075	$25	$80	150	150
$19	$00	100	090	$2A	$80	170	170
$19	$40	101 (ch2)	C02	$32	$00	200	None

P=0, R0=R1=0

Figure 7. Sample Frequency Control Bytes

sample MC44802A interface to this MCU. A full listing of the code is included in the Appendix. An HC11 program is written without line numbers. The code shown is the 'program.lst' version created by the assembler which inserts the line numbers and machine code.

Pin Descriptions

Note that only the HC11 pins used in this exercise are shown in Figure 8. Many of the I/O pins can be configured for different functions throughout the execution of a program. This is noted by pins labeled name1/name2. The names in bold indicate the functions used. They will be referred to by their functional name from here forward and are briefly explained below. Refer to the Appendix for code lines.

IC3 — (Input Capture 3) is an edge triggered interrupt pin that can be configured for rising, falling, or both edges. It is configured to respond to rising edges (code lines 70 and 71). All controller output changes are initiated at this pin.

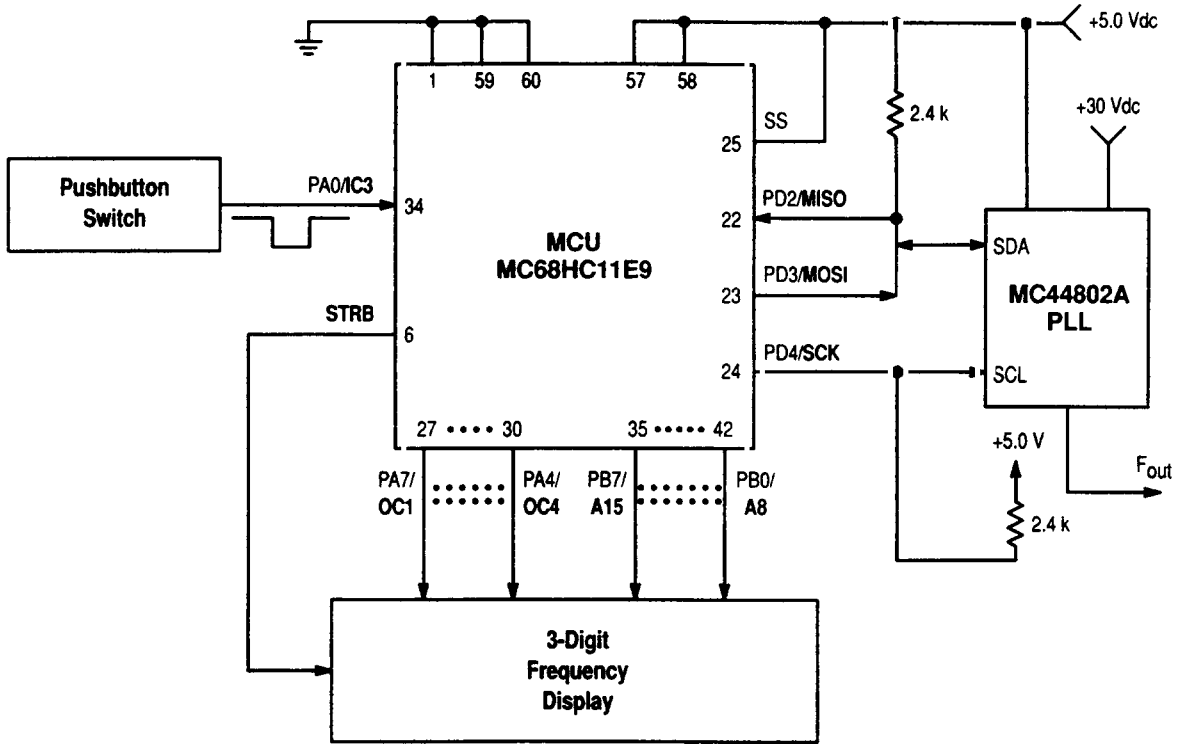

Figure 8. MCU Implementation

SPI (Serial Peripheral Interface) Pins:

MOSI — (Master Out Slave In) is the serial output line used for I^2C data communication with the PLL chip. The controller is configured as the master device in this exercise. This line is referred to as PORT D, bit 3 when the SPI is disabled (the SPI is enabled only during a serial transfer). It is essential that this be configured as an open drain output (an external pullup is used) when programming the SPI Control Register (SPCR, code lines 122–123). This allows the slave device (the PLL) to acknowledge by pulling the data line low.

