SX IrDA Virtual Peripheral Implementation

SCENIX

Application Note 16

February 15, 1999

1.0 Introduction

The IrDA (Infra-Red Data Association) standard is wireless replacement for traditional wired connections between computing devices and peripherals. It uses an infra-red LED and photo-diode to transmit information at up to 4Mbps over one metre.

This application note describes the implementation of the lower levels of the IrDA protocol stack and the high level IrComm protocol for a secondary device on the SX communications controller. The implementation of this reliable protocol on a small communications controller is achieved by relying on the application's ability to re-send data upon request rather than the traditional use of large data buffers.

The implementation communicates at up to 115.2kbps and makes use of the SX's high clock speed to shape the IrDA pulses without external hardware. It uses two virtual peripheral UARTs: one for the IrDA port and one for a debugging serial port.

The next section gives an overview of the IrDA stack and describes the features of the SX implementation. Each layer of the stack is then explained in detail. Finally, descriptions of the demonstration applications and hardware are given.

This documentation should be read in conjunction with the IrDA Specification, IrDA Lite Specification and IrComm available from `www.irda.org`.

1.1 THE IrDA STACK

Figure 1-1 shows the IrDA protocol stack. The boxes in white are included in the SX implementation and are described in this document

The physical layer converts octets of data to bit streams and transmits them in the form of bursts of IR light (and vice-versa). The rated operating range for IrDA is one metre for all speeds, however under typical conditions this range is usually higher.

The framing layer encapsulates the payload data within a frame so that it can be transmitted and identified as a frame of data when it is received by another station. The frame data is protected by a CRC to confirm the validity of the data.

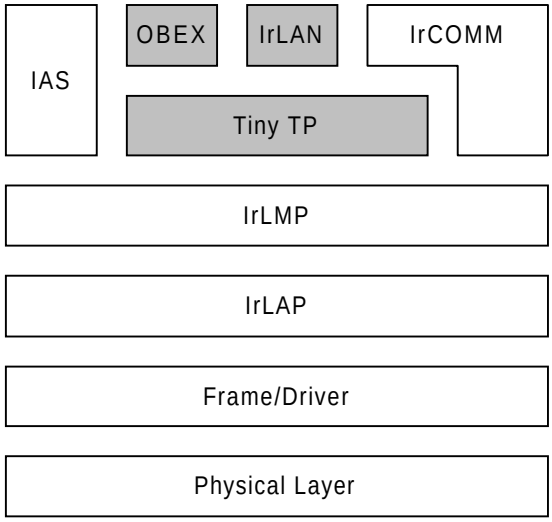


Figure 1-1. IrDA Protocol Stack

The link-access-protocol (LAP) layer controls the flow of frames and provides a connection-based bi-directional reliable data transfer service. It manages the discovery process, the connection and negotiation process, and the transfer of reliable data.

The link-management-protocol (LMP) layer serves to multiplex the LAP connection between a number of higher layers and/or applications using the reliable data transfer service of the LAP layer.

The information-access-server (IAS) provides a way for an IrDA device to get specific details about the high level services another IrDA device offers.

The IrComm interface provides a reliable bi-directional virtual COM or LPT interface between to devices. The API interface can be considered to be that of a UART and is intended to allow the use of software designed only to work with a wired connection to be able to used a IrDA virtual wired connection.

1.2 SX IrDA SPECIFICATIONS

The implementation is based on the IrDA Lite specifications for a secondary only device with support for connection speeds up to 115200bps and with the following primary features:

- Support for initiating XID discovery.
- Support for sending unnumbered-information frames.

1.2.1 Physical Layer

Multi-speed IrDA UART Virtual Peripheral supporting 115200, 57600, 38400, 19200, and 9600bps

1.2.2 Framing Layer

Byte-at-a-time processing of data including 16-bit CRC protection of data.

1.2.3 LAP Layer

- SX can discover other devices by initiating the discovery process.
- SX can be discovered by other devices by responding a discovery request.
- SX can accept a connection request and negotiate a high speed for the connection.
- SX can maintain a bi-directional flow of reliable data within a connection.
- SX can transmit unreliable data frames outside of a connection.
- SX can accept unreliable data frames outside of a connection.

1.2.4 LMP Layer

Minimal implementation to allow multiplexing between the IAS and the IrCOMM service.

1.2.5 IAS

Support for “GetValueByClass” queries to provide general and IrCOMM specific device information.

1.2.6 IrCOMM

Support for the 3-wire raw IrCOMM protocol providing both serial (COM) and LPT (IrLPT) services.

2.0 Programming Methodology

The IrDA Virtual Peripheral consists of both interrupt and mainline code. The interrupt service routine implements the IrDA UART. Rather than implementing an application by calling the IrDA sub-routines the code works on a call-back system. Creating a new application involves implementing certain functions which will be called by the IrDA mainline code. This will become clearer in the section describing the application layer.

An IrDA frame is processed one byte at a time through every layer.

For reception the framing layer strips off the header and passes the payload data one byte at a time to the payload layer (part of the LAP layer). The payload layer strips off the address and command bytes and if the frame contains I data then it will pass the data one byte at a time to the LAP layer which will pass it to the LMP layer. The LMP layer will strip off the LMP address bytes and if the frame contains IrComm data then it will pass it to the application layer. At this point the frame has not yet been validated by the frame-check-sequence (FCS) CRC nor has it been validated as the correct frame number by the LAP layer. Once the integrity of the frame has been checked by the framing layer it will inform the payload layer which will inform the LAP layer. The LAP layer will verify the frame numbering and will inform the LMP layer of the validity of the frame which will in-turn inform the IrComm application layer.

The protocol stack is implemented as a number of layers, each one based on a state-machine. All the state-machines must operate in parallel and to achieve this the state machines are event-driven. No state machine may hold up the processor waiting for an event, but instead must complete its processing for the incoming event and return. The event handlers for a layer are subroutines where the name of the routine is predefined so that the call to the event handler can be coded in the originating code.

There are two types of events:

- Asynchronous Events
- Synchronous Events

Asynchronous events are events that are triggered by the interrupt service routine (ISR). When an event (e.g. a received IrDA byte) has been detected by the ISR will set the appropriate flag in a global ISR status register. The main code is a loop that tests these flags and will call any event handlers as appropriate.

Synchronous events are events that are generated by a layer to inform a higher layer of a situation.

For example, when a byte has been received by the IrDA UART in the ISR the IrDA UART data available flag will be set. These asynchronous flags are polled by the main routine and when the data available flag is detected the data available event handler in the framing layer will be called. The framing layer will obtain the data byte, process it, and return. However, if the framing layer detects the byte as payload data then before returning it will call the payload data available event handler in the payload layer. The payload layer will process the data, call any appropriate higher event handlers, and return. When the end-of-frame byte is received by the framing layer it will test the FCS bytes and will call either the valid or the error event handler in the payload layer. If appropriate the payload layer will inform its higher layer and so on.

The following rules apply to inter-layer events or calls:

- The routine name must start with the abbreviation of the calling layer followed by a ‘2’ followed by the abbreviation of the called layer.
- The call instruction must be proceeded by a page instruction.
- The return instruction from the called event must be a ‘retp’ instruction.
- The called code may change any global or shared registers.
- Upon returning the bank is assumed to have been changed.
- A paged call may be replaced by a paged jump if directly after the inter-layer call has been completed the code will issue a return instruction to return from a inter-layer call.

