Keywords: wireless automatic meter reading, AMR, transceiver, transmitter, receiver, ISM, RF, radio, PCB, layout, schematic, hardware, firmware, microcontroller, utility company, smart grid, protocol, antenna, link, range

**REFERENCE DESIGN 5391 INCLUDES: √Tested Circuit √Schematic √BOM √Board Available √Description √Test Data √Software √Layout**

# LFRD002: Wireless Automatic Meter Reading Reference Design

**By: Martin Stoehr, Principal Member of the Technical Staff, Applications**
**May 04, 2012**

*Abstract: This reference design provides a complete demonstration platform for using industrial/scientific/medical radio frequency (ISM-RF) products in a wireless automatic meter reading (AMR) application. This document includes the hardware, firmware, and system structure requirements for implementing an AMR design.*

More Information
- Wireless Home
- Application Notes and Tutorials
- EV Kit Software
- Technical Support

## General Description

Click here for an overview of the wireless components used in a typical radio transceiver.

The MAX7032 transceiver reference design (RD) is a self-contained evaluation platform for exercising the device as a wireless automatic meter reading (AMR) demo system. With the use of the Maxim USB-to-JTAG board (MAXQJTAG-USB), the MAXQ610 on both the slave meter (MTR) and the master reader (RDR) board can be programmed by the end user.

The meter board enables basic human interaction through a single momentary switch input and an LED for visual feedback. The MTR is designed to be compact, providing a self-contained transceiver board with a radio, microcontroller, and multiple "ports" for connecting various meter inputs to the system. Two separate designs are provided: one with a built-in printed circuit board (PCB) antenna, and the other with a small-footprint antenna-mounting option. Input to the MTR system can be configured with up to six ports, and the primary input interfaces with a "pulse" or dry contact (reed) output meter. This board can be operated from any 3V power source (1.7V to 3.6V for the MAXQ610, 2.1V to 3.6V for the MAX7032).

The reader board has seven menu keys for user input, a reset switch, a 102 pixel x 64 pixel LCD display with multicolor LED backlighting for menu interactions, plus a receive signal indicator (RSI) LED. The shape of the reader unit (RDR) fits within a Series 55 BOX enclosure, and has both an edge-connected antenna mount and a built-in PCB trace antenna.

Both systems are preprogrammed with operational firmware to demonstrate a simple wireless AMR meter (slave)/reader (master) system. Gerber files are available for simple cut-and-paste designs of the radio sections or the full implementation.

## Features

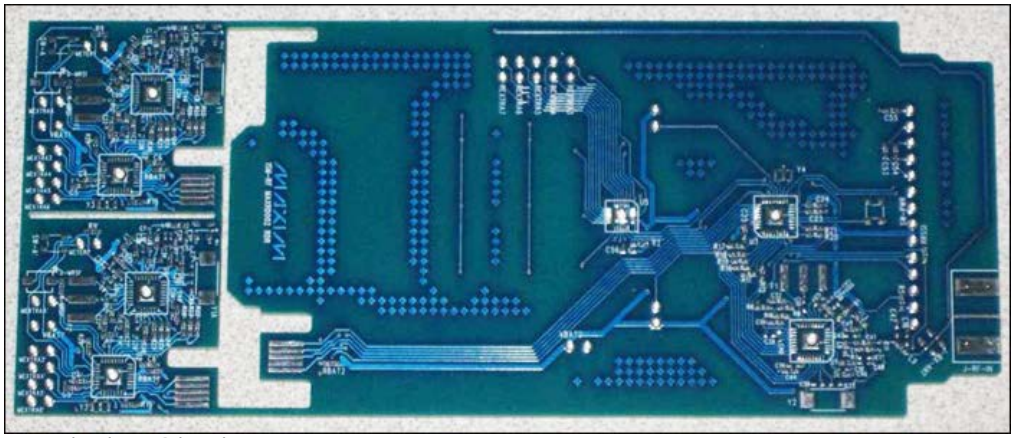
- Proven printed circuit board (PCB) layout
- Proven component parts list
- Preprogrammed transceiver (TRX) pair for quick demonstration capabilities
- Free MAXQ® microcontroller programming tools available for flexible operation

## Quick Start

1. Pull the three boards (two MTR boards and the RDR board) out of box and install the batteries.
2. Connect the RDR antenna.
3. Navigate the RDR menu system MAIN MENU ⟶ AMR MENU ⟶ START COMM

The RDR will communicate with each of the MTR units in turn and will return the value of 00 after each communication.

## Meter and Reader Board Description

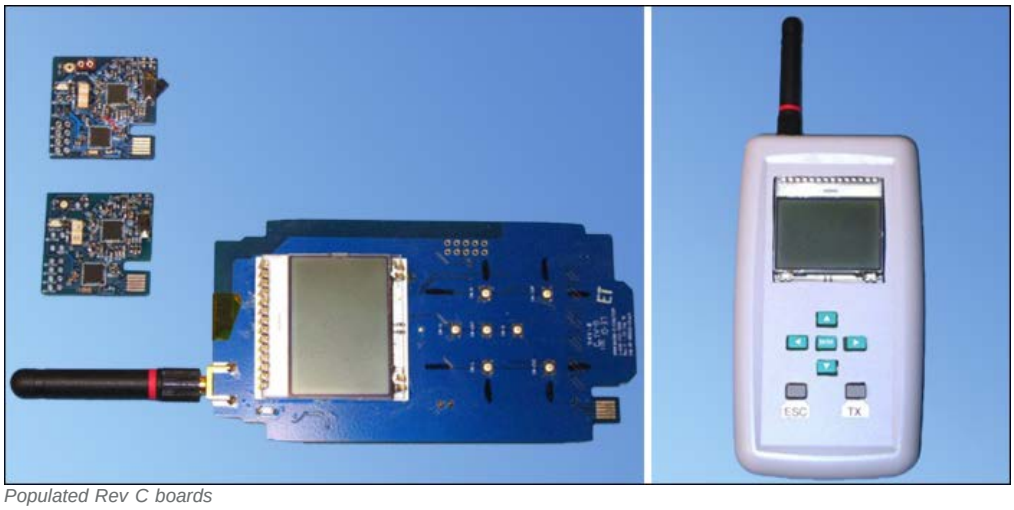


*Unpopulated Rev C boards*

## Form Factor

The LFRD002 was designed as a demonstration platform for both the MAX7032 transceiver and the MAXQ610 microcontroller. This reference design targets a low-cost, low bill-of-materials (BOM) count, RF link, showing how an inexpensive design can still work in a variety of metering environments. Both the MTR and the RDR systems incorporate a MAX7032 radio IC mated with a MAXQ610 microcontroller that comes preprogrammed to operate as a wireless AMR demonstration system.

Both boards allow the end user to program the MAXQ610 through a JTAG interface. In its smallest form, the MTR board is 3cm x 3cm while the RDR board measures in at 11.5cm x 6.1cm.




*Populated Rev C boards*

## I/Os and Switches

Power to the MTR board is typically supplied by a 3.6V lithium (Li) battery but can be powered through the JTAG interface as well. The RDR board is typically powered by a pair of AA batteries, and also has the JTAG power provision.

**RDR Switch Function Table**

| Switch | Position | Function | Connection |
|---|---|---|---|
| SW-PWR | Momentary | "Reset" | Pin 11, P1.2 of µC |
| (up) | Momentary | "^" or "Up" | Pins 12 & 23 of MAX7359 |
| (left) | Momentary | "<" or "Left" | Pins 13 & 24 of MAX7359 |
| ENTER | Momentary | "Enter" | Pins 12 & 24 of MAX7359 |
| (right) | Momentary | ">" or "Right" | Pins 11 & 24 of MAX7359 |
| (down) | Momentary | "v" or "Down" | Pins 1 & 12 of MAX7359 |

| | | | |
|---|---|---|---|
| (Esc) | Momentary | "Escape" | Pins 1 & 13 of MAX7359 |
| (TX) | Momentary | "Transmit" | Pins 1 & 11 of MAX7359 |

**MTR Switch Function Table**

| Switch | Position | Function | µC Connection |
|---|---|---|---|
| SW-A | Momentary | User TX | Pin 11, P1.2 |

**RDR and MTR I/O Edge Connectors**

| Signal | Description | µC Connection |
|---|---|---|
| JTAG-1 | TCK—clock | Pin 24, P2.4 |
| JTAG-2 | GND—ground | |
| JTAG-3 | TDO—data out | Pin 27, P2.7 |
| JTAG-4 | VBAT—external supply | |
| JTAG-5 | TMS—master select | Pin 26, P2.6 |
| JTAG-6 | nRST—reset | Pin 28, Reset |
| JTAG-7 | N/A | |
| JTAG-8 | N/A | |
| JTAG-9 | TDI—data in | Pin 25, P2.5 |
| JTAG-10 | GND—ground | |

**MTR LED Indicator Table**

| LED | Function | µC Connection |
|---|---|---|
| MRSI | Tx/Receive Signal Indicator/heartbeat | Pin 7, P0.6 |

**RDR LED Indicator Table**

| LED | Function | µC Connection |
|---|---|---|
| RSSI | Tx/Receive Signal Indicator | Pin 7, P0.6 |
| LEDBL1 | Red backlight | Pin 21, P1.7 |
| LEDBL2 | Green backlight | Pin 20, P1.6 |

## Data Frame Structure

The basic structure of the data frame is ASK modulated, Manchester encoded, 4.8kbps (0.2083ms/bit), and has 144 bits per frame (18 bytes or 9, 2-byte words), 30ms per frame, a pause of 70ms between frames, and 3 frame transmissions per burst. For information on Manchester encoding, refer to application note 3435, "Manchester Data Encoding for Radio Communications."

This structure is directly compatible with other reference design communication formats, and utilities are provided in the RDR system to work directly with other reference designs. Appendix I further describes each section of the remote keyless entry (RKE) frame structure.

**Frame structure**

| Preamble | | | | ID | | | | Function | | Data | | Sync | | Bat | Sig | Chk Sum | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FF | FF | FF | FD | 02 | 00 | 00 | 00 | 00 | 01 | 00 | 00 | 43 | 21 | 11 | 22 | 01 | 68 |

The structure of this frame is arbitrary, but provides an example of the information that can be contained in any frame related to the many industrial, scientific, and medical (ISM) RF applications.

**ID Structure**

| ID | | | |
|---|---|---|---|
| 02 | 00 | 00 | 00 |

This design has been preprogrammed to use a 0x02 00 00 00 identification code in the RDR, with the last byte being adjusted between MTR systems (typically 0x01 and 0x02). This allows the RDR to communicate with each of the two MTR boards separately by addressing them one at a time.

| Function Structure | |
|---|---|
| **Function** | |
| FF | 00 |

This reference design uses the Function field to communicate "group" and "individual" functions to the various MTRs within reception of the RDR system. Again, this structure can be modified to suit the purposes of the user. In this application, the first byte of the Function field is used for high-level commands such as ATTN and CLOSE. The second byte of the Function field is used for various hand-shaking or command operations. **Table 1** describes the various functions and their hexadecimal values as seen on the display and within the firmware:

**Table 1. Functions and Hexadecimal Values Within the LFRD002 Firmware**

| Function | Frame Valueh | | Source |
|---|---|---|---|
| | Func[0] | Func[1] | |
| ATTN | FF | XX | RDR |
| REQ | 00 | 8P | RDR |
| DAT | 00 | 1P | MTR |
| SET | 00 | 1P | RDR |
| CLR | 00 | 2P | RDR |
| ACK | 00 | 4P | RDR |
| CLOSE | AA | XX | RDR |
| ECHO | 00 | 00 | Node |
| Red (A) | 00 | 01 | RKE TX |
| Green (B) | 00 | 02 | RKE TX |
| Blue (C) | 00 | 04 | RKE TX |
| Amber (D) | 00 | 08 | RKE TX |

P = Addressed port number, X = don't care, NN = node list index

These values are an arbitrary definition for the structure of the Func field, which can always be modified by the user.

| Data Structure | |
|---|---|
| **Data** | |
| 00 | 00 |

The Data section of this frame is provided for transmitting information such as a temperature, pressure, or volume measurement. In this design, the Data field tallies pulse outputs from a water meter. Again, the use of this Data value is arbitrary and can be modified by the user.