MISO — (Master In Slave Out) is a serial input line to the controller. Tied to MOSI, it forms a bi-directional data port permitting the MCU to read the acknowledge pulse.

SCK — is the clock line in the I^2C protocol. It is referred to as PORT D, bit 4 when the SPI is disabled.

SS — is a slave select line that must be tied high (inactive) to set the MCU as the master.

Port A Pins:

PA7 — is a general I/O pin. It must be configured as input or output depending on the desired function. It is configured here as an output (code lines 46–48) to drive a bit in the seven segment display (in conjunction with PA6–PA4).

PA6 to PA4 — are fixed direction output pins also used for the seven segment displays.

Port B Pins:

PB7 to PB0 — are fixed direction output pins used for the seven segment displays.

STRB — is an enable line that provides an active low pulse each time new data is written to Port B. This is used to latch data into the display decoders.

Software Description

The software is written in two functional blocks — a main program and an interrupt service routine (ISR). The main program sets up the MCU ports and control registers. It then goes into a low power stopped state until an interrupt is initiated. The interrupt service routine creates the required serial and parallel output signals, and then returns control to the main program which waits for another interrupt.

The interrupt structure provides flexibility for expansion of this system. Other functions can be easily added to the main program without affecting performance of the serial interface. But for this exercise, the main program is kept simple. It sets up memory address references (lines 20–38), parallel Port A (lines 46–48), parallel Port B (line 51) and the interrupt control (lines 66–73). The main program then goes into its low power wait state. It does nothing until control is transferred to the ISR. An ISR flow diagram is included as Figure 10 for clarification.

The following program was written under the assumption that eventually the system will be run as a stand alone. Thus, the serial bytes pertaining to tuning requirements must be stored in the MCU EEPROM. To avoid program modification each time such requirements change, data space has been allocated for this function beginning at location B700. The program requires a specific data format while maintaining application flexibility.

The first requested transfer will output bytes starting at location B700. Transmission continues until a null data byte ($00) is encountered (which is not outputted). The two bytes following contain the display information. Transmissions of this format should follow consecutively as desired with another null after the last display value. The program will then reset the

data pointer to B700. Figure 9 shows the frequency data space for the sample system. It contains bytes for various frequencies from 75 MHz to 170 MHz. Band switching is done between the 90 MHz and 101 MHz (VHF Channel 2) frequency values.

B700> **c2 88 04 12 c0** 00 0f 75 **c2 16 10** 00 0f 90 **c2 88**
B710> **01 19 40** 00 cf 02 **c2 1a c0** 00 cf 03 **c2 1c 40** 00
B720> cf 04 **c2 1e c0** 00 cf 05 **c2 20 40** 00 cf 06 **c2 25**
B730> **80** 00 1f 50 **c2 2a 80** 00 1f 70 **00** ff ff ff ff ff

Figure 9. Sample System Control Data

Note that this example contains two five-byte transmissions and the remainder are three-byte transmissions. Three-byte transmissions are useful as unchanged control, and band information need not be repeated. The displays will cycle through '075', '090', 'C02', 'C03', 'C04', 'C05', 'C06', '150', and '170' which is a mix of frequency (in MHz) and VHF channel displays. The lower four-bits of the first display value (set to f) are ignored since they are unconnected.