2.1 SOURCE CODE ORGANIZATION

The source code is so large it has been split among several files to make editing easier. These source files are concatenated together to form `IrDA.src` which is programmed into the SX.

Each separate source file implements a macros for code, data and register definitions. The file `Project.src` uses these macros to form the complete code. From `Project.src` it can be seen how the code is arranged in the program memory of the SX. This arrangement is quite important because it ensures that instruction restrictions (such as jump tables residing in the lower half of a page) are met.

The three application demonstrations can be selected by removing the commenting next to the appropriate macro in `Project.src`.

A simple batch file (`Asm.bat`) is supplied which will concatenate the source files together.

3.0 IrDA Layer Descriptions

The following sections describe each of the layers in the IrDA stack in more detail.

3.1 PHYSICAL LAYER

The physical layer consists of an IrDA UART to convert octets of data to an IR pulse stream and to process the received pulse stream into octets of data.

IR communication is inherently half-duplex in nature as the IR emitter used for transmission is physically close to the IR detector used for reception thus the detector will always detect what is being transmitted. Furthermore a minimum turnaround time must be respected to allow the daylight correction on the detector to recover from the transmission.

The octet encoding is dependant on the class of the speed, and for “ASYNC” speeds (9600-115200bps) the bit timing is the same as a conventual UART but based on pulses. A ‘0’ is encoded as a pulse and a ‘1’ is encoded as no pulse. The bit format is 1 start bit, 8 data bits, and 1 stop bit (no parity). The octet will start with a pulse for the start bit (start bit = ‘0’), followed by 8 data pulses/no-pulses, followed by a no-pulse for the stop bit (stop bit = ‘1’ - return to idle). The pulse can be of any width between 1.6us (3/16ths of the bit time at 115200bps) and 3/16ths of the total bit time for the UART speed.

3.1.1 SX Timing

The IrDA Virtual Peripheral implementation uses a constant pulse time of 2.16us (within the IrDA specifications) regardless of the speed. This time is 4/16ths of the bit time at 115200 allowing the use of a divide by 4 rather than a divide by 16 counter and reducing the counter resolution to 8-bit. The timer ISR routine will be executed every 2.16us (108 clock cycles @ 50MHz) thus 4 interrupts will be received per bit at the highest speed of 115200bps.

The HP HDSL1001 IrDA transceiver (which is described in the hardware section) has the following signal properties:

- RxIdle High
- TxIdle Low

It should also be noted that the IR transceiver LED timing is completely software controlled. If the SX device leaves the LED on continually then the IR transceiver may be permanently damaged as it will be operating at 4.5 times its absolute maximum average current rating.

For reception the pulse will generate an interrupt flag (falling edge) but will not generate an interrupt. If an interrupt occurs during the ISR execution the SX will not re-enter the ISR. By using the interrupt flag (rather than raising an interrupt) no interrupts are missed, even during ISR processing. The flag will be detected the next time the timer interrupts, and will be processed then.

3.1.2 Physical Layer API

When a byte has been received the global IrdaRxAvail flag will be set and the data will be available from the `IrdaRxData` register in the `IsrBank`.

A byte can be transmitted by first storing it in the `IrdaTxData` register in the `IsrBank` and then by setting the global flag `IrdaTxStart`. When the byte has been transmitted the ISR will set the global flag `IrdaTxEmpty`.

3.2 FRAMING LAYER

For transmission the framing layer adds the required framing information to the payload data. For reception the framing layer removes the framing information to recover the payload data. The framing information consists of turn-around delay bytes, beginning and end of frame bytes, a 16-bit CRC check, and the application of transparency bytes.

3.2.1 Frame Format

Framing is described in the IrDA IrLAP documentation (pages 112-118). The format of a frame is shown in Figure 3-1.

The initial beginning-of-frame bytes (BOFs) take care of the turnaround delay required for a IR receiver to recover after transmission. Initial BOFs are send as $FF rather than the BOF byte $C0 as recommended by the IrLAP specification.

The frame-check-sequence (FCS) is a 16-bit CCITT CRC covering all the payload data prior to the application of any transparency bytes required. The traditional look-up table approach requires a 512 byte look-up table thus is unrealistic to implement on a small communications controller. The FCS is instead calculated on a byte-by-byte basis as they are transmitted/received using just 21 words of sequentially executed code. The derivation of the FCS calculation can be found in Appendix A.

For reception all bytes received after the BOF (and after transparency recovery) are included in the FCS calculation until the end-of-frame (EOF) byte is received (i.e. including the FCS bytes). When the EOF is received the calculated FCS in memory is compared to the constant $F0B8 to determine the validity of the frame. It should be noted that the FCS is defined as the last two bytes prior to the EOF byte thus to remove the FCS all payload data passes through a 2-byte FIFO buffer before being passed as payload data to the higher layer.

Initial BOFs	BOF ($C0)	Payload Data	FCS (2 bytes)	EOF ($C1)

Figure 3-1. Frame Format

3.2.2 Frame Transparency

Prior to transmission extra escape characters are added to ensure payload data cannot prematurely terminate the frame by conflicting with a control byte. The application of transparency covers all data between the BOF and EOF bytes including the FCS. The control escape (CE) byte is defined as $7D. Any instance of a BOF, EOF, or CE in the data is encoded by inserting a CE byte followed by the original data byte xored with $20.

After transparency bytes have been applied the only instance of a BOF will be at the start of a frame thus if a BOF is received at any point any existing data will be discarded and the payload data started again.

3.2.3 Receive API

As a frame is being received the payload bytes are passed to the payload layer using fl2plRxData. When the end of the frame is detected, fl2plRxValid or fl2plError will be called indicating that the frame is complete and the payload data was valid /erroneous.

The frame can be rejected by the payload layer by calling pl2flRxIgnore or by calling pl2flTxStart to transmit a frame (see Transmit API). When a frame is rejected in this manner, the receive state is reset to idle, thus fl2plRxValid/fl2plRxError will not be called.

It should be noted that the framing layer does not time out. If transmission is interrupted part way through a frame then fl2plRxError will not be called until the start of a new frame is detected. If a higher layer times out and requests that a frame be transmitted then the receive state is reset thus fl2plRxError will not be called.

The fl2lapMediaBusy indication is used by the lap layer to reset the media idle test and is called before any other receive call.

fl2plRxData (w=Data)	A byte of payload data has been received
fl2plRxValid ()	The payload data is complete and has been validated.
fl2plRxError ()	The payload data passed is invalid.
pl2flRxIgnore ()	The payload layer has no interest in the frame.
fl2lapMediaBusy ()	A byte has been received

Figure 3-2. Framing Layer Receive API

3.2.4 Receive State machine

Table 3-1 shows the Framing Layer Receive State Machine.

