Order this document
by AN1833/D

AN1833

A Low-Power Wireless Remote Control Transmitter and Receiver

By C. William Dawson
Semiconductor Products Sector, Woburn, MA
Harry Swanson
Semiconductor Products Sector, Tempe, AZ

Introduction

A low-cost solution for an OOK (on-off keying) transmitter and receiver in the periodic operation part 15 band from 260 to 470 MHz is described in this application note.

The receiver is designed to track and lock to the carrier of a SAW (surface acoustical wave) resonator or crystal-controlled PLL transmitter. The simplest form of ASK (amplitude shift keying) is OOK in which the carrier is simply switched on and off by the PCM (pulse code modulation) waveform. Generally, a low-cost OOK transmitter consists of a SAW resonator and a transistor used as a Colpitts oscillator.

Frequency stability is only fair, compared to a crystal-controlled source, offering excellent frequency stability. Because of poor frequency stability due to time and temperature effects, the frequency of a 300-MHz SAW resonator oscillator typically shifts ±150 kHz. A standard OOK receiver is designed with a wideband response to capture the drifted transmitter at the expense of sensitivity and selectivity.

The OOK receiver and transmitter presented in this application note offer greater range of operation and more robust data recovery due to improved receiver sensitivity and the receiver's ability to tune automatically to the received carrier from the low-power crystal-controlled transmitter. The transmitter has high-frequency stability and low-harmonics power from unintentional radiators.

NOTE: *Throughout this document, numbers appearing in square brackets ([1], [2], etc.) refer to the information in* ***References****.*

Application Overview

In the United States, the Federal Communications Commission (FCC) has allocated the 260- to 470-MHz band for periodic part 15 operation. The FCC regulations are found in the *Code of Federal Regulations* (CFR), title 47, part 15 (paragraph 15.231).

NOTE: *It is strongly recommended that this document be reviewed before pursuing an application in this band.*

In the United States, most of the consumer applications are regulated under part 15, which covers non-licensed intentional radiators. CFR title 47 [1] can be found at most libraries in the reference section or may be obtained from the U.S. Government Printing Office or the Federal Communications Commission's Web site.

Periodic operation refers to the amount of time a system is active. A manually operated transmitter will employ a switch that will deactivate the transmitter automatically within five seconds. Likewise, a transmitter that is activated automatically will cease transmission within five seconds after activation. Periodic transmissions at regular intervals are not allowed. However, polling to determine the integrity of the transmitters used in security or safety applications is allowed if the periodic rate of transmission does not exceed one transmission of more than one second duration per hour for each transmitter in the system.

The field strength of emissions from intentional radiators from 260 to 470 MHz is limited to 12,500 microvolts per meter measured at three meters. At band edges, the tighter limit of 3750 microvolts per meter applies. Field strength of spurious emissions are 1/10 of the fundamental frequency of the intentional radiator. The bandwidth of the emission shall not be wider than 0.25 percent of the center frequency. Bandwidth is determined at the points 20 dB down from the modulated carrier.

Although continuous transmissions such as voice, video, and data are not permitted, the transmission of recognition codes is allowed. These codes are used to identify a sensor or switch that is activated or to identify the particular component as being part of the system. The intentional radiator is restricted to the transmission of control signals such as those used with security alarm systems, door openers, keyless entry, remote switches, and remote sensing/telemetry, etc.

The general system may consist of a number of transmitters used to provide control signals from remote sites. The receiver demodulates the signal and recognizes the remote sites; then the system may initiate a response in several ways such as sounding an alarm, making an automatic distress call, turning on a light, opening a door, and turning on sprinklers, etc. Each site may use a coded address that would be broadcast to indicate the intended receiver. Some applications are more critical than others, such as a security alarm or distress calling systems. It is essential that the system is dependable.

The receiver has the job of extracting and reproducing information from the modulated RF (radio frequency) signal that has been corrupted by noise in the channel. Consistent recovery and error-free replication of the modulating signal requires a receiver which has reliable performance.

The two main classes of radio frequency receivers are:

- Tuned radio frequency (TRF)
- Superheterodyne receivers

The TRF receiver may consist of several cascaded high gain RF amplifier and tunable bandpass filter stages followed by an appropriate detector (envelope detector, product detector, FM (frequency modulation) detector, etc.). A reliable TRF receiver at UHF (ultra-high frequency) frequencies is difficult to achieve because appreciable parasitic feedback between the output and input of the high gain RF amplifier chain creates oscillation at the center frequency. Reducing the gain in the RF amplifier chain may prevent oscillations but lower gain will reduce the sensitivity of the receiver. These receivers generally use discrete semiconductors since integrated circuits are not available for low-voltage, low-power applications.

The more reliable and stable receiver described in this application note is a superheterodyne type receiver. In this receiver, the RF carrier is filtered and amplified using an RF preamplifier and is down converted using a mixer and local oscillator (LO) to a convenient, lower frequency called the intermediate frequency (IF). The IF signal is amplified and filtered and then demodulated by using an appropriate detector.

The OOK receiver shown in **Figure 1a** has many advantages over a TRF receiver. It utilizes two monolithic integrated circuits (IC), the MC13144 [2] as a cascade low-noise amplifier (LNA), and the MC13158 [3] as the main building block components of the receiver. This receiver offers very stable operation with more than 115 dB of stable gain. Typical sensitivity is –100 dBm which allows for a greater operating distance between transmitter and receiver. By utilizing a varactor-controlled LO locked by an external automatic frequency control (AFC) circuit, the receiver has the ability to lock onto the transmitter. Thus, this receiver system has no need to use a more expensive crystal oscillator for the LO.

The OOK transmitter (**Figure 1b**) is implemented by using a versatile 1-chip, fixed frequency PLL (phase-locked loop) system IC, the MC13176D [4] which operates under an inexpensive crystal reference source from a microcontroller unit, the MC68HC705J1A [5].

Baseband processing functions are performed by MCUs in both the remote receiver and handheld transmitter.

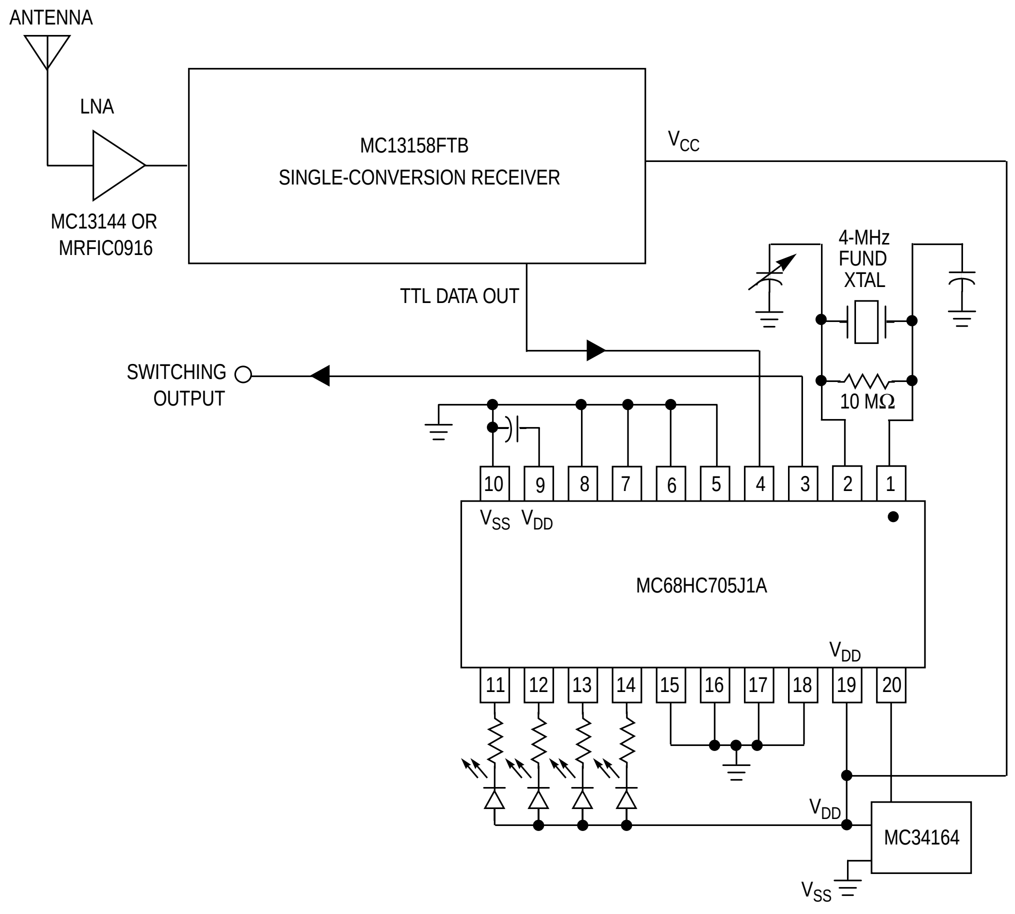


Figure 1a. OOK Receiver Block Diagram

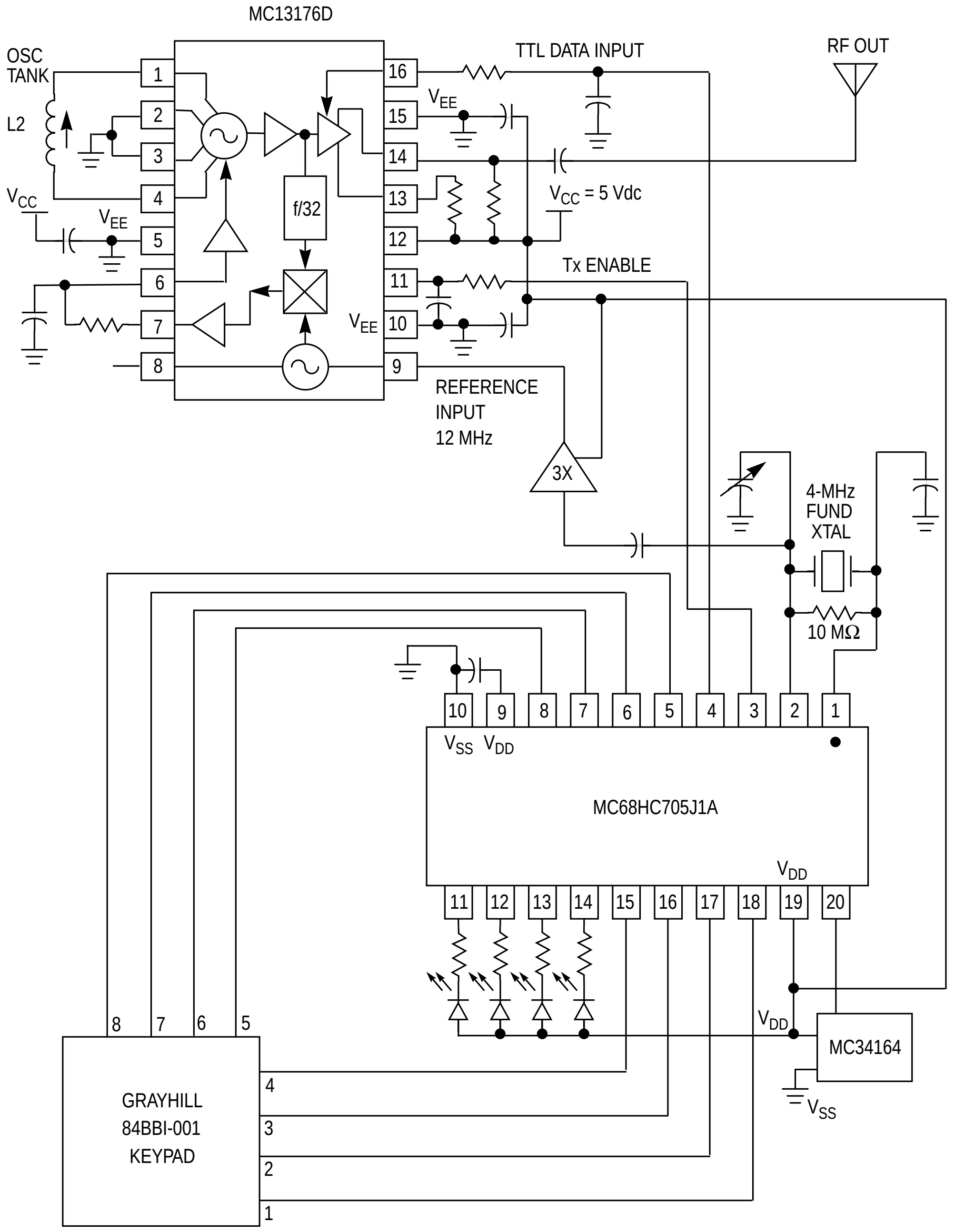


Figure 1b. OOK Transmitter Block Diagram

Description of the MC13158

The MC13158 is an integrated, single-conversion, wideband FM receiver system intended for digital and analog modulation formats. It was developed as a high-performance IC to be used in battery-operated radio systems, as it can be operated below 2 Vdc. The MC13158 is available in a 32-lead, low-profile quad flat package (LQFP), which offers small size layout and stable performance.

