Application Report
SLAA202B–February 2005–Revised December 2018

# Implementing IrDA With MSP430™ MCUs

*Melisa Nunez-Arzuaga and Andreas Dannenberg* *MSP430 Applications*

**ABSTRACT**

The development of wireless communications has occurred rapidly throughout the past decade. One of the standards used is the Infrared Data Association (IrDA) specification. The protocol introduced by this entity consists of three basic layers: IrPHY, IrLAP, and IrLMP, which supply a base for many other applications.

This application report implements the IrDA Lite protocol (IrPHY, IrLAP, and IrLMP) on select MSP430™ microcontrollers (MCUs), as well as Tiny Transfer Protocol (TTP) and IrCOMM 3-wire services as a passive secondary-only device. IrPHY implementations are provided using a Timer_A-based approach and also using the USCI_A hardware module.

The source code and software described in this application report can be downloaded from http://www.ti.com/lit/zip/slaa202.

**Contents**

**List of Figures**

## Trademarks

## 1 Introduction

It is helpful if the reader of this application report has some prior knowledge of the Infrared Data Association (IrDA) specifications. Some general information on the stack is provided in Section 5 but this is by no means interchangeable with the full documentation and specifications for IrDA.

This implementation follows the standards defined by IrDA Lite. The application uses IrPHY, IrLAP, IrLMP, TTP, and IrCOMM 3-wire services to implement an IrDA serial port connection as a passive, secondary-only device. When a primary IrDA peer transmits the string "t" to the MSP430 MCU, the MCU detects the string, reads the ADC internal temperature sensor, and responds with the temperature reading. Within this application report, three different projects are included:

- IrDA demonstration application running on an MSP430F149 device, using Timer_A to implement IrPHY encoding and decoding and using the ADC12 to obtain the temperature reading
- IrDA demonstration application running on an MSP430FG4619 device, using USCI_A0 to implement IrPHY encoding and decoding and using the ADC12 to obtain the temperature reading
- IrDA demonstration application running on an MSP430F2274 device, using USCI_A0 to implement IrPHY encoding and decoding and using the ADC10 to obtain the temperature reading

To provide a more complete solution, a demonstration application for Windows®-based PCs is also included. This application is written in the C programming language and shows how to establish an IrDA connection between a PC and the MSP430 IrDA stack using only standard Windows API calls.

## 2 Hardware Description

### 2.1 Hardware Overview

The hardware design for this application focuses on the interfacing of the MSP430F149, the MSP430FG4619, and the MSP430F2274 with the Sharp® GP2W0110YPSF IrDA transceiver device. Other MSP430 MCUs can also be used, depending on the requirements of the end application.

The Sharp GP2W0110YPSF was selected because it follows all ISO specifications for IrDA V1.0. The fact that this part can be used at a 3.0-V level is a benefit when interfacing with the MSP430 MCU, because no external circuitry is necessary to adapt the voltage levels. It also needs only three signals for interfacing with the microcontroller: transmit, receive, and shutdown. This leaves most of the MCU pins free for other purposes.

The DIr169 evaluation board is compatible with the MSP430F149-based software presented in this application report.

### 2.2 Circuit Description

The circuitry around the GP2W0110YPSF is simple (see Figure 1). Two pins are connected to the 3.0-V supply voltage (VCC and LEDA), one pin is connected to the common ground, and three pins interface to the MSP430 (TxD, RxD, and SD). Two bypass capacitors are placed in parallel between VCC and ground close to the transceiver to compensate for current pulses that occur when the transmit IR LED is operated. The TxD and RxD connections are made to Timer_A pins in case of the MSP430F149 (see Figure 1), and to USCI_A0 pins in case of the MSP430FG4619 (see Figure 2) and the MSP430F2274 (see Figure 3).

For the MSP430F149-based design, the device is sourced by a 32-kHz watch crystal that provides a reference clock to calibrate the internal high-speed DCO using a software FLL. This configuration is needed to meet the requirements of the IrPHY physical layer timing specification.

The MSP430FG4619-based design also uses a 32-kHz watch crystal. However, in this case, the built-in FLL circuit of the device clock module generates the system clock.

No external crystal is needed for the MSP430F2274-based design. In this case, the factory-provided DCO calibration constants are loaded into the DCO during device startup, to provide an accurate and stable system clock.

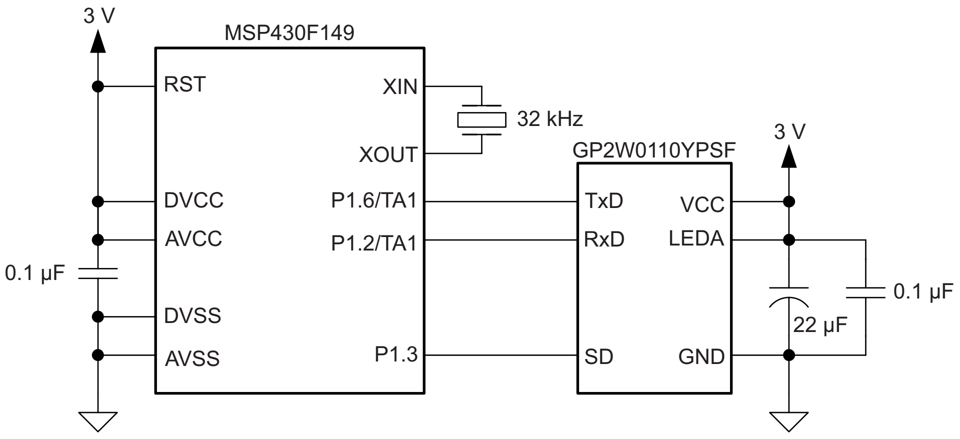

**Figure 1. Schematic Using MSP430F149**

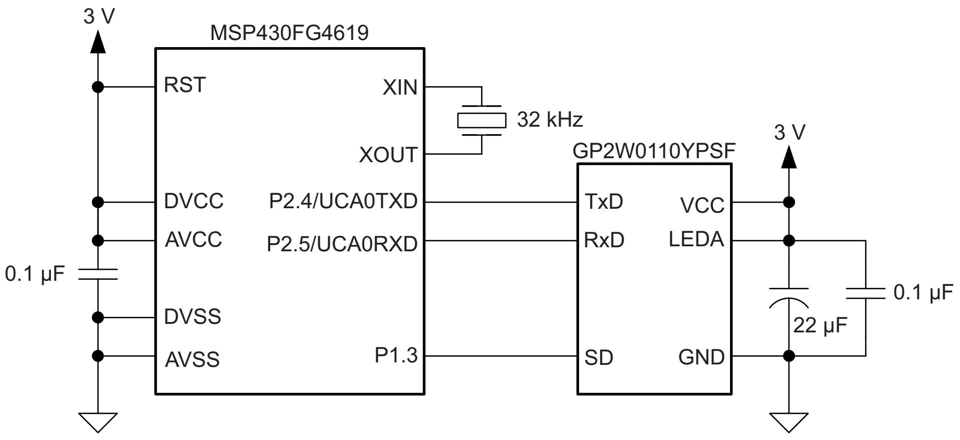

**Figure 2. Schematic Using MSP430FG4619**

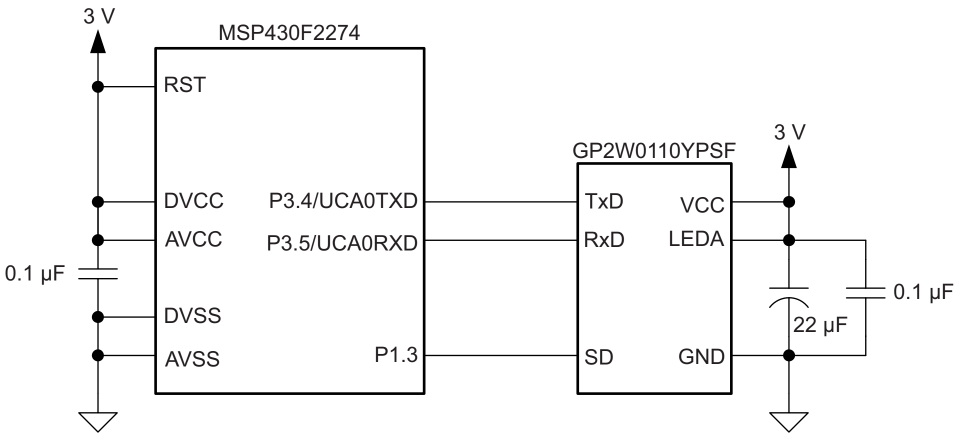

**Figure 3. Schematic Using MSP430F2274**

## 3 Software Description

This section provides a description of the implemented IrDA protocol stack. The entire demonstration application is written in assembly language and resides in one file. Functions have been named in ways which make it simple to understand which layer they belong to and what their functionality is. All service primitives for the IrPHY, IrLAP, and IrLMP layers have been implemented as specified by the IrDA Lite documentation. The TTP and IrCOMM 3-wire cooked services have been implemented to provide a demonstration of the working stack.

This chapter discusses two different implementation methods of the IrPHY layer. The first method uses Timer_A, and is utilized by the MSP430F149-based design discussed in this application report. The second method uses the USCI_A0 hardware communication module, and is used by both MSP430FG4619- and MSP430F2274-based designs. However all designs use the exact same IrDA communication algorithms.

### *3.1 Implementing IrPHY Layer Using Timer_A*

The Timer_A peripheral module found on all MSP430s can be used to implement the IrPHY layer, in case that no hardware module with IrDA support is available.