Table 3-1. Framing Layer Receive State Machine

Current State	Event	Action	Next State
Idle	RxAvail = BOF		Begin
	RxAvail = Other		Idle
Begin	RxAvail = BOF		Begin
	RxAvail = EOF		Idle
	RxAvail = CE	Reset FCS. Add data to FCS. Pass RxData to payload layer via FIFO buffer.	Control
	RxAvail = Other	Indicate data error to payload layer (fl2plRxError).	Payload
Payload	RxAvail = BOF	Indicate data valid to payload layer (fl2plRxValid).	Begin
	RxAvail = EOF & FCS = valid	Indicate data error to payload layer (fl2plRxError).	Idle
	RxAvail = EOF & FCS = invalid		Idle
	RxAvail = CE	Add data to FCS Pass RxData to payload layer via FIFO buffer.	Control
	RxAvail = Other	Indicate data error to payload layer (fl2plRxError).	Payload

Table 3-1. Framing Layer Receive State Machine

Control	RxAvail = BOF	Indicate data error to payload layer (fl2plRxError).	Begin
	RxAvail = EOF	XOR RxData with $20. Add to FCS. Pass to payload layer via FIFO buffer.	Idle
	RxAvail = Other		Payload
All States	pl2flRxIgnore		Idle
	RxAvail = EOF & FCS = valid	Indicate data error to payload layer (fl2plRxError).	Idle
	RxAvail = EOF & FCS = invalid		Idle
	RxAvail = CE	Add data to FCS Pass RxData to payload layer via FIFO buffer.	Control
	RxAvail = Other	Indicate data error to payload layer (fl2plRxError).	Payload

3.2.5 Transmit API

The pl2flTxStart call starts the transmission of a frame header immediately and resets the receive state to idle.

When the framing layer can transmit a payload byte it calls fl2plTxData to obtain the data to transmit. The payload byte is returned in w and the z flag is set if this is the last byte to be sent. This process will be repeated until a last indication is detected.

Finally fl2plTxComplete is called once the frame has been completely sent (i.e. the stop bit of the EOF byte has been sent).

pl2flTxStarT ()	The payload layer wants to transmit a frame.
fl2plTxData (ret w=Data, z=Last)	Request the next payload byte.
fl2plTxComplete ()	The frame has been completely sent.

Figure 3-3. Framing Layer Transmit API

3.2.6 Transmit State Machine

Table 3-2 shows the Framing Layer Transmit State Machine

Table 3-2. Framing Layer Transmit State Machine

Current State	Event	Action	Next State
Idle	pl2flTxStart ()	Initialize FF counter. Send FF byte.	Begin
Begin	TxEmpty & Counter-- <> 0	Send FF byte.	Begin
	TxEmpty & Counter-- = 0	Initialize FCS. Send BOF byte.	Payload

Table 3-2. Framing Layer Transmit State Machine

Current State	Event	Action	Next State
Payload	TxEmpty	Get data from payload layer (fl2plTxData). Add to FCS.	
		Last flag not set. Jump to Send.	Payload
FCS Low	TxEmpty	Data = FCS Low XOR $FF. Jump to Send	FCS High
FCS High	TxEmpty	Data = FCS High XOR $FF. Jump to Send	Send End
Send End	TxEmpty	Send EOF byte	Wait End
Wait End	TxEmpty	Indicate complete to payload layer (fl2plTxComplete)	Idle
(Send)	Data <> BOF or EOF or CE	Send data byte	no change
	Data = BOF or EOF or CE	Store current state. Send CE byte	Control
Control	TxEmpty	Recover stored state. Send data byte XOR $20	previous state

3.3 LAP LAYER

The primary purpose of the link-access-protocol (LAP) layer is to manage the link to provide a connection-based bi-directional reliable data transfer service. It manages the discovery process, the connection process, and the transfer of data.

The LAP layer implementation is secondary only with two additional primary features:

- Support for initiating XID discovery.
- Support for sending unnumbered-information frames.

The LAP layer is implemented in the IrDA Virtual Peripheral as two parts – payload layer and LAP layer.

3.3.1 Discovery

In order for an IrDA device to open a connection with another IrDA device it must first know it is there and the discovery process must be able to cope with multiple IrDA devices in range. The discovery (XID) frames pass a 32-bit device address, hint bits as to the type of device, and a ‘nickname’ string that can be displayed to the user.

Both command and response XID processes have been implemented so that the SX can discover other devices and be discovered itself.

The discovery process is outside of any connection and so all frames relating to the discovery process are sent at 9600bps with a minimum turnaround of 10ms.

When a device wishes to discover devices in range it will transmit a XID command frame indicating slot 0 of N slots (typically 8). Every device receiving a XID command frame will generate a random slot number between 0 and N-1 to send a XID response frame. If the random slot number is 0 then it will transmit a reply immediately, otherwise it will wait until it receives a XID command frame with a matching slot number. Once the commander has sent out the N XID command frames with slot numbers from 0 to N-1 it will send its own details in a XID command frame with a slot number of $FF.

The hint bits and nickname field are referred to as the XID discovery information field and are included in all XID response frames. For XID command frames only the final frame with a slot number of $FF will contain this field.

The 32-bit device address is a random number that is chosen at start up. It is possible that 2 devices will choose the same device address and so the XID frame can be addressed to specific devices and can request that a new 32-bit address be generated.

The IrDA standard recommends that if a XID command frame is received with a slot number greater that 0 while a slot number has not been chosen (i.e. the slot 0 command frame was missed) then a slot number should be chosen between the received slot number and the maximum. This complicates generating the random slot number and requires registers to keep track of the XID process in case more XID frames are missed. To avoid this the SX generates a random slot number every time a slot 0 XID command frame is detected and will send a XID response when the slot number matches the chosen slot number. The SX will respond correctly to a XID discovery as long as the first frame of the discovery process is received correctly. If the first frame is missed then the SX will not respond and the discovery process will have to be repeated.

The random 32-bit device address is chosen at start-up and will be re-generated if a XID command frame is received that requests a new address be generated. Address conflicts can only detected and corrected by primary IrDA devices. The IrDA Virtual Peripheral is classed as a secondary only device and so does not need to detect conflicts. The primary service of initiating XID discovery is only provided for information gathering of other devices in range.

3.3.2 Connection Set-up

Once a primary IrDA device has discovered a another device it can open a connection with it allowing the bi-directional transfer of reliable data. The IrDA Virtual Peripheral is secondary only and so cannot initiate a connection.

The request for connection frame (set-normal-response-mode SNRM frame) from the primary IrDA device contains the destination 32-bit address, its own 32-bit address, a 1 byte connection address, and a list of supported connection parameters.

If the destination address matches the 32-bit IrDA Virtual Peripheral device address then a connection acknowledge frame (unnumbered-acknowledgment UA frame) will be sent in reply indicating the actual connection parameters. The actual connection parameters are the highest commonly supported options between the two devices.

Once the UA reply frame has been sent by the secondary station it applies the determined connection parameters. The primary station receiving the UA frame will read the connection parameters from the frame, apply them, and reply with a receive-ready (RR) frame at the new speed. The connection has now been established and data can now be transferred.

3.3.3 Connection Parameters

The main parameter that is negotiated is the baud rate. The IrDA Virtual Peripheral supports 9600, 19200, 38400, 57600, and 115200bps. The IrDA standard also includes optional support for 2400bps only devices, but this is not supported by the IrDA Virtual Peripheral.

All other parameters are set to the lowest values so they do not need to be negotiated but instead can be sent as fixed data (as suggested in the IrDA Lite documentation).

The window size is the maximum number of unacknowledged data frames that can exist at one time. The IrDA Virtual Peripheral does not buffer the data frames and so negotiates the window size to be 1. This means that a station can only send 1 frame before passing control back to the other station.

The maximum payload data size is set to the minimum (and default) of 64 bytes to avoid having to process and remember the maximum supported size of the initiating station.