While it is an ideal receiver for the recovery of GFSK (Gaussian-filtered shift keying), FSK (frequency-shift keying), or FM (frequency modulation) signals, the circuit detailed in this application note shows how the MC13158 system can be reconfigured to recover OOK signals. Further details about the MC13158 not covered in this application note are found in comprehensive device documentation [2].

The MC13158 functional blocks include:

- Differential input mixer
- 40-dB IF amplifier
- 55-dB limiter amplifier
- Externally tuned FM quadrature detector
- Received signal strength indicator (RSSI)
- Data slicer comparator
- Carrier detection

The mixer has differential RF inputs allowing it to be used in balanced and single-ended configurations. Its single-ended input impedance is 1 kΩ in parallel with 4 picoFarads. Access is provided to the base and emitter of the MC13158 internal oscillator transistor for attaching the oscillator resonant tank circuit. The oscillator transistor internally drives the mixer and can be operated as an LC oscillator to greater than 450 MHz. The application circuit (see **Figure 2**) uses a varactor-controlled oscillator (VCO) to provide LO drive at approximately 395 MHz.

The IF chain is split into two sections to allow IF bandpass filtering in two places:

1. Between the mixer and IF amplifier
2. Between the IF and limiter amplifiers

Output impedance driving each filter port is 330 Ω, which is ideal for common ceramic IF filters centered at 10.7 MHz. The IF and limiter amplifier's internal inputs are 330 Ω to ground to match the 10.7-MHz ceramic filters.

The limiter is coupled internally to the quadrature detector inputs, and it is also pinned out. An external RLC (resistance-inductance-capacitance) tank provides selection of the demodulator tuned frequency and detection bandwidth. The detected signal is brought out by the inverting amplifier buffer. Its output amplitude swing is set by an external feedback resistor from the output (pin 17) to the input of the inverting amplifier (pin 15). It is combined with a second resistor from the input to the negative supply (pin 16) to select the output dc level.

The RSSI (received signal strength indicator) output provides an output current that is proportionate to the logarithm of the receiver input voltage.

NOTE: *Externally load it with a resistor to V_{EE}.*

Its typical transfer gain is 0.4 μA per dB of input drive level. The RSSI dynamic range is higher than 70 dB, which is achieved by sensing signal strength at each stage of the IF and limiter amplifiers and summing them together. The RSSI linearity is affected by the loss placed between the IF amplifier and the limiter amplifier; a total midband attenuation of 10 dB is optimal.

The data slicer comparator has more than 500 kHz of toggling capability. Its inverting input (pin 18) and non-inverting input (pin 20) have back-to-back diodes across them. The recovered signal may be dc coupled to one data slicer input while the other one is ac grounded; it is allowed to swing ±1 V_{BE} about the dc level to the comparator input.

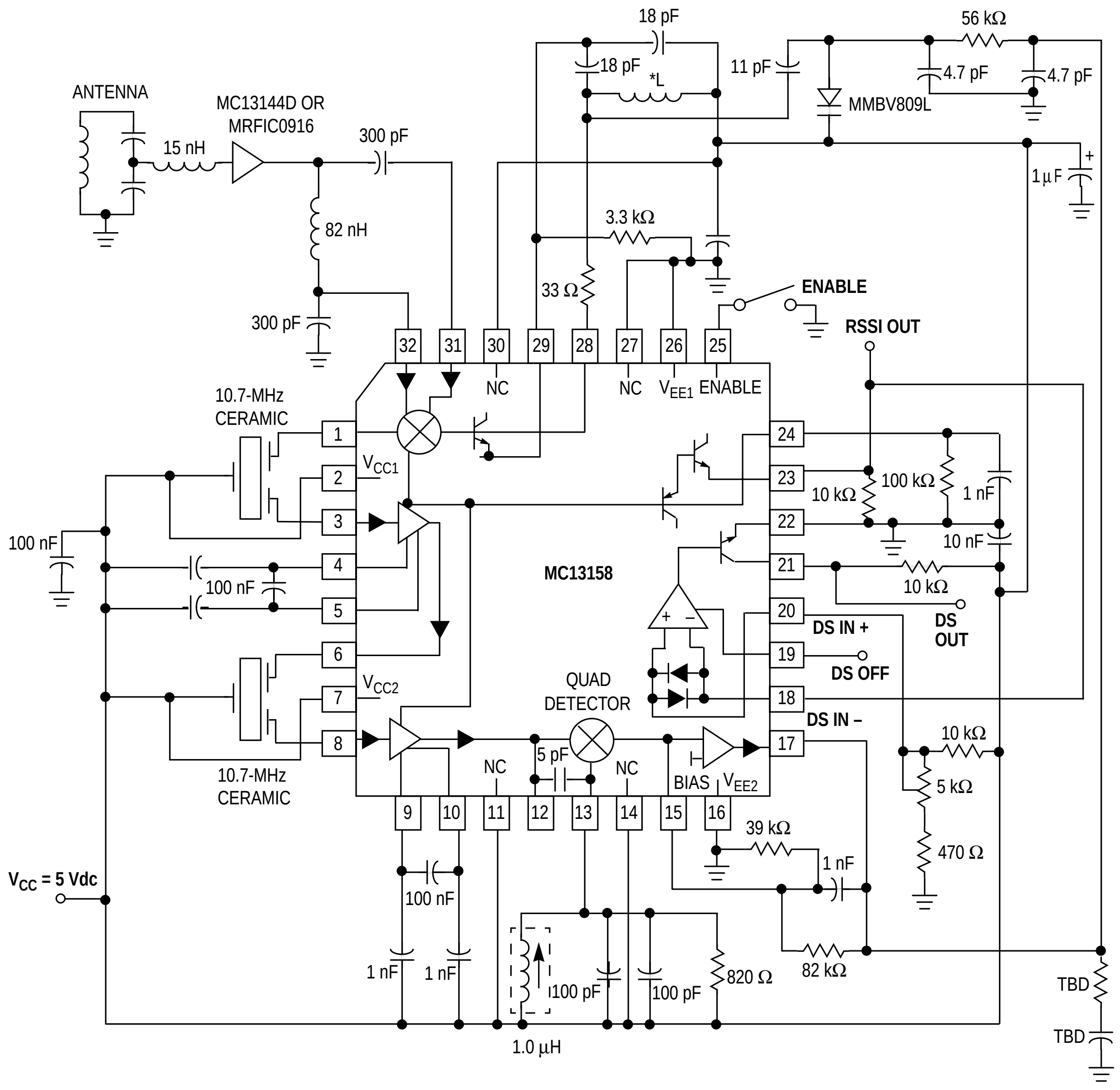
The polarity of the output data depends on which input is referenced to V_{EE} (ground). For positive logic data output, the data signal goes to the non-inverting input. Where the reference input pin of the data slicer is capacitively grounded, the size of this capacitor and the nature of the data signal determine how faithfully the data slicer shapes up the recovered signal. A control pin is provided to shut the data slicer output off (DS off, pin 19).

Receiver Application Circuit Description/Design

The block diagram of the OOK receiver and its controller is shown in **Figure 1a**. The complete schematic of the MC13144 and MC13158 single conversion OOK receiver circuit is shown in **Figure 2**. The internal on-board transistor is used as the LO source to down convert the received 384-MHz RF carrier signal. Its VCO tuning frequency is controlled by an automatic frequency control (AFC) circuit created with the dc output from the MC13158 quadrature detector buffer amplifier and an external loop filter.

OOK demodulation is achieved by using the MC13158 fast RSSI port. The RSSI output is fed to the data slicer, which shapes the recovered RSSI data into a rail-to-rail logic swing.

In the following sections, each part of the OOK receiver is explained in more detail.



* L is 5T #28AWG enameled wire wound on a #74 drill bit.

Figure 2. Detailed OOK Receiver Circuit

External Varactor Controlled Local Oscillator

Figure 3 is a schematic of the receiver external local oscillator circuit. For easy reference, the components are labelled in the following discussion.

The LC oscillator uses the on-board transistor. The MMBV809L is a low-voltage varactor suitable for UHF applications; it is a single varactor in an SOT-23 package. This oscillator is tunable over a range of approximately 1.5 MHz (393.8 to 395.3 MHz) with the control voltage V_{VCO} from the AFC.

Figure 4 is a curve of VCO frequency versus AFC V_{VCO}. Note that the VCO frequency decreases with increasing control voltage since the varactor diode cathode is taken to V_{CC} and the control voltage is applied to the anode.

This circuit is breadboarded using a custom PC board in which the RF ground is V_{CC} and RF path lengths are minimized. High quality surface mount components were used except where otherwise specified. The absolute values of the components used will vary with layout placement and component parasitics.

In the VCO circuit, R1 is the emitter bias resistor. R2 forms the base bias circuit. C1 and C2 make up the Colpitts capacitive divider network. CS and CB are capacitive trimmers for the LC oscillator. The shunt capacitor CS across the varactor diode, MMBV809L [6], and series capacitor CB are used to adjust the tuning range by varying the capacitance of the LC oscillator circuit.

This equation shows the relationship:

1. $f = (0.159) / [\{ [[(C1)(C2)] / [(C1 + C2)] + C] * [L] \}1/2]$

where C is the equivalent capacitance of CB, CS, and CV.

The relationship for C is:

2. $C = (CS + CV)(CB) / (CB + CS + CV)$

The oscillator tuning range is controlled by ΔC resulting from the voltage variable ΔCV. The following equation by similarity to equation (2) above shows this relationship.

3. $\Delta C = (CS + \Delta CV)(CB) / (CB + CS + \Delta CV)$

The oscillator operating frequency may be estimated by the next equation, which is derived by substituting equation (2) for C in equation (1).

4. $f = (0.159) / [\{ [[(C1)(C2)] / [(C1 + C2)] + [(CS + CV)(CB) / (CB + CS + CV)] * [L] \} 1/2]$

Solving (4) for L, results in:

5. $L = (0.159)^2 / \{ [(C1)(C2)] / [(C1 + C2)] + [(CS + CV)(CB) / (CB + CS + CV)] * f^2 \}$

where C1 = C2 = 18 pF

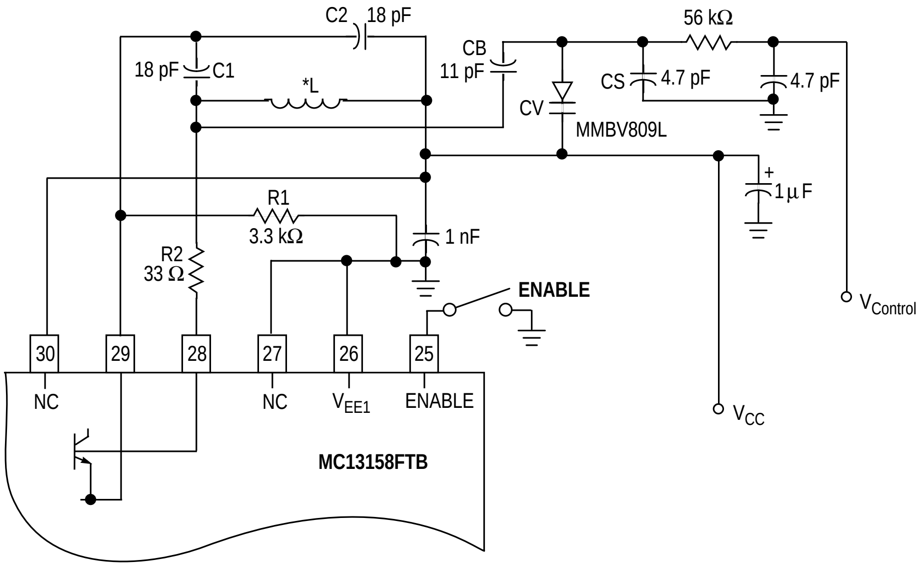
CS = 4.7 pF

CV = 5 pF (at 2.2 Vdc)

CB = 11 pF

For an oscillator center frequency of 394.7 MHz (high side LO is used), the value of L is calculated to be 11.5 nH.