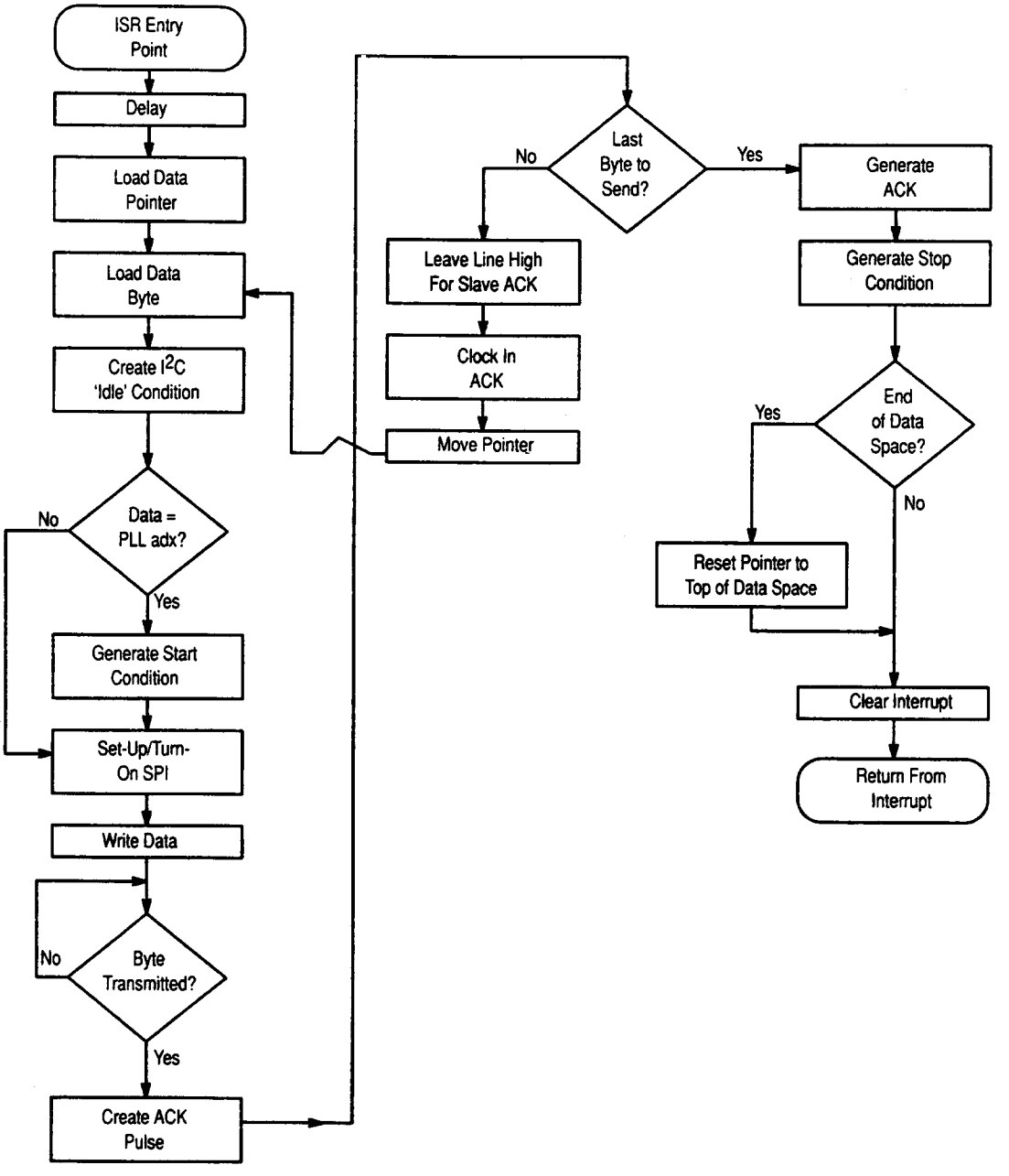

Figure 10. ISR Flow Diagram

Figure 11 is a picture of the first byte of a transmission (the PLL address). Note that the start condition is generated at the scope trigger point and the bit stream 11000010 ($C2) is clocked in on rising edges. After the eighth clock pulse, the data line is released by the MCU and quickly acknowledged (pulled low) by the PLL chip. Refer back to Figure 2.