## Communication Protocol

The basic structure defines the interaction of the RDR (master) system and any number of MTR (slave) systems. (See Appendix V).

To conserve power, the MTR systems are configured to operate with a predefined OFF/ON or sleep/listen (S/L) duty cycle. In this demo system, after power is applied to the MTR board, it acknowledges startup with four flashes of the LED, then enters into a 3.0s sleep/0.5s listen cycle. During the ON cycle, the radio is configured for Rx mode, and the system will listen for any in-frequency, ASK broadcasts. If a valid ATTN frame is received, the MTR system will suspend its S/L cycle and enter into a full wake state.

All communication is initiated with a broadcast from the RDR system. Since the MTRs are presumably in a 3.0s/0.5s S/L cycle, the RDR begins with a 5s transmission of Attention (ATTN) frames to awaken all MTRs within range. The ATTN frame is defined as Don't Care values for all of the fields, except for the checksum and the first byte of the Function Structure, which is set to 0xFF00.

After issuing the ATTN signal, the RDR will communicate with all of the MTRs listed in the system (see the MTR List in the RDR Menu section). In this case, two MTRs should be predefined in the system, with one port set up for reading on MTR ID 01, and two ports set up for reading on MTR ID 02. Port 0 on the MTR is configured to act as the pulse-tally port, and MTR port 1 has a preset value of 0x2222. The communication can occur either as a batch process (all the MTRs in the list are automatically contacted) or as a user-sequenced process (the user presses the ENTER key to initiate a REQ/DATA exchange with each MTR entry in the MTR list individually).

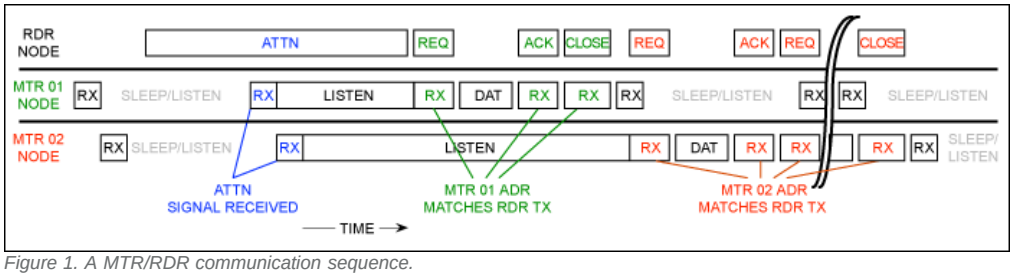



*Figure 1. A MTR/RDR communication sequence.*

A full communication sequence (**Figure 1**) takes the form of the RDR sending a Request (REQ) frame burst followed by a short pause, and the RDR switching to RX mode. The MTR with the matching ID will decode the REQ frame, prepare the data, and switch to TX mode. The MTR transmits a Data (DAT) frame to the RDR, then switches back to RX mode. The RDR receives the DAT frame, and then switches to TX mode and sends an Acknowledge (ACK) frame to the MTR. At this point, the RDR system can either transmit another REQ frame (starting another communication sequence, possibly addressing a different port on the MTR) or it can close the communication session with that MTR. After all of the measurement ports of interest have been read from that MTR, the session with that ID is terminated by sending a Close (CLOSE) frame to the MTR. After a MTR has received the CLOSE frame, it will revert to the S/L cycle until it receives a new ATTN frame.

### Handling Lost Frames and Errors

After the RDR receives the requested information from the targeted MTR, it returns an ACK frame. Since this exchange consists of three functional commands, both ends of the communication can be confirmed. If the REQ frame was not received properly, the RDR should recognize that when it does not receive a DAT frame response; if the RDR did not receive the DAT frame, the MTR will not see an ACK frame response. This process allows for retransmission of the REQ frame by the RDR (if no DAT frame is received) and similarly, the DAT frame can be retransmitted by the MTR (if no ACK frame is received).

- It should be assumed that the RDR has the potential for a stronger TX signal than the MTR.
- If a MTR receives an ATTN signal but no other communication (within 30s), it reverts to its sleep/listen cycle.
- If a RDR does not receive the DAT frame (after 1s), it will repeat a REQ frame (two more times). If the DAT frame is still not received, the MTR should be skipped. In this case, the MTR will be flagged as NON-COMM.
- If a MTR sends a DAT frame but does not receive an ACK frame (within 1s), it will repeat the DAT frame (two more times). In this case, assume the session was closed and revert to its sleep/listen cycle.

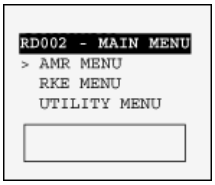
This process will assure a minimum number of reattempts to communicate. If the three-burst frame is not sufficient and the multiple attempts also fail, both units will revert to their default state (sleep/listen for the MTR, and NON-COMM for the RDR). This lost frame process is not currently implemented.

## RDR Menu

The RDR unit operates with a basic user interface structure using a simple menu system and a few navigation keys. To browse through the menu system, there are seven keys: four directional arrows to scroll up (^) and down (v), right (<) and left (>), the ENTER key, the ESC key, and a TX key. The menu interface indicates the selectable line with a ">" character (cursor) found in the left-most column of the display.

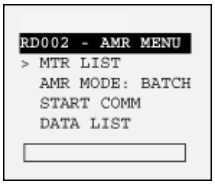
### Main Menu

From the Main Menu, the user can navigate to the three primary functions of the RDR unit: the AMR functions, the RKE functions (that can operate in conjunction with an LFRD001), and the utility functions.




### AMR Menu

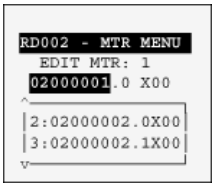
The primary menu of interest for the LFRD002 system will be the AMR Menu. From this screen, the user can review and edit the MTR list, change the operating mode of the AMR sessions, begin an AMR communication session, or review the last set of data received from the MTRs.



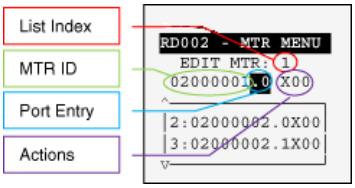

### MTR Menu

Under the AMR Menu, the user can navigate to the list of MTRs to be serviced during the communication sessions. This

screen will initially present a field showing the MTR List Index (the list entry being edited), the fields for that entry, and a display of the next two MTRs in the list (shown in the box). The fields available for editing consist of the Address or MTR ID of the unit to be contacted, the port of that MTR to be queried, and three flags for the actions to be performed. These actions are an X flag for including (or executing) that MTR in the AMR session, a C flag for clearing the data stored on the MTR, and a S flag for setting the data stored on the MTR. The "C" and "S" flag settings are available in the menu, but the clear and set actions are not currently implemented.

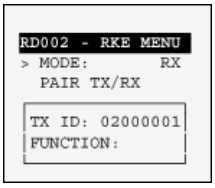



To select the field for editing, first scroll the list up or down until the index of the desired MTR entry is shown next to the EDIT MTR: prompt. The values for that entry will be displayed in the editing area. Choose the field to edit by using the right and left arrow keys to select the item (ID, Port, X, C, or S Actions). Once selected, press ENTER. Each field can be adjusted by scrolling through the values with the up and down arrow keys. The MTR ID is edited one character at a time by using the right and left arrow keys to select the digit and the up or down arrow keys to change the value. After setting the value for any of the fields, press ENTER to accept the changes. Continue scrolling through the other fields or leave the MTR Menu by pressing the ESC key. The 4-byte ID field is edited one character at a time with a range of 0 to F on each character. The port can be set from 0 to 5. The flag fields are set to 0 for OFF; X, C, or S for ON.



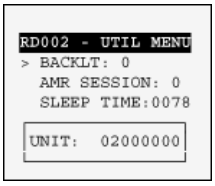

### RKE Menu

This menu provides a number of user-oriented functions and is directly compatible with the RKE protocol functions. The mode can be used to switch the RDR unit between transmit (TX) and receive (RX) modes of the MAX7032 transceiver. The PAIR TX/RX function allows the RDR to act as an RKE transmitter or an RKE using the ASK data frame. Once the RDR unit is paired with another RDR unit, it will flash different-colored backgrounds on the display, based on the function transmitted. This menu also displays the ID and function of any transmitter that is broadcasting the defined ASK frame. This menu can be helpful in debugging any transmitters using the RKE or AMR protocol.




### Utility Menu

The utility menu available in the RDR system allows for basic setting and interaction of the user with internal microcontroller values. This menu shows the default backlight color (0 = none, 1 = red, 2 = green (default), and 3 = amber), the "index" of the last communication attempt with a MTR in the MTR LIST, the sleep time setting (adjustable in 30s increments with 120s as the default), and the ID of the RDR unit itself.




## Firmware Structure

### MTR

Functional operation of the MTR and RDR systems is very similar. The MAXQ610 provides a number of inputs to the MAX7032, such as SCLK, DIO, and CS for the SPI interface and DATA, ENABLE, TR, and RSSI for various other RF controls. The purpose of the MTR unit is twofold: first is for the microcontroller to act as a manager and collect data from the meter system; second is to control the radio and communicate the data upon request. One user input is available to force the MTR unit to transmit a basic data frame. The MAXQ610 and the MAX7032 are configured to be in a "stop mode" unless one of three interrupts occur: a wakeup command from the timer, a switch press from the user, or an incoming edge from the meter port. The last two items will cause an external interrupt to be triggered, whereas the wakeup timer generates an internal interrupt.

The interrupt is serviced by the microcontroller, which decodes the source of the interrupt, then takes the appropriate action. A switch press event will cause the microcontroller to go directly into transmit mode and send an ID frame. A positive edge on the meter port is accumulated in a counter and stored in memory until the information is requested. A wakeup command is the most complex of the three processes and involves timers, branching decisions, and possibly both modes (RX and TX) for the MAX7032. See Appendix II for the MTR firmware code.