For a more in-depth study, Randall Rhea discusses several oscillator circuits in his book on oscillator design [7].



* L is 5T #28AWG enameled wire wound on a #74 drill bit. The VCO is fine tuned by adjusting the spacing between windings of this air core inductor.

Figure 3. Receiver 394.7-MHz Local Oscillator

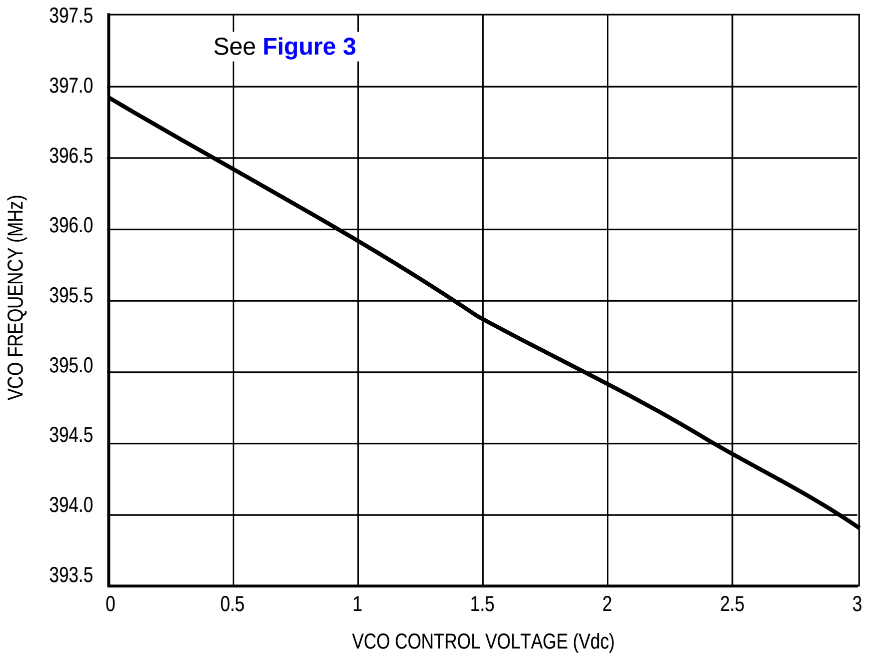


Figure 4. VCO Frequency versus VCO Control Voltage

Automatic Frequency Control

The MC13158 OOK receiver uses automatic frequency control (AFC) to adjust the VCO with respect to the RF to down convert it to the desired IF frequency. The VCO's output frequency is adjusted by the output voltage from a quadrature detector buffer amplifier circuit and the loop filter. The loop filter input is taken from the buffered output of the frequency discriminator of the MC13158. The discriminator V_{Out} versus frequency response is set by the resonant frequency and bandwidth of the external quadrature tank circuit; while the dc output component is determined by the combination of the external load feedback resistor, R17, and the resistor from inverting input to ground, R15.

The detector dc level, V17, is calculated in this equation:

$$V17 = \{ [(R15/R17) + 1] / (R15/R17)\} * V_{BE}$$

where V_{BE} is a diode drop = 0.75 Vdc.

In the application circuit, a V17 of nominally 2.2 Vdc is chosen, yielding standard resistor values for R17 of 82 kΩ and R15 of 39 kΩ.

The AFC loop gain and settling time are set by the buffer amplifier feedback resistor and the capacitor, respectively. The values of these loop characteristics are left to the designer's discretion, depending on the system requirements. Stable frequency acquisition responses within milliseconds have been achieved with the schematic values of this simple single-pole circuit.

The closed loop frequency error is 12 kHz per 100 kHz change in RF frequency. This provides:

- A bias to the discriminator output
- An offset to dc match the loop filter frequency versus voltage out response to VCO control characteristic

The AFC output (varactor control voltage) decreases with increasing IF frequency. This system was designed for high side mixer conversion (the LO is above the RF). The ground reference of the varactor is taken to V_{CC} (ac ground); thus, the LO frequency increases as the varactor control voltage decreases.

The RLC values shown will tune the tank to 10.7 MHz and allow for approximately 900-kHz AFC tuning bandwidth. **Figure 8** shows the pull-in and hold-in range of the AFC loop. An adjustable coil is used to tune the quadrature detector. The resistor sets the bandwidth by lowering the Q of the LCR tank circuit. The following relationship shows how to calculate the response of the discriminator resonant tank.

6. $Q = R / X_L$

for R = 820 Ω; $X_L = 2\pi(10.7)(1.0) = 67.2\ \Omega$

then Q ~ 12.2

7. BW 3 dB = fo / Q

thus, the 3-dB bandwidth = 10.7 MHz / 12.2 = 880 kHz

Should the overall transmit/receive system be prone to extreme frequency drift, the quadrature coil circuit bandwidth may need to be increased to provide adequate tracking range.

IF Interstage Filtering

Ceramic bandpass filters are placed between:

- Mixer output and the IF amplifier input
- IF output and the limiter input (see **Figure 2**)

They allow improved sensitivity by bandwidth, limiting the noise to the RSSI detectors, especially the limiter. The filters shown in the schematic have a 3-dB bandpass response centered at 10.7 MHz which is 280 kHz wide. Choice of IF filter bandwidth will affect the fidelity of the recovered baseband signals. As the data rate is increased, the filter bandwidth should be wider.

The filter bandwidth is affected by its source impedance and output load impedances which are 330 Ω. The MC13158 mixer output and IF amplifier input and output impedances are 330 Ω to match typical 10.7-MHz ceramic filters.

Data Recovery Circuit

The received OOK signal is converted from RF to baseband by the MC13158 RSSI function. The RSSI output sources a current to an external load connected to V_{EE} to produce a voltage. The choice of this load resistance affects the receiver performance. Obviously, the resulting RSSI gain (V_{Out}/RF_{In}) is directly proportionate to this load. When this load is large enough, the RSSI output will saturate, which will compress the high RF input end of the RSSI transfer response. A load of 100 kΩ is a good choice to obtain 70 dB of monotonic RSSI dynamic range with no compression. The load impedance chosen also affects the RSSI rise and fall times (see **Figure 5**). The RSSI output rise and fall times are more than adequate to follow signal amplitude variations to rates greater than 50 kHz.

For data rates of 10 kHz or less, 680 pF and 100 k may be connected from pin 24 to V_{EE}. These components form a low pass filter with its corner at approximately 10 kHz; this helps clean up the RSSI response under low signal conditions. Data rates up to 50 kHz have been achieved without low pass filtering.

The RSSI is dc coupled to the data slicer non-inverting input (pin 20). The data slicer inverting input terminal is referenced to V_{EE} with a voltage divider or a diode to ground. Temperature stability of this reference should be considered for receiver operating over an extended temperature range.

The data slicer can be disabled by connecting the data slicer off pin to greater than 1.5 Vdc (V_{CC} will suffice). Should the receiver system be integrated with a transmitter, this will prevent an invalid data slicer output from causing problems.

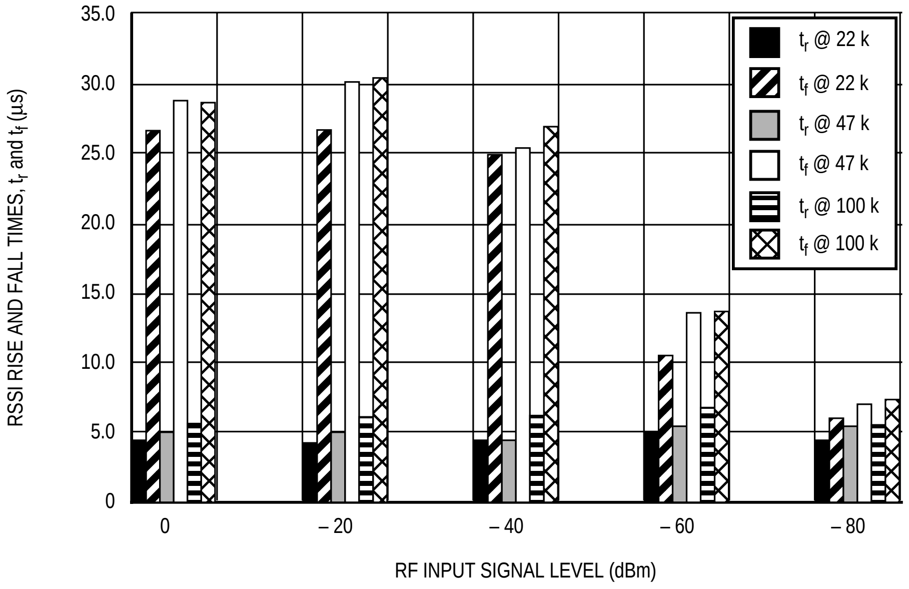


Figure 5. RSSI Output Rise and Fall Times versus RF Input Signal Level

Low-Power Transmitter

The schematic in **Figure 6** shows the handheld OOK transmitter circuit. It is comprised of the MC13176 UHF PLL transmitter, the MC68HC705J1A MCU, and the reference/tripler.

The PLL reference at pin 9 is derived from tripling the 4-MHz crystal reference from the MCU. In this tripler, a low-cost transistor, MMBT3904L [8], is used with a 7-mm shielded transformer that provides the 12-MHz resonant LC tank at the output of the transistor and also the means to easily supply dc to the collector output. The transformer is a part supplied by Toko America [9] (part number 600GCS-8519N). Twelve MHz, the third harmonic from the 4-MHz crystal, is selected by the LC resonant circuit formed by the primary windings between pins 1 and 2 of the transformer and an external capacitor of 330 pF.

The secondary output of the transformer uses a tapped capacitor transformer to provide a lower source impedance to the reference input of the MC13176 (pin 9) while forming a second LC resonant network. Thus, the tripler circuit is a double tuned LC network which provides increased selectivity of the 12-MHz signal.

The MC13176 is enabled by the MCU at pin 10 and data is input at pin 16 creating the OOK modulation. At pins 1 and 4, an external inductor is used as the resonator for the current controlled oscillator (CCO). This RF resonator must be a high Q component and adjustable to align the CCO to the reference source.

The application circuit uses a Coilcraft adjustable 7-mm shielded inductor [10] (part number 150-04J08S).

Phase noise of the MC13176 measures at –85 dBc/Hz at 10-kHz separation (**Figure 7**). This offers ample performance for a wideband system such as described in this application note.

Layout and Practical Considerations

These layout considerations are suggested:

1. Use double sided PCB in which one side is the circuit side where all surface mounted components are placed and where the interconnect and circuit traces are done; the other side is the V_{EE} ground side where leaded components are mounted and only a few necessary circuit traces are done.
2. Keep all RF traces as short as possible.
3. Avoid point-to-point wiring; it is best to run the trace on the PCB.
4. Use controlled impedance microstrip lines in high frequency portions of the circuit.
5. Use ground return paths through ample size holes to the ground side of the PCB.
6. Decouple V_{EE} where the pins of the IC contact the trace on the PC board.

In mixed signal applications, isolation of the digital and RF ground and supply lines is a must. In addition, the receiver is properly shielded and ac decoupled to eliminate interference by switching and high EMI/RFI sources.

Performance in Application Circuit

In the receiver, an input carrier level of –80 dBm allows recovery of the PCM waveform without jitter or tearing at 2 kbps (see **Figure 2**). This is improved approximately 15 dB by using a preamp and bandpass filtering before the mixer. Design of the matching network to the preamplifier should consider interfacing to an antenna which would be "application specific," depending on the size and type of antenna. A loop antenna suitable for this application is discussed in a book by Constantine Balanis [11].

The pull-in range and hold-in range are dependent on the input drive level. In general, under higher drive the hold-in and pull-in ranges are greater. **Figure 8** shows a plot of the hold-in range and pull-in range versus the RF input level. The pull-in range is consistent with the selectivity of the IF bandwidth.

The application circuit supports data rates up to 50 kHz. This has been confirmed in the lab with relaxed interstage and baseband filtering. The wider filtering is much more susceptible to interference.