The minimum turnaround time should be sent as the correct time for the specific IR transceiver used. The value transmitted is 5 ms and can be changed in the string segment of the payload layer (`plSNRMParam`). Related to the turnaround time is the additional BOFs parameter. This is a request for additional BOF characters to be sent at the beginning of a frame to allow for a slow interrupt response. The IrDA Virtual Peripheral does not process or store the minimum turnaround time or the additional BOF request from the initiating station but instead always sends 10 ms worth of additional BOF bytes at the start of each frame. The 10 ms time is the worst possible case so that the IrDA Virtual Peripheral can communicate with any IrDA device however it should be possible to reduce this time to 5 ms and maintain compatibility. The number of additional BOFs sent can be changed in the lap layer (`lmpMinTurnaround`).

The turnaround time is the maximum time a station can have control of the IR medium before passing control back to the other station. For “ASYNC” speeds (speeds up to 115200bps) the turnaround time must be 500ms.

The link disconnect threshold time is the maximum time to wait without hearing from the remote station before assuming that the connection has been lost. The disconnect time is set to the minimum of 3 seconds.

3.3.4 Data Flow

During a connection, only one frame will be sent by a station before it passes control (permission to send a frame) back to the other station. The passing of control is achieved by setting the “final” bit in the command byte and so this bit should be set for all frames. Any frames received that do not have the final bit set will be ignored as suggested in the IrDA Lite documentation.

One frame (S or I) will be sent from the primary station and the control will be passed to the secondary station. The secondary station will reply with one frame (S or I) and pass control back to the primary station.

If there is data to be sent then an I frame will be sent. The command byte of an I frame contains a 3 bit frame number as well a 3-bit acknowledge number. The acknowledge number is used to confirm the receipt of frames numbered up to but non including the acknowledge number. If there is no data to be sent then a S frame will be sent. S frames do not contain data and so are not numbered, but do contain a 3-bit acknowledge number used in a similar way as a I frame acknowledgment. If a error is detected in a received frame then a S frame will be sent in reply to explicitly reject the frame.

A connection can be maintained indefinitely without data by use of S frames.

3.3.5 Disconnection

The connection is disconnected by an incoming disconnect (DISC) frame from the primary station. Upon receipt of a DISC frame the default connection parameters will be reinstated (9600bps, 10 ms minimum turnaround). The secondary can send a request disconnect (RD) frame if it wishes to disconnect which should result in a DISC reply from the primary, however, the RD frame can only be sent when the secondary station has control (i.e. in its turn). The connection is also covered by a 3 second watchdog (WD) timer. If no valid frames are received within the 3 second timeout then the connection will be assumed to be dead and will be handled in the same way as for a received DISC frame.

3.3.6 Frame Structure

Figure 3-4 shows the LAP Frame Format. The address byte consists of a 7 bit connection address and a 1 bit command/response bit (LSB). The IrDA Virtual Peripheral implementation uses an 8-bit connection address register and will only accept frames with the matching byte in the address field. When the LAP layer is idle (i.e. the normal-disconnect-mode (NDM) state) the connec-

tion address register is set to $FF to accept broadcast command frames and reject response or addressed frames. When the LAP layer is initiating discovery (i.e. the QUERY state) the connection address is set to $FE to accept broadcast response frames and reject command or addressed frames. When connected – normal-response-mode (NRM) state the connection address is set to the given connect address (from the connection request frame) with the command bit set therefore accepting addressed command frames and rejecting broadcast, response, and incorrectly addressed frames.

The command byte determines the frame type. There are three basic types of command – unnumbered (U), supervisory (S), and information (I). U commands do not relate to reliable data and are used for tasks such as discovery and link connection/disconnection. S commands are used to acknowledge or reject received I frame data and are also used to maintain the connection when there is no data to be transferred. I frames are used to transfer reliable data and also contain an acknowledgment for received I frame data encoded in the command byte.

The information field is command specific. For I frames it contains the reliable data to be transferred, for U frames it contains parameters relating to the command, and for S frames it will be empty (0 bytes in length).

I frame numbering for acknowledgment/reject is based on a 3 bit frame number to protect against duplicate frames.

Initial BOFs	BOF ($C0)	Address (1 byte)	Command (1 byte)	Information (up to 64 bytes)	FCS (2 bytes)	EOF ($C1)

Figure 3-4. LAP Frame Format

Table 3-3. Payload Frame Types

Type Constant	Command Type	Description
iFrame	Information	Information data frame.
sRRFrame	Supervisory	Acknowledge I frame and receive-ready indication.
sRNRFrame	Supervisory	Acknowledge I frame and receive-not-ready indication.
sREJFrame	Supervisory	Reject I frame.
sSREJFrame	Supervisory	Selective reject I frame.
uUIFrame	Unnumbered	Unnumbered-information data frame.
uDISCFrame	Unnumbered	Disconnect frame.
uUARspFrame	Unnumbered	Response to a SNRM or DISC frame.
uNRMFrame	Unnumbered	SNRM or RNRM – Open or close a connection.
uTESTFrame	Unnumbered	Test frame.
uFRMRRspFrame	Unnumbered	Frame reject response frame.
uDMRspFrame	Unnumbered	Disconnected-mode indication response frame.
uXIDCmdFrame	Unnumbered	Discovery command frame.
uXIDRspFrame	Unnumbered	Discovery response frame.

3.3.7 Implementation - Payload Layer Receive

For incoming frames the payload layer tests the connection address, and if it is correct it will inform the LAP layer of the type of the frame. The information field will be processed or passed (in the case of a data frame) as appropriate for the type of frame, and a Valid/Error message will be sent to the LAP layer when complete.

When the LAP layer is informed of the incoming frame type it may choose to ignore the frame by calling lap2plRxIgnore which will cause the payload layer to ignore the frame.

Figure 3-5 shows the Payload Layer Receive API.

pl2lapRxFrame (w=Type)	A correctly addressed frame of the given type is being received.
pl2lapRxValid ()	The frame is complete and has been validated.
pl2lapRxError ()	The frame is invalid.
pl2lapRxXIDData (w=Data)	Data from the XID information field
pl2lapRxIData (w=Data)	Data from an I frame
pl2lapRxUIData (w=Data)	Data from an UI frame
lap2plRxIgnore ()	The lap layer has no interest in the frame.

Figure 3-5. Payload Layer Receive API

If the lap layer calls lap2plRxIgnore to ignore the frame then the payload layer will inform the framing layer that the frame should be ignored thus no more data will be received from the framing layer and no Valid/Error indication will be given.

The command byte is recorded by the payload layer so that the LAP layer can examine the it once a frame has been completed. This allows the LAP layer to gain access to the frame numbering bits encoded in the command byte.

For I (reliable data) and UI (unreliable data) frames all data is passed to the LAP layer using pl2lapRxIData or pl2lapRxUIData.

For XID frames the frame information field is internally processed by the payload layer except for the XID discovery information field which is passed to the LAP layer using pl2lapRxXIDData. The received 32-bit source address is stored so that it can be used in a reply if the frame is an XID command frame and the LAP layer requests that an XID response be sent. The received 32-bit destination address is tested to ensure it is either broadcast or specifically destined for the IrDA Virtual Peripheral, and the XID flags and slot number are stored for future reference by the LAP layer. If there is any error in the frame such as a wrong destination address or an unsupported version then the lap layer will receive a pl2lapRxError message. Note that the payload layer only keeps the current source address in memory and does not generate a list of received source addresses.