The approach is to design a unit that behaves as a UART but processes data received from and sent to the IR transceiver. The IR-pulse duty cycle is 3/16th of a bit period as specified in the IrPHY documentation. The transmission and reception schemes are similar.

#### 3.1.1 Transmission

For transmission, the initial XBOF flags are sent first followed by the BOF flag. Then the rest of the frame is transmitted. The FCS is calculated and sent after the whole frame has been sent, followed by the EOF flag. The actual sending of the individual bytes is performed by fitting each bit into 3/16th of a pulse, where a 0 is represented by a pulse and a 1 is represented by no pulse (see Figure 4). The transmission process is interrupt driven and is handled by the Timer_A module.

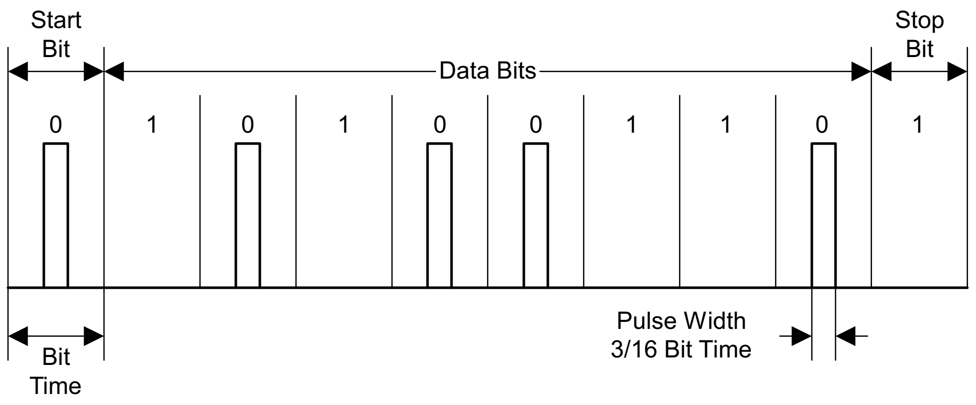

**Figure 4. IR Byte**

The actual encoding is manipulated in its entirety by the Timer_A module.

1. The bit length and 3/16th of the bit length are calculated given the speed at which the Timer_A clock source is operating, and they are stored as constants for use in the transmission process.
2. Timer_A is cleared and the clock source is set to SMCLK.
3. The pin being used for transmitting data is configured for the CCR1 peripheral function, and its direction is set to output.
4. The current state of TAR is stored in register CCR1 which, given that TAR has been previously cleared, holds a value of 0.
5. One bit length is added to the value stored in CCR1, and then CCR1 is copied into register CCR0.
6. The number of counts that equal 3/16th of a pulse width is added to CCR0.
7. The stop bit and start bit are added to the 8 bits of data to be transmitted.

8. The transmission counter is loaded with 10, to include the start and stop bits.
9. The CCR0 interrupt is enabled in the CCTL0 control register.
10. Output Mode 3 Set/Reset is selected for CCR1 to transmit the start bit.
11. Timer_A is started in continuous mode.
12. The CCIE flag is polled to determine when the transmission has ended, because the ISR clears the CCIE flag when all bits of the byte have been transmitted. The CCIE flag is polled by a subroutine inside the transmission routine and as soon as the CCIE flag is cleared, the transmission routine returns.
13. When TAR reaches CCR1, the output goes high and it returns back low when TAR reaches CCR0. This generates the start bit.

Figure 5 shows the switching between the output modes as well as the resulting output signals.

1. The Timer_A module first adds 1 bit length to CCR1 and CCR0. This keeps them 3/16th of a bit apart from each other but sets them up in time to transmit the next bit.
2. If the bit is 0, then it is sent using Output Mode 3 Set/Reset which produces a pulse, given that all bits must be sent inverted.
3. If the bit is 1, then Output Mode 3 is changed to Output Mode 5 Reset to make sure that the output is held low.
4. When the bit counter reaches 0, the interrupts for CCR0 are disabled and the ISR returns.

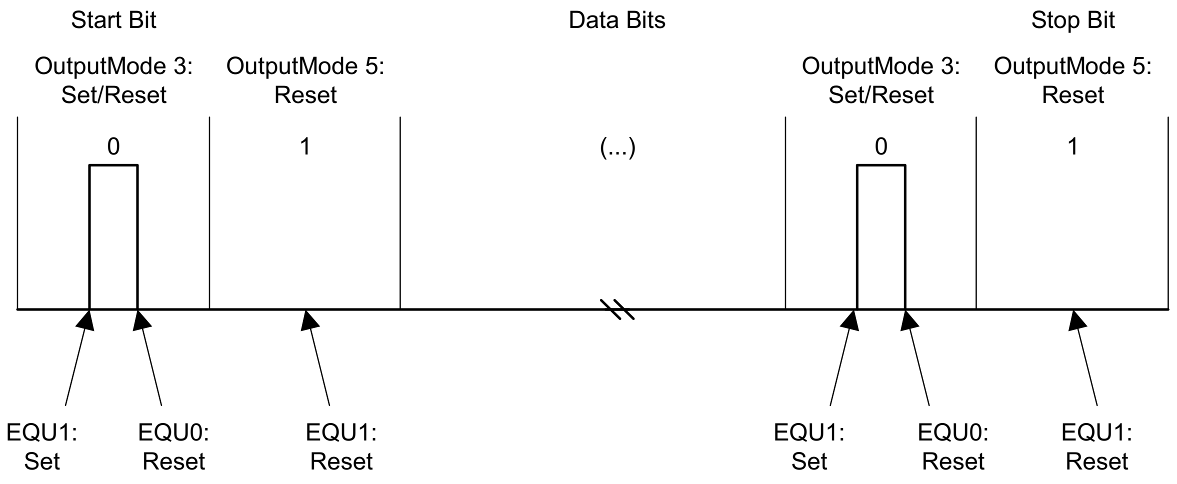

**Figure 5. Transmission With Timer_A**

### 3.1.2 Reception

Reception is based not on trying to capture the pulse when it comes in but is instead based on latching the input and testing for the presence of a pulse some time after the bit has been sent by the primary. Therefore, the interrupt flag for the port pin is checked before the time allocated for sending the bit is over. The main reason for this implementation is that 3/16th of a pulse does not allow much room for error and if reception is off by a small percentage, then more bits could be missed and the frame would be corrupted.

The pointer to the buffer is incremented, and the following bytes are stored through the same process until the EOF flag is encountered. Finally, the pointer is offset to the next location in memory. When idle, the application is always in receive mode waiting for an incoming start bit.

The process through which each byte is received depends on the IrDA_RX routine and the TA1_ISR. To receive:

1. A bit counter is loaded with 8 bits, the clock source for Timer_A is set to SMCLK, and the mode control bits are set to use the timer in continuous mode.
2. The capture/compare block 1 is configured to capture the falling edge of the input signal synchronized with the timer clock (SMCLK), and interrupts are also enabled.
3. When the ISR is done collecting the bits, it clears the CCIE flag, indicating that the reception of one byte has finished.

Every time the TA1_ISR is called, it adds the time to the next bit to CCR1 and tests CCTL1 for its capture/compare setting. If it is in capture mode, then the start bit is being handled. The routine adds an additional quarter of the bit length to the value in CCR1 which now holds a total of one and a quarter bit length and then sets the pin to I/O function. The I/O interrupt edge select is set to be triggered on a high-to-low transition (register P1IES), and the interrupt flag is cleared (register P1IFG) to clear the latched edge. After this, a switch is made to compare mode before returning from the ISR.

If the test for the capture/compare setting of CCTL1 results in a 0, there are incoming data bits. The P1IFG flag for the I/O pin is then tested. If the bit is set, a 0 was received, and if the bit is not set, a 1 was received. Next, the received bit is rotated through the use of an RRC instruction into the IR_DATA variable which holds the byte being received.

The received bit is then inverted, and the edge latch is cleared. Finally, the counter for bit reception is decremented by 1. If the counter is not 0, then the interrupts continue to be enabled and because, at the return from the interrupt, the CCIE flag is still not 0, the next incoming data bits are processed until the counter reaches 0 and interrupts are stopped. At this point, the entire byte has been successfully received. Figure 6 shows this process.

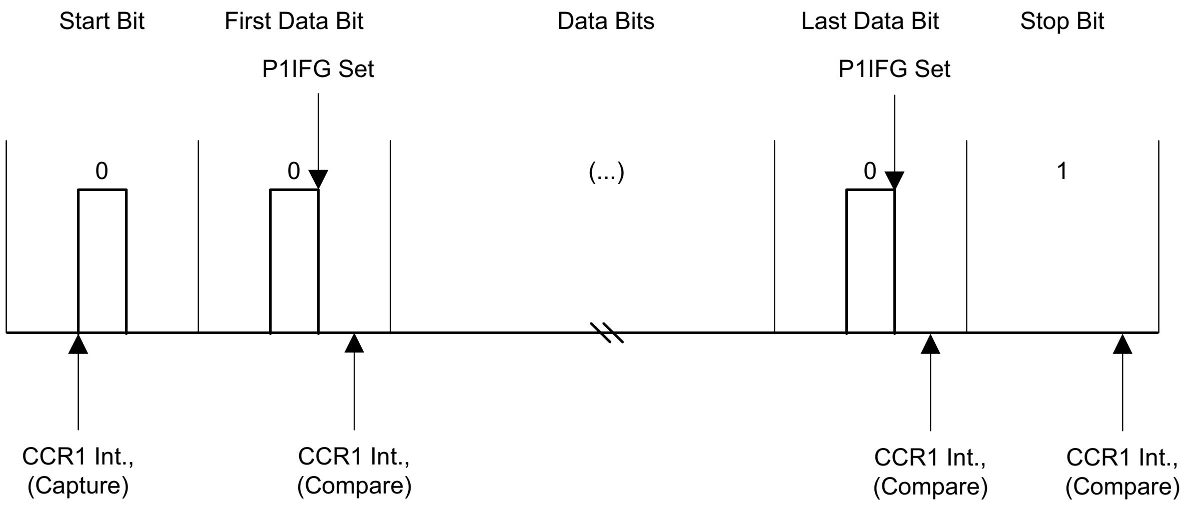

**Figure 6. Detailed Byte Reception**

## 3.2 *Implementing IrPHY Layer using USCI_A0*

On MSP430 devices with a USCI peripheral module, the USCI_A0 can be used in UART mode to conveniently decode and encode the IrDA PHY pulse train. To activate the hardware bit shaping, the bit UCIREN in the UCA0IRCTL control register needs to be set.