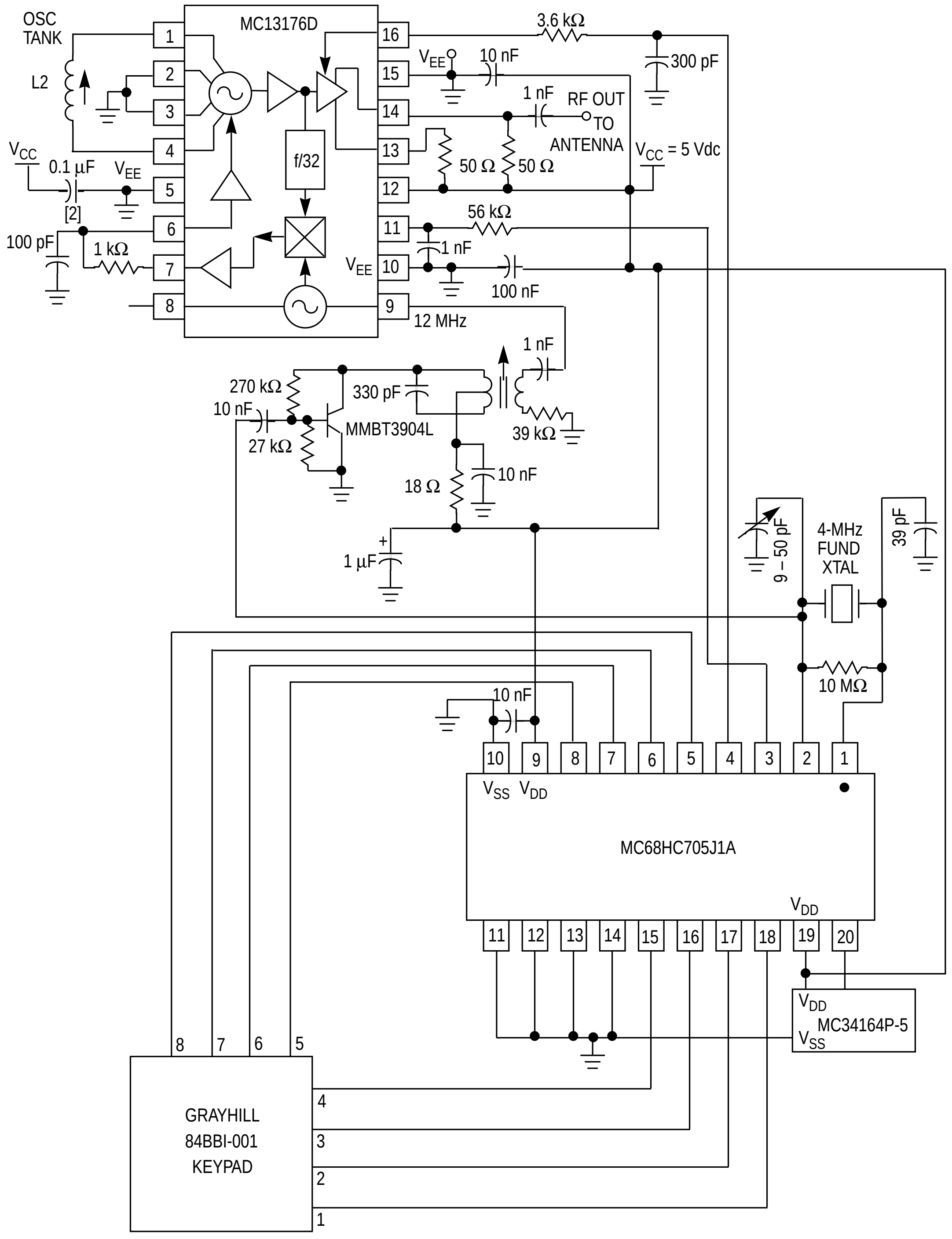


Figure 6. 384-MHz OOK Transmitter

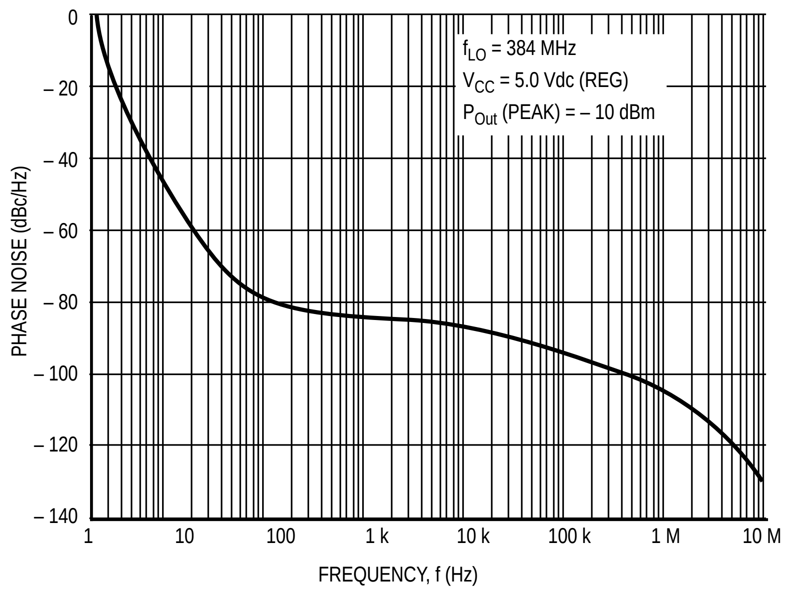


Figure 7. Phase Noise versus Frequency

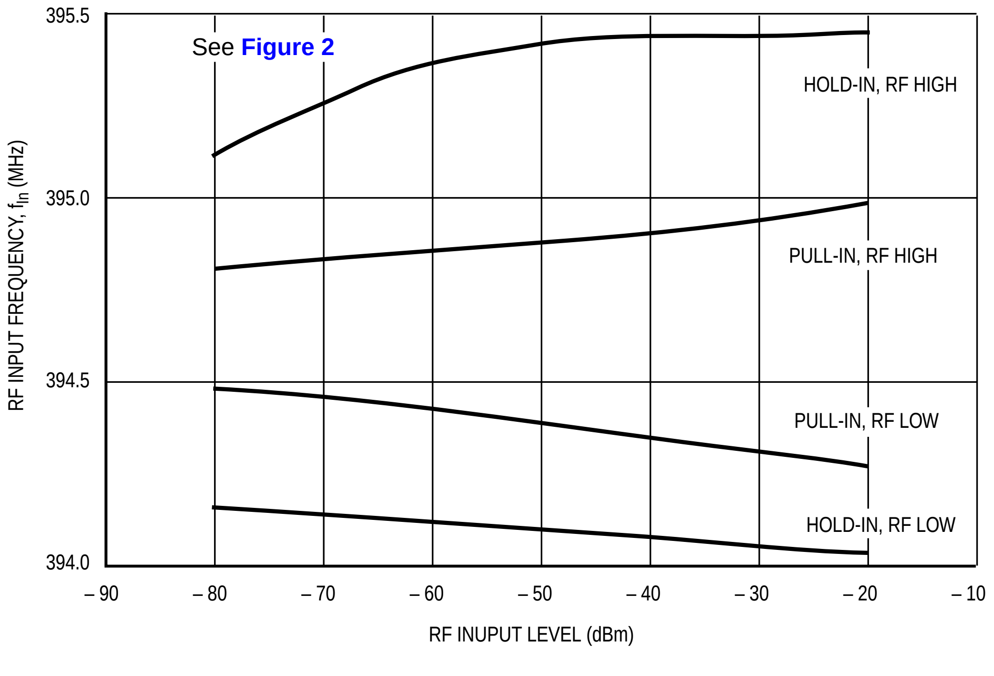


Figure 8. AFC Capability: Hold-In and Pull-In Limits versus RF Input Level

MCU Introduction

A low-power wireless remote control system has four pieces:

1. Method of initiating a request such as a button on a remote control unit
2. Sending a message from an RF transmitter
3. Receiving and validating a message
4. Determining action to be taken such as releasing a solenoid or turning on a light

The microcontroller unit (MCU) in the remote control unit decodes the button that has been pressed and sends a bitstream to an RF transmitter. The receiver acquires the RF bitstream, converts it to digital format for an MCU, which analyzes the digital data, and determines what operation is to be performed.

Microcontroller Software Overview: Initiate an Access Request

A battery-powered remote control unit waits in an extremely low-power or sleep mode until someone presses a button or series of buttons on the unit. (Low-power mode means a battery will power the unit longer.) Pressing buttons causes the MCU to wake up and begin executing an internal sequence of commands that determine what keys have been pressed. The MCU in this design is an MC68HC705J1A [5]. The keystrokes are stored by the MCU and determine the format of an access request message.

Interrupt Service Routine

When a keypad button is pressed, it asserts an input pin that triggers the execution of the interrupt service routine (ISR). The MCU clock feeding the MC13176 turns on when a keypad button is depressed. Decoding of which key is depressed is done using the routines written by David Yoder [12]. The ISR examines the port A input values when certain port B outputs are asserted.

Sending a Message

When two keypad values have been received, the MCU ensures a logic low is on the TTL (transistor-to-transistor logic) data input to the transmitter and enables the MC13176 via output pins on port B. After waiting for the transmitter to warm up, a logic 1 is presented to the TTL data inputs of the MC13176 for a 500-microsecond interval representing a “front porch." Three preamble characters (55 hex) and the two keypad values received are sent serially to the transmitter by the MC68HC705J1A using the routines written by Scott George [13]. The MCU turns off the transmitter enable pin on port B, initializes all software flags and buffers, prepares for the next keypad interrupt, and then re-enters sleep mode. The transmitter flowchart is shown in **Figure 9** and **Figure 10**.

Acquire and Validate an Access Request

The receiver activates itself when it sees a specific signal exceed a strength threshold and passes the data bits received in a digital format to an MCU input port. When the MCU sees at least a 500-microsecond logic 1 on the receiver input pin, a “shutter timer” is activated that has a timing interval equal to 8.2 milliseconds.

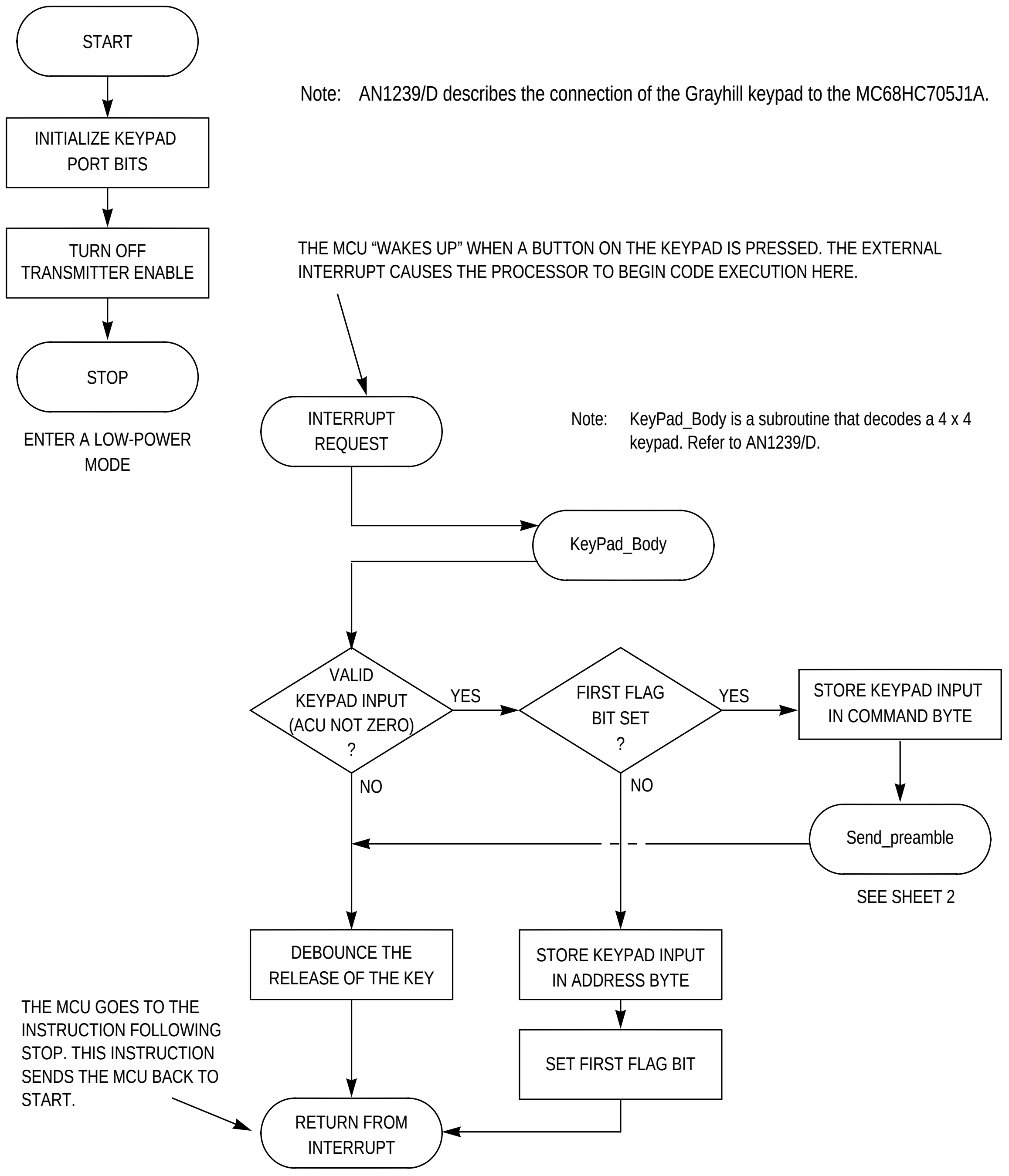
The MC68HC705J1A examines the received data, using the routines written by Scott George [13], for ASCII character sequences of:

- Start bit
- Eight bits of logic high or low data, and
- A stop bit during the “shutter timer” period

The character data is put together via several subroutines and placed in the MCU buffer memory. Data acquisition is terminated when an interrupt caused by the “shutter timer” interval expiring occurs. The receiver flowchart is shown in **Figure 11**, **Figure 12**, and **Figure 13**.