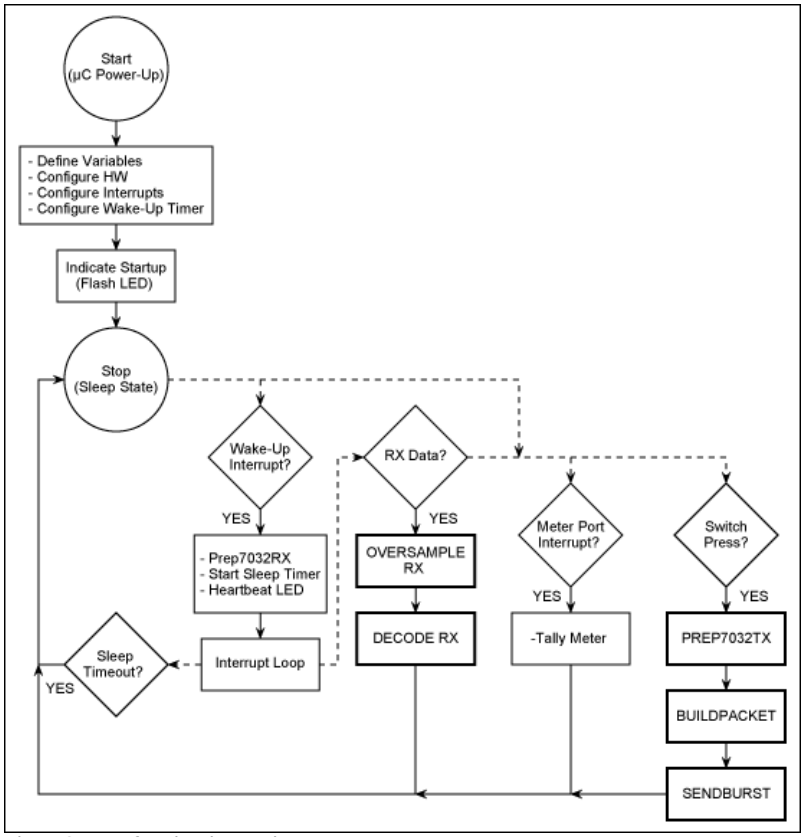



*Figure 2. MTR functional operation.*

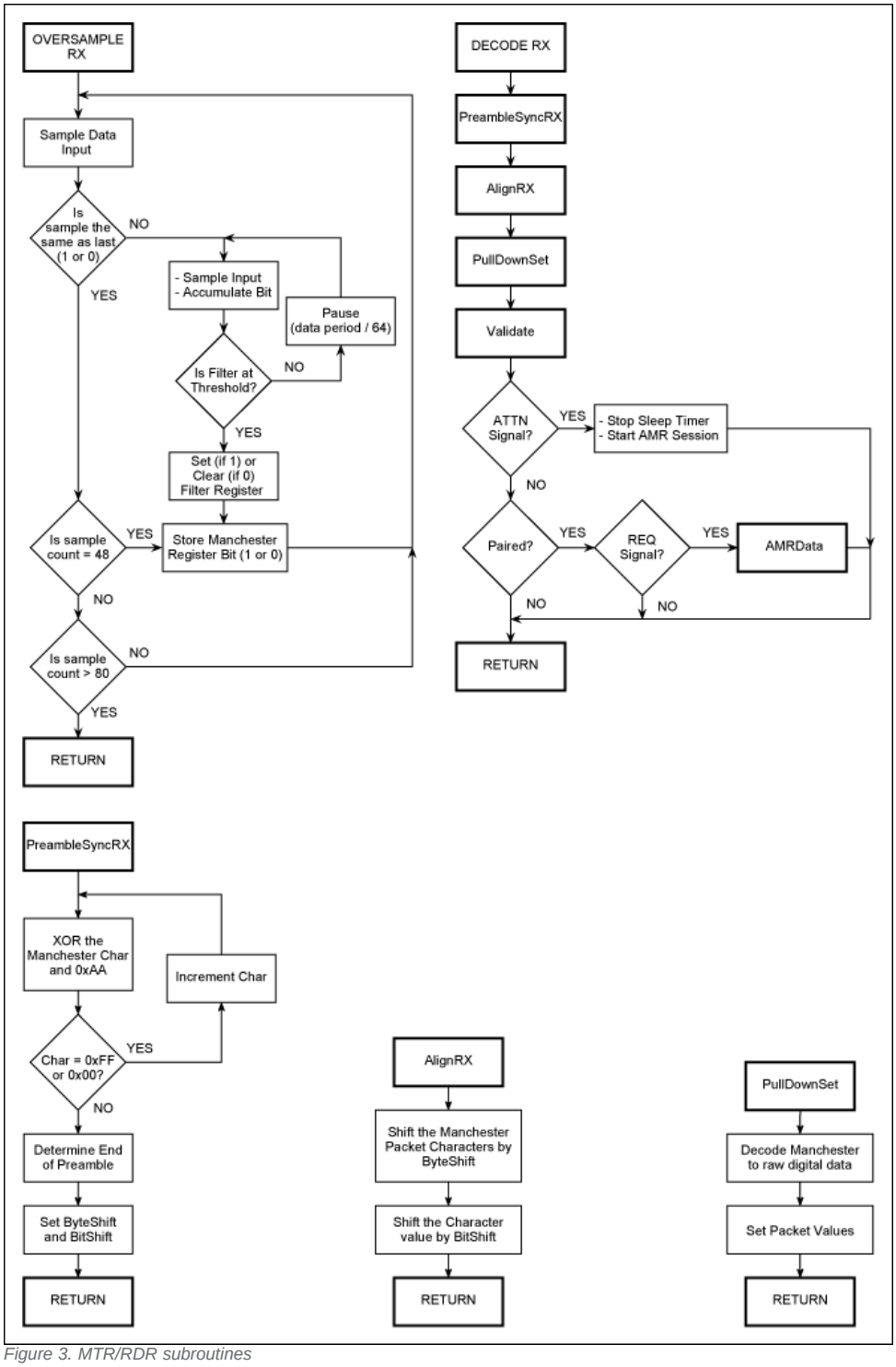



*Figure 3. MTR/RDR subroutines*

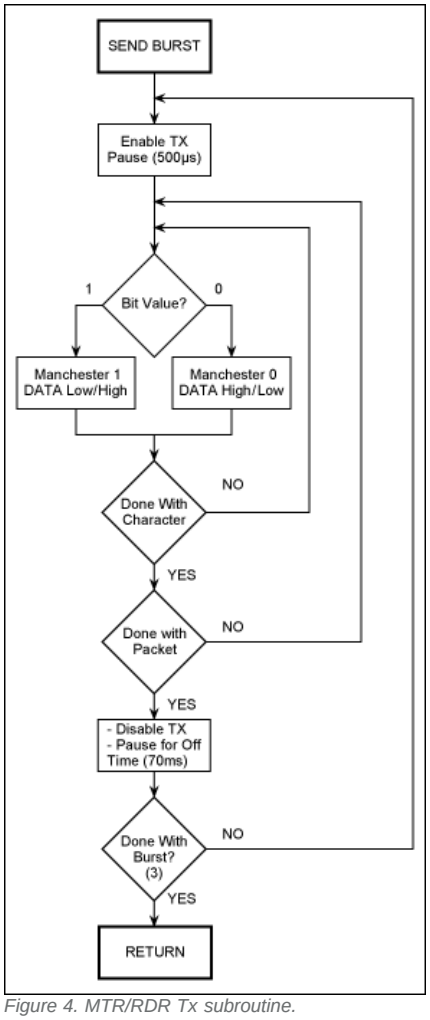



*Figure 4. MTR/RDR Tx subroutine.*

### RDR

As noted, the functional operation of the RDR and MTR systems are very similar. The MAXQ610 microcontroller has the same interface to the MAX7032 transceiver to simplify and share firmware code between the two units. In the RDR unit, there is a great deal of user interface functionality in addition to the operational aspects of the radio. The RDR unit acts as the master in the communication process, initiating the AMR session, requesting frames from each MTR unit in turn, and releasing the MTR units back to their S/L cycle.

Human interfacing is provided through the Electronic Assembly DOGS102-6 pixel display, along with the backlighting, an RSI LED, and the menu keys. The display operates with an SPI interface and thus shares the SCLK and DIO lines with the MAX7032 and the MAX7359. The key controller operates with an I²C interface and shares the clock and data lines from the SPI devices as well as providing an interrupt signal to the microcontroller.

The interrupts are serviced by first determining the source of the interrupt then operating with that source appropriately. A key press event will cause the microcontroller to "pop" the MAX7359 stack, a switch press will wait for about 4s and reset the RDR system. An incoming transmission will be decoded as needed, if the RDR is configured in RX mode for either an AMR session or RKE debug use. Raw transmissions can be sent from the RDR system if the RKE mode is set to TX and the user presses the TX key. See Appendix III for the RDR firmware code.

## Microcontroller Programming SW

The firmware in this reference design was developed within the IAR Embedded Workbench® software. A full version of this software (4k KickStart Edition) can be obtained at www.iar.com/. The IAR EW works in concert with the MAXQUSB-JTAG interface and the programming adapter, to flash the MAXQ610 on both the TX and RX boards.

### Installation

Please refer to the IAR EW documentation for installation and guidance. The firmware in this projected was developed with MAXQ plug-in: IAR

EWMAXQ2.20I.

### Operation

Be sure to have the USB port properly configured within the IAR EW: Project → Options... General Options—Debugger—JTAG. The COM Port should be set to match the "USB Serial Port (COM XX)" as indicated in the Windows® Device Manager.

For best performance during programming and debug of the MAXQ610, we suggest adjusting the Advanced Settings for the COM port (Device Manager → USB Serial Port (COM XX) Properties → Port Settings → Advanced...). Recommended values for receive and transmit buffer sizes are 512 bytes and a recommended latency timer of 4ms should provide optimal operation.

The USBJTAG translator (MAXQJTAG-USB) board uses a FTDI UART and a MAXQ2000 to convert the PC's serial port into a JTAG port. This board's firmware should be checked for the latest version. To determine the firmware revision, connect the USBJTAG board to a USB port; open the Microcontroller Tool Kit (MTK2) software; select Dumb Terminal in the Select device window during startup (click OK); choose Options → Configure Serial Port and select the appropriate COM port and choose the 115200 Speed (click OK); choose Target → Open COMXX...; hit Enter a few times; type q and hit Enter. The terminal should read back the firmware revision as "83."

A JTAG interface adapter (JIA) must be used to program these systems because of the different supply levels. The USB port provides a 5V power supply that must be regulated down to the 3.3V level needed on the RD002 boards. One of these JIA boards is included in the LFRD002 kit.

The edge connector should be oriented with JTAG pin 1 on the top of the MTR or RDR boards. The system can be programmed with batteries in place or without batteries installed (3.3V regulated USB power is used). Within the C code, there are some lines that can be uncommented to help with debug of firmware changes.

## Operational Setup and Use

### AMR Functionality

#### Reviewing/Editing the MTR List

The RDR has been configured to communicate with the each of the two MTR units included in the kit. These communication entries can be reviewed or adjusted as described in the previous **MTR Menu** section. Under the `AMR Menu`, select the `MTR List` to review or edit the entries. The default entries are listed as the following:

```
EDIT MTR:
1       02000001.0 X00
2       02000002.0 X00
3       02000002.1 X00
4       02000003.0 000
...     ...
```

The first entry in this list (`EDIT MTR: 1`) is referring to the first MTR unit (with ID `02000001`), Port 0 (`.0`), and the action to be taken is to include this unit in the communications process (indicated by the `X` field). The second entry is addressing the second MTR unit (with ID "02000002"), Port 0 (".0"), and again, this unit is included in the communications list the ("X" entry). The third item is indicating the second MTR unit again, this time Port 1 is addressed. Finally, the fourth entry is referring to an undefined MTR unit (ID ...03), and it is set to take no action on that unit (indicated by the "0" entry in the execute field).

#### Communicating with the Meters

To actually communicate with the MTRs listed, the user must initiate the communications process. Before initiating the communication, the user can choose between a fully automated communication session (referred to as a batch process) or a partial user-controlled session (called a sequential process). These two settings are controlled by the AMR MODE listed in the AMR MENU. To change between the two settings, simply select the AMR MODE line and press ENTER.