The provided MSP430FG4619 and the MSP430F2274 demo applications operate with an SMCLK frequency of 4 MHz and use the exact same USCI initialization sequence. To achieve an IrDA baud rate of 9600 bps using a BRCLK frequency of 4 MHz with the oversampling mode enabled (UCOS16 = 1), the BITCLK16 frequency must be 9600 bps × 16 = 153.6 kHz. For this, the baud rate prescaler is set to 26, and a first-stage modulator setting of 1 is used, giving a very close match to the desired divider.

The IrDA transmitter is configured to generate pulses with a pulse length of exactly 3/16th of a bit time by setting UCIRTXCLK = 1 and UCITXPLx = 5. This setting can be reduced if a shorter pulse length is permissible, resulting in current savings during the operation of the externally connected IrDA transceiver. The byte transmission in IrDA mode is exactly the same as in UART mode. The application ensures that the transmit buffer can be loaded by ensuring that UCA0TXIFG is clear, and then simply loads the transmit data byte into UCA0TXBUF. The USCI module will then start transmitting and bit-shaping the IrDA pulse train.

The receiver configuration includes the setting of UCIRRXPL to match the polarity of the receive signal provided by the IrDA transceiver. An optional analog deglitch filter can be configured to improve system robustness; however, this feature is not used here. To receive data, the USCI_A0 receive interrupt is enabled, and the low-power mode is entered. Upon the reception of a data byte, the receive interrupt service routine is entered, and the data is fetched from the UCA0RXBUF receive buffer.

For more information on the configuration and operation of the USCI module, see the MSP430F4xx or MSP430F2xx family user's guides [12][13].

## 3.3 Implementing IrLAP

The approach for the implementation of the IrLAP layer was to develop functions performing all the tasks described in the primitives that also behave according to the state tables found in the IrDA Lite documentation. This layer depends on the services provided by the IrPHY layer to function properly. This layer provides a reliable connection for data transfer.

To process the frames correctly, first it is necessary to know which of the three classes of frames is being handled. The three types of frames in IrLAP are supervisory (S), information (I), and unnumbered (U). The most effective way to identify the frame is through parsing of the bytes. As soon as the byte pattern that identifies a certain type of frame is found, the program jumps to the routines responsible for its handling. Then, as the bytes are identified and decoded, actions are taken depending on the primitive to which they correspond as shown below in a code excerpt from the application. This routine is called IR_MSG_PROCESS, and it is the piece of code responsible for parsing of the frame and sending the data to the right primitive for processing.

It is simple to identify which type of frame was received, given the bytes used to identify each command. It is enough to check one or two bits of the byte which are in a certain position to identify the IrLAP command that was received. After the command has been identified, the proper response is generated and sent through the IrPHY layer to the transceiver. After this, the program goes back to the main loop and waits until the next frame is received from the primary.

### 3.3.1 Discovery Services

IrLAP routines are responsible for the negotiation of the link and identification exchange while in NDM mode. The first communication that happens between devices is the identification exchange (XID) process. During this time, the primary issues a cycle of XID frames according to the set number of discovery slots. The frame is received and parsed. First, it is identified as an unnumbered (U) frame by checking bits 0 and 1 of the IrLAP control byte, and if they are both 1, then the frame is in the unnumbered format. To identify the unnumbered frame as an XID command frame, bit 5 of the byte is tested next. If this bit is 1, then the identification process of an XID command frame is positive (control bytes shown in Figure 7).

**Unnumbered:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| X | X | X | P / F | X | X | 1 | 1 |

**XID command control:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | P | 1 | 1 | 1 | 1 |

**Figure 7. Unnumbered and XID Command Frame Formats**

As soon as an XID command frame is identified, the discovery flag is checked to see if the XID command has been answered already. If discovery has not happened yet and the frame is not the final slot frame, which is marked by a 0FFh slot number, then an XID response is sent out. The routine in charge of assembling the XID response frame first verifies that the frame was sent for the broadcast address (0FFh) and then produces a frame with the format of an IrLAP XID response frame as the one seen in Figure 8.

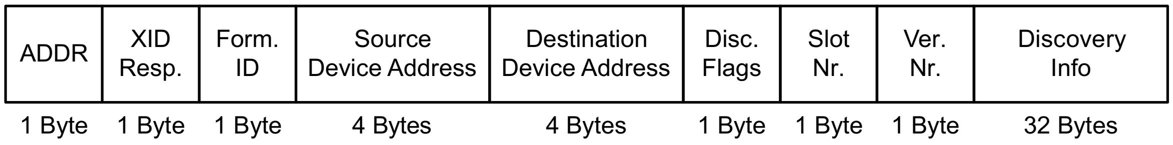

**Figure 8. Discovery Response XID Format**

The MSP430 is set to always answer to discovery slot 0. Therefore, when the first discovery slot is sent by the primary, a frame is sent that claims that spot by issuing the XID response frame. This means the following XID command frames are ignored.

## 3.3.2 Connect Services

After the exchange identification frames have been handled, it is time to negotiate the connection parameters. For the negotiation procedure, accepted connection parameters are established using constants. For the purpose of this application, the constants defined in the IrDA Lite specification are used as the default transmission parameters (for example, 9.6 kbps, 64 bytes, and 1 frame). Then, the primary device produces a set of parameters which it supports and sends it as soon as the discovery process is completed successfully. The other important value introduced by the set normal response mode (SNRM) frame is the connection address (CA). The CA is the byte which is used to contact the specific peer by the primary device instead of the broadcast address.

The format of this SNRM frame is the IrLAP unnumbered type (see Figure 9). An SNRM frame can be easily identified by its control field 093h at byte 2 of the frame (IrLAP control byte). The only bit that is necessary to check is bit 7 because this bit is not set for any other control field of an unnumbered format frame.

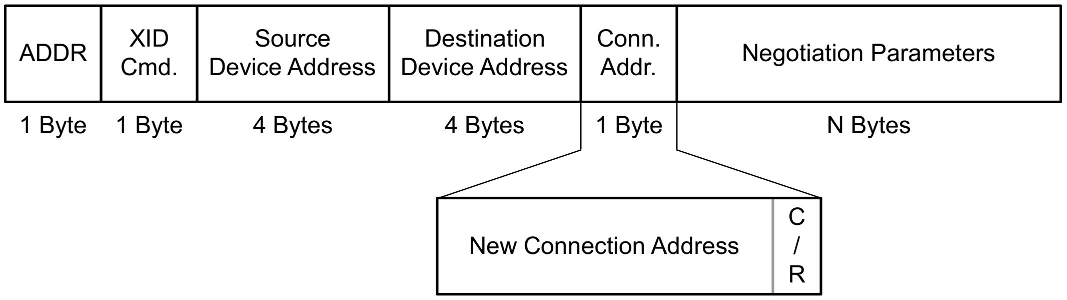

**Figure 9. SNRM Frame Format**

When entering the connection process, the UA response to the SNRM command is issued. The routine responsible for issuing this response verifies that the frame is coming from the device with the same address as the one which caused the discovery sequence to take place. It also includes in the response frame the parameters supported by the secondary for the connection. As part of the routine responsible for assembling the UA response frame to the SNRM command, the counters for amount of sent and received frames (Ns and Nr, respectively) are reset, and the connected flag is set high to let other services know that a connection has been successfully established. On the success of the connection process, NRM is entered.

### 3.3.3 Data Services

After a connection has been established, a change occurs from NDM to NRM. Once in NRM mode, higher layers take over the connection, and IrLAP frames are used as a means to transfer their data reliably. All the frames exchanged must follow the IrLAP information (I) format. The main feature of this frame format is the inclusion of the Nr, Ns, and Poll/Final (P/F) bits as seen in Figure 10.