Permit or Reject Access

The receiver buffer is examined for a specific data sequence when data acquisition is terminated. If at least two of the preamble characters are received as alternating data bits of logic 1 and 0 (ASCII 55 hex) and the receiver number or a broadcast number is in the lower nibble of the address byte, then the MCU will perform actions specified in the command byte.



START
INITIALIZE KEYPAD PORT BITS
TURN OFF TRANSMITTER ENABLE
STOP
ENTER A LOW-POWER MODE
Note: AN1239/D describes the connection of the Grayhill keypad to the MC68HC705J1A.
THE MCU "WAKES UP" WHEN A BUTTON ON THE KEYPAD IS PRESSED. THE EXTERNAL INTERRUPT CAUSES THE PROCESSOR TO BEGIN CODE EXECUTION HERE.
INTERRUPT REQUEST
Note: KeyPad_Body is a subroutine that decodes a 4 x 4 keypad. Refer to AN1239/D.
KeyPad_Body
VALID KEYPAD INPUT (ACU NOT ZERO) ?
YES
NO
FIRST FLAG BIT SET ?
YES
NO
STORE KEYPAD INPUT IN COMMAND BYTE
Send_preamble
SEE SHEET 2
DEBOUNCE THE RELEASE OF THE KEY
STORE KEYPAD INPUT IN ADDRESS BYTE
SET FIRST FLAG BIT
THE MCU GOES TO THE INSTRUCTION FOLLOWING STOP. THIS INSTRUCTION SENDS THE MCU BACK TO START.
RETURN FROM INTERRUPT

Figure 9. Transmitter MCU Flowchart (Sheet 1 of 2)

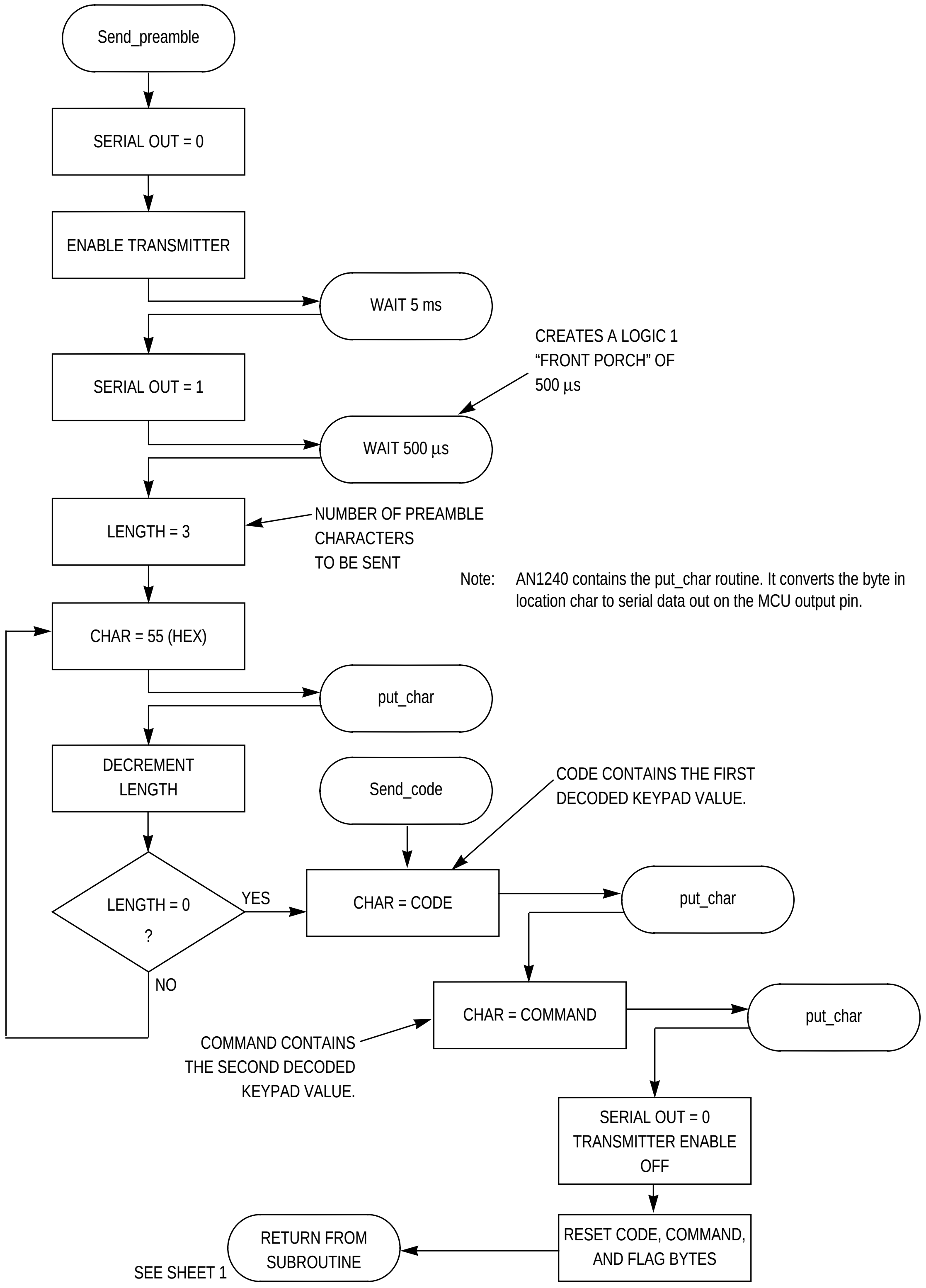

Send_preamble
SERIAL OUT = 0
ENABLE TRANSMITTER
WAIT 5 ms
SERIAL OUT = 1
CREATES A LOGIC 1 "FRONT PORCH" OF 500 µs
WAIT 500 µs
LENGTH = 3
NUMBER OF PREAMBLE CHARACTERS TO BE SENT
Note: AN1240 contains the put_char routine. It converts the byte in location char to serial data out on the MCU output pin.
CHAR = 55 (HEX)
put_char
DECREMENT LENGTH
Send_code
CODE CONTAINS THE FIRST DECODED KEYPAD VALUE.
LENGTH = 0 ?
YES
NO
CHAR = CODE
put_char
CHAR = COMMAND
put_char
COMMAND CONTAINS THE SECOND DECODED KEYPAD VALUE.
SERIAL OUT = 0 TRANSMITTER ENABLE OFF
RESET CODE, COMMAND, AND FLAG BYTES
RETURN FROM SUBROUTINE
SEE SHEET 1

Figure 10. Transmitter MCU Flowchart (Sheet 2 of 2)

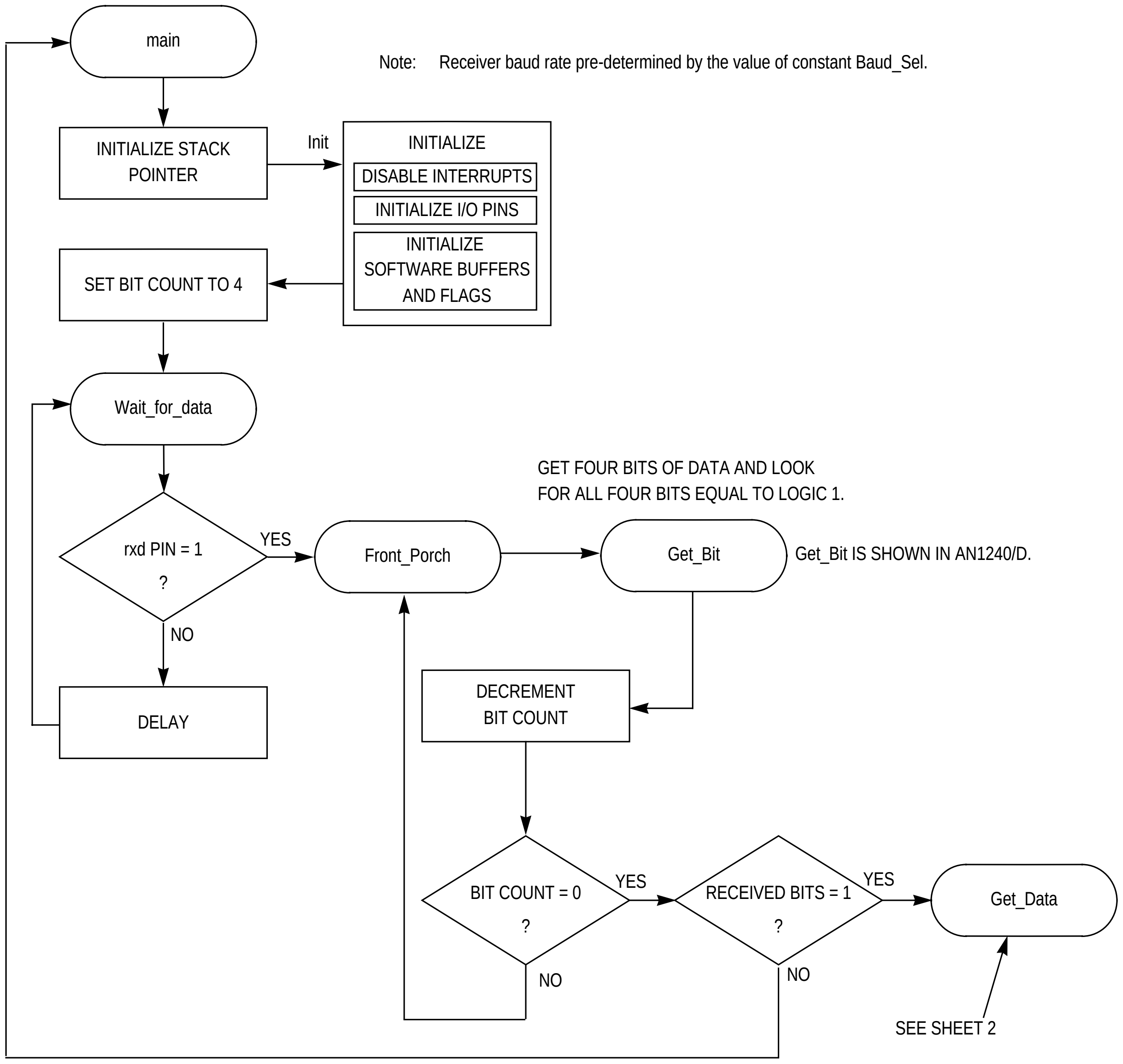


Figure 11. Receiver MCU Flowchart (Sheet 1 of 3)