### RKE Functionality

#### Pairing the MTR and RDR Units

In this reference design, each of the MTR units has been preprogrammed with a semi-unique ID (0x02000001 and 0x02000002). This 4-byte value has been flashed into the MAXQ610 as part of the FW encoding process and can be easily changed by the user by reprogramming the microcontroller. Similarly, the RDR unit has a preprogrammed ID (0x02000000) that can be changed by the user with a FW edit. This system does not require any form of "pairing" in the traditional sense, because the use of this communication protocol only requires the RDR unit to know the ID of the MTR unit of interest. Other MTRs can be added to the MTR List by the user and included in the AMR session.

The RKE menu items do provide an RKE-style pairing function. Therefore, if the RDR is to be used for debugging purposes or for decoding information from a MTR unit or another RDR, the unit can be paired with any TX unit by selecting the "PAIR TX/RX" item in the RKE menu. The RDR unit will wait until a valid transmission is received and will then link with that unit ID, displaying unique background colors for the different key fob switches pressed.

To link the RDR system to the transmitter system, the user simply needs to set the "PAIR TX/RX" item in the RKE menu and send a valid signal with the transmitter to be paired. This process must be repeated whenever the batteries are replaced on the RDR system.

1. Select the "PAIR TX/RX" item in the RKE menu.
2. Press any button on the TX system (MTR or RDR) to be paired.
3. The RDR system should cycle the backlight colors when a valid frame is received.
4. The two systems are now paired, test the link with any button press on the TX

### Range

The predicted range in a flat unobstructed outdoor area is based on the following assumptions.

$f_0$ = 433.92MHz
$P_{PA}$ = +10dBm
$G_T$ = -18dBi (small loop antenna typical -18dBi)
$h_{TX}$ = 1m
$h_{RX}$ = 1m
$G_R$ = 4.14dBi (ideal 1/4λ antenna = 5.14dBi)
$L_{ConR1}$ = -0.57dB
Path loss varies as $R^{-4}$ because of ground bounce interference
RX sensitivity set at -114dBm

The calculated estimate of "open field" range is approximately 370m (see application note 5142, "Radio Link-Budget Calculations for ISM-RF Products" for more information).

Indoor range testing resulted in a consistent useable distance of 30m using a Linx reduced-height antenna connected to the RDR; 35m was achieved with a 1/4λ antenna on the MTR, with the MTR unit placed ~2m above the floor (cube wall); 30m was also reached in a lab environment with the MTR positioned 1m above the floor (bench top).

## Battery Usage Analysis

### Microcontroller

[1.8V nominal core voltage, 1.0V RAM (min) data retention/power-on-reset voltage]
The MAXQ610 microcontroller burns a maximum 12µA (with power-fail off) during "stop" mode.
The MAXQ610 microcontroller (with 12MHz SysClk) burns a maximum of 5.1mA during normal operation.

### TRX

[2.1V to 3.6V operation]
The MAX7032 transceiver burns a maximum 8.8µA (3V, 85°C) during sleep mode.
The MAX7032 transceiver during TX operation burns a typical 12.4mA at 434MHz (max 20.4mA) with "always on"; when running at 50% duty cycle at 434MHz, it burns a typical 8.4mA (ASK) and a maximum of 13.6mA.
The MAX7032 transceiver typically burns 6.4mA at 434MHz, ASK (3V, 85°C) with a maximum of 8.3mA during RX operation.

### Key Controller

[1.62V to 3.6V operation]
The MAX7359 key switch controller burns a maximum 5µA during sleep mode.
The MAX7359 key switch controller burns typical 25µA, maximum 60µA; with one key pressed the controller burns a typical 45µA during normal operation.

### MTR

LED: configured with 75Ω limiting resistors, yellow ($V_F$ = 2.2), nominal 10.7mA on current.

The average MTR system sleep current was measured to be 233µA. If left in sleep mode consistently, using a 750mAh, CR2 battery, the MTR unit would last about 134 days. For the sleep/listen duty cycle of 14.3% (3.0s sleep/0.5s listen), with the microcontroller and MAX7032 cycling, the measured current averaged a maximum of 11.6mA. Using the CR2 battery, the typical MTR life (without any TX communication) would be approximately 403hrs [11.6mA × 0.143 + 0.233mA × 0.857 = 1.859mA] or about 16.8 days.

The average maximum current during communications (TX/RX with the RDR unit) was approximately 15.62mA. If a 10s communication session was held once a day, the overall life of the CR2 battery would be reduced by 0.73mA/day—a negligible amount of time.

This system has not yet been optimized for low-current "stop" mode.

### RDR

In addition to the same radio configuration as above, the RDR unit also has the MAX7359 used for key inputs and the DOGS serial interface display with backlighting. A standby mode is implemented, but the current consumption of the RDR unit has not been measured under varying conditions.

This system has not yet been optimized for mid-low current "standby" mode or low-current "stop" mode.

## Hardware Details

### Transceiver Specifications

| | |
|---|---|
| Supply current ($I_{DD}$) at $f_{RF}$ = 433MHz, TX 50% duty cycle | 8.4mA (typ) |
| Output power ($P_{OUT}$) into 50Ω | +10.0dBm (typ) |

### Receiver Specifications

| | |
|---|---|
| Supply current ($I_{DD}$) at $f_{RF}$ = 433MHz | 6.4mA (typ) |
| Deep-sleep supply current ($I_{DD}$) at $f_{RF}$ = 433MHz | 0.8µA (max) |
| Sensitivity (average power level) | -113dBm (typ) |

## Component List

The following table provides a list of components used to populate both the MTR and RDR boards. Maxim recommends high-quality, wire-wound inductors for components used on both boards.

**MTR**

| Designation | Qty | Description |
|---|---|---|
| C49–50 | 2 | CAP, 0.01µF, 10% |
| C21–22 | 2 | CAP, 0.047µF, 10% |
| C5–6 C51 | 3 | CAP, 0.1µF, 10% |
| C4 | 1 | CAP, 1.0µF, 10% |
| C2 | 1 | CAP, 1.8pF, 5% |
| C1 C7–10 | 5 | CAP, 100pF, 5% |
| C29 | 1 | CAP, 10pF, 5% |
| C20 | 1 | CAP, 1500pF, 5% |
| C3 C26 C31 C33–34, C45 | 6 | CAP, 220pF, 10% |
| C27 | 1 | CAP, 470pF, 10% |
| C28 | 1 | CAP, 6.8pF, 5% |
| C30 | 1 | CAP, 6.8pF, 5% |
| C52 | 1 | CAP, 680pF, 10% |
| Y3 | 1 | CERAMIC-SMD, 12.000MHz |
| F2 | 1 | FLT\MURATA\SFTLA10M7FA00-B0, 10.7MHz |
| L4 | 1 | IND-MOLDED, 10nH, 5% |
| L2 | 1 | IND-MOLDED, 16nH, 5% |
| L10–12 | 3 | IND-MOLDED, 22nH, 5% |
| L1 | 1 | IND-MOLDED, 68nH, 5% |
| D-MRSI | 1 | LED-1 |
| U4 | 1 | MAX7032 |
| U2 | 1 | MAXQ610A-0000+ |
| R1 R9 RBAT1 | 3 | RES, 0Ω |
| R22–26 R29–30 | 7 | RES, 100Ω |
| R13–14 | 2 | RES, 100kΩ |
| R15 | 1 | RES, 10kΩ |

| | | |
|---|---|---|
| R7 | 1 | RES, 1MΩ |
| R-MRSI | 1 | RES, 75Ω |
| R4 | 1 | RES, Open |
| J-MX-ANT | 1 | RF-SWITCH-SWF |
| SW-A | 1 | SW-SPST-NO-B, SPST NO |
| Y1 | 1 | XTAL-SMD, 17.63416Mhz |

**RDR**

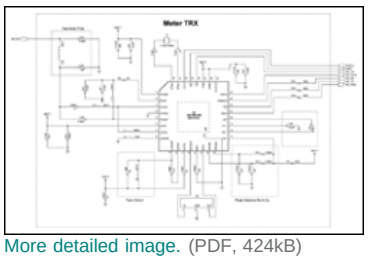
| Designation | Qty | Description |
|---|---|---|
| | 1 | BOX, Series 55 enclosure |
| | 1 | 433MHz Antenna |
| C44 C46 | 2 | CAP, 0.01µF, 10% |
| C14 C32 | 2 | CAP, 0.047µF, 10% |
| C24-25 C47 C53-56 | 7 | CAP, 0.1µF, 10% |
| C23 | 1 | CAP, 1.0µF, 10% |
| C12 | 1 | CAP, 1.8pF, 5% |
| C11 C35-38 | 5 | CAP, 100pF, 5% |
| C42 | 1 | CAP, 10pF, 5% |
| C13 | 1 | CAP, 1500pF, 5% |
| C15 C17-19 C39 C43 | 6 | CAP, 220pF, 10% |
| C40 | 1 | CAP, 470pF, 10% |
| C16 | 1 | CAP, 6.8pF, 5% |
| C41 | 1 | CAP, 6.8pF, 5% |
| C48 | 1 | CAP, 680pF, 10% |
| Y4 | 1 | CERAMIC-SMD, 12.000MHz |
| U6 | 1 | DISPLAY/DOGS102-6 |
| F1 | 1 | FLT\MURATA\SFTLA10M7FA00-B0, 10.7MHz |
| L6 | 1 | IND-MOLDED, 10nH, 5% |
| L5 | 1 | IND-MOLDED, 16nH, 5% |
| L7-9 | 3 | IND-MOLDED, 22nH, 5% |
| L3 | 1 | IND-MOLDED, 68nH, 5% |
| D-RSSI | 1 | LED-1 |
| U1 | 1 | MAX7032 |
| U5 | 1 | MAX7359ETG+ |
| U3 | 1 | MAXQ610A-0000+ |
| R3 R5 RBAT2 | 3 | RES, 0Ω |
| R11-12 R17-21 | 7 | RES, 100Ω |
| R8 R10 | 2 | RES, 100kΩ |
| R6 | 1 | RES, 1MΩ |
| R2 | 1 | RES, 4.7kΩ |
| RRSSI | 1 | RES, 75Ω |
| R16 | 1 | RES, Open |
| J-RX-ANT | 1 | RF-SWITCH-SWF |
| J-RF-IN | 1 | SMA |

| | | |
|---|---|---|
| SW-D SW-ENT SW-ESC SW-L SW-PWR SW-R SW-TOP SW-U | 8 | SW-SPST-NO-B, SPST NO |
| Y2 | 1 | XTAL-SMD, 17.63416Mhz |

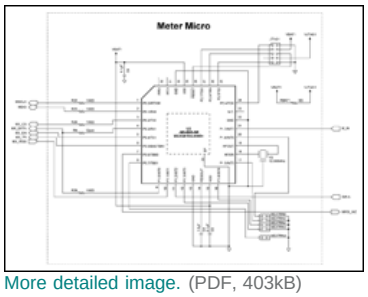
## Schematics

(Revision C1: detailed 11"x 17" copy available here)