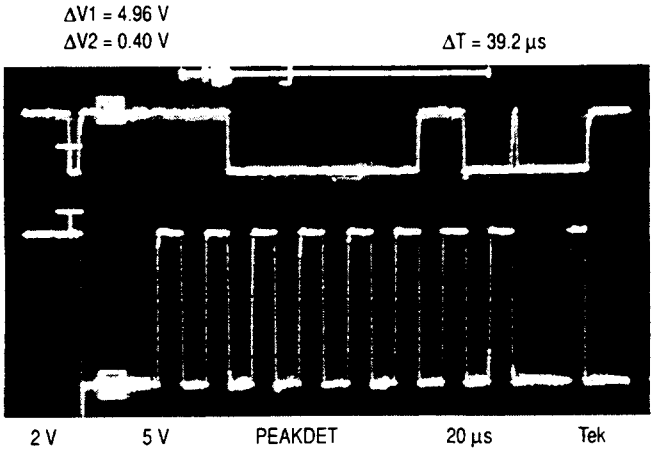

Figure 11. PLL Address Transmission

If the PLL were not responding, the data line would have remained high rather than looking like a spike. This acknowledge is clocked in and the next byte is ready for transmission. Figure 12 illustrates this by showing a full three-byte transmission that updates the PLL tuning frequency.

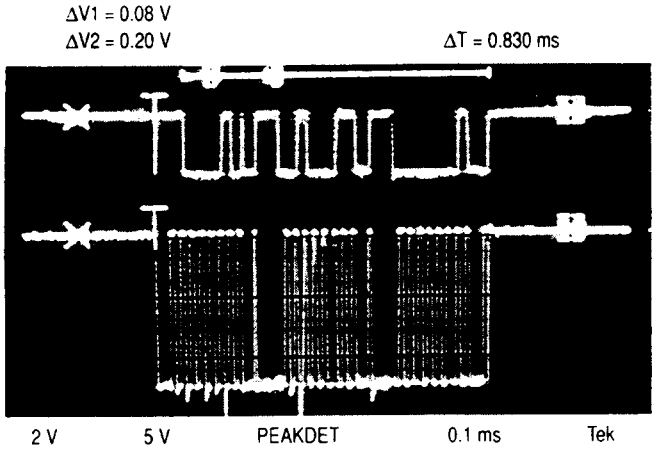

Figure 12. Three Byte Data Transmission

Interrupt Circuitry Implementation

The interrupt circuit (Figure 13) is designed as a simple debounced momentary pushbutton switch. The switch must have a normally open (N/O) and a normally closed (N/C) contact. The output of the circuit is normally at V_{CC} (+5.0 V). When the button is pushed the output goes low. It comes back high when the button is released. The cross-coupled NAND gates eliminate the effect of switch bounce.

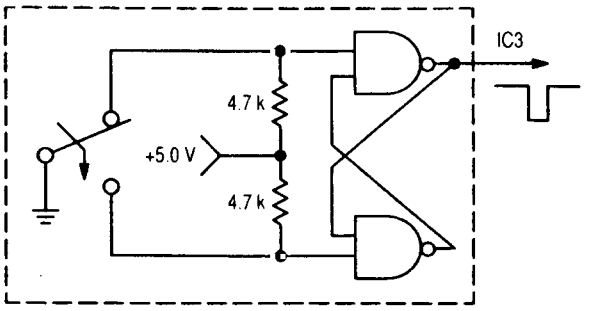

Figure 13. Pushbutton Interrupt Circuit

This will provide a clean low-going pulse to trigger one of the controller's edge-sensitive interrupts (IC3). IC3 is programmed to respond to the rising edge of the pulse to facilitate further debouncing in software.

Frequency/Channel Display Implementation

The display is implemented using three seven-segment (common cathode) LEDs. They are driven by parallel ports (A and B) of the controller in the ISR. These ports send the display information to the hexadecimal-to-seven segment decoders (MC14495-1). The STRB output from the controller is pulsed low each time data is written to Port B and is used to latch the decoders.

Display information is programmed in data space as shown in Figure 9. Outputs are done in the ISR to Port A (lines 147–148) and then to Port B (lines 150–151), and are done in this order because a write to Port B causes the STRB decoder enable pulse. Figure 14 shows the frequency display circuit.

The MC14495-1 is a hexadecimal-to-seven segment Latch/Decoder Driver. It is an improved version of the MC14495 with CMOS input levels and decreased propagation delays. This permits them to be operated directly from the limited duration pulse (STRB) generated by the MCU. The MC14495-1 has internal series output resistors (typically 290 Ω) allowing direct connection to a common cathode LED display.

SUMMARY

This application note should serve as a reference for using an MC44802A for various tuning applications. It is not intended as a replacement for the MC44802A Data Sheet nor the *MC68HC11 Reference Manual*. Its intention is to help bring these tools together to build a working system.