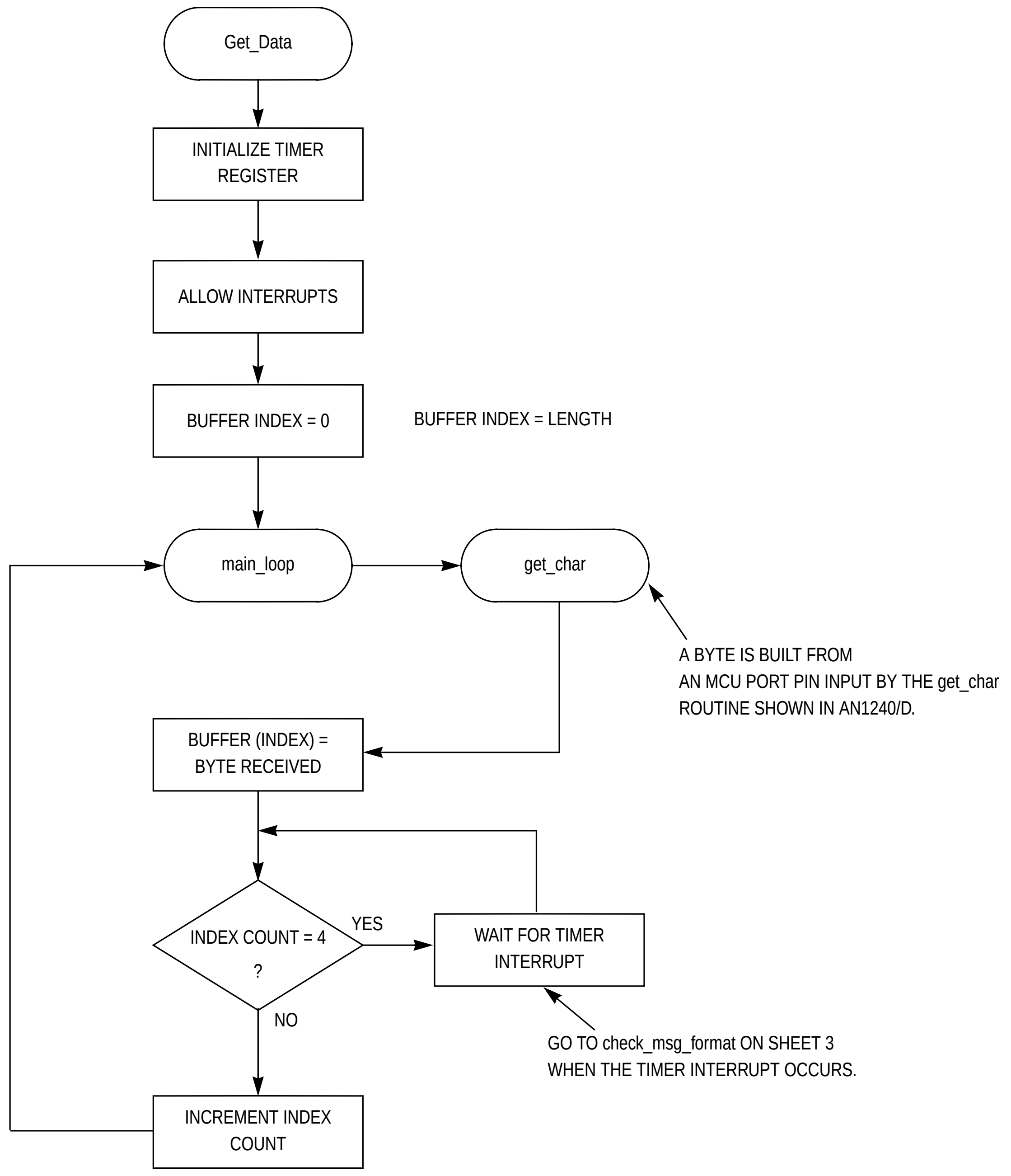
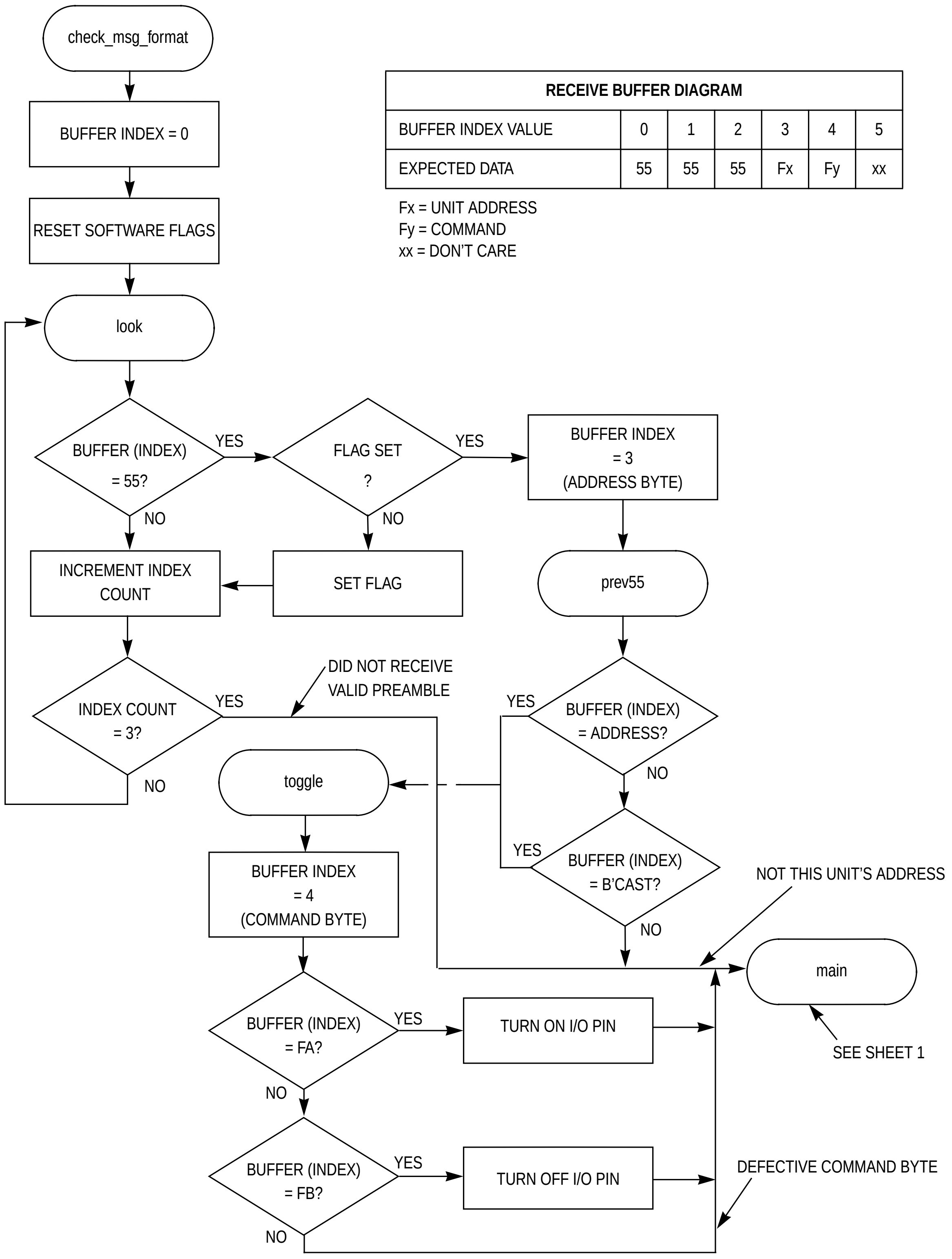

Get_Data
INITIALIZE TIMER REGISTER
ALLOW INTERRUPTS
BUFFER INDEX = 0
BUFFER INDEX = LENGTH
main_loop
get_char
A BYTE IS BUILT FROM AN MCU PORT PIN INPUT BY THE get_char ROUTINE SHOWN IN AN1240/D.
BUFFER (INDEX) = BYTE RECEIVED
INDEX COUNT = 4 ?
YES
WAIT FOR TIMER INTERRUPT
GO TO check_msg_format ON SHEET 3 WHEN THE TIMER INTERRUPT OCCURS.
NO
INCREMENT INDEX COUNT

Figure 12. Receiver MCU Flowchart (Sheet 2 of 3)



RECEIVE BUFFER DIAGRAM

BUFFER INDEX VALUE	0	1	2	3	4	5
EXPECTED DATA	55	55	55	Fx	Fy	xx

Fx = UNIT ADDRESS
Fy = COMMAND
xx = DON'T CARE

Figure 13. Receiver MCU Flowchart (Sheet 3 of 3)

Conclusions

The MC13144 and MC13158 are excellent solutions for a stable OOK receiver to fit part 15 applications such as keyless entry and security systems where high reliability and performance is essential. High performance attributes have been demonstrated and can be successfully integrated into a low-cost OOK communication system.

The varactor-controlled LC oscillator is capable of tuning over a suitable frequency range at low-supply voltage. Driving the VCO with the AFC network provides compensation for expected transmitter carrier drift. This overcomes the need for an expensive crystal-controlled transmitter/receiver system. Narrow band filters improve selectivity and assist in achieving the receiver's low sensitivity of –100 dBm. With only three ICs and a minimum number of external components, the low-cost objective is met.

The MC13176 offers a versatile UHF PLL transmitter using few external components and exceeds the performance criteria needed to adhere to the FCC regulations. It also fits manufacturing cost goals.

Both RF components interface well with the MC68HC705J1A MCU to provide more robust and dependable data recovery which ensures error-free switching of remote devices.

References

[1] *Code of Federal Regulations, Title 47*. U.S. Government Printing Office, Washington, DC, 1992, Part 15, § 15.231.

[2] *VHF – 2.0 GHz Low Noise Amplifier Advance Information;* document order number MC13144/D; Motorola, Inc.

[3] *Wideband FM IF Subsystem for Dect and Digital Applications*, document order number MC13158/D; Motorola, Inc.

[4] *UHF FM/AM Transmitter*; document order number MC13176/D; Motorola, Inc.

[5] *MC68HC/HRC/HSC/HSR705J1A Technical Data*; document order number MC68HC705J1A/D; Motorola, Inc.

[6] *Small Signal Transistors, FETs, and Diodes*, Motorola document order number DL126/D.

[7] Rhea, Randall W. *Oscillator Design and Computer Simulation.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1990.

[8] *General-Purpose Transistor, NPN Silicon*; document order number MMBT3904L/D; Motorola, Inc.

[9] *600GCS-8519N Coil Specification*. Toko America, Inc., Mt. Prospect, IL.

[10] *150-04J08S Coil Specification*, Coilcraft, Cary, IL.

[11] Balanis, Constantine A. *Antenna Theory: Analysis and Design*. Wiley, NY, 1982, pp.169-184.

[12] Yoder, David. *HC05 MCU Keypad Decoding Techniques Using the MC68HC705J1A*; document order number AN1239/D, Motorola, Inc.

[13] George, Scott. *HC05 Software-Driven Asynchronous Serial Communication Techniques Using the MC68HC705J1A*; document order number AN1240/D; Motorola, Inc.

[14] Sibigtroth, James M. *Understanding Small Microcontrollers*, document order number M68HC05TB/D; Motorola, Inc.