For SNRM frames the information field is internally processed by the payload layer. The received 32-bit source address is stored so that it can be used in a reply if requested by the LAP layer. The received 32-bit destination address is tested to ensure it is either broadcast or specifically destined for the IrDA Virtual Peripheral, and if not then the lap layer will receive a pl2lapRxError indication. The received connection address is stored but not applied (it cannot be applied until the frame has been validated and the LAP layer requests that the connection be accepted). And finally the baud rate parameter is extracted from the received supported connection parameters and stored for future use. When the frame is validated the LAP layer will accept the connection and the stored data will be used to form the connection-accept reply.

For all other frame types the information field is ignored.

3.3.8 Implementation - Payload Layer Transmit

The payload layer provides the LAP layer with the ability to request that a frame of a given type be sent.

When the LAP layer requests that a frame be sent it will be sent immediately and the receive state will be reset to idle.

For a XID command frame with a slot number of $FF a XID frame will be sent with the XID discovery information field, otherwise the discovery information field will be omitted.

A request to send an UI command frame will be followed by requests for data using pl2lapTxUIData until there is no more data to be sent (indicated by the last flag being set by the LAP layer).

A lap2plSNRMAccept will cause the connection address to be applied and a UA connection acknowledgment frame to be sent with appropriate negotiation parameters. This routine can only be called directly following a valid received SNRM frame as information from the received frame is used to transmit the UA reply frame.

The transmit I response routine differs from the I frame receive routine in that the payload layer will internally send the DSLAP and SLSAP bytes of the LMP layer rather than the two bytes being passed as data (refer to the LMP section of this document). These two bytes will be the first two data bytes sent in the information field and so they must have been correctly set prior to calling lap2plTxIRsp. Once the DLSAP and SLSAP bytes have been sent further data will be requested by use of the pl2lapTxIData call until the last flag is set indicating that there is no more data to send. Note that the DLSAP and SLSAP bytes are part of the LMP layer data and as such are of no interest to the LAP or payload layers. They are sent by the payload layer in this manner to simplify the LMP layer.

An XID response frame can only be send directly following a valid received XID command frame as information from the received frame is used to transmit the XID response frame.

lap2plTxXIDCmd (w=slot)	Send an XID command frame with the given slot number.
lap2plTxUICmd ()	Send an UI command frame.
lap2plSNRMAccept ()	Apply connection address and send UA connection accept frame with appropriate negotiation parameters.
lap2plTxSimpleRsp (w=cmd)	Send a frame with the given command and with no information field.
lap2plTxIRsp ()	Send an I response frame.
lap2plTxXIDRsp ()	Send an XID response frame
pl2lapTxIData (ret w=data, z=last)	Request for the next I data byte.
pl2lapTxUIData (ret w=data, z=last)	Request for the next UI data byte.
pl2lapTxComplete ()	The frame has been completely sent.

Figure 3-6. Payload Layer Transmit API

3.3.9 Implementation - LAP Layer API

lap2lmpConnectIndication ()	A LAP connection has been established.
lap2lmpDisconnectIndication ()	The LAP connection has been terminated.
lmp2lapDisconnectRequest ()	The LMP layer wishes to terminate the LAP connection.

Figure 3-7. LAP Layer Connect API

The `DisconnectionRequest` will result in the link being disconnected following the IrDA specifications. When the SX has control of the link it will request disconnection from the primary station and the link will be formally disconnected when the primary station returns a disconnect frame. During this time no further data frames will be accepted and so no further data will be passed to the LMP layer. When the disconnect frame is received any outstanding non-validated data will be rejected and then the `DisconnectIndication` will be called.

The LMP layer uses reliable I frames for data transmission and reception. Received data will confirmed by a `RxValid` message once both the FCS and the frame numbering have been checked. Transmitted data will be confirmed by a `TxValid` message once the remote station has correctly acknowledged the receipt of the transmitted frame.

Because the acknowledge of transmitted data can be part of a received data frame the LMP layer may be passed received data before the transmitted data has been confirmed. This means that the LMP layer must be capable of re-transmitting a frame after more incoming data has been received.

For solid bi-directional data flow the events will occur in the following order:

lap2lmp**Tx**Start
lap2lmp**Tx**Data * n
lap2lmp**Rx**Data * n
lap2lmp**Tx**Valid/Error
lap2lmp**Rx**Valid/Error
lap2lmp**Tx**Start
etc.

lap2lmpConnectIndication ()	A LAP connection has been established.
lap2lmpDisconnectIndication ()	The LAP connection has been terminated.
lmp2lapDisconnectRequest ()	The LMP layer wishes to terminate the LAP connection.
lap2lmpRxData (w=Data)	I frame data has been received.
lap2lmpRxValid ()	The I frame data received since last Valid/Error message has been validated.
lap2lmpRxError ()	The I frame data received since last Valid/Error message is invalid.

Figure 3-8. LAP Layer LMP Receive API

The application layer can request the discovery process be initiated by calling DiscoveryRequest. If there has been no IR detected in the past 500 ms (as required by the IrLAP specifications) then the discovery process will be initiated and z will be returned clear. If there has been IR detected in the past 500 ms then the request will be refused and z will be returned true. Any attempt to initiate the discovery process during a connection or during a discovery process will be rejected.

The application layer can request that a unreliable broadcast data (UI) frame be sent outside of a connection by calling TxUIStart. If there has been no IR detected in the past 500 ms (as required by the IrLAP specifications) then the frame will be transmitted and z will be returned clear. If there has been IR detected in the past 500 ms or there is a connection open then the request will be refused. The data will be requested from the application layer as it can be sent by use of the TxUIData event. The LAP layer does not enforce the 64 byte limit on the length of the data within a UI frame and so it is up to the application layer to limit the data to 64 bytes if it wants to maintain IrDA compliance.

All broadcast UI data frames received during the disconnected (NDM) state will be passed to the application layer. Figure 3-9 shows the LAP Layer Application API..

app2lapDiscoveryRequest (ret z=busy)	Request that the discovery process be initiated.
lap2appRxUIData (w=data)	Incoming UI data frame.
lap2appRxUIValid ()	Frame was validated.
lap2appRxUIError ()	Frame was invalid.
app2lapTxUIStart (ret z=busy)	Request that a frame be sent.
lap2appTxUIData (ret w=data, z=last)	A request for UI data to be transmitted.
lap2appTxUIComplete ()	The UI frame has been completely transmitted.

Figure 3-9. LAP Layer Application API

3.3.10 Implementation – LAP Layer State Machine

The IrDA Virtual Peripheral LAP layer manages the LAP secondary-only state machine from the IrDA Lite specification.

The implementation differs from the specification in the following ways:

- There is no OFFLINE state (i.e. the LAP layer can always receive frames).
- The XID response to an XID command frame as stated in Discovery section (i.e. slot 0 must be received for the SX to respond).
- The addition of the QUERY state so the IrDA Virtual Peripheral can initiate XID discovery following the IrDA Lite primary state machine.
- The IrDA Virtual Peripheral will always automatically accept an incoming connection request rather than asking a higher layer if it should accept the connection request.
- The ability to transmit UI command frames (NDM state only).

Note that the payload layer automatically filters the frames such that the LAP layer will only be informed of frames that are correctly addressed command frames with the final bit set.