Bibliography

(1) MC44802A Data Sheet
(2) MC1648 Data Sheet
(3) M68HC11E9 Data Sheet
(4) M68HC11EVBU/AD1
(5) MC14495-1 Data Sheet

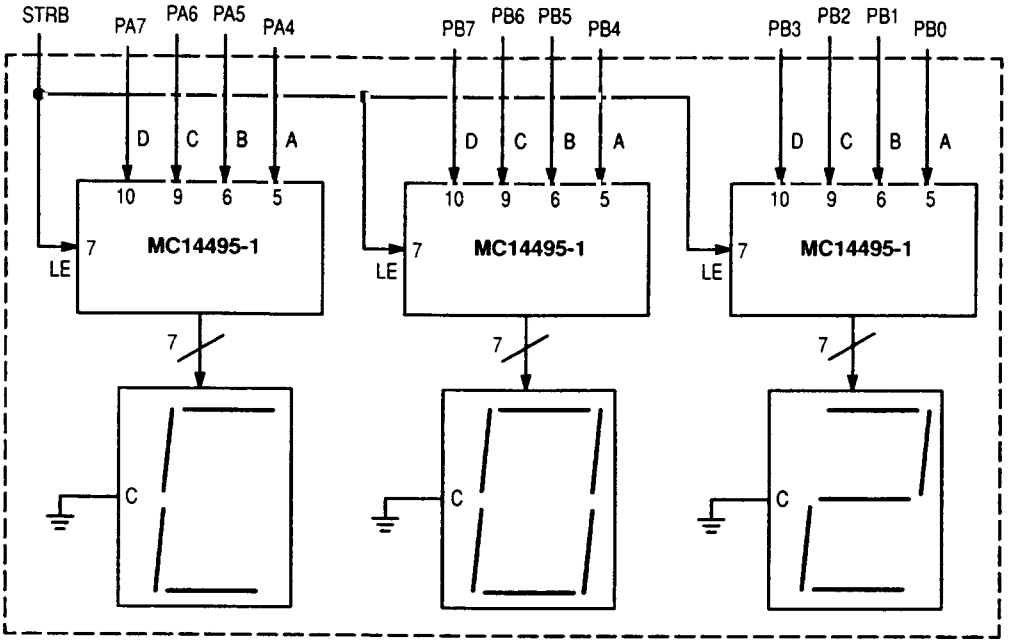

Figure 14. Three Digit Display Circuit

APPENDIX 1 — Microprogramming Basics/ Program Listing

The M68HC11 EVBU (Universal Evaluation Board) provides a friendly environment for developing an HC11 system. Programming is a three step process which includes writing software, assembling it, and downloading it to the MCU.

Writing/Modifying Software

Software should be created as a text file (e.g., program.asm) following the format of HC11 assembly commands. Full description of each command can be found in the *M68HC11 Reference Manual.*

Program Assembly

Once a program has been written, it is run through an assembler. This program will generate the necessary object code and, if desired, a listing file. The object file (xxxx.list) is then downloaded into the HC11:

program.list — listing file,
program.s19 — file to be downloaded.

Downloading/Debugging

Performance of the software and hardware should be evaluated with the help of a personal computer (Macintosh or a PC compatible) and a terminal emulation package such as Freeterm or Kermit. This program allows communication between the EVBU and computer.