Code Listing

```
************************************************************************************
* Transmit Microcode for a Low Power Wireless Remote Control Unit
*
* Revision 7.1.98.2
* Grayhill Keypad                                                                  *
* Decode two keypad inputs, activate transmitter and send a message
*
* code is the first keypad input . command is the second keypad input
*
************************************************************************************

*** I/O Pin Equates:
serial_port    equu     $01                ; port used for serial port pins
status_port    equ      $00                ; port used for driving LED's.
noise          equ      4                  ; pin # for noise LED
frame          equ      5                  ; pin # for frame LED
rxd            equ      0                  ; pin # for receive data pin
txd            equ      4                  ; pin # for transmit data pin
xmt            equ      5                  ; transmitter enable
BAUD_SEL       equ      $08                ; BAUD_SEL   4MHz osc    2 MHz osc
                                           ; $04          19.2k       9600
                                           ; $08          9600        4800
                                           ; $10          4800        2400
                                           ; $20          2400        1200
                                           ; $40          1200         600
                                           ; $80           600         300

               org      0c0                ; start of RAM space
char           rmb      1                  ; data register for sci
count          rmb      1                  ; temp storage variable
length         rmb      1                  ; preamble counter
code           rmb      1                  ; debounced key output storage #1
command        rmb      1                  ; debounced key output storage #2
flag           rmb      1                  ; software flags
                                           ;  bit 0 - first key input done flag

************************************************************************************
* Start of Program Space                                                           *
************************************************************************************
************************************************************************************
*Start - Initializes MCU hardware and enables interrupts.                         *
************************************************************************************
               org     300                 ;
Start:
KeyPdInt_Init:
               ldx      #$00               ;
               rsp                         ; reset stack pointer
               lda      #$0                ; set port A register to all zeroes
               sta      PORTA              ;
               sta      DDRA               ; set Port A direction as Inputs
```

```
              sta      PORTB                   ; set Port B register to all zeroes
              lda      #$3F                    ; set Port B direction as
                                               ; bit 5 output (transmitter switch)
                                               ; bit 4 output (serial data out)
                                               ; bit 3 output (output to switch)
                                               ; bit 2 output (output to switch)
                                               ; bit 1 output (output to switch)
                                               ; bit 0 output (output to switch)
              sta      DDRB                    ; implement direction bits on Port B
              lda      #$0F                    ; load the lower nibble of port B
                                               ; so when a keypad button is pushed
                                               ; a port A interrupt occurs
              sta      PORTB
              clr      PDRA                    : ensure puldowns on port a are enabled
              CLI                              ; Allow interrupts - IRQ pin 19 is
                                               ; hardwired to logic 1 (no interrupt
                                               ; from this pin) waiting for an
                                               ; interrupt from Port A bits

shutdown      STOP                             ; stop to conserve power - waiting
                                               ; for a keyboard interrupt
              bra      start                   ; for getting back to stop when RTI
                                               ; comes back
*********************************************************************************
* Send_preamble - routine that sends out a character stream to the transmitter  *
* Character string is 55 55 55 code command                                     *
*********************************************************************************
Send_preamble sta      code
              bclr     txd,serial_port         ; ensure serial out is low
              bset     xmt,serial_port         ; turn on the transmitter and wait for
                                               ; stable frequency
                                               ; generation
              lda      #$05                    ; load 5 millisecond constant
              jsr      DelaymS2_Body           ; go to delay routine

*             *******************************************************************
*             * generate 500 microsecond logic 1 “front porch”                  *
*             *******************************************************************
              bset     txd,serial_port         ; ensure serial out is high
              lda      #$50                    ; generate a 500 micrsecond
              jsr      delay_13a               ; delay to trigger receiver

*             *******************************************************************
*             * send three preamble characters of 55 hex                        *
*             *******************************************************************
              lda      #$3                     ; load count for sending
              sta      length                  ; number of 'preamble' char
alt_ones      lda      #55                     ; load preamble character
              sta      char                    ;
              jsr      put_char                ; send the character in char
              dec      length                  ; decrement count
              bne      alt_ones                ; go send another if not zero
```

```
*              ********************************************************************
*              * route the address in ‘code’ to the routine that sends it out     *
*              ********************************************************************

Send_code      lda      code                 ; get the key-pad character
               sta      char                 ; send out key-pad character
                                               ; insert a 12 cycle timing stall so
                                               ; receiver and transmitter routines take
                                               ; the same time between sent characters
               sta      char
               sta      char
               sta      char
               jsr      put_char               ; send address byte

*              ********************************************************************
*              * route the command byte in “command” to the routine that sends it out *
               ********************************************************************
               lda      command                ; get the command code
               sta      char
                                               ; insert a 12 cycle timing stall so
                                               ; receiver and transmitter routines
                                               ; take the same time between sent
                                               ; characters
               sta      char
               sta      char
               sta      char
               jsr      put_char               ; send command byte

*              ********************************************************************
*              * turn off transmitter and initialize buffers for next time        *
*              ********************************************************************

               bclr     txd,serial_port        ; ensure serial out is zero
               bclr     xmt,serial_port        ; turn off transmitter
               lda      #$0
               sta      code
               sta      command
               sta      flag
               rts

***********************************************************************************
* Routine from AN1240 written by Scott George                                     *
***********************************************************************************
put_char       ldx      #9                     ;[2] be sending 8 data bits
               clc                             ;[2] clear carry for start bit
put_data_bits  bcc      send_0                 ;[3] if carry<>0, then
               bset     txd,serial_port        ;[5]    send out a 1
               bra      jmp_bit                ;[3]    finished sending a 1
send_0         bclr     txd,serial_port        ;[5] else send a 0
               bra      jmp_bit                ;[3]    finished sending a 0
jmp_bit lda    #2*(BAUD_SEL-1)-1               ;[2] prepare for a 1 bit delay
               bsr      delay_13a              ;[13a+12] execute delay routine
               tsta                            ;[3] for timing purposes only
               ror      char                   ;[5] get next data bit to send
```

```
              decx                          ;[3] one bit sent, so dec count
              bne     put_data_bits         ;[3] loop if more bits to send

put_stop_bit  nop                           ;[2] for timing purposes only
              bset    txd,serial_port       ;[5] send out a one
              lda     #2*(BAUD_SEL-1)       ;[2] prepare for a 1 bit delay
              bsr     delay_13a             ;[13a+12] execute delay routine
              rts                           ;[6] exit (put_char)

*********************************************************************************
* Delay routine from AN1240  written by Scott George                            *
*********************************************************************************
delay_13a     nop                           ;[2] this is a 13-cycle loop
              nop                           ;[2]
              tsta                          ;[3]
              deca                          ;[3] decrement loop count
              bne     delay_13a             ;[3] loop if count not zero
              rts                           ;[6] exit (delay_13a*

*********************************************************************************
* Delay routine from AN1240  written by Scott George                            *
*********************************************************************************

DelaymS2_Body:                              ;JSR EXT to get here    6
DelaymS2010   ldx     #$F8                  ;Load delay into X      2--\
DelaymS2020   decx                          ;   Decrement delay     3-\|
              nop                           ;   burn 2 bus cycles   2 ||
              bne     DelaymS2020           ;   Branch if not done  3-/|
              stx     COPR                  ;Service the WDOG       5  |
                                            ;Note that X will          |
                                            ;always be zero here       |
              brn     *                     ;Burn 3 bus cycles      3  |
              deca                          ;decrement # of mS      3  |
              bne     DelaymS2010           ;branch if not done     3--/
DelaymS2030   rts                           ;return

*********************************************************************************
* Original Routine from AN1239 written by David Yoder and modified for use in   *
* A low-power Wireless Remote Control Tranmitter and Receiver                   *
*                     IRQ Interrupt Service Routine                             *
*                                                                               *
* This is the external interrupt service routine. Both the external             *
* interrupt pin IRQ_ and the keypad interrupts use this routine.                *
* Note that in this version, any interrupt is interpreted as having only a keypad *
* source                                                                        *
*********************************************************************************

KeyPdInt_Isr:                               ; Any decoding of external interrupts
                                            ; should be done here.
                                            ; The external and keypad interrupt share
                                            ; this vector.
KeyPdInt_Isr010:
              jsr     KeyPad_Body           ;See if a key is pressed
              beq     KeyPdInt_Isr090       ;If no key down, return to save power
```

```
               lda      #$4                    ;Debounce key for 4mS
               jsr      DelaymS2_Body          ;Jump to delay routine
               jsr      KeyPad_Body            ;Get the keypress
KeyPdInt_Isr020:
               beq      KeyPdInt_Isr090        ;If no key down, return
                                               ;no key down = 00 in acu
KeyPdInt_Isr030:
               brset    first,flag,get2nd      ; branch if first key seen
               sta      code                   ; store address code
               bset     first,flag             ; set first key flag
               bra      KeyPdInt_Isr080        ; go wait for key release
get2nd         sta      command                ; store command byte
               jsr      Send_preamble          ;go send preamble & code

KeyPdInt_Isr080:
               jsr      KeyPad_Body            ; go check if the key has been released
               beq      KeyPdInt_Isr090        ; no key pressed so exit
               bra      KeyPdInt_Isr080        ; else wait for key release

KeyPdInt_Isr090:
               lda      #!10                    ;Delay 10 mS
               jsr      DelaymS2_Body          ;Debounce the release
               bset     IRQR,ISCR              ;Clear any interrupt requests
                                               ; generated due to key bounce
               bclr     xmt,serial_port         ; turn off the transmitter
               rti                             ;Return from Interrupt  (KeyPdInt_Isr)
                                               ;Interrupts can happen in any code in
                                               ; the main routine after this ISR has
                                               ; been called once.
                                               ;Remember this when changing the main
                                               ; routine!
************************************************************************************
* This keypad decode  routine is from AN1239  written by David Yoder                *
*                                                                                  *
* KeyPad_Body                                                                      *
*                                                                                  *
* This subroutine decodes a 4x4 matrix keypad on port B.                           *
*                                                                                  *
************************************************************************************

KeyPad_Body:                                   ;Load X with the offset of the last
                                               ;entry in the table
               ldx      #{KeyPad_Table_Top - KeyPad_Table}
KeyPad010:
               lda      portb                  ;Get value in port B
               and      #$f0                   ;Do not allow high nibble to change
               ora      KeyPad_Table+1,x       ;Get key decode value from table
               sta      portb                  ;Write to Port

               lda      porta                  ;Get value in port A
               and      #$0F                   ;Throw out columns to read only rows
               cmp      KeyPad_Table,x         ;See if high nibble bit was pulled low
               beq      KeyPad030              ;If key found, branch
               decx                            ;Decrement X thrice to point to
```

```
              decx                           ; next value in table
              decx
              bpl      KeyPad010             ;If not below bottom of table
                                             ; try again.
              ldx      #$00                  ;A key was not decoded, so:
              bra      KeyPad035             ;Return with null character
KeyPad030:
              lda      KeyPad_Table+2,x      ;Load key code into Acc.
              tax                            ;Store in X for now.

KeyPad035:    lda      portb                 ;'Help' the pulldowns by driving the
              and      #$F0                  ; lines low. This minimizes current
              sta      portb                 ; draw while debouncing.

              txa                            ;Get result back to Acc
              tsta                           ;Set the flags so calling routine
                                             ; can use them for decisions.
KeyPad040     rts                            ;Return with result value in Acc

                             ;Table of keypad decode values and codes.
                             ;Fill in your own key codes. Codes must be 1
                             ; byte each. Currently row 1 col 1 = 0, which
                             ; duplicates no key found. This is a result
                             ; of limiting the codes to 4 bits for display
                             ; on PA[4..7]
                                             ; Row   Column
KeyPad_Table:
                       DB    $01,$01,$F1     ; PA0   PB0
                       DB    $01,$02,$F2     ; PA0   PB1
                       DB    $01,$04,$F3     ; PA0   PB2
                       DB    $01,$08,$FA     ; PA0   PB3
                       DB    $02,$01,$F4     ; PA1   PB0
                       DB    $02,$02,$F5     ; PA1   PB1
                       DB    $02,$04,$F6     ; PA1   PB2
                       DB    $02,$08,$FB     ; PA1   PB3
                       DB    $04,$01,$F7     ; PA2   PB0
                       DB    $04,$02,$F8     ; PA2   PB1
                       DB    $04,$04,$F9     ; PA2   PB2
                       DB    $04,$08,$FC     ; PA2   PB3
                       DB    $08,$01,$FF     ; PA3   PB0
                       DB    $08,$02,$FE     ; PA3   PB1
                       DB    $08,$04,$FF     ; PA3   PB2
KeyPad_Table_Top       DB    $08,$08,$FD     ; PA3   PB3
************************************************************************************
#INCLUDE 'H705J1A.FRK'                 ;Include the equates for the HC705J1A

************************************************************************************
              org      MOR
                fcb     06                   ;Enable Port A Interrupts
                                             ;If used on a mask rom part,
                                             ; be sure to specify this option.
                                             ;Enable level sensitivity on IRQs
                                             ; SOSCD = Short Oscillator delay
                                             ; enable
```

```
                                        ; EPMSEC = EPROM Security
                                        ; OSCRES = Ocscillator Parallel
                                        ; Resistor
                                        ; SWAIT = Stop Instruction Mode
                                        ; SWPDI = Port Pulldown Inhibit
                                        ; PIRQ = Port A Interrupt Function
                                        ; LEVEL = IRQ Edge Sensitivity
                                        ; COPEN = COP Watchdog enable
                                        ;Software pull down inhibit = off
                                        ;Stop = Halt disabled
                                        ;OSC parallel resistor disabled
                                        ;EPROM security off
                                        ;Short OSC recovery disabled

                  org     RESET
                  fdb     Start
                  org     IRQ_INT
                  fdb     KeyPdINt_Isr

**********************************************************************************
* Receiver Microcode for a Low Power Wireless Remote Control Unit                *
* REVISION 7.9.98.2                                                              *
*                                                                                *