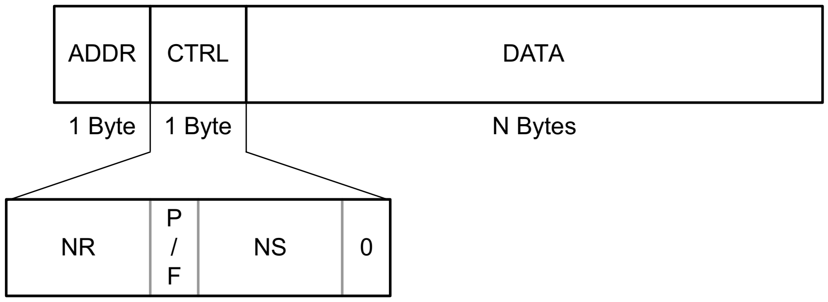

**Figure 10. Information Frame Format**

First, the connection address is verified to ensure that a reply to a frame that was not meant for the MSP430 is ignored. If the connection address was the one assigned to the MSP430, then the frame sequence is verified through the comparison of the value stored in Nr and the value of Ns in the incoming frame. If this check is successful, then the value stored in Nr is now incremented by 1, and the frame is passed to the upper layers for proper manipulation. If the check is unsuccessful, then the frame is ignored, and the connection is dropped. The other condition while in NRM that could cause the connection to be dropped from the IrLAP layer, not including a disconnect request, is if an SNRM frame is received, in which case a request disconnect (RD) response is issued to the primary.

If data is requested by an upper layer while in NRM, then data is queued and buffered, the parts of the frame corresponding to IrLAP are prepared, and the frame is the responsibility of the higher layer. If an I-command frame is received, then it means that the secondary stack is now allowed to transmit frames. Depending on the state of the Ns and Nr counts, either an I- response frame carrying data or an S-response of the type receive ready (RR) or receive not ready (RNR) is sent. The response to receiving a RR command frame is similar to the one explained above as are the responses to RNR and reject (REJ) commands. The same response is also issued in the case of an unknown command frame with the P bit set and from an unknown type frame as it is described in the IrLAP state table for IrDA Lite.

On the other hand, if a U-frame carrying a disconnect command is received, then the proper UA response is issued and the connection state is now NDM.

### 3.3.4 Disconnect Services

A disconnect command frame is of unnumbered format and has a control field equal to 053h. The IrLAP frame size is 2 bytes only (see Figure 11).

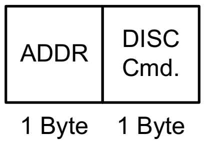

**Figure 11. Disconnect Command Frame Format**

When this frame is identified, a UA response is issued immediately, and the NDM state is entered. A disconnect command can only be issued by a primary and is received by the disconnect primitive whose only responsibility is to issue the UA response and to notify the success of the disconnection to the user layer.

## 3.4 Implementing IrLMP

The implementation of the IrLMP layer was accomplished by following the specifications of the IrDA Lite documentation. The main task of this layer is to verify that the DLSAP-SEL and SLSAP-SEL values are correct and that the correct service is using them. It also verifies that the services requested by the peer are supported as specified in the GetValueByClass function. Finally, it provides the frame format that carries the data to be exchanged by the two devices.

### 3.4.1 Discovery Services

These services are in charge of starting an IrLAP discovery procedure when requested by the user layer. Its only involvement is in adding the service hints and the device information field to the IrLAP XID frame. As shown in Figure 8, the XID response frame includes a field for the device information named discovery info. This field, as described by the IrLMP documentation, is composed of the service hints and the device nickname. The device nickname is itself divided into two other fields which are the character set and the actual name of the device as shown in Figure 12. The service hints provided in this application are 08204h. As can be seen from Table 1, these bits correspond to an extension bit, PDA/Palmtop support, and IrCOMM implementation. The total length of the device information field must not exceed 23 bytes, although the IrLAP XID frame reserves 32 bytes.

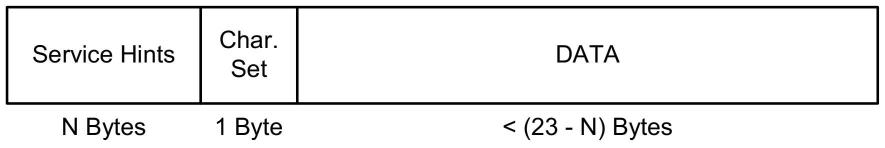

**Figure 12. Device Information Field Format**

### 3.4.2 Link Connect and Connect Services

Because all IrLMP data is exclusively in I-format frames from IrLAP, the first parsing that occurs to identify an IrLMP frame is at an IrLAP level. If the frame is not recognized as an IrLAP packet, then IrLMP parsing takes place. First, there is a check to see if it is an IrLMP data frame by checking byte 3 of the IrLAP frame and testing bit 7. This is a characteristic of data transfer packets in IrLMP which provides easy identification of frames carrying IrLMP data. Then, by checking byte 5 of the frame it is possible to recognize the IrLMP command being issued by the primary. If bit 0 of this byte is set, then the command is an IrLMP connect command. On the receipt of this command, the program enters the IrLMP connect confirm routine.

The responsibilities of this routine are to store the DLSAP-SEL and the SLSAP-SEL bytes from the IrLMP connect command frame that was received. These are values used by the two devices when communicating at the IrLMP level and, whenever called later by another service, other LSAP-SELs are assigned and stored as the ones corresponding to the particular service.

Next, the value of Nr is incremented by 1 and the value of Ns is placed in the following byte. The F bit is set because the connection parameters specify one window frame. This means that each device is allowed to transmit only one frame on each turn. The next two bytes are occupied by DLSAP-SEL and SLSAP-SEL, respectively, and then the opcode for a connect confirm 081h is placed on the next byte. The following byte is reserved and must be set to 0. The next field is the LMP-User Data field which is used by the user layer. The total number of bytes in the frame is loaded into the transmit counter variable, the buffer containing the assembled byte is passed to the transmission pointer, and the routine responsible for sending the frame is called. After this, the program waits for the next incoming frame from the primary peer.

### 3.4.3 Data Services

After the frame has been identified as an IrLAP I-frame carrying user-data, the first test in the IrLMP parsing process is to determine if it is an IrLMP frame carrying data, IrLMP-connection parameters, or TTP-connection parameters. After bit 7 of the third IrLAP byte has been tested and if it is a 0, then the frame is positively identified as an IrLMP frame carrying data. Figure 13 shows the encoding of an IrLMP data frame.

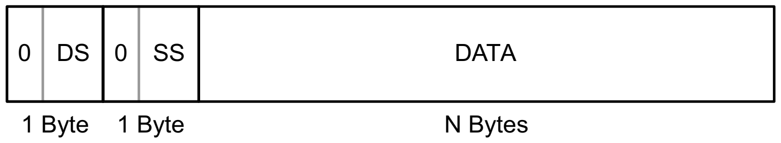

**Figure 13. IrLMP Data Transfer Frame Format**

### 3.4.4 Disconnect Services

Disconnect services as specified by the IrDA Lite documentation are linked directly to IrLAP disconnect. As soon as a disconnect frame is identified, given its IrLMP opcode 002h, the following byte is passed to the user layer. Immediately afterwards, an IrLAP disconnect command is issued, dropping all connections and entering the NDM state once more until another discovery session occurs.

## 3.5 *IAS Implementation*

The IAS services store information about other devices. It also provides information about the services supported by the implementation. As specified by the IrDA Lite documentation, the only service primitive required is GetValueByClass. Figure 14 shows the IAP IrLMP data frame format.

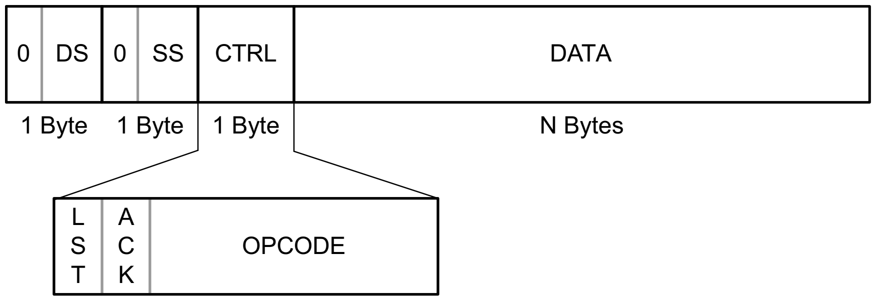

**Figure 14. IAP Frame Format**

The GetValueByClass call issued by the primary carries the class and attribute names which are supported by its IrDA implementation and that are relevant to the service to be used for the connection. As soon as this frame is identified by GetValueByClass opcode 084h which is in byte 3 of the IrLMP data frame, the first thing that must be done is to confirm that the class and attributes advertised are supported by the implementation. The routine CHECK_CLASS_ATTRIB verifies that the class and attributes are both supported by the implementation by checking those received in the frame against the values stored in a memory table.

First, the class is checked. If the class name is unknown, then the GetValueByClass return frame is assembled with the return code corresponding to: "No such class: no other results". If the class check passes, then the attributes are checked. If the attributes are not known, then the return code used corresponds to: "No such attribute: no other results". If the check for both class and attributes is successful, then the frame contains not only the return code for success but also the results. Results include a list length encoded as a 16-bit unsigned integer for which this implementation has a value of 1, followed by the object identifier encoded also as a 16-bit unsigned integer with a value of 3, and finally an attribute value of type integer with a fixed length of 4 octets equal to 003h. This is the extent of the implementation of the IAS capabilities according to the IrDA Lite documentation.

## 3.6 *TTP Implementation*

The operation of TTP involves the exchange of data TTP-PDUs (protocol data units). Effectively, this adds a single octet of header to the IrLMP-MUX data LM-PDUs. This additional octet is used to convey increments (credits) to the number of data TTP-PDUs that may be exchanged in each direction using the underlying IrLMP service.

In this implementation, a simple policy for advancing credit is used. This means that the credit is held longer at AvailCredit rather than being advanced at the earliest opportunity. This leaves buffers available to be reclaimed and redeployed to other needy TTP connections or to relieve resource problems elsewhere in a system.