APPENDIX 2 — Program Listing

```
0001                    * Motorola SPS — Bipolar Analog IC Division
0002                    * Written by Paul Brownlee
0003
0004                    * This M68HC11 code provides control bytes to operate
0005                    * an MC44802A (Motorola PLL Tuning Circuit) via I2C protocol
0006                    * The bytes are to be determined by the user and placed in
0007                    * memory starting with location B700 (see technical data sheet
0008                    * for control byte information).
0009
0010                    * Communication is achieved using the HC11's Synchronous Serial
0011                    * Peripheral Interface (SPI) to generate both the clock and data
0012                    * signals. The main program is a short monitor loop. Output is
0013                    * implemented as an interrupt service routine for the edge
0014                    * triggered interrupt IC3. Thus, the location to the routine,
0015                    * B640, must be entered in user RAM as a jump destination for the IC3
0016                    * service routine. The interrupt can then be implemented as a
0017                    * simple debounced switch.
0018
0019                        * REFERENCED TO X-OFFSET ($1000)
0020 0000               PORTA   EQU   $00     PORT A DATA REGISTER
0021 0004               PORTB   EQU   $04     PORT B DATA REGISTER
0022 0002               PIOC    EQU   $02     PARALLEL I/O CONTROL
0023 0026               PACTL   EQU   $26     PULSE ACC CNTRL REG (PORT A)
0024 0008               PORTD   EQU   $08     PORT D DATA REGISTER
0025 0028               SPCR    EQU   $28     SPI CONTROL REGISTER
0026 002a               SPDR    EQU   $2A     SPI DATA REGISTER
0027 0029               SPSR    EQU   $29     SPI STATUS REGISTER
0028 0009               DDRD    EQU   $09     PORT D DATA DIRECTION REGISTER
0029 0022               TMSK1   EQU   $22     REGISTER FOR INPUT CAPTURE ENABLE
0030 0023               TFLG1   EQU   $23     REGISTER FOR INPUT CAPTURE STATUS
0031 0021               TCTL2   EQU   $21     REGISTER FOR INPUT CAPTURE CONTROL
0032                    * REFERENCED TO Y-OFFSET (STARTS AT $B700)
0033 0000               DATA    EQU   $00     DATA SPACE (REL DATA POINTER)
0034 0001               NEXTD   EQU   $01     NEXT DATA BYTE POINTER
0035                    * REFERENCED TO 0000
0036 0000               YSTOR   EQU   $0000   RAM LOC FOR CONTROL DATA
0037 00e2               IC3JMP  EQU   $E2     THE LOCATION FOR IC3 JUMP INST
0038 00e3               IC3JMP1 EQU   $E3     LOC. TO PLACE THE JMP ADX
0039
0040
0041                    ************************** MAIN PROGRAM **************************
0042 b600               ORG $B600
0043 b600 ce 10 00              LDX   #$1000        * BASE FOR CONTROL REGISTERS
0044
0045                    * PORT A SET-UP (FOR HIGH ORDER 7 SEG DISPLAY OUTPUT)
0046 b603 a6 26                 LDAA  PACTL,X       * SET PORTA, BIT 7 TO
0047 b605 8a 80                 ORAA  #$80          * AN OUTPUT PORT
0048 b607 a7 26                 STAA  PACTL,X
0049
0050                    * PORT B SET-UP (FOR 2 LOW ORDER 7 SEG DISPLAY OUTPUTS)
0051 b609 1d 02 ff              BCLR  PIOC,X $FF    * SIMPLE HANDSHAKE MODE
0052
0053                    * TEST OUTPUTS
0054 b60c 86 af                 LDAA  #$AF          * PUT AN 'A' IN THE HIGH
0055 b60e a7 00                 STAA  PORTA,X       * HEX DIGIT
0056 b610 86 bc                 LDAA  #$BC          * AND A 'BC' IN THE LOW
```

```
0057 b612 a7 04                 STAA    PORTB,X         * FOR LED DISPLAYS
0058
0059 b614 18 ce b7 00           LDY     #$B700          * SET MEMORY POINTER
0060 b618 18 df 00              STY     YSTOR
0061
0062                    * INITIALIZE USER STACK POINTER
0063 b61b 8e 00 ff              LDS     #$FF            * STACK STARTS AT $FF WHICH
0064
0065                    * INTERRUPT PREPARATIONS
0066 b61e 86 7e                 LDA     #$7E            * OPCODE FOR JMP INST
0067 b620 97 e2                 STAA    IC3JMP          * LOADED INTO RAM
0068 b622 cc b6 40              LDD     #$B640          * SET THE JUMP LOCATION FOR
0069 b625 dd e3                 STD     IC3JMP1         * THE INT SERVICE ROUTINE
0070 b627 86 01                 LDAA    #$01            * INPUT CAPTURE (IC3) SET FOR
0071 b629 a7 21                 STAA    TCTL2,X         * RISING EDGE
0072 b62b 1c 22 01              BSET    TMSK1,X $01     * ENABLE THE IC3
0073 b62e 0e                    CLI                     * ENABLE ALL NON-MASKED
                                                          INTERRUPTS
0074
0075                    * MAIN PROGRAM DO NOTHING LOOP
0076 b62f               MONITOR         EQU             *
0077 b62f 01                    NOP                     SIT HERE AND DO NOTHING UNTIL
0078 b630 cf                    STOP                    * SAVE POWER IN STANDBY MODE
0079 b631 20 fc                 BRA     MONITOR         * INTERRUPT
0080
0081
0082
0083                    ********** INTERRUPT SERVICE ROUTINE **********
0084 b640               ORG $B640
0085
0086 b640               START           EQU             *
0087 b640 86 64                 LDAA    #100
0088 b642 18 ce 03 e8   OUTERD  LDY     #1000           * DELAY FOR SOFTWARE
0089 b646 18 09         DELAY   DEY                     * DEBOUNCING OF