Table 3-4. LAP State Machine

Current State	Event	Action	Next State
NDM	DiscoveryRequest & Media is idle	Slot number = 0. Send XID command frame. Start slot timer.	QUERY
	Send UI request & Media is idle	Send a UI command frame	NDM
	uXIDCmd frame	If the GenerateNewDeviceAddr flag is set then ask payload layer to generate a new device address. If received slot number = 0 the generate a random slot number. If received slot number = chosen slot number = 0 then send a XID response frame.	NDM
	uNRM frame	Apply connection address. Send UA frame to accept connection. Apply connection parameters. Initialize frame numbering registers. Indicate connection to LMP layer.	NRM
	uUI frame	Pass data to application layer	NDM
QUERY	Timeout & slot number < 8	Increment slot number. Send XID command frame. Start slot timer.	QUERY
	Timeout & slot number = 8	Send XID command frame with slot number of $FF	NDM
	uXIDRsp frame	Pass data to user	QUERY
NRM	DisconnectRequest		SCLOSE
	I frame	Jump to TestNr.	NRM
	sRR Frame	RemoteBusy = false. Jump to TestNr.	NRM
	sRNR Frame	RemoteBusy = true. Jump to TestNr.	NRM
	sREJ Frame or sSREJ Frame	Jump to TestNr.	NRM
	SNRM Frame	Send RD frame. Start WD timer	SCLOSE
	DISC Frame	Send UA response Apply default connection parameters. Indicate disconnect to LMP layer.	NDM
	All other frames (command, final)	Send S frame	NRM
	Timeout	Apply default connection parameters. Indicate disconnect to LMP layer.	NDM
NRM (TestNr)	Received Nr = NrAck	Start WD Timer. NrNotAck := NrAck. Inform LMP layer that transmitted data has been acknowledged if appropriate. Jump to TestNs.	same

Table 3-4. LAP State Machine

	Received Nr = NrNotAck	Start WD Timer. NrAck := NrAck – 1. Inform LMP layer that transmitted data must be resent if appropriate. Jump to TestNs.	same
	Received Nr = other	Send RD frame. Start WD timer	SCLOSE
NRM (TestNs)	Received Ns = Ns	Ns := Ns + 1 Inform LMP layer that received data is valid if appropriate. Jump to SendData.	same
	Received Ns <> Ns	Inform LMP layer that received data is valid if appropriate. Send S frame.	same
	Frame was a S frame	Jump to SendData	same
NRM (SendData)	LMP layer has data to send	Send I frame. NrAck := NrAck + 1.	same
	LMP layer does not have data to send	Send S frame.	same
SCLOSE	uRDFrame	Send UA response Apply default connection parameters. Indicate disconnect to LMP layer.	NDM
	All other frames (command, final)	Send RD frame. Start WD timer	SCLOSE
	Timeout	Apply default connection parameters. Indicate disconnect to LMP layer.	NDM

3.4 LMP LAYER & IAS SERVER

The IrDA Link-Management-Protocol (IrLMP) layer serves to multiplex the IrLAP connection between a number of higher layers and/or applications using the reliable service of the IrLAP layer. The LMP layer requires that a LMP connection be created for every instance of a service or application that wishes to communicate across the link so that all the data can be correctly multiplexed to the correct destination services/applications. The addressing is achieved by adding a one byte destination Link-Service-Access-Point (DLSAP) address and a one byte source LSAP (SLSAP) to every reliable I frame.

The Information-Access-Server (IAS) allows IrDA devices to get specific details about the high layer services the IrDA device offers. A request for information will consist of a class-name in text form (e.g. IrDA:IrCOMM for IrComm) and a attribute-name in text form (e.g. Parameters for IrComm). If the IAS server has a entry of the correct class-name then it will respond by returning the data associated with the requested attribute. The text names are generally ASCII but are considered to be language-independent byte sequences and are case sensitive The IAS implementation is secondary only and as such can reply to requests for information but cannot request information itself.

In the IrDA Virtual Peripheral both the IrLMP layer and the IAS have been implemented together in the one layer referred to as the LMP layer.

3.4.1 LMP LSAP Addresses

The following services are provided:

- IAS server (LSAP = 0)
- IrComm service (LSAP = 5)

3.4.2 LMP Behavior

The IrComm service LMP state is remembered as either connected or not connected by the use of the `lmpAppLSAP` variable. This variable holds the LSAP of the remote service that opened the connection to the IrComm service and if 0 the IrComm service is considered to be not connected.

The IAS service LMP state is not stored. Any attempt to open the IAS service is accepted and any attempt to close it is ignored.

Any attempt to access an unknown LSAP or the IrComm LSAP while it is not connected will result in a `BadLSAP`

event which once the frame has been confirmed will result in a LAP link disconnect.

Any received LMP command other that a “connect” or a “disconnect” will be ignored.

3.4.3 LMP IrComm API

The LMP API provides the IrComm API to the application layer and is described in the Application layer section of this document.

3.4.4 LMP Implementation Notes

The IrLMP layer uses reliable I-frames and as the information passed to/from the LAP layer is unreliable until confirmed, the LMP implementation must be capable of rejecting or re-transmitting data. The LMP layer implementation must be able to retransmit a frame after a frame has been received thus all transmit and receive resisters must be separate. Normally if the transmitted data is not-acknowledged then the reply will be a S frame thus no data would have been sent, however it is still possible to receive data followed by a transmit not-acknowledge if a frame is lost.

For solid bi-directional data flow the events will occur in the following order:

lap2lmp**Tx**Start
lap2lmp**Tx**Data * n
lap2lmp**Rx**Data * n
lap2lmp**Tx**Valid/Error
lap2lmp**Rx**Valid/Error
lap2lmp**Tx**Start
etc.

For reception, all I-frame data bytes are passes to the LMP layer. The first two bytes will be the DLSAP and the SLSAP, followed by the higher-layer data (up to 62 bytes).

For transmission, the DLSAP and SLSAP bytes will be transmitted by the payload layer. This allows the LMP layer to use a single TxState value for each of the possible types of transmission thus if a retransmission is requested then the state will remain unchanged. When the LAP layer gives permission to send a frame the DLSAP and SLSAP bytes are copied into the appropriate payload registers for transmission. All data requested from the LAP layer will consist of the higher-layer data. Also note that if a transmitted frame is requested to be repeated then the DLSAP and SLSAP registers in the payload layer will have been lost and so are transferred again.

The LAP layer will send a `RxError` and/or a `TxError` message if required before sending a `DisconnectIndication` thus the LMP layer does not need to do any special processing on a `DisconnectIndicaton` to inform the application layer.

3.4.5 Information Access Service

The IAS provides the required GetValueByClass service with no optional services.

The IAS holds two classes:

- “Device” class
- “IrDA:IrCOMM” class

3.4.5.1 IAS Device Class

The IAS Device class holds two attributes:

- “DeviceName” which returns the user string “SX IrDA IrComm Demonstration”
- “IrLMPSupport” which returns IrDA version 1 with no additional IAS services and no additional MUX services supported (This attribute is not requested by a Win95 connection and so has not been completely tested).

3.4.5.2 IAS IrComm Class

The IAS IrComm class holds two attributes:

- “Parameters” which returns that 3-wire raw serial or parallel is supported (no port name is sent).
- “IrDA:IrLMP:LsapSel” which returns the LSAP of the IrComm service = 5 (This number is arbitrary, between 1 and 6F).