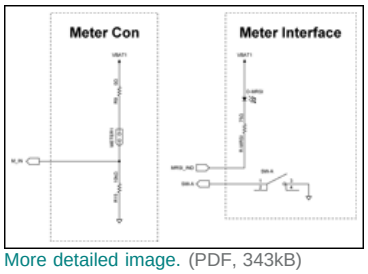
MTR Blocks



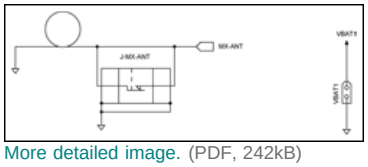

More detailed image. (PDF, 424kB)




More detailed image. (PDF, 403kB)

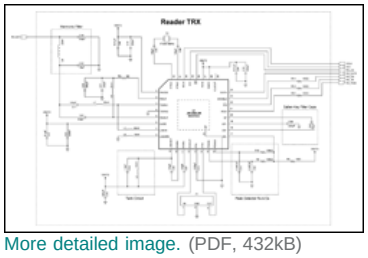



More detailed image. (PDF, 343kB)

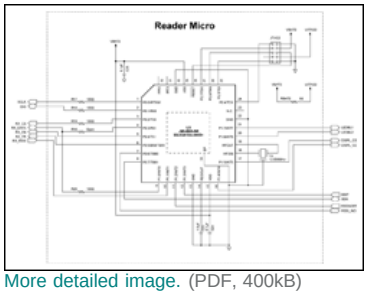


More detailed image. (PDF, 242kB)

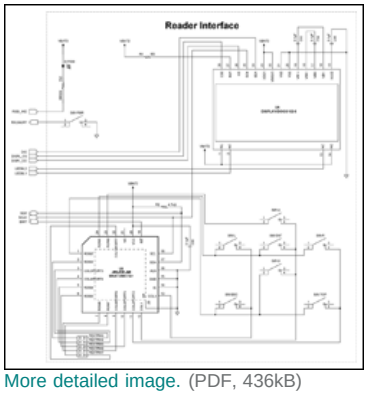
RDR Blocks



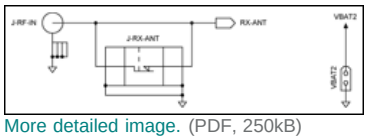

More detailed image. (PDF, 432kB)



More detailed image. (PDF, 400kB)



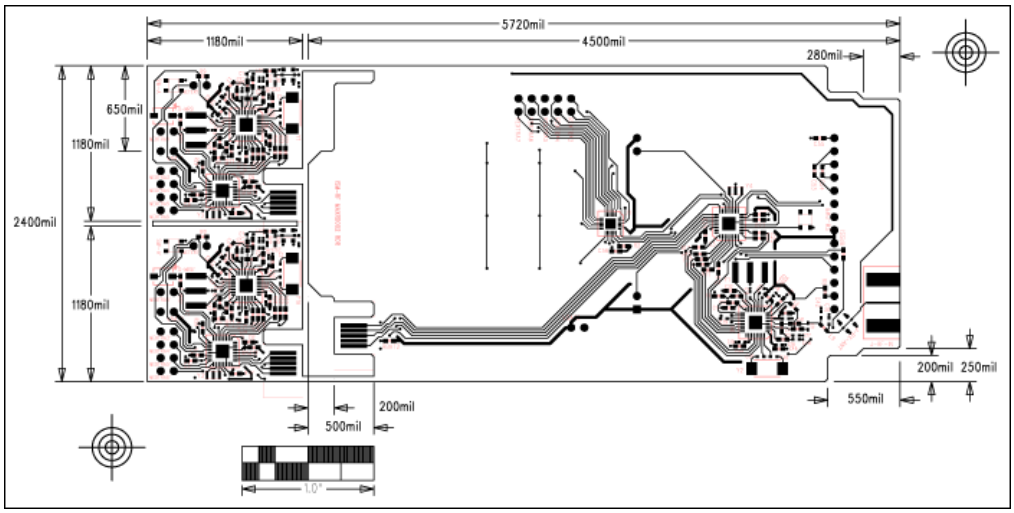
More detailed image. (PDF, 436kB)

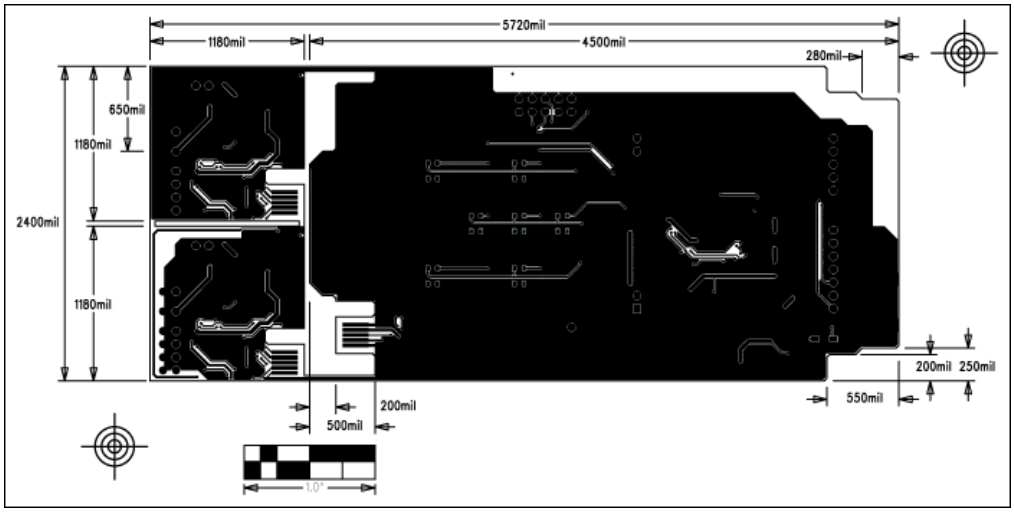


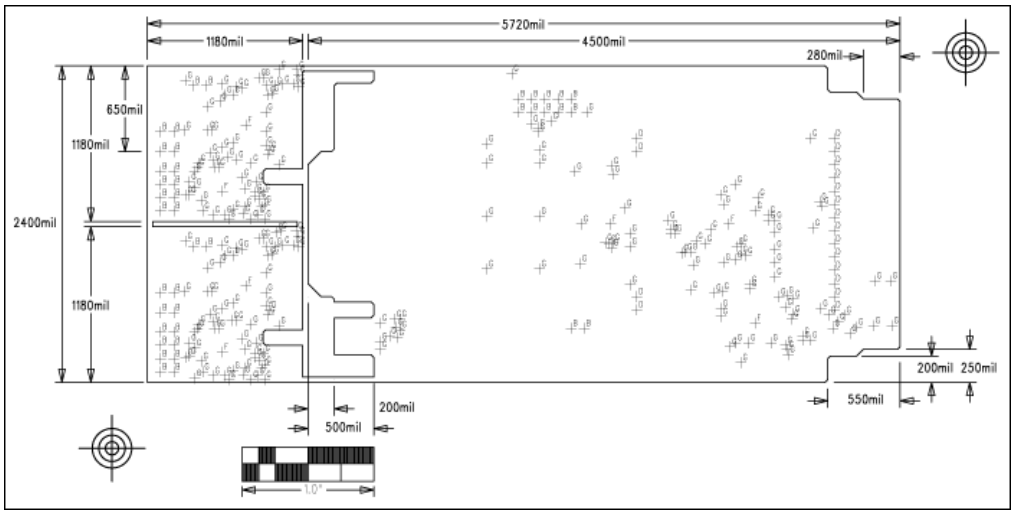
More detailed image. (PDF, 250kB)

## Layout

(Revision C1: detailed scale plots available here.)






## References


1. MAXQ610 data sheet
2. MAXQ Family User's Guide
3. MAXQ610 User's Guide
4. MAXQ610 Revision A3 Errata
5. MAXQ610 evaluation kit
6. MAXQ USB-to-JTAG evaluation kit
7. Application note 4465, "Using the Serial Port on the MAXQ610 Microcontroller"
8. Application note 4314, "Getting Started with the MAXQ610 Evaluation Kit (EV Kit) and the IAR Embedded Workbench"
9. MAX ISM-RF Ref Des 002 Schematic
10. ISM-RF Reference Design 002 Layout
11. IAR Embedded Workbench IDE User Guide
12. EWMAXQ_AssemblerReference.pdf, MAXQ IAR Assembler Reference Guide
13. EWMAXQ_CompilerReference.pdf, MAXQ IAR C Compiler Reference Guide
14. Electronic Assembly, DOGS displays


## Related Application Notes


Application note 2815, "Calculating the Sensitivity of an ASK Receiver"
Application note 3401 "Matching Maxim's 300MHz to 450MHz Transmitters to Small Loop Antennas"
Application note 3435 "Manchester Data Encoding for Radio Communications"
Application note 3671 "Data Slicing Techniques for UHF ASK Receivers"
Application note 4302 "Small Antennas for 300MHz to 450MHz Transmitters"

Application note 4314, "Getting Started with the MAXQ610 Evaluation Kit (EV Kit) and the IAR Embedded Workbench"
Application note 4465, "Using the Serial Port on the MAXQ610 Microcontroller"
Application note 4636 "Avoid PC-Layout "Gotchas" in ISM-RF Products"
Application note 5142, "Radio Link-Budget Calculations for ISM-RF Products"

## Appendix I: RKE Frame Structure

The basic structure of the data frame is ASK modulated, Manchester encoded, 4.8kbps (0.2083ms/bit), has 144 bits per frame (18 bytes or 9, 2-byte words), 30ms per frame, a pause of 70ms between frames, and 3 frame transmissions per burst. For information on Manchester encoding, refer to application note 3435, "Manchester Data Encoding for Radio Communications."

**Frame Structure**

| Preamble | | | | ID | | | | Function | | Data | | Sync | | Bat | Sig | Chk Sum | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FF | FF | FF | FD | 02 | 00 | 00 | 01 | 00 | 01 | 00 | 00 | 43 | 21 | 11 | 22 | 01 | 68 |

The structure of this frame is arbitrary but has been established to provide an example of the information that can be contained in any frame related to the many ISM-RF applications.

**Preamble**

| Preamble | | | |
|---|---|---|---|
| FF | FF | FF | FD |

The default preamble for this reference design is 4 bytes of high data with a pair of stop bits at the end. Since this is transmitted with Manchester encoding, the waveform appears as a 4.8kHz square wave lasting for 6.67ms. Having a preamble longer than roughly 1ms should provide ample time for the Rx system to wake up, given a strong received signal strength. The MAX1473 has a typical wake-up time of 250µs to start receiving valid data. Extra time can be padded onto this initial wake-up time to allow the Rx system to properly settle the baseband slicing circuit, which in turn provides optimum sensitivity. Providing a preamble of over 6ms gives ample opportunity for a relatively weak signal to wake up the receiver.

Assuming the MAXQ610 microcontroller used in the Rx is sitting in a stop mode, the system will consume a certain number of received bits to start up the microcontroller before it can begin decoding the data stream. The nanopower ring oscillator in the MAXQ610 typically runs at 8kHz (the wakeup timer interval can be $1/f_{NANO}$ to $65535/f_{NANO}$). If the Rx system is configured to use the RXSIG or the RXDATA line as an interrupt generator, the µC would have a warm-up time of $8192 \times t_{HFXIN}$. With $t_{HFXIN}$ = 83.3ns ($f_{XCLK}$ = 12MHz clock), the warm-up time works out to approximately 0.6827µs. This is well within the time available in one preamble transmission, and at a baud rate of 4.8kbps, the µC should be in a warmed-up state within 3.28 bits. The FD pattern at the end of the preamble is used to indicate the following start of the data frame and is the key to synchronizing the received bit stream.