Connect TTP-PDUs exchanged during connection establishment are not regarded as requiring or consuming credit. Segmentation and reassembly is not implemented. Because the IrLAP window size is equal to 1, a single TTP connection can take full advantage of the underlying IrLAP window.

For this particular implementation, when the initial TTP connection frame is identified, credit is issued so that the peer entity can transfer its data. As soon as the MSP430 responds and after it has received an RR command from the primary, it then issues more credit for the peer entity to continue transmitting data.

### 3.7 IrCOMM Implementation

This is the layer that is in charge of the actual data exchange. The only times when it does not use the IrLMP services is for identifying the frames that carry user data, for constructing the frames that send data to the peer device, and when dealing with TTP services. For connection and disconnection, it uses the respective IrLMP services.

IrCOMM services are the same services as provided by the IrLMP layer. IrCOMM uses the services provided by the service layer, and this call would propagate down the protocol stack. It uses the data PDUs from IrLMP to transmit all data and control channel information. These two types of frames can be differentiated because when sending data there is an overhead of two bytes. The first byte is equal to the amount of credit that the peer has left to transmit after the data-carrying frame is received, and the second one is equal to 0 following the IrLMP LM-MUX overhead. In the case of the connection frames, a control field follows the TTP overhead where the first byte indicates the length in bytes of the connection parameters. As the connection parameters for IrLAP and TTP, these are in groups of three.

### 3.8 Application Layer

The application layer is the layer that uses the services provided by the stack to perform a specific task. In this case, for demonstrating functionality, the user layer waits for the ASCII character 't'. After the character has been received, the MSP430 obtains a temperature reading from the ADC module integrated temperature sensor and appends it through the use of the IrCOMM layer to the user data field of a data carrying IrLMP frame. The temperature sent then is displayed by the peer device, and the MSP430 then issues more credit to the peer entity so that more data can be sent.

## 4 PC Demonstration Application

The PC demonstration application supplied with this application report is provided as both C source code and as an executable single-file Win32 command line application. It has been decided to develop the software as a command line application to make it both easy to understand and also compatible with all common Windows-based C development systems such as Microsoft Visual C++ or Borland C++. The demonstration software requires at least Windows 2000 or Windows XP operating system and a properly installed infrared port. This IR port can either be integrated (such as found in notebook computers) or provided by an external Windows-supported IrDA adaptor (for example, the Actisys ACT-IR220L+).

The current Windows operating systems come with a built-in IrDA stack that implements various IrDA communication modes, such as the 9-wire IrCOMM mode. This mode can easily interface with external devices like PDAs, cell-phones, and the presented MSP430 IrDA stack. Microsoft decided to expose the IrDA stack through the Windows sockets library rather than by providing virtual COM ports. If virtual COM ports need to be used, then additional third-party driver software such as IrCOMM2k is required [9]. However, the more elegant approach is to perform IrDA communication by using the standard Windows built-in driver model as shown in this demonstration application.

The PC demonstration application provides feedback about every Windows API call it does. If one of the calls fail, the application terminates and displays the Windows error code as obtained by the Windows sockets function WSAGetLastError(). For detailed information about the error codes, see the Microsoft Windows Platform SDK documentation [8].

When starting the application, it tries to open the Winsock system library (DLL). At least version 2.2 is required for proper operation of the IrDA communication. The program exits with an error message if an incompatible version is located. After opening the Windows IrDA socket, the reception and transmit timeouts are configured to 3 seconds to prevent the software from indefinitely waiting for the end of the communication. The software then generates a list of IrDA devices which have been discovered already. For the sake of simplicity, the first device is taken from the list and assumed to be the IrDA peer the user wants to communicate with. At this point, the user could be provided with a list of the discovered devices. Next, the peer's IAS database is scanned to check whether or not it supports the 9-wire communication mode. If the 9-wire mode is supported, then it is activated. Otherwise, the application terminates with an error message.

From this point on, using the Windows IrDA stack does not differ from using any other Windows sockets such as TCP or UDP. The connection is opened by calling the connect() function. After that, a single t character is transmitted using send(). When the MSP430 IrDA stack receives this character, it initiates an A/D conversion of the ADC12 module internal temperature sensor and then sends back the current MCU temperature as an ASCII string. Finally, the data that was received and buffered by the Windows IrDA stack is read out using the sockets function recv() and displayed on the screen. The Windows socket and therefore the IrDA stack are now closed by calling closesocket(). It is important that the socket is closed every time it has been used, even if an error occurred. Otherwise, internal Windows IrDA stack resources could remain locked and the IrDA port can no longer be accessed until the next system start-up. Therefore, make sure not to interrupt the demo application by not pressing Ctrl+C. Detailed information about the built-in Windows IrDA functionality can be found in the Microsoft Windows Platform SDK.

## 5 IrDA Protocol Basics

Communication protocols are commonly divided into layers. These layers have their own responsibilities and dependencies to the layers above and below them, thus creating the concept of a protocol stack. Figure 15 shows the IrDA stack. This stack is built from the bottom up, with each layer dependant on the layers below. The first three layers shown are mandatory while those on top of IrLMP, except IAS, are optional protocols required only for specific applications. The required layers are:

- IrPHY: Specifies optical characteristics following ISO standards, data encoding/decoding, and framing for various speeds.
- IrLAP: Establishes the basic reliable connection, frame formatting, and parameter negotiation procedures.
- IrLMP: Multiplexes services and applications on the IrLAP connection level.
- IAS: Provides a database of services on a device.

Some of the optional layers are:

- TTP: Manages per-channel flow control.
- OBEX: Object exchange protocol, used to easily transfer files or other data objects.
- IrCOMM: Serial and parallel port emulation enabling devices that use these services to use IrDA interchangeably without problems.

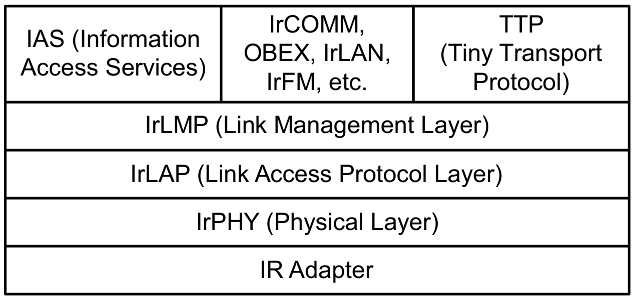
IAS (Information Access Services)
IrCOMM, OBEX, IrLAN, IrFM, etc.
TTP (Tiny Transport Protocol)
IrLMP (Link Management Layer)
IrLAP (Link Access Protocol Layer)
IrPHY (Physical Layer)
IR Adapter

**Figure 15. IrDA Stack**

## 5.1 Physical (IrPHY) Layer

The physical (IrPHY) layer is responsible for the sending and receiving of frames, some framing as beginning- and end-of-frame flags, and cyclic redundancy checks (CRC) or check sums. The latter are used as means of error detection mechanisms. Even though there is no means by which to repair the data (if the data is known to be corrupted), it could be requested again.

A frame at the IrPHY level is identified by 10 extra beginning-of-frame (XBOF) flags, 1 byte each as specified by the IrDA Lite documentation. After the XBOF flags are sent, a single beginning- of-frame (BOF) flag, 1 byte, is sent followed by data (payload). At the end of each frame, there are 2 bytes which compose the frame check sum (FCS) followed by a single end-of-frame (EOF) flag marking the end of the frame. Figure 16 shows the frame structure.

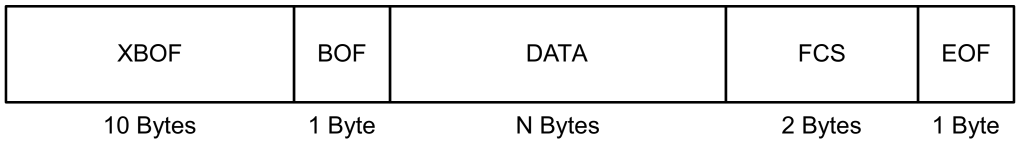

**Figure 16. IrPHY Frame**

## 5.2 Link Access Protocol (IrLAP) Layer

The link access protocol (IrLAP) layer is responsible for reliable data transfer at a low level; upper layers can then rely on the services provided by this layer. The data is delivered and if delivery is not possible, then the upper layer is aware of this fact and acts accordingly. The IrLAP layer provides a point-to-point half-duplex connection with no data collision control mechanism. It is possible to simulate a full-duplex connection when the timing requirements are not critical.

The devices connected to the IrLAP layer are known as the primary and secondary devices which correspond to a master-slave relationship, respectively. The responsibilities of each device vary depending on the role it performs. Although many times a primary station can take on the role of a secondary, the opposite is not true when a secondary-only implementation is followed.

The primary station is responsible for:

- Sending command frames (commencing connections and transfers)
- Control and organization of data
- Manipulating data link errors

The secondary station:

- Sends response frames only

The IrLAP layer is built around two modes of operations: normal disconnect mode (NDM) and normal response mode (NRM). NDM is used when a connection does not yet exist. When the devices identify the IR media to be free, then the discovery process can begin with the default parameters for negotiating a connection. NRM is used when a connection exists and then upper layers proceed to exchange information. Figure 17 shows the basic IrLAP frame format.