**********************************************************************************
*                                                                                *
* Full Functional Description of Routine Design:                                 *
*   Program flow:                                                                *
*       Main/Reset:  Call init to intialize port pins and interrupts             *
*               Clean up the buffers                                             *
*                                                                                *
**********************************************************************************

********************************************
* Part Specific Framework Include Section  *
********************************************
**********************************************************************************
#INCLUDE 'H705J1A.FRK'                  ; Include the equates for the
                                        ; HC705J1A so all labels can be found.

**********************************************************************************
*                                                                                *
*                                Equates and RAM Storage                         *
*                                                                                *
**********************************************************************************
*** I/O Pin Equates:
serial_port   equ      $01              ; port used for serial port pins
status_port    equ      $00             ; port used for driving LED's.
noise          equ      4               ; pin # for noise LED
frame          equ      5               ; pin # for frame LED
rxd            equ      4               ; pin # for receive data pin
txd            equ      1               ; pin # for transmit data pin
passflag       equ      7               ; timer monitor bit

*** Program Constant Equates:           ; Baud rate select table:
```

```
BAUD_SEL        equu     $08                ; BAUD_SEL   4MHz osc    2MHz osc
                                            ; $04 19.2 k 9600
                                            ; $08        9600    4800
                                            ; $10 4800   2400
                                            ; $20 2400   1200
                                            ; $40 1200    600
                                            ; $80  600    300
*** RAM variable allocation:
              org     0C0
char          rmb     1                      ; incoming data register
count         rmb     1                      ; temp storage variable
cmpr          rmb     1                      ; compare register
xcnt          rmb     1                      ; counter
length        rmb     1                      ; index pointer storage
buffer        rmb     8                      ; receiver buffer
flag          rmb     1                      ; flag bits  - bit 0 = got a 55
*************************************************************************************
*                     *******************************
*                     * beginning of EPROM space    *
*                     *******************************
              org     300                   ; start at the top of ROM
main          rsp                           ; reset the stack pointer
              ldx     #$0                   ; clear flags
              jsr     init                  ; initialize port pins & ints
              lda     #$4                   ; load bit counter to 3
              sta     count
Wait_for_data brset   rxd,PORTB,front_porch ; branch if incoming data =1
              inc     xcnt                  ; increment a count
              bcc     Wait_for_data         ; check for incoming data
              lda     #$00                  ; carry set so clear the count
look_again    sta     xcnt                  ; and go back to examining data
              bra     Wait_for_data
*************************************************************************************
* frontporch -takes a 4-bit sample and looks for logic 1                            *
*************************************************************************************
front_porch   equ     *
              jsr     get_bit               ;[39] sample data bit
              rora                          ;[3] noise bit -> carry
              rorx                          ;[3] carry -> noise data reg
              rora                          ;[3] filtered data bit -> carry
              ror     char                  ;[5] carry -> char
              lda     #2*(BAUD_SEL-3)       ;[2] prepare for 1 bit delay
              jsr     delay_13a             ;[13a+12] execute delay routine
              tsta                          ;[3] for timing purposes only
              dec     count                 ;[5] bit received, dec count
              bne     front_porch           ;[3] loop if more bits to get
              lda     #$f0                  ; check for bit value received
              cmp     char                  ; looking for 4 one bits
              beq     get_data              ; four logic 1 bits received - go wait
                                            ; for incoming data
              jmp     main                  ; invalid frontporch -restart
```

```
*************************************************************************************
* get_data - the routine that acquires data and puts it into the receive buffer.   *
* A timer interrupt is the expected exit path.  timer interval is 8.2 msec         *
*************************************************************************************
get_data       lda     #$1C                  ; initialize timer register
               STA     TSCR                  ; and allow interrupts
               cli                           ; clear I bit in CCR
               lda     #$0                   ; initialize the index
               sta     length                ; store in memory
main_loop      jsr     get_char              ; receive one byte of data from rxd pin
               lda     char                  ; get the current character
               ldx     length                ; load buffer index
               sta     buffer,x              ; store character in recv buffer
               lda     #$3                   ; check for buffer full
               cmp     length                ; is count 3? (buffer full)

waiting_for_int beq     waiting_for_int      ; wait here for interrupt if receive
                                             ; buffer is full
*      ***************************************************************
*        * waiting for timer interrupt- expected exit point - see RTI_int *
*      ***************************************************************
               inc     length                ; increment buffer pointer
               bra     main_loop             ; go get another byte
********************************************
**                                              **
***                                            ***
**** Entry point for timer interrupt routine    ****
***                                            ***
**                                              **
********************************************
Timer_SVR      equ     *                     ;
RTI_int        equ     *                     ; Real Time Interrupt routine
               sei                           ; disable interruptss
               lda     #$0C                  ; clear timer overflow flag
                                             ; clear RTI flag
                                             ; Timer overflow disable
                                             ; Real Time Interrupt disable
                                             ;
               sta     TSCR
check_msg_forma lda    #$0                   ; initialize the index
               sta     length                ; in location length
               sta     flag                  ; clear all flags
               ldx     length                ; and load the index register
               lda     #$55                  ; looking for a 55
               sta     char
look           lda     buffer,x              ; get a buffer character
               cmp     char                  ; check if it's a 55
               beq     got_one               ; branch if a 55
next           incx
               inc     length                ; looked for all possible 55s?
               lda     #$3                   ; in the first three buffer
               cmp     length                ; locations
               beq     done                  ; branch if so
               bra     look                  ; go back and look some more
```

```
got_one        brset    0,flag,prev55          ; branch if saw a 55 previously
               bset     0,flag                 ; set the flag
               bra      next                   ; go look for another 55
prev55         ldx      #$3                    ; get code character in buffer
               lda      #$f5                   ; compare for address desired - this
value changes
               cmp      buffer,x               ;
               beq      toggle                 ; go execute the command byte if address
                                               ; byte match
               lda      #$fc                   ; compare for broadcast address
               cmp      buffer,x
               beq      toggle                 ; go execute the command byte
                                               ; if a broadcast address match
done           jmp      main                   ; else- go reset everything and wait for
                                               ; the front porch

**************************************************
* command byte checking
**************************************************

toggle         ldx      #$4                    ; set index to command byte
               lda      #$FA                   ; check for an on command
               cmp      buffer,x
               beq      turnon                 ; branch if turn on command
               lda      #$FB                   ; compare for turn off command
               cmp      buffer,x
               beq      turnoff                ; branch if turn off command
               bra      done                   ; else invalid command so...
turnon
               bset     5,portb                ; set the port bit
               bra      done                   ; command executed so...
turnoff        bclr     5,portb                ; reset the port bit
               bra      done                   ; command executed so ...

**********************************************************************************
* init - initialize port pins, disable interrupts. Clear all buffers and flags     *
*        called by main                                                           *
*                                                                                 *
* input cond.     - none                                                          *
* output cond.    - Port B bit 5 = output , RXD = input                           *
* stack used      - 0 bytes                                                       *
* variables used  - none                                                          *
**********************************************************************************
init           lda      #$0                    ; all port a pins are inputs
               STA      DDRA                   ; (all port a pins are grounded)
               lda      #$20                   ; port b pin 5 output - to device to be
                                               ; turned on and off
               STA      DDRB                   ; pin 4 input  - data from the receiver
               lda      #$00                   ; load ISCR to 00 - no external
                                               ; interrupts from IRQ Port A
               STA      ISCR
               lda      #$0C                   ; load TSCR (timer Control) so 0= turn
                                               ; off RTIC= RTI enable,
                                               ; clear RTIFlag
```

```
              STA      TSCR                 ; C= RTI = enable, clear RTI Flag
              lda      #$0                  ; clear buffers
              sta      flag                 ; clear 55 flag
              sta      char                 ; cleaar receiver memory location
              sta      length               ; clear index counter
              ldx      #$0
              sta      buffer,x             ; clear receive buffer
              incx                          ; location 0
              sta      buffer,x
              incx                          ; location 1
              sta      buffer,x
              incx                          ; location 2
              sta     buffer,x
              incx                          ; location 3
              sta     buffer,x
              rts                           ;

**********************************************************************************
* incoming/get_char -   Routine from AN1240 written by Scott George              *
* called by get_data                                                             *
* input cond.          - RXD pin defined as an input pin                         *
*                 - expecting a preamble character of where four bits are logic 1  *
*                 - followed by a series of 55 55 55  fx  fx                     *
* get_char - receive one byte of data from RXD pin; called by main               *
*                                                                                *
* input cond.          - RXD pin defined as an input pin                         *
* output cond.         - char contains received data; X,ACC undefined;           *
*                   half carry = 1 (frame error occurred) or 0 (no frame error);   *
*                   carry = 1 (noise and/or frame error occurred) or 0 (no noise)  *
* stack used           - 2 bytes                                                 *
* variables used       - char: storage for received data (1 byte)                *
*                   count: temporary storage (1 byte)                            *
* ROM used             - 63 bytes                                                *
**********************************************************************************
incoming:
get_char      lda      #8                   ;[2] receiving 8 data bits
              sta      count                ;[4] store value into RAM
              clrx                          ;[3] used to store noise data

get_start_bit brclr    rxd,serial_port,*    ;[5] wait until rxd=1
              brset    rxd,serial_port,*    ;[5] wait for start bit
              lda      #BAUD_SEL-3          ;[2] prepare for 1/2 bit delay
              bsr      delay_13a            ;[13a+12] execute delay routine
              bsr      get_bit              ;[39] sample start bit
              lsra                          ;[3] noise bit -> carry;
                                            ;    acc=filtered start bit
              bne      get_start_bit        ;[3] if false start, start over
              tsta                          ;[3] for timing purposes only
              tsta                          ;[3] for timing purposes only

              lda      #2*(BAUD_SEL-2)      ;[2] prepare for 1 bit delay
              bsr      delay_13a            ;[13a+12] execute delay routine
```

```
get_data_bits bsr     get_bit                ;[39] sample data bit
              rora                           ;[3] noise bit -> carry
              rorx                           ;[3] carry -> noise data reg
              rora                           ;[3] filtered data bit -> carry
              ror     char                   ;[5] carry -> char
              lda     #2*(BAUD_SEL-3)        ;[2] prepare for 1 bit delay
              bsr     delay_13a              ;[13a+12] execute delay routine
              tsta                           ;[3] for timing purposes only
              dec     count                  ;[5] bit received, dec count
              bne     get_data_bits          ;[3] loop if more bits to get

get_stop_bit  bsr     get_bit                ;[39] sample stop bit
              lsra                           ;[3] noise bit -> carry
                                             ;    acc=filtered stop bit
              sta     count                  ;[4] store stop bit in count
              bcc     yes_noise              ;[3] if noise, then branch

              txa                            ;[2] noise data -> acc
               eor    char                   ;[3] XOR noise with char,
              beq     no_noise               ;[3] and if result=0,
                                             ;    then no noise in data
                                             ;    reception

yes_noise     lda     #$08                   ;[2] set noise bit (half carry)
              add     #$08                   ;[2] by adding $8 to $8

no_noise      lda     count                  ;[3] retrieve stop data bit,
               coma                          ;[3] complement it,
              lsra                           ;[3] and shift it into carry
                                             ;    for frame error bit
              rts                            ;[6] exit (get_char)

************************************************************************************
* Routine from AN1240 written by Scott George                                      *
* get_bit - receive one bit of filtered data and noise info;                       *
*             called by get_char and frontporch                                    *
*                                                                                  *
* input cond.          - RXD pin defined as an input pin                           *
* output cond.         - ACC = 000000dn, where d = filtered data, n = noise info   *
* stack used           - 0 bytes                                                   *
* variables used       - none                                                      *
* ROM used             - 17 bytes                                                  *
************************************************************************************
get_bit       clra                             ;[3] used to add sampled bits
              brset   rxd,serial_port,samp_1;[5] sample 1st bit into carry
samp_1 adc    #0           ;[3] add it to acc
              brset   rxd,serial_port,samp_2 ;[5] sample 2nd bit into carry
samp_2 adc    #0           ;[3] add it to acc
              brset   rxd,serial_port,samp_3 ;[5] sample 3rd bit into carry
samp_3        adc     #0                     ;[3] add it to acc
              rts                            ;[6] exit (get_bit)
```

```
************************************************************************************
* Routine from AN1240 written by Scott George                                      *
* delay_13a - delay for 13*ACC + 12 cycles; called by get_char and front porch     *
*                                                                                  *
* input cond.     - ACC set to appropriate value (13*ACC + 12 cycles)              *
* output cond.    - ACC = 0                                                        *
* stack used   - 0 bytes                                                           *
* variables used- none                                                             *
* ROM used    - 7 bytes                                                           *
************************************************************************************
delay_13a     nop                         ;[2] this is a 13-cycle loop
              nop                         ;[2]
              tsta                        ;[3]
              deca                        ;[3] decrement loop count
              bne     delay_13a           ;[3] loop if count not zero
              rts                         ;[6] exit (delay_13a)
************************************************************************************
*                                                                                  *
*           MOR Bytes Definitions for the Main Routine                             *
*                                                                                  *
************************************************************************************
              org     MOR
              fcb     $00
************************************************************************************
*                                                                                  *
*        Interrupt and Reset vectors for Main Routine                              *
*                                                                                  *
************************************************************************************
              org     RESET               ; When reset occurs, go to routine main
              fdb     main
              org     Timer_INT           ; When interrupt occurs, go to routine
                                          ; Timer_SVR
              fdb     Timer_SVR
```

Application Note

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™ semiconductor