**ID Structure**

| ID | | | |
|---|---|---|---|
| 01 | 23 | 45 | 67 |

The ID section is configured with 4 bytes of identification. This permits $2^{32}$ unique identifiers or over 4.29 billion codes. If one byte is used for class identification (256 different car models for example), $2^{24}$ unique identifiers or over 16 million codes remain. The structure can be modified to suit the purpose of the user. This RD has been preprogrammed to use an identification code as noted above, with the last byte being adjusted between RD systems. This allows for simultaneous operation of multiple, independent RKE systems.

**Function Structure**

| Function | |
|---|---|
| 00 | 01 |

This simple reference design has only four input switches on the transmitter, so 2 bytes of "functions" is overkill. Again, this structure can be modified to suit the purposes of the user. In this application, when button A is pressed, the function value would be 00 01. When button B is pressed, the function value would be 00 02, and so forth. The function value is used to convey information (with individual buttons having their own bit), so it is possible to press multiple buttons simultaneously. In that instance, if buttons B and D were pressed together, the function value would be represented as 00 0A (this multibutton operation has not been implemented in the design). This is an arbitrary definition for the structure of the function value, and it can always be modified by the user.

**Data Structure**

| Data | |
|---|---|
| 00 | 00 |

The data section of this frame is provided for transmitting information, such as a temperature or pressure measurement, a speed indicator, etc. In this design, the data section could work in concert with the function section to convey information whenever a button on the transmitter has been pressed. Again, the use of this data value is arbitrary and can be modified by the user. This operation is not currently implemented in the design.

| Sync Structure | |
|---|---|
| Sync | |
| 43 | 21 |

The synchronization block is set up to enable encryption coding. Users can work with this section to provide rolling code sync, a public key, etc. This operation is not currently implemented in the design.

| Battery Gauge Indicator |
|---|
| Bat |
| 11 |

This single byte allows the transmitter to send an indication of battery strength to the receiver. This section of the frame could have value when indicating a need to change the Tx battery. This operation is not currently implemented in the design.

| Signal Strength Indicator |
|---|
| Sig |
| 22 |

A possible use for a transceiver configuration, the received signal strength of the return channel could be shared between the nodes. This operation is not currently implemented in the design.

| Checksum | |
|---|---|
| Chk Sum | |
| 01 | 67 |

In this reference design, the checksum is used as a go/no-go gate for valid data. The frame values (except the preamble) are summed up during transmission, one byte at a time, and the full sum is tacked on to the end of the frame as the checksum. This value is compared to a received data stream, and a decision to use or discard the frame is made. The format of this checksum operation is arbitrary, but as long as the Tx/Rx and the encode/decode methods are equivalent, the checksum process will operate as intended.

## Appendix II: MTR Firmware

main.h (17 Sep 10, Rev 0.3):

```
/*******************************************************************************
 * Copyright (C) 1999-2010 Maxim Integrated Products, All Rights Reserved.
 *
 * See main.c for additional information.
 *
 *******************************************************************************/

/* Main Subroutines */
  void MasterInt();
  void AMRData(unsigned char MTRPort);

/* TX Subroutines */
  void SendBurst();
  void BuildPacket();

/* RX Subroutines */
  void DecodeRX();
  void OversampleRX();
  void PreambleSyncRX();
  void AlignRX();
  void PullDownSet();
  char PullDownPair(short int MPIndex);
  char Validate();
  char Paired();
```

```
/* 7032 Subroutines */
  void Init7032();
  void Prep7032TX();
  void Prep7032RX();
  void Prep7032Sleep();

/* SPI Subroutines */
  unsigned int Write7032Reg(unsigned char Adr, unsigned char Data);
  unsigned int Read7032Reg(unsigned char Adr);
  void ClockOutSPI(unsigned char InBit, unsigned int Delay);
  unsigned char ClockInSPI(unsigned int Delay);

/* Common Subroutines */
  void Lights7();
  void Pause(long int Count);
  void WriteFlash(unsigned int Address, unsigned int Data);
  unsigned int ReadFlash(unsigned int Address);
  void EraseFlash(unsigned int Address);
  void GoToSleep();
  void ExtISR();
  void Sleep();
  void WakeUp();

/* ------------------------------------------------------------------------- */
/* Global Constants */
  static int IntFlag = 0;
//  static int SleepTime = 0x0FA00;     // ~ 5.3sec sleep time (almost max), ~2F2B per sec
  static int SleepTime = 0x08D81;       // ~ 3sec sleep time
  static int SetHold = 29;              // default SPI setup and hold time, ~0.05ms
  static char WakeForSession = 0;       // Flag for RX
  static char FWRev = 0x04;             // Firmware revision

/* ------------------------------------------------------------------------- */
/* AMR Constants/Variables */
  static char AMRSession = 0;   // flag for active AMR session
  static char Meter[8][2] =    {{0x00, 0x00},   // initial values for each meter
                                {0x22, 0x22},
                                {0x00, 0x00},
                                {0x00, 0x00},
                                {0x00, 0x00},
                                {0x00, 0x00}};
/* ------------------------------------------------------------------------- */
/* TX Constants/Variables */
  static int PacketChars = 18;
  static char Preamble[4] = {0xFF, 0xFF, 0xFF, 0xFD},
              ID[4] = {0x02, 0x00, 0x02, 0x01},
              Func[2] = {0x00, 0x00},
              Data[2] = {0x00, 0x00},
              Sync[2]= {0x43, 0x21},
              Bat = 0x11,
              Sig = 0x22,
              CheckSum[2] = {0x00, 0x00};
  static char Packet[19];               // store the packet data as a string: 18 char + null
  static int BitMask[8] = {128, 64, 32, 16, 8, 4, 2, 1};  // store a bit mask
  static int TXWUCnt = 273;             // TX Warm-Up Count (~500us)
  static int ManCnt = 61;               // pause time for 1/2 Manchester bit
  static long int TXOffCnt = 40000;     // TX Off time Count

/* ------------------------------------------------------------------------- */
/* RX Constants/Variables */

  static char UnitID[4] = {0x02, 0x00, 0x02, 0x01};  // ID of the MTR unit
  static char ManPacket[39];  // store the Manchester data of with 2x18+1 char
  // packet (including full preamble) + 2 'noise' characters
  static int ByteShift = 0;
  static int BitShift = 0;
```

```
/* ------------------------------------------------------------------------- */
```

See the latest RD002-MTR*.zip file for all the firmware code.
main.c (17 Sep 10, Rev 0.3)
isr.c (17 Sep 10)