0090 b648 26 fc                 BNE     DELAY           * INTERRUPT CIRCUIT
0091 b64a 4a                    DECA
0092 b64b 26 f5                 BNE     OUTERD
0093
0094 b64d 18 de 00              LDY     YSTOR           * LOAD POINTER
0095 b650 1c 08 10              BSET    PORTD,X $10     * SET D BIT 4 HIGH (IDLE)
0096
0097                    * THE REMAINING LOOP IS EXECUTED AS MANY TIMES AS THERE ARE
0098                    * BYTES TO BE OUTPUTTED. IT STARTS AT B700 (OR WHEREVER IT LEFT
0099                    * OFF ON PREVIOUS INTERRUPT HANDLED) AND OUTPUTS UNTIL A NULL
0100                    * BYTE (00) IS FOUND (00 IS NOT OUTPUTTED). THE NEXT TWO BYTES
0101                    * ARE DISPLAYED AND THE POINTER UPDATED.
0102
0103 b653               LOOP            EQU             *
0104 b653 18 e6 00              LDAB    DATA,Y          * LOAD THE PRESENT BYTE
0105 b656 1d 28 40              BCLR    SPCR,X $40      * DISABLE SPI
0106 b659 1c 08 08              BSET    PORTD,X $08     * SET D BIT 3 HIGH (IDLE)
0107 b65c 86 38                 LDAA    #$38            * SS=1, SCK=MOSI=1
0108 b65e a7 09                 STAA    DDRD,X
0109 b660 c1 c2                 CMPB    #$C2            * CHECK DATA TO SEE IF A
0110 b662 26 03                 BNE     NOSTART         * START CONDITION IS REQ
0111                    * (IF FIRST DATA BYTE)
0112
```

```
0113                    * This segment transfers a byte from the HC11's SPI
0114                    * to the I2C peripheral. Upon Entry, data is in Acc B.
0115                    * w_start is the entry point for sending a start bit.
0116                    * nostart is the entry point for transferring data
0117                    * without a start condition.
0118
0119 b664              W_START          EQU     *
0120 b664 1d 08 08                      BCLR    PORTD,X $08     * START CONDITION
0121 b667              NOSTART          EQU     *
0122 b667 86 73                         LDAA    #$73            * ENABLE SPI (SPE=1); MASTER
0123 b669 a7 28                         STAA    SPCR,X          * CPOL=CPHA=0; BITRATE=CLK/32
0124 b66b 1c 08 08                      BSET    PORTD,X $08     * RETURN PD3 TO IDLE STATE
0125 b66e e7 2a                         STAB    SPDR,X          * WRITE DATA
0126 b670 a6 29        WAIT             LDAA    SPSR,X          * WAIT FOR END OF XMISSION
0127 b672 2a fc                         BPL     WAIT            * IF NOT, WAIT
0128                                                            *
0129 b674 1d 08 10                      BCLR    PORTD,X $10     * LEAVE SCLK (PD4) LOW
0130 b677 a6 28                         LDAA    SPCR,X          * CREATE ACK PULSE
0131 b679 84 bf                         ANDA    #$BF            * CLEAR SPE, DISABLE SPI
0132 b67b a7 28                         STAA    SPCR,X          * CAUSES PD4 (SDA) TO GO HIGH
0133
0134 b67d 18 6d 01                      TST     NEXTD,Y         * TEST NEXT BYTE, IF <> 0
0135 b680 26 3a                         BNE     HI_ACK          * SLAVE GENRTS ACK (LOW)
0136
0137 b682 1d 08 08     LO_ACK           BCLR    PORTD,X $08     * ELSE, CLEAR ACK BIT
0138 b685 1c 08 10                      BSET    PORTD,X $10     * GEN ACK CLOCK
0139 b688 21 f8                         BRN     LO_ACK          * INSURE PULSE WIDTH
0140 b68a 1d 08 10                      BCLR    PORTD,X $10     * CLOCK LOW
0141 b68d 1c 08 10                      BSET    PORTD,X $10     * GEN STOP
0142 b690 1c 08 08                      BSET    PORTD,X $08     * CONDITION
0143
0144 b693 18 08                         INY
0145 b695 18 08                         INY                     * PNT TO FREQ VALU
0146 b697 1d 02 ff                      BCLR    PIOC,X $ff      * SIMPLE HANDSHAKE MODE
0147 b69a 18 a6 00                      LDAA    DATA,Y          * LOAD MSB OF FREQ VAL
0148 b69d a7 00                         STAA    PORTA,X         * AND OUTPUT IT
0149 b69f 18 08                         INY                     * MOVE POINTER
0150 b6a1 18 a6 00                      LDAA    DATA,Y          * LOAD 2 LS DIGITS
0151 b6a4 a7 04                         STAA    PORTB,X         * AND OUTPUT THOSE
0152
0153 b6a6 18 08                         INY                     * POINT TO NEXT GROUP
0154 b6a8 18 df 00                      STY     YSTOR           * SAVE NEW POINTER
0155 b6ab 18 6d 00                      TST     DATA,Y          * CHECK FOR LAST GROUP
0156 b6ae 26 07                         BNE     MODATA          * IF NOT, KEEP YSTOR
0157 b6b0 18 ce b7 00  SETPTR           LDY     #$B700          * ELSE RESET POINTER
0158 b6b4 18 df 00                      STY     YSTOR           * TO TOP OF DATA
0159
0160 b6b7 86 01        MODATA           LDAA    #$01
0161 b6b9 a7 23                         STAA    TFLG1,X         * CLEAR INTERRUPT
0162 b6bb 3b                            RTI                     * STOP SERVICE OF OUTPUT
0163
0164
0165
0166 b6bc 1c 08 10     HI_ACK           BSET    PORTD,X $10     * GENERATE ACK CLOCK
0167 b6bf a6 08                         LDAA    PORTD,X         * CHECK FOR SLAVE ACK
0168 b6c1 84 04                         ANDA    #$04            * BEING A LOW BIT 3
0169 b6c3 26 09                         BNE     ERROR           * IF NOT, BRANCH TO ERROR
0170 b6c5 21 f5                         BRN     HI_ACK          * ENSURE CLK PULSE WIDTH
```