3.4.6 IAS Implementation Notes

The incoming strings (classname and attribute) are made up of a one byte length followed by length bytes of case-sensitive text. All the strings that the IrDA Virtual Peripheral needs to check for are of different lengths and so the IrDA Virtual Peripheral uses the length byte to determine which string to test for. If the string does not match then a error reply will be generated with an appropriate error number indicating the cause of the error.

3.5 APPLICATION LAYER

3.5.1 IrComm

The IrComm interface provides a reliable bi-directional virtual COM/LPT interface. The IrComm interface can be considered to be that of a UART with a cable connecting it to the UART of the other station.

The IrDA Virtual Peripheral does not buffer frames and so the API is not quite this ideal. All data sent must be able to be resent and all data received must be able to be discarded if requested. Up to 62 bytes of data can be sent/received before being confirmed.

lmp2appRxCommData (w=data)	Incoming IrComm data.
lmp2appRxCommValid ()	Data bytes passed since last Valid/Error message have been validated.
lmp2appRxCommError ()	Data bytes passed since last Valid/Error message are erroneous.
lmp2appTxCommStart (ret z=none)	A request to find out if the application has IrComm data waiting to be sent.
lmp2appTxCommData (ret w=data, z=last)	A request for IrComm data so that it can be transmitted.
lmp2appTxCommValid ()	All data passed since last Valid/Error message were acknowledged as received by the remote station.
lmp2appTxCommError ()	All data passed since last Valid/Error message were discarded and will need to be sent again.

Figure 3-10. IrComm API

If the application was simplex (one direction only), for example a printer, then the data could be buffered as the SX would be capable of buffering a 62 byte frame. It is not, however, capable of buffering both a 62 byte transmit and a 62 byte receive frame.

The application layer cannot call the LMP layer. Instead the LMP layer will call the “event handlers” in the application layer as specified in Figure 3-10. An event handler should be in the following form:

```
lmp2appRxCommData
      bank    ApplicationBank       ;Correct local register bank
      ...                           ;Event handling code
      retp                          ;Paged return
```

For reception the LMP layer will call the application layer’s lmp2appRxCommData event handler with the data byte in the w register. The application is free to process the byte however it must return when it has finished processing the data and cannot hold up the CPU waiting for something to happen. This event handler will be called for every IrComm data byte in the incoming frame (up to 62 times) before the LMP layer calls lmp2appRxCommValid to validate the data or lmp2appRxCommError to reject the data.

For transmission the LMP layer will ask if there is any data to be sent by calling lmp2appTxCommStart. The application layer should return z clear if there is data to send. For example:

```
lmp2appTxCommStart
      bank    ApplicationBank       ;Correct local register bank
      test    DataCounter           ;Z is set if no data to send
      retp                          ;Z is clear if data to send
```

If the application layer indicates that there is data to be sent then the LMP layer will call lmp2appTxCommData when it is ready to transmit the first byte of data. The application layer must return with w containing the data byte to be sent and z indicating if there is more data to be sent. If z is true then this is the last byte to be sent and no more data will be requested. If z is false then another byte will be requested when the LMP layer is ready to accept it. The LMP layer will accept up to 62 bytes of data for a frame thus the 62nd byte will be considered to be the last byte by the LMP layer regardless of the z flag.

```
lmp2appTxCommData
      bank    ApplicationBank       ;Correct local register bank
      mov     w, DataPointer        ;w = Pointer to register with data
      inc     DataPointer           ;increment pointer to next register
      mov     FSR, w                ;apply register pointer
      mov     w, INDF               ;w = data to send
      bank    ApplicationBank       ;ensure correct register bank
      dec     DataCounter           ;Z is set if no more data to send
      retp                          ;Z is clear if more data to send
```

No indication is given when a frame has been sent. The `lmp2appTxValid` or `lmp2appTxError` handlers will not be called until the remote station responds with an acknowledge. This poses a problem in that data can be received before the transmitted data has been acknowledged and so transmitted data may need to be resent after more data has been received. Normally if the transmitted data is not-acknowledged then the reply will be a S frame thus no data would have been sent, however it is still possible to receive data followed by a transmit not-acknowledge if a frame is lost.

For solid bi-directional data flow the events will occur in the following order:

lmp2app**Tx**CommStart
lmp2app**Tx**CommData * n (up to 62)
lmp2app**Rx**CommData * n (up to 62)
lmp2app**Tx**CommValid/Error
lmp2app**Rx**CommValid/Error
lmp2app**Tx**CommStart
etc.

If the connection is terminated or times-out while there is unconfirmed data then the appropriate error message will be passed.

3.5.2 Discovery

The XID discovery process can be initiated by the application layer by calling `DiscoveryRequest`.The application should call the lap layer in the following way:

```
...
page    app2lapDiscoveryRequest     ;Set page bits for call
call    app2lapDiscoveryRequest     ;call lap layer
bank    ApplicationBank             ;change to correct local bank
...
```

The request will be accepted if there is no connection and there has been no IR detected in the past 500ms. Acceptance will be indicated by z being returned false and refusal by z being returned true. The XID Information field will not be passed to the application layer but will be displayed out the debug port if the `ShowXIDInfo` debug has been enabled.

3.5.3 UI Transport

The unnumbered-information (UI) service provides a broadcast unreliable connectionless data service for the application layer..

lap2appRxUIData (w=data)	Incoming UI data frame.
lap2appRxUIValid ()	Frame was validated.
lap2appRxUIError ()	Frame was invalid.
app2lapTxUIStart (ret z=busy)	Request that a frame be sent.
lap2appTxUIData (ret w=data, z=last)	A request for UI data to be transmitted.
lap2appTxUIComplete ()	The UI frame has been completely transmitted.

Figure 3-11. UI Transport API

The reception of a UI data frame works in the same way as for IrComm data. The LAP layer will call the application layer's `lap2appRxUIData` event handler with the data byte in the w register. The application is free to process the byte however it must return when it has finished processing the data and cannot hold up the CPU waiting for something to happen. This event handler will be called for every data byte in the incoming frame (up to 64 times) before the LAP layer calls `lap2appRxUIValid` to validate the data or `lap2appRxUIError` to reject the data.

For transmission the application layer must call the LAP layer's `app2lapTxUIStart` function to request that a UI frame be sent. The LAP layer can only send a UI command frame outside of a connection and only if no IR has been detected in the past 500ms. The LAP layer will return z false if it will send the frame or z true if it refuses the request.

If the frame can be sent the LAP layer will request the data as it can send it by calling the application layer's `lap2appTxUIData` event handler. This function return with w containing the data byte to be sent and z indicating if there is more data to be sent. If z is true then this is the last byte to be sent and no more data will be requested. If z is false then another byte will be requested when the LAP layer is ready to accept it. When the frame has been completely sent by the physical layer the LAP layer will call the `lap2appTxUIComplete` event handler to indication completion. Note that the LAP layer does not enforce the 64 byte limit on the length of the data within a UI frame and so it is up to the application layer to limit the data to 64 bytes if it wants to maintain IrDA compliance.

The application should call the lap layer in the following way:

```
...
page    app2lapTxUIStart;Set page bits for call
call    app2lapTxUIStart;call lap layer
bank    ApplicationBank       ;change to correct local bank
...
```

4.0 Sample Applications Using the Virtual Peripheral

The IrDA Virtual Peripheral can be used in two different scenarios: for communication with another SX running the Virtual Peripheral or with a Windows 95 computer (or similar) with an IrDA port. The next two sections describe three demonstrations of the Virtual Peripheral in these two scenarios.