## Appendix III: RDR Firmware

main.h (21 Sep 10, Rev 0.3):

```
/*****************************************************************************
 * Copyright (C) 1999-2010 Maxim Integrated Products, All Rights Reserved.
 *
 * See main.c for additional information
 *
 *****************************************************************************/

/* Main Subroutines */
  void MasterInt();

/* AMR Subroutines */
  void AMRStart();
  void AMRRX();
  void AMRAttn();
  void AMRReq();
  void AMRAck(unsigned char Reset);
  void AMRClose();

/* RKE Subroutines */
  void RKERX();

/* MAXBee Subroutines */
  void Echo();
  char EchoCheck();

/* RX Subroutines */
  void DecodeRX();
  void OversampleRX();
  void PreambleSyncRX();
  void AlignRX();
  void PullDownSet();
  char PullDownPair(short int MPIndex);
  char Validate();
  char Paired();

/* TX Subroutines */
  void SendBurst();
  void BuildPacket();

/* 7032 Subroutines */
  void Init7032();
  void Prep7032TX();
  void Prep7032RX();
  void Prep7032Sleep();

/* Menu Subroutines */
  void MainMenu();    // Header
  void TopMenu();     // Page 0
  void AMRMenu();     // Page 1
  void MTRMenu();     // Page 2
  void RKEMenu();     // Page 3
  void UtilMenu();    // Page 4
  void CommMenu();    // Page 5
  void DataMenu();    // Page 6
  void MenuNav(unsigned char Key, unsigned char TopSub);
  void ListNav(unsigned char Key);
  void ColNav(unsigned char Key);
  void PrintEditMTR();
```

```
  void MenuEnt();
  void MenuEsc();
  void EditNav(unsigned char Key);

/* Display Subroutines */
  void InitDisp();
  void ClrDisp();
  void PrintDisp(unsigned char AlphaNum, unsigned char x, unsigned char y);
  void PrintByteDisp(unsigned char Byte, unsigned char x, unsigned char y) ;
  void PrintByteBDisp(unsigned char Byte, unsigned char x, unsigned char y) ;
  void PrintLineDisp(unsigned char AlphaNum[17], unsigned char y);
  void WriteEADogs(unsigned char ComDat, unsigned char Data);
  void DispSPI(unsigned char InBit);

/* 7359 and I2C Subroutines */
  void InitKeys();
  void WriteKeys(unsigned char Cmd, unsigned char Data);
  unsigned char ReadKeys(unsigned char Cmd);
  unsigned char MOSII2C(unsigned char SendByte);
  unsigned char MISOI2C();
  void StartI2C();
  void StopI2C();

/* SPI Subroutines */
  unsigned int Write7032Reg(unsigned char Adr, unsigned char Data);
  unsigned int Read7032Reg(unsigned char Adr);
  void ClockOutSPI(unsigned char InBit, unsigned int Delay);
  unsigned char ClockInSPI(unsigned int Delay);

/* Common Subroutines */
  void Lights7();
  void BackLight(unsigned char Mode);
  void Pause(long int Count);
  void WriteFlash(unsigned int Address, unsigned int Data);
  unsigned int ReadFlash(unsigned int Address);
  void EraseFlash(unsigned int Address);
  void GoToSleep();
  void WakeUp();
  void ExtISR();
  void Sleep();
  void WURX();

/* ------------------------------------------------------------------------- */
/* Global Constants/Variables */
  static char IntFlag = 0;  // Interrupt flag
  static char SetHold = 29; // default SPI setup and hold time, ~0.05ms
  static char TRXMode = 0;  // RX = 0, TX = 1
  static char BGColor = 2;  // Default background color
  static char SleepFlag = 0;  // Auto power down flag
  static char SleepTick = 0;  // count for seconds until sleep
  static int SleepTime = 120; // adjustable auto power-down time (seconds)
  static char FWRev = 0x05;   // Firmware revision

/* ------------------------------------------------------------------------- */
/* MAXBee Constants/Variables */
  static char TargetID[4] = {0x02, 0x00, 0x01, 0x04};  // ID of the "other" radio

/* ------------------------------------------------------------------------- */
/* AMR Constants/Variables */
  static char AMRSession = 0;   // flag for active AMR session
  // Initial MTR IDs
  static char MTRID[16][5] =   {{0x02, 0x00, 0x02, 0x00, 0x00},  // This Unit
                                {0x02, 0x00, 0x02, 0x01, 0x10},  // first MTR Unit ID
                                {0x02, 0x00, 0x02, 0x02, 0x10},
                                {0x02, 0x00, 0x02, 0x02, 0x11},
                                {0x02, 0x00, 0x00, 0x03, 0x00},
                                {0x02, 0x00, 0x00, 0x03, 0x00},
                                {0x02, 0x00, 0x00, 0x04, 0x00},
```

```
                                {0x02, 0x00, 0x00, 0x04, 0x00},
                                {0x00, 0x00, 0x00, 0x08, 0x00},
                                {0x00, 0x00, 0x00, 0x09, 0x00},
                                {0x00, 0x00, 0x00, 0x0A, 0x00},
                                {0x00, 0x00, 0x00, 0x0B, 0x00},
                                {0x00, 0x00, 0x00, 0x0C, 0x00},
                                {0x00, 0x00, 0x00, 0x0D, 0x00},
                                {0x00, 0x00, 0x00, 0x0E, 0x00},
                                {0x00, 0x00, 0x00, 0x0F, 0x00}};
  static char AMRMode = 1;  // flag for batch processing of AMR sessions
  static char MTRData[16][2] = {{0x00, 0x00}, {0x00, 0x00},
                                {0x00, 0x00}, {0x00, 0x00},
                                {0x00, 0x00}, {0x00, 0x00},
                                {0x00, 0x00}, {0x00, 0x00},
                                {0x00, 0x00}, {0x00, 0x00},
                                {0x00, 0x00}, {0x00, 0x00},
                                {0x00, 0x00}, {0x00, 0x00}};

/* ------------------------------------------------------------------------ */
/* RX Constants/Variables */

  static char UnitID[4] = {0x02, 0x00, 0x00, 0x00};  // ID of the RDR unit
  static char ManPacket[39];  // store the Manchester data of with 2x18+1 char
  // packet (including full preamble) + 2 'noise' characters
  static int ByteShift = 0;
  static int BitShift = 0;

  static char ID[4] = {0x02, 0x00, 0x00, 0x01},  // storage of the RX data
              Func[2] = {0x00, 0x00},
              Data[2] = {0x00, 0x00},
              Sync[2]= {0x00, 0x00},
              Bat = 0x00,
              Sig = 0x00,
              CheckSum[2] = {0x00, 0x00};

  static char PairID[4] = {0x02, 0x00, 0x00, 0x01};  // initial paired ID
  static char PairON = 0; // flag to indicate the pairing function

/* ------------------------------------------------------------------------ */
/* TX Constants/Variables */
  static int PacketChars = 18;
  static char Preamble[4] = {0xFF, 0xFF, 0xFF, 0xFD};
  static char Packet[19];               // store the packet data as a string: 18 char + null
  static int BitMask[8] = {128, 64, 32, 16, 8, 4, 2, 1};  // store a bit mask
  static int TXWUCnt = 273;             // TX Warm-Up Count (~500us)
  static int ManCnt = 61;               // pause time for 1/2 Manchester bit
  static long int TXOffCnt = 40000;     // TX Off time Count

/* ------------------------------------------------------------------------ */
/* Display Constants/Variables */
  static int DispSetHold = 2; // SPI setup and hold time for Display

  static char MenuPage = 0;   //  Page selection
  static char MenuRMin  = 2;  //  Minimum Row selection
  static char MenuRow  = 2;   //  Row selection
  static char MenuRMax  = 3;  //  Maximum Row selection
  static char MenuCol  = 0;   //  Column selection
  static char MenuSel  = 0;   //  Selection flag
  static char Inverse = 0;    //  0 = B on W,  1 = W on B
  static char ListBox = 0;    //  List box flag
  static char ListIndex = 0;  // List box index
  static char Edit = 0;       //  Edit flag
  static char IDEdit = 0;    //  Edit flag for the MTR ID field (reflects ID bit)
  static char DataBox = 0;    //  Data box flag

/* ------------------------------------------------------------------------ */
/* Font/Print Lines */
```

```
__data16 static unsigned char font6x8[0x50][6] = {   // initiate the 6x8 font
    {0x3E, 0x51, 0x49, 0x45, 0x3E, 0x00}, // 0 [0x00]
    {0x00, 0x42, 0x7F, 0x40, 0x00, 0x00}, // 1 [0x01]
    {0x42, 0x61, 0x51, 0x49, 0x46, 0x00}, // 2 [0x02]
    {0x21, 0x41, 0x49, 0x4D, 0x33, 0x00}, // 3 [0x03]
    {0x18, 0x14, 0x12, 0x7F, 0x10, 0x00}, // 4 [0x04]
    {0x27, 0x45, 0x45, 0x45, 0x39, 0x00}, // 5 [0x05]
    {0x3C, 0x4A, 0x49, 0x49, 0x30, 0x00}, // 6 [0x06]
    {0x41, 0x21, 0x11, 0x09, 0x07, 0x00}, // 7 [0x07]
    {0x36, 0x49, 0x49, 0x49, 0x36, 0x00}, // 8 [0x08]
    {0x06, 0x49, 0x49, 0x29, 0x1E, 0x00}, // 9 [0x09]
    {0x7C, 0x12, 0x11, 0x12, 0x7C, 0x00}, // A [0x0A]
    {0x7F, 0x49, 0x49, 0x49, 0x36, 0x00}, // B [0x0B]
    {0x3E, 0x41, 0x41, 0x41, 0x22, 0x00}, // C [0x0C]
    {0x7F, 0x41, 0x41, 0x41, 0x3E, 0x00}, // D [0x0D]
    {0x7F, 0x49, 0x49, 0x49, 0x41, 0x00}, // E [0x0E]
    {0x7F, 0x09, 0x09, 0x09, 0x01, 0x00}, // F [0x0F]

    {0x3E, 0x41, 0x49, 0x49, 0x7A, 0x00}, // G [0x10]
    {0x7F, 0x08, 0x08, 0x08, 0x7F, 0x00}, // H [0x11]
    {0x00, 0x41, 0x7F, 0x41, 0x00, 0x00}, // I [0x12]
    {0x20, 0x40, 0x41, 0x3F, 0x01, 0x00}, // J [0x13]
    {0x7F, 0x08, 0x14, 0x22, 0x41, 0x00}, // K [0x14]
    {0x7F, 0x40, 0x40, 0x40, 0x40, 0x00}, // L [0x15]
    {0x7F, 0x02, 0x1C, 0x02, 0x7F, 0x00}, // M [0x16]
    {0x7F, 0x04, 0x08, 0x10, 0x7F, 0x00}, // N [0x17]
    {0x3E, 0x41, 0x41, 0x41, 0x3E, 0x00}, // O [0x18]
    {0x7F, 0x09, 0x09, 0x09, 0x06, 0x00}, // P [0x19]
    {0x3E, 0x41, 0x51, 0x21, 0x5F, 0x00}, // Q [0x1A]
    {0x7F, 0x09, 0x19, 0x29, 0x46, 0x00}, // R [0x1B]
    {0x26, 0x49, 0x49, 0x49, 0x32, 0x00}, // S [0x1C]
    {0x01, 0x01, 0x7F, 0x01, 0x01, 0x00}, // T [0x1D]
    {0x3F, 0x40, 0x40, 0x40, 0x3F, 0x00}, // U [0x1E]
    {0x1F, 0x20, 0x40, 0x20, 0x1F, 0x00}, // V [0x1F]

    {0x3F, 0x40, 0x38, 0x40, 0x3F, 0x00}, // W [0x20]
    {0x63, 0x14, 0x08, 0x14, 0x63, 0x00}, // X [0x21]
    {0x03, 0x04, 0x78, 0x04, 0x03, 0x00}, // Y [0x22]
    {0x61, 0x51, 0x49, 0x45, 0x43, 0x00}, // Z [0x23]
    {0x00, 0x40, 0x38, 0x18, 0x00, 0x00}, // , [0x24]
    {0x00, 0x60, 0x60, 0x00, 0x00, 0x00}, // . [0x25]
    {0x08, 0x08, 0x08, 0x08, 0x08, 0x00}, // - [0x26]
    {0x08, 0x08, 0x3E, 0x08, 0x08, 0x00}, // + [0x27]
    {0x16, 0x16, 0x16, 0x16, 0x16, 0x00}, // = [0x28]
    {0x2A, 0x1C, 0x7F, 0x1C, 0x2A, 0x00}, // * [0x29]
    {0x20, 0x10, 0x08, 0x04, 0x02, 0x00}, ///[0x2A]
    {0x02, 0x01, 0x51, 0x09, 0x06, 0x00}, // ? [0x2B]
    {0x04, 0x02, 0x01, 0x02, 0x04, 0x00}, // ^ [0x2C]
    {0x00, 0x08, 0x14, 0x22, 0x41, 0x00}, // < [0x2D]
    {0x00, 0x41, 0x22, 0x14, 0x08, 0x00}, // > [0x2E]
    {0x00, 0x10, 0x20, 0x40, 0x20, 0x10}, // v [0x2F]

    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // space [0x30]
    {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}, // full block [0x31]
    {0x00, 0x00, 0x14, 0x00, 0x00, 0x00}, // : [0x32]
    {0x00, 0x40, 0x34, 0x00, 0x00, 0x00}, // ; [0x33]
    {0x00, 0x00, 0xFF, 0x00, 0x00, 0x00}, // | [0x34]
    {0x00, 0x00, 0xF8, 0x08, 0x08, 0x08}, // , - [0x35]
    {0x08, 0x08, 0xF8, 0x00, 0x00, 0x00}, // -, [0x36]
    {0x00, 0x00, 0x0F, 0x08, 0x08, 0x08}, // '- [0x37]
    {0x08, 0x08, 0x0F, 0x00, 0x00, 0x00}, // -' [0x38]
    {0x40, 0x60, 0x30, 0x18, 0x0C, 0x00}, // *M     [0x39]
    {0x7F, 0x7F, 0x30, 0x18, 0x0C, 0x00}, // *M     [0x3A]
    {0x7F, 0x7F, 0x00, 0x40, 0x60, 0x30}, // *M *A  [0x3B]
    {0x18, 0x0C, 0x40, 0x7F, 0x7F, 0x00}, // *A     [0x3C]
    {0x41, 0x63, 0x36, 0x18, 0x1C, 0x36}, // *X     [0x3D]
    {0x63, 0x41, 0x00, 0x7F, 0x7F, 0x00}, // *X *I  [0x3E]
```

```
    {0x7F, 0x7F, 0x00, 0x00, 0x00, 0x00}, // *M     [0x3F]

    {0xFF, 0xFF, 0xC3, 0xC3, 0xC3, 0xC3}, // 'B  [0x40]
    {0xE3, 0xF7, 0xBE, 0x1C, 0x00, 0x00}, //  B' [0x41]
    {0x03, 0x03, 0x03, 0x03, 0x00, 0x00}, //  E' [0x42]
    {0xF8, 0xFE, 0x0F, 0x03, 0x03, 0x03}, // 'O  [0x43]
    {0x03, 0x0F, 0xFE, 0xF8, 0x00, 0x00}, //  O' [0x44]
    {0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00}, // 'L  [0x45]
    {0xFF, 0xFF, 0x03, 0x03, 0x03, 0x03}, // 'D  [0x46]
    {0x3F, 0x3F, 0x30, 0x30, 0x30, 0x30}, // , B  [0x47]
    {0x30, 0x39, 0x1F, 0x0E, 0x00, 0x00}, //  B. [0x48]
    {0x30, 0x30, 0x30, 0x30, 0x00, 0x00}, //  E. [0x49]
    {0x07, 0x1F, 0x3C, 0x30, 0x30, 0x30}, // , O  [0x4A]
    {0x30, 0x3C, 0x1F, 0x07, 0x00, 0x00}, //  O. [0x4B]
    {0x08, 0x08, 0x08, 0x08, 0x08, 0x08}, // - [0x4C]
    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // space [0x0D]
    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // space [0x0E]
    {0x00, 0x00, 0x00, 0x00, 0x00, 0x00} // space [0x0F]
  };

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// Top Menu items
  static unsigned char MainMenuLn[17] =  {0x1B, 0x0D, 0x00, 0x00, 0x02, 0x30,
                                          0x26, 0x30, 0x16, 0x0A, 0x12, 0x17,
                                          0x30, 0x16, 0x0E, 0x17, 0x1E};

  static unsigned char AMRLn[17] =       {0x30, 0x30, 0x0A, 0x16, 0x1B, 0x30,
                                          0x16, 0x0E, 0x17, 0x1E, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char RKELn[17] =       {0x30, 0x30, 0x1B, 0x14, 0x0E, 0x30,
                                          0x16, 0x0E, 0x17, 0x1E, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char UtilLn[17] =      {0x30, 0x30, 0x1E, 0x1D, 0x12, 0x15,
                                          0x12, 0x1D, 0x22, 0x30, 0x16, 0x0E,
                                          0x17, 0x1E, 0x30, 0x30, 0x30};
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// AMR Menu items
  static unsigned char AMRMenuLn[17] =   {0x1B, 0x0D, 0x00, 0x00, 0x02, 0x30,
                                          0x26, 0x30, 0x0A, 0x16, 0x1B, 0x30,
                                          0x16, 0x0E, 0x17, 0x1E, 0x30};

  static unsigned char MTRListLn[17] =   {0x30, 0x30, 0x16, 0x1D, 0x1B, 0x30,
                                          0x15, 0x12, 0x1C, 0x1D, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char AMRModeLn[17] =   {0x30, 0x30, 0x0A, 0x16, 0x1B, 0x30,
                                          0x16, 0x18, 0x0D, 0x0E, 0x32, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char StartCommLn[17] = {0x30, 0x30, 0x1C, 0x1D, 0x0A, 0x1B,
                                          0x1D, 0x30, 0x0C, 0x18, 0x16, 0x16,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char DataListLn[17] =  {0x30, 0x30, 0x0D, 0x0A, 0x1D, 0x0A,
                                          0x30, 0x15, 0x12, 0x1C, 0x1D, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// MTR Menu items
  static unsigned char MTRMenuLn[17] =   {0x1B, 0x0D, 0x00, 0x00, 0x02, 0x30,
                                          0x26, 0x30, 0x16, 0x1D, 0x1B, 0x30,
                                          0x16, 0x0E, 0x17, 0x1E, 0x30};

  static unsigned char MTREditLn[17] =   {0x30, 0x30, 0x0E, 0x0D, 0x12, 0x1D,
                                          0x30, 0x16, 0x1D, 0x1B, 0x32, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// RKE Menu items
  static unsigned char RKEMenuLn[17] =   {0x1B, 0x0D, 0x00, 0x00, 0x02, 0x30,
                                          0x26, 0x30, 0x1B, 0x14, 0x0E, 0x30,
                                          0x16, 0x0E, 0x17, 0x1E, 0x30};

  static unsigned char ModeLn[17] =      {0x30, 0x30, 0x16, 0x18, 0x0D, 0x0E,
                                          0x32, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char TXIDLn[17] =      {0x34, 0x1D, 0x21, 0x30, 0x12, 0x0D,
                                          0x32, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x34};

  static unsigned char FuncLn[17] =      {0x34, 0x0F, 0x1E, 0x17, 0x0C, 0x1D,
                                          0x12, 0x18, 0x17, 0x32, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x34};

  static unsigned char PairLn[17] =      {0x30, 0x30, 0x19, 0x0A, 0x12, 0x1B,
                                          0x30, 0x1D, 0x21, 0x2A, 0x1B, 0x21,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char WaitLn[17] =      {0x30, 0x30, 0x20, 0x0A, 0x12, 0x1D,
                                          0x12, 0x17, 0x10, 0x30, 0x0F, 0x18,
                                          0x1B, 0x30, 0x1D, 0x21, 0x30};

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// Util Menu items

  static unsigned char UtilMenuLn[17] =  {0x1B, 0x0D, 0x00, 0x00, 0x02, 0x30,
                                          0x26, 0x30, 0x1E, 0x1D, 0x12, 0x15,
                                          0x30, 0x16, 0x0E, 0x17, 0x1E};

  static unsigned char BGColorLn[17] =   {0x30, 0x30, 0x0B, 0x0A, 0x0C, 0x14,
                                          0x15, 0x1D, 0x32, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char AMRSesLn[17] =    {0x30, 0x30, 0x0A, 0x16, 0x1B, 0x30,
                                          0x1C, 0x0E, 0x1C, 0x1C, 0x12, 0x18,
                                          0x17, 0x32, 0x30, 0x30, 0x30};

  static unsigned char SleepLn[17] =     {0x30, 0x30, 0x1C, 0x15, 0x0E, 0x0E,
                                          0x19, 0x30, 0x1D, 0x12, 0x16, 0x0E,
                                          0x32, 0x30, 0x30, 0x30, 0x30};

  static unsigned char UnitIDLn[17] =    {0x34, 0x1E, 0x17, 0x12, 0x1D, 0x32,
                                          0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x34};

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// Comm Menu items
  static unsigned char CommMenuLn[17] =  {0x1B, 0x0D, 0x00, 0x00, 0x02, 0x30,
                                          0x26, 0x30, 0x16, 0x1D, 0x1B, 0x30,
                                          0x0C, 0x18, 0x16, 0x16, 0x30};

  static unsigned char CommMTRLn[17] =   {0x30, 0x30, 0x0C, 0x18, 0x16, 0x16,
                                          0x30, 0x16, 0x1D, 0x1B, 0x32, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char DataLn[17] =      {0x34, 0x0D, 0x0A, 0x1D, 0x0A, 0x32,
                                          0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x34};

  static unsigned char MTRIDLn[17] =     {0x34, 0x16, 0x1D, 0x1B, 0x32, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x34};
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// Data Menu items
  static unsigned char DataMenuLn[17] =  {0x1B, 0x0D, 0x00, 0x00, 0x02, 0x30,
                                          0x26, 0x30, 0x0D, 0x0A, 0x1D, 0x0A,
                                          0x30, 0x16, 0x0E, 0x17, 0x1E};

  static unsigned char DataDispLn[17] =  {0x30, 0x30, 0x0D, 0x0A, 0x1D, 0x0A,
                                          0x30, 0x16, 0x1D, 0x1B, 0x32, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
// Misc
  static unsigned char BlankLn[17] =     {0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char MaximLn[17] =     {0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E,
                                          0x39, 0x3A, 0x3F, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30};

  static unsigned char Bold1Ln[17] =     {0x34, 0x30, 0x40, 0x41, 0x40, 0x42,
                                          0x30, 0x40, 0x41, 0x43, 0x44, 0x45,
                                          0x30, 0x46, 0x44, 0x30, 0x34};

  static unsigned char Bold2Ln[17] =     {0x34, 0x30, 0x47, 0x48, 0x47, 0x49,
                                          0x30, 0x47, 0x48, 0x4A, 0x4B, 0x47,
                                          0x49, 0x47, 0x4B, 0x30, 0x34};

  static unsigned char Box1Ln[17] =      {0x35, 0x4C, 0x4C, 0x4C, 0x4C, 0x4C,
                                          0x4C, 0x4C, 0x4C, 0x4C, 0x4C, 0x4C,
                                          0x4C, 0x4C, 0x4C, 0x4C, 0x36};

  static unsigned char Box2Ln[17] =      {0x37, 0x4C, 0x4C, 0x4C, 0x4C, 0x4C,
                                          0x4C, 0x4C, 0x4C, 0x4C, 0x4C, 0x4C,
                                          0x4C, 0x4C, 0x4C, 0x4C, 0x38};

  static unsigned char Box3Ln[17] =      {0x34, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
                                          0x30, 0x30, 0x30, 0x30, 0x34};
```