APPENDIX 2 — Program Listing (continued)

```
0171 b6c7 1d 08 10          BCLR    PORTD,X $10     * BCLR 4, PORTD
0172 b6ca 18 08             INY                     * POINT TO NEXT DATA BYTE
0173 b6cc 20 85             BRA     LOOP
0174
0175 b6ce 86 ee     ERROR   LDAA    #$EE            * PRINT OUT AN 'EEE'
0176 b6d0 a7 00             STAA    PORTA,X         * TO INDICATE THAT THE
0177 b6d2 a7 04             STAA    PORTB,C         * SLAVE DIDN'T ACK
0178 b6d4 7e b6 b0          JMP     SETPTR          * END XMISSION ATTEMPT
0179
0180
```


How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

Customer Focus Center: 1–800–521–6274

Mfax™: RMFAX0@email.sps.mot.com – TOUCHTONE 1–602–244–6609
Motorola Fax Back System – US & Canada ONLY 1–800–774–1848
– http://sps.motorola.com/mfax/

HOME PAGE: http://motorola.com/sps/

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 4–32–1,
Nishi–Gotanda, Shinagawa–ku, Tokyo 141, Japan. 81–3–5487–8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,
51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852–26629298

MOTOROLA

31850-20 PRINTED IN USA 9/97 IMPERIAL LITHO 100

AN1122/D