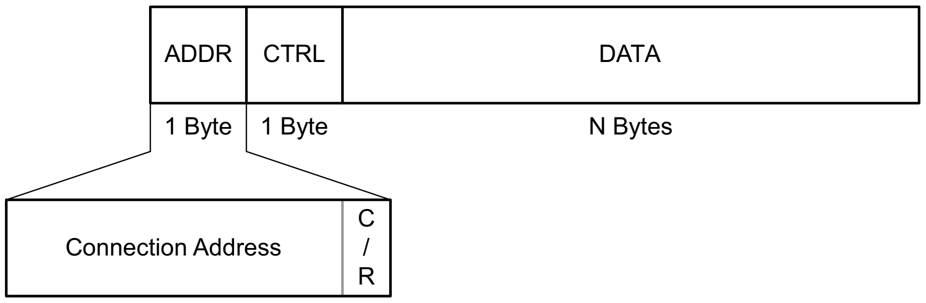

**Figure 17. IrLAP Frame Format**

The address and control fields only take 1 byte each; thus, adding little overhead to user data. Three different framings exist that are applied to the data before it is sent. Proper framing depends solely on the speed at which the data is sent. The scope of this report is up to 115.2 kbps; therefore, asynchronous framings schemes are followed.

The operations performed by this layer are defined in its service primitives; these can be understood as IrLAP API definitions. Figure 18 shows an example of how these service primitives work. Not all the primitives defined are required to have an IrDA implementation. The primitives defined by the IrDA Lite protocol as required are:

- Discovery and address conflict services
- Connect services
- Data services
- Disconnect services

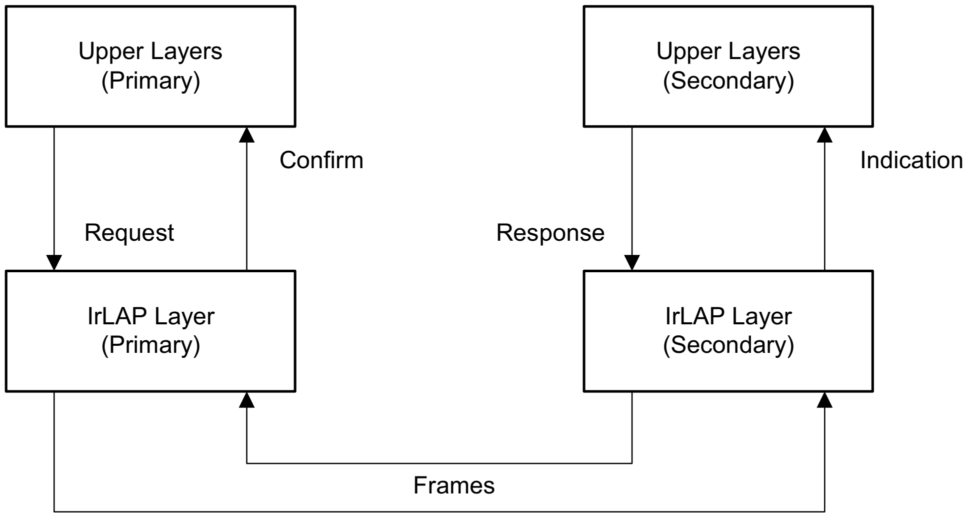

**Figure 18. IrLAP Service Primitives**

## 5.3 *Link Management Protocol (IrLMP) Layer*

The link management protocol (IrLMP) layer depends completely on the reliable connection services and the negotiated link provided by the IrLAP layer. The services provided by the IrLMP layer include: multiplexing of the link, address conflict resolution above the IrLAP layer, and information access services (IAS).

To be able to have multiple LMP connections on one single LAP connection, the IrDA developed a way of addressing these connections. Its method consists of having various logical service access points (LSAPs) which access a service or application. This is accomplished through the use of LSAP-SELs (LSAP selectors), which are 1-byte numbers that depending on the range they belong to is the service they provide.

Just like the IrLAP layer, the IrLMP layer has a series of primitives that stipulate the services it provides, which are, according to the IrDA Lite specification:

- Discovery services
- Link connect services
- Connect services
- Data services
- Disconnect services

The IrLMP layer adds 2 bytes of overhead in the information field of the IrLAP frame. Figure 19 shows the IrLMP frame format.

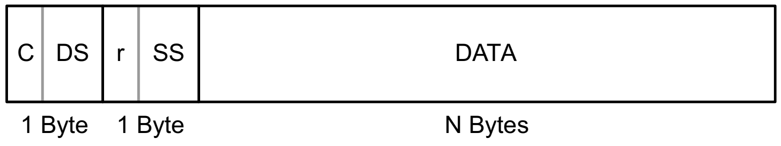

**Figure 19. IrLMP Frame Format**

The control bit (C) identifies the frame as a command (C = 1) or data frame (C = 0) and the r bit is reserved. DLSAP-SEL (DS) and SLSAP-SEL (SS) are the service addresses of the destination of the frame and for the sender of the frame, respectively.

As part of the discovery services, the IrLMP layer adds overhead to the IrLAP XID frame. This overhead is constituted by the device discovery information. This field is then divided in three fields which are: service hint bits, character set, and device nickname. The service hint bits are 2 bytes that, depending on which bits are set, determine which services that specific device supports (see Table 1). The character set most widely used is ASCII, although other character sets are permitted. Finally, the device nickname is the name by which the device is identified in the IAS database service.

**Table 1. Service Hint Bits**

| **Byte 1** | | **Byte 2** | |
|---|---|---|---|
| **Bit** | **Function** | **Bit** | **Function** |
| 0 | PnP compatible | 0 | Telephony |
| 1 | PDA/Palmtop | 1 | Fileserver |
| 2 | Computer | 2 | IrCOMM |
| 3 | Printer | 3 | Reserved |
| 4 | Modem | 4 | Reserved |
| 5 | Fax | 5 | OBEX |
| 6 | LAN access | 6 | Reserved |
| 7 | Extension (additional hint byte follows if this bit is set) | 7 | Extension |

## 5.4 Information Access Services (IAS)

Information access services (IAS) provide a directory of services for a device. All the applications available for connections must have entries in the IAS which determine their LSAP-SEL. This address is used when querying the application for further information about the services it provides.

According to the IrDA Lite standard, the only service primitive which is mandatory when implementing IAS is the GetValueByClass primitive. The application which makes the inquiry must provide the following information: class and attribute name. The application responding provides the LSAP-SELs that provide the requested service or an indication that the specific attribute does not exist in its database.

## 5.5 Tiny Transfer Protocol (TTP)

Tiny transfer protocol (TTP) provides a higher level of flow control than that of the IrLAP layer. It makes use of the multiplexing capabilities of the IrLMP layer to enable two different LMP connections to use the same underlying IrLAP connection without the loss of any data in the process. The way in which TTP is able to provide this service is by issuing a credit to each party. This credit is directly proportional to the buffer space that one side has. Sending data causes one credit to be used. Periodically, the receiver issues more credit to the sender to keep receiving the intended data. TTP also has segmentation and reassembly capabilities; this feature is not described in this application report.

TTP has its own set of primitives that describe the behavior of this layer. The primitives for TTP are:

- Connect services
- Disconnect services
- Reliable or unreliable data services
- Local flow control services

Two frame formats are used in TTP. Both add overhead to the User-Data field of the IrLMP

frame. These frames, as shown in Figure 20, establish a connection where initial credit (IC) is issued or transfer data where delta credit is accounted for. Delta credit (DC) is the amount of data frames that a device is allowed to send before it has to turn the connection over to another device.

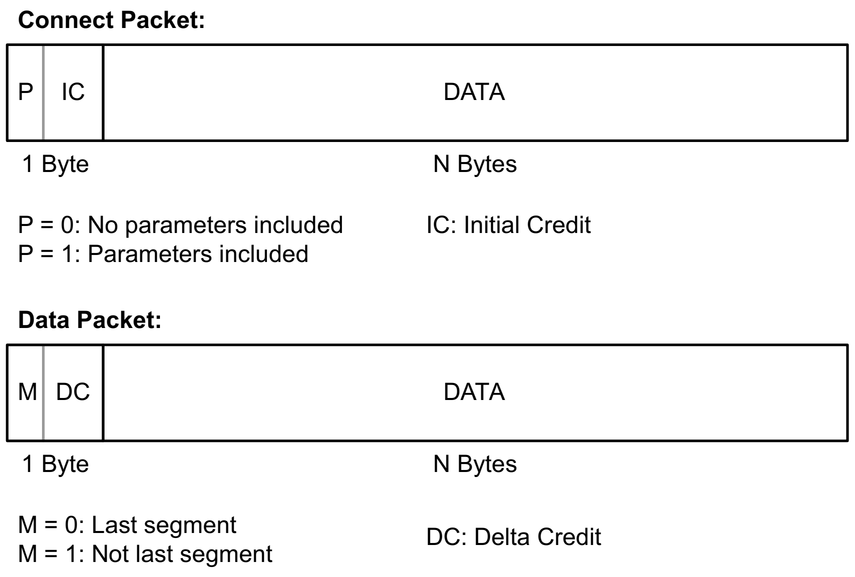

**Figure 20. TTP Frame Formats**

## 5.6 IrCOMM

IrCOMM is serial and parallel port emulation over IR, also known as wire replacement. It provides four different services: 3-wire raw, 3-wire cooked, 9-wire, and Centronics. For this application, the IrCOMM 3-wire services are implemented. 9-wire services are needed to communicate with Windows. Because the Windows IrDA stack does not actually handle the control channel, it is possible to achieve communications with Windows by the same means as 3-wire cooked as long as the 9-wire service is advertised.