See the latest RD002-RDR*.zip file for all firmware code.
main.c (21 Sep 10, Rev 0.3):
isr.c (21 Sep 10):

## Appendix IV: RX Timing


## Appendix V: Preliminary MAXBee Protocol Definition

1. **Radio Nomenclature**
   - All radios are transceivers.
   - The meter consists of a water flow meter, a microcontroller that counts switch contacts (indicating a unit of water volume), and a transceiver that sends the flow information to a remote reader. The receiver part of the radio listens for commands from the reader.
   - The reader consists of a handheld display, a key (as in keypad) decoder for pushbutton commands, a microcontroller that processes the flow rate information sent by the meter and interfaces with the display, and a transceiver that sends commands to multiple meters and receives flow rate information from these meters.
   - Both the meter and reader are battery-powered, so they cannot be on all the time. The meter radio is smaller than the reader and has a smaller, lower-life battery, so its on-off duty cycle should be lower than the reader.

2. **Communication Protocol Overview**
   - A single reader will communicate with multiple meters. The meters are connected to water pipes and are therefore stationary. The reader can be fixed or mobile.
   - The purpose of the reader's communication link is to receive water usage information remotely from the meters via the radio link. The simplest way to do this is "on demand, " which means that the reader initiates the information-gathering process. A repetitive or scheduled reading, is more reliable and predictable, but would require time synchronization of all the meters and the reader.
   - All information between the reader and the meters is contained in a predefined frame, specifically chosen for the reader-meter demonstration system. This frame contains ID information, data, functions, plus preamble, synch, and basic error detection (checksum). All information transfer takes place through these frames.

3. **Communication Protocol Procedure**
    - All radio transmissions are done at a single frequency. There are no channels or frequency hopping.
    - Each meter periodically turns on its receiver to listen for a command from the reader. The period and duty cycle are the same for each meter, but because their clocks are not synchronized, there will be no structured on and off timing for the meters. It is possible to set each meter's duty cycle differently to achieve a condition that is closer to random listening periods. If the meter does not recognize a command from the reader during its "on" time, it turns off until its next "on" cycle.
    - The reader issues a Broadcast command when it wants to receive readings from the meters. The format of this command is such that it is recognized by every meter within range. When a meter receives this command, it goes into another mode, in which it keeps its receiver on continuously. Once all the meters have been set to a steady listening mode, the reader is ready to poll each meter. The steady listening mode is most likely determined by a time-interval setup on the reader that accounts for the time it takes for every meter to wake up in its low duty-cycle listening mode.
    - The reader issues sequential bursts of frames, each burst containing the ID code of the particular meter that is to report its water volume or flow data to the reader. Upon reception of its unique code, the meter transmits its data to the reader. All other meters will receive this transmission, but the frame will not contain their codes, so they will continue to keep their receivers on until they recognize their ID. After the reader confirms reception from a meter and displays or stores the information, it changes the ID number in its transmitted frame and starts the reading process from another meter.
    - Each meter keeps its receiver on after it has transmitted information. It reverts to its periodic low duty-cycle listening mode either after it has gone for a predetermined interval without receiving another frame with its ID, or upon receiving another Broadcast command from the reader telling it to revert to its low duty-cycle mode.



| Related Parts | | |
|---|---|---|
| MAX7032 | Low-Cost, Crystal-Based, Programmable, ASK/FSK Transceiver with Fractional-N PLL | Free Samples |
| MAX7359 | 2-Wire Interfaced Low-EMI Key Switch Controller/GPO | Free Samples |
| MAXQ610 | 16-Bit Microcontroller with Infrared Module | Free Samples |

**More Information**
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 5391: http://www.maximintegrated.com/an5391
REFERENCE DESIGN 5391, AN5391, AN 5391, APP5391, Appnote5391, Appnote 5391