The 3-wire cooked service class makes use of TTP flow control, so that it may coexist with other connections that employ a higher level (not IrLAP) flow control (including other cooked IrCOMM connections). Therefore, it is not limited to a single IrLMP connection. It also supports a control channel for sending information like data format. Because of the need for flow control and the use of the control channel, the 3-wire service type uses a more elaborate frame format.

IrCOMM introduces a new service hint bit to indicate that the device supports IrCOMM services. The IrCOMM IAS entry is IrDA:IrCOMM with the following attributes: LsapSel and Parameters. The LsapSel attribute is needed to make a connection. The Parameters attribute allows the client application to distinguish among multiple IrCOMM services, because many different applications can use serial and parallel ports to communicate. It has at least the service-type parameter with at least the 3-wire cooked-bit set. The LsapSel attribute is also the unique address of the service within the context of one device and is needed to connect to that service. If the IrDA:IrCOMM IAS entry is for 3-wire cooked service, then the format to be used is IrDA:TinyTP:LsapSel.

The Parameters attribute contains one or more values which characterize the IrCOMM service being provided. The attribute Parameters is mandatory, and it is to have a value type of octet sequence (002h). Each parameter in the Parameters attribute consists of tuples of 3 (PI, PL, and PV). The parameters that can be set are service type, port type, and fixed port name.

3-wire connections can coexist with other nonexclusive IrLMP connections. This is because 3-wire cooked uses TTP flow control, a method which allows various connections with their own flow control to exist concurrently. Despite the presence of a control channel, 3-wire cooked cannot emulate hardware handshaking, because nondata signals are not emulated. If hardware handshaking is required, then the 9-wire or Centronics service type is necessary.

In 3-wire cooked, the control channel is used for three purposes:

- Selecting the service type
- To exchange port communication settings (data rate, data format, and flow control information) when emulating a serial port
- For certain Type 2 devices to deliver port line status (overrun, parity, and framing errors) back to Type 1 devices

## 6 IrDA Communication Diagram

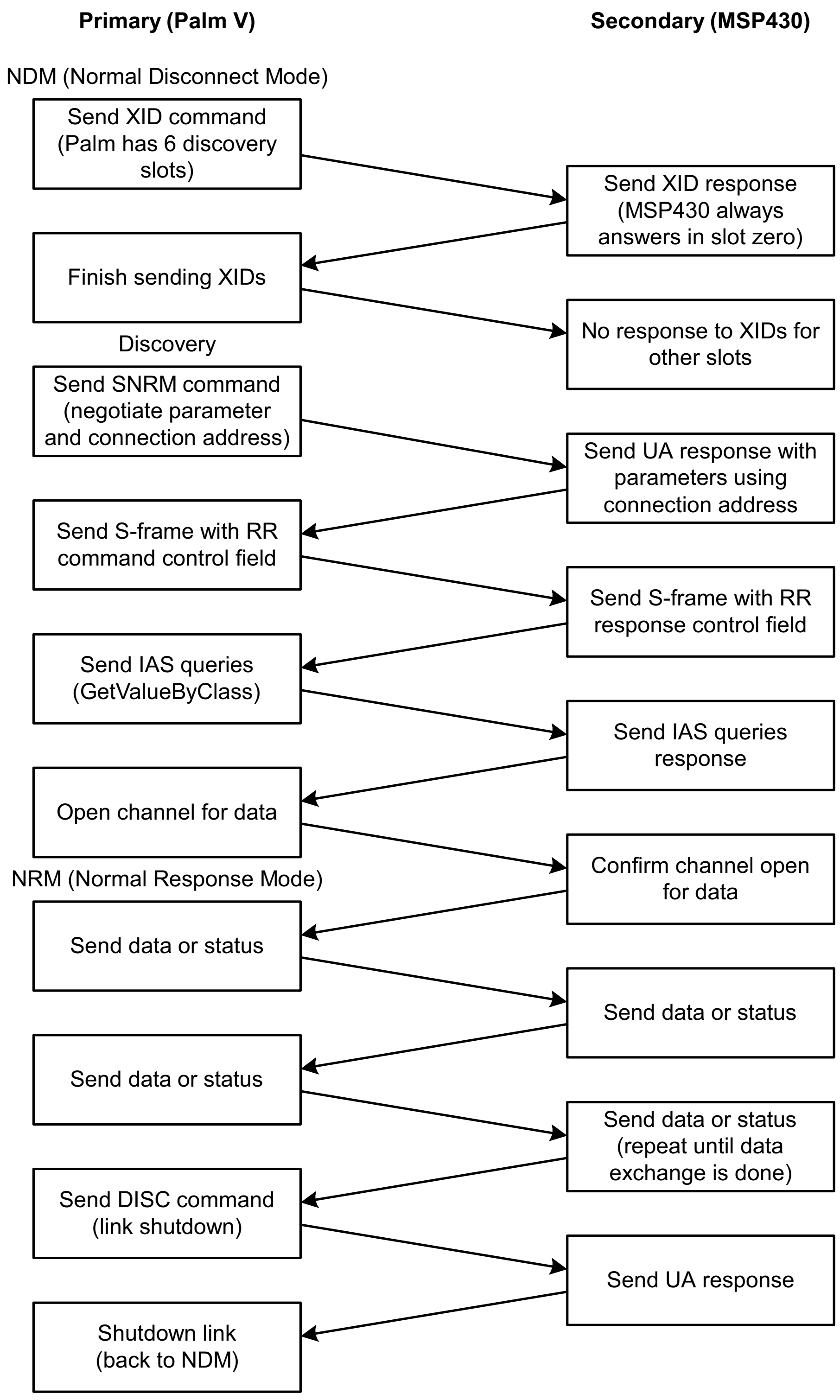
Primary (Palm V)
Secondary (MSP430)
NDM (Normal Disconnect Mode)
Send XID command (Palm has 6 discovery slots)
Send XID response (MSP430 always answers in slot zero)
Finish sending XIDs
No response to XIDs for other slots
Discovery
Send SNRM command (negotiate parameter and connection address)
Send UA response with parameters using connection address
Send S-frame with RR command control field
Send S-frame with RR response control field
Send IAS queries (GetValueByClass)
Send IAS queries response
Open channel for data
Confirm channel open for data
NRM (Normal Response Mode)
Send data or status
Send data or status
Send data or status
Send data or status (repeat until data exchange is done)
Send DISC command (link shutdown)
Send UA response
Shutdown link (back to NDM)

**Figure 21. IrDA Communication Diagram**

## 7 Frame Exchange Log

This section shows the frames exchanged in a connection between the IrCOMM2k driver for the PC and the MSP430 IrDA stack. This is only meant to be an example, and it shows communication with no errors. When a frame is similar to the previous frame, only the different fields are commented on. All the numbers are in hexadecimal format, unless noted otherwise.

**NOTE:** IrCOMM2k is a driver for Windows that makes possible to use the IrCOMM services in the Windows IrDA stack. It also enables the user to be able to use the Linux-IrDA stack ported to Windows, if so desired (see IrCOMM2k, Virtual Infrared COM Port for Windows 2000/XP).

**(001 from IrCOMM2k)** XID command: FF 3F 01 29 23 BE 84 FF FF FF FF 01 00

Address Field: FF, where the last bit is the C/R bit. When set, the C/R bit indicates a command frame. The previous seven bits, equal to 7F are the broadcast address.

Control Field: 3F, 001x1111b indicates an unnumbered type frame. 2F is the opcode for exchange station identification. Bit 4 is the Poll/Final bit, which, when set by the primary, indicates the frame is the final frame.

Format Identifier: 01 is a set value as defined in the IrLAP documentation.

Source Device Address: 2923BE84 is the address of the device sending the frame. Destination Device Address: FFFFFFFF is the broadcast address during discovery. Discovery Flags: 01. The last two bits indicate the amount of slots to be used during the

discovery, in this case six. Bit 2 is the conflict bit which is not set in this case. Slot Number: 00, marks the first discovery slot.

Version Number: 00, as specified in the IrLAP documentation.

**(002 from MSP430)** XID response: FE BF 01 AB CD 12 34 29 23 BE 84 01 00 00 82 04 00 20 4D 53 50 34 33 30 20

Destination Device Address: 2329BE84 is the address of the device for which the frame is intended.

Service Hint Bits: 8204. These bits are set according to the IrLMP documentation and in this case they are set to support PDA/Palmtop and IrCOMM.

Character Set: 00 indicates ASCII is used.

Device Nickname: 204D535034333220 reads MSP430 and is the name that stores this device information in the peer device IAS database.

**(003 from IrCOMM2k)** XID command: FF 3F 01 29 23 BE 84 FF FF FF FF 01 01

**(004 from IrCOMM2k)** XID command: FF 3F 01 29 23 BE 84 FF FF FF FF 01 02

**(005 from IrCOMM2k)** XID command: FF 3F 01 29 23 BE 84 FF FF FF FF 01 03

**(006 from IrCOMM2k)** XID command: FF 3F 01 29 23 BE 84 FF FF FF FF 01 04

**(007 from IrCOMM2k)** XID command: FF 3F 01 29 23 BE 84 FF FF FF FF 01 FF 00 84 04 00 4C 54 41 30 38 36 36 32 36 35

Slot Number: FF marks the final discovery slot in the discovery sequence.

Service Hint Bits: 8404. This device supports PCs, PDA/Palmtops, and IrCOMM.

**(008 from IrCOMM2k)** SNRM command: FF 93 29 23 BE 84 AB CD 12 34 70 01 01 3E 82 01 01 83 01 3F 84 01 7F 85 01 80 86 01 1F 08 01 07

Control Field: 93, 100x0011$_b$ = Unnumbered: Set Normal Response Mode (83) with the final bit set

Connection Address: 38 this corresponds to the middle six bits of 70.

Negotiation Parameters:

- Baud Rate: (01, 01, 3E) 9600, 19200, 38400, 57600, and 115200 bps supported
- Maximum Turnaround Time: (82, 01, 01) 500 ms
- Data Size: (83, 01, 3F) 64, 128, 256, 512, 1024, and 2048 bytes
- Window Size: (84, 01, 7F) 1, 2, 3, 4, 5, 6, and 7 frame windows
- Additional BOFs: (85, 01, 80) 0 additional BOFs at 115200 bps
- Minimum Turnaround Time: (86, 01, 1F) 10, 5, 1, 0.5, and 0.1 ms

- Link Disconnect/Threshold Time: (08, 01, 07) 3/0, 8/3, and 12/3 seconds

**(009 from MSP430)** UA Response: 70 73 AB CD 12 34 23 29 BE 84 01 01 02 82 01 01 83 01 01 84 01 01 85 01 01 86 01 01 08 01 01

Address Field: 70. Bit 0 is the C/R bit indicating a response, and the upper 7 bits use from now on the 'Connection Address' received in the SNRM frame number 38.

Control Field: 73, 011x0011b = Unnumbered: Unnumbered Acknowledgement (63) with the poll bit set.

**(010 from IrCOMM2k)** RR command: 71 11

Address Field: 71 with the C/R bit set and the CA (connection address) set to 38.

Control Field: 11 where bits 0-3 indicate a supervisory frame with a receive ready opcode, bit 4 is the P/F bit, and bits 5-7 indicate the received frame count (Nr).

**(011 from MSP430)** RR response: 70 11

**(012 from IrCOMM2k)** IrLMP Connect Command: 71 10 80 5A 01 00

Control Field: 10 where bit 0 indicates an information type frame, bits 1-3 indicate the sent frame count (Ns), bit 4 is the P/F bit, and bits 5-7 indicate the received frame count.

Destination: 80 where bit 7 is the control bit, and bits 0-6 are the DLSAP.

Source: 5A where bit 7 is reserved, and bits 0-6 are the SLSAP.

Opcode: 01 this corresponds to the connect command.

Reserved: 00 for future use.

**(013 from MSP430)** IrLMP Connect Confirm: 70 30 DA 00 81 00

Opcode: 81 corresponds to the connect confirm.

**(014 from IrCOMM2k)** IAS query: 71 32 00 5A 84 0B 49 72 44 41 3A 49 72 43 4F 4D 4D 0A 50 61 72 61 6D 65 74 65 72 73

IAP Control Field: 84 where bit 7 corresponds to the last frame bit, bit 6 is the acknowledge bit, and bits 0-5 are the opcode to the GetValueByClass IAS query.

Class Name: 0B497244413A4972434F4D4D where the first byte represents the length of the class name, and the rest is the ASCII hex for the class name: "IrDA:IrCOMM"

Attribute Name: 0A506172616D6574657273 where the first byte represents the length of the attribute, name and the rest is the ASCII hex for the attribute name: "Parameters".

**(015 from MSP430)** IAS result: 70 52 5A 00 84 00 00 01 00 01 02 00 03 00 01 07

IAP Control Field: 84 where bit 7 corresponds to the last frame bit, bit 6 is the acknowledge bit, and bits 0-5 are the opcode to the GetValueByClass IAS response.

Return: 00, success

List Length: 01

Object Identifier: 0001

Type: 02 for octet sequence

Sequence Length: 03

Sequence: 000107, where the first byte means service type, second byte is the length of result, and the last byte has bits set for: 3-wire raw, 3-wire, or 9-wire service.

**(016 from IrCOMM2k)** IrLMP Disconnect: 71 54 80 5A 02 01

IrLMP opcode: 02

Reason: 01 meaning user request.

**(017 from MSP430)** RR response: 70 71

**(018 from IrCOMM2k)** IrLMP Connect Command: 71 56 80 5B 01 00

**(019 from MSP430)** IrLMP Connect Confirm: 70 94 DB 00 81 00

**(020 from IrCOMM2k)** IAS query: 71 78 00 5B 84 0B 49 72 44 41 3A 49 72 43 4F 4D 4D 13 49 72 44 41 3A 54 69 6E 79 54 50 3A 4C 73 61 70 53 65 6C

IAP Control Field: 84 where bit 7 corresponds to the last frame bit, bit 6 is the acknowledge bit, and bits 0-5 are the opcode to the GetValueByClass IAS query.

Class Name: 0B497244413A4972434F4D4D where the first byte represents the length of the class name, and the rest is the ASCII hex for the class name: "IrDA:IrCOMM".

Attribute Name: 13497244413A54696E7954503A4C73617053656C where the first byte represents the length of the attribute name, and the rest is the ASCII hex for the attribute name: "IrDA:TinyTP:LsapSel".

**(021 from MSP430)** IAS result: 70 B6 5B 00 84 00 00 01 00 01 01 00 00 00 02

Type: 01 indicating an integer result.

Value: 000002 indicating the LSAP to use for TTP.

**(022 from IrCOMM2k)** IrLMP Disconnect: 71 9A 80 5B 02 01

**(023 from MSP430)** RR response: 70 D1

**(024 from IrCOMM2k)** IrLMP/TTP Connect Command: 71 9C 82 55 01 00 10

TTP Connect Frame: 10 where bit 7 is the P bit indicating a parameter-less connection frame, and where the remaining 7 bits indicate the Initial Credit given to the peer entity.

**(025 from MSP430)** IrLMP/TTP Connect Confirm: 70 F8 D5 02 81 00 01

**(026 from IrCOMM2k)** RR command: 71 B1

**(027 from MSP430)** RR response: 70 F1

**(028 from IrCOMM2k)** IrCOMM Parameter Set: 71 BE 02 55 00 12 00 01 04 10 04 00 00 25 80 11 01 03 12 01 0C 20 01 0C

IrCOMM header: 00

Length of Parameters: 12

Service Type: (00, 01 04) Service selected, equal to highest service in common, 9-wire

Data Rate: (10, 04, 00002580) 9600 bps

Data Format: (11, 01, 03) 8 bits, 1 stop bit, no parity

Flow Control: (12, 01, 0C) RTS/CTS on input and output

DTE Line Settings and Changes: (20, 01, 0C) DTR and RTS state

**(029 from MSP430)** IrCOMM Parameter Accept and TTP credit: 70 1A 55 02 02

TTP Credit: 02, where bit 7 is the M (more) bit, and the remaining six bits indicate the delta credit

**(030 from IrCOMM2k)** RR command: 71 D1

**(031 from MSP430)** RR response: 70 11

**NOTE:** The RR command and response and exchange continue (frames 32-37) until a device starts sending.

**(038 from IrCOMM2k)** IrLMP PDU/IrCOMM: 71 D0 02 55 00 00 74

IrCOMM data header: 0000

User Data: 74, the hex value for the ASCII character t

**(039 from MSP430)** IrLMP PDU/IrCOMM: 70 3C 55 02 00 00 38 37 46 20

IrCOMM data header: 0000

User Data: 38374620, the hex sequence for the ASCII string 87F

**(040 from IrCOMM2k)** RR command: 71 F1

**(041 from MSP430)** TTP advancing credit: 70 3E 55 02 02

TTP Credit: 02, where bit 7 is the M (more) bit, and the remaining six bits indicate the delta credit.

The exchange of RR command/response frames, IrCOMM user data and TTP credit continues until one of the peers requests disconnection after finishing the data transfer.

## 8 References

1. Serial Infrared Physical Layer Specification V1.4, Infrared Data Association, 2001
2. Serial Infrared Link Access Protocol (IrLAP) V1.1, Infrared Data Association, IBM Corporation, Hewlett-Packard Company, Apple Computer, Inc., Counterpoint Systems Foundry, Inc., 1996
3. Link Management Protocol V1.1, Infrared Data Association, 1996
4. Minimal IrDA Protocol Implementation (IrDA Lite) V1.0, Infrared Data Association, Counterpoint Systems Foundry, Actisys Corporation, 1996
5. 'Tiny TP': A Flow Control Mechanism for use with IrLMP V1.1, Infrared Data Association, 1996
6. IrDA Object Exchange Protocol OBEXTM V1.3, Extended Systems, Microsoft Corporation, 2003
7. 'IrCOMM': Serial and Parallel Port Emulation over IR (Wire Replacement) V1.0, Counterpoint Systems Foundry, Inc., Hewlett-Packard Company, Lexmark International, Inc., Sharp Corporation, NTT Corporation, Nokia, 1995
8. Microsoft Windows Platform SDK
9. IrCOMM2k, Virtual Infrared COM Port for Windows 2000/XP
10. MSP430x13x, MSP430x14x Mixed-Signal Microcontrollers
11. MSP430x1xx Family User's Guide
12. MSP430x2xx Family User's Guide
13. MSP430x4xx Family User's Guide
14. GP2W0110YPSF Low Power IrDA Transceiver Module Datasheet, Sharp Corporation, 2002

## Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

**Changes from June 22, 2007 to December 4, 2018** **Page**

- Editorial and formatting changes throughout document ........ 1