Note:Since the Virtual Peripheral complies with the IrDA standard it will communicate with any other IrDA compliant device (such as a 3Com PalmPilot or digital camera). For this to be useful, the other device would also need to support IrComm (there are many high protocols at the top level of the IrDA stack and all are optional) or the SX would need to support the particular protocol used by the device.

4.1 SX TO WINDOWS COMMUNICATION (IRCOMM)

4.1.1 Transparent IrComm Application Description

The 'Transparent IrComm' sample application connects the Debug UART port to the IrComm virtual COM port of a PC using the IrDA IrComm reliable connection based protocol. Once a connection has been established by the PC any characters sent down the PC's virtual com port will appear out the SX Debug port and vice-versa.

The application does not buffer the data and does not send the valid/error messages out the debug port. In the rare case that an error occurs both the corrupt data and the valid data will be displayed. All frame errors, transmit not-acknowledges, and receive frame numbering errors will still be shown on the 'ERR' LED.

The application cannot transmit data until it has control of the IR link and so any characters received down the debug port for transmission must be stored until transmission is possible. To achieve this the application includes an 8-byte buffer for data pending transmission.

4.1.2 SX to PC IrComm Application Description

The 'SX to PC IrComm' sample application will respond to user commands entered into a terminal window connected to the virtual IrComm port of the PC. The SX will respond as follows:

- '?' will return the text string 'SX IrComm Terminal' to the user.
- 'c' will return the value of port c (raw byte) to the user.
- 'r' will return the 128 byte general register file (raw bytes) to the user.

The application is completely reliable in that it will correctly discard invalid received data and will correctly resend any transmitted data as requested.

4.1.3 IrComm requirements

- MS Windows 95/98 machine.
- Infrared IrDA port.
- Infrared drivers installed
  (The Windows 95 IrDA driver can be found at `www.microsoft.com/windows/95downloads/`).
- A serial connection to the SX if debug is of interest (The serial cable should be a straight-through cable with at least TX, RX, and GND connections.
- Windows terminal software capable of connecting to the IrDA virtual com port.

Note:Windows 95 HyperTerminal can only connect to COM1-4 and if the PC has a physical COM4 then the virtual com port will be higher.

4.1.4 IrComm Operation

- Double-click on the "Infrared" icon in control panel to show the Infrared window.
- Under 'Options' ensure that 'Search for devices in range" is selected (typically every 3 seconds).
- Under 'Options' note the 'Providing application support' virtual LPT and COM port numbers.
- Open a terminal program for the debug on the appropriate COM port at 115200,n,8,1, no handshaking.
- Power up the SX. Power is indicated by the 'PWR' LED.
- The Infrared monitor should indicate that there is a device in range called 'SX IrComm Device'
- The IR RX indication LED should be indicating received data about every 3 seconds. The IR TX indication LED should flash at a random interval into the RX phase indicating a reply in a random slot number.
- Open another terminal program for the IrDA virtual com port. Use the 'application support' COM port number noted from the Infrared monitor with any parameters (they are irrelevant). Note that Windows does not make these virtual ports available to a DOS window.
- The Infrared monitor should say 'Communicating with – Name: SX IrComm Device, Description: SX IrDA IrComm Demonstration', 'Good at 115.2 kbps'.
- If the 'ShowConnect' debug is enabled in the SX code then the debug window will show a '['when the connection has been established and a ']' when the connection has been terminated or lost.
- If the 'Transparent IrComm' application code is being used on the SX then anything entered into the virtual COM port terminal window will appear in the debug terminal window and vice-versa.
- If the 'SX 2 PC IrComm' application code is being used on the SX then the SX will respond to a '?' (in the virtual COM port terminal window) by returning the text string 'SX IrComm Terminal', a 'c' by returning the value of port c, and to a 'r' by returning the 128 byte register dump.
- Connection will be maintained until the virtual COM port is closed or the devices cannot see each other for more than 3 seconds.
- Another interesting test is to install a printer driver in windows connected to the virtual LPT port. With the 'Transparent IrComm' application, all the graphical printer data will be sent out the debug port of the SX.

4.2 SX TO SX COMMUNICATION

4.2.1 SX to SX Application Description

The ‘SX 2 SX’ sample application allows SX devices to initiate XID discovery and send unreliable broadcast data packets. The SX is controlled by commands received from the debug port as follows:

- ‘d’ will start the discovery process
- ‘c’ will broadcast the value of port c (raw byte).
- ‘s’ will broadcast a hello string.

The SX will respond to the command with a ‘^’ on the debug port if it has accepted the request or a’!’ if it has refused the request. A request will be refused if IR communication has been detected in the past 500 ms (as required by the IrDA IrLAP specification to prevent collisions).

Note that this connectionless protocol does not require the LMP layer and could be implemented with much simplified LAP and payload layers freeing up over half the code space.

4.2.2 Requirements

- Two SX boards (at least) each with a serial connection to a PC through the debug ports.

Note:The serial cable should be a straight-through cable with at least TX, RX, and GND connections.

4.2.3 Operation

- Open a terminal program on the appropriate physical COM port at 115200,n,8,1, no handshaking on each of the computers.
- Power up the SX boards. Power is indicated by the ‘PWR’ LED.
- Press ‘d’ on one computer to request that the discovery process be initiated. The SX should respond with a ‘^’ followed by the text names of all the IrDA devices in range. If a ‘!’ is returned then other IR communication has been detected so the request cannot be accepted.
- Observe the LEDs during the discovery process. The initiating SX’s TX indication LED should be flickering for about one second and any responses should be seen with the RX indication LED. The response will occur at different intervals through the transmission process each time as a different random reply slot will have been chosen by the responder.
- Press ‘c’ on one computer to request that the contents of port c be broadcast. The SX should respond with a ‘^’ and all other SX devices in range should display the received byte. If a ‘!’ is returned then other IR communication has been detected so the request cannot be accepted.

5.0 Serial Port / Debug

The demonstration PCB includes a RS232 serial port for debug and external communication. The UART-Virtual Peripheral supports full duplex communication at 115200bps (1 start bit, no parity, 1 stop bit) with no hand-shaking (RTS/CTS hand-shaking is supported by the hardware but has not been implemented in software).

A data byte can be transmitted using the following code:

```
mov     UartTxData, w
setb    UartTxStart
```

The register and bit are both global (no bank instruction needed). Debug data is the same as `UartTxData` and `DebugSend` is the same as `UartTxStart`. When the data byte has been sent the global `UartTxEmpty` flag will be set.

If the UART is idle then the transmission will start at the next timer interrupt (108 cycles @ 50MHz), otherwise it will start once the current byte is finished. There is no additional buffering and so a second byte cannot be sent until the first byte has started transmission (indicated by `UartTxStart` being cleared).

When a data byte is received the global `UartRxAvail` flag will be set. The received data byte is in the `UartRx-Data` register in the `IsrBank` (not global).

5.1 DEBUG MACROS

Table 5-1 shows three debug macros for showing debug data.

Table 5-1. Debug Macros

Name	Description	Example	Affects	Words	Cycles
Debug	Output the value in w	debug 1	none	2	2
Debugl	Output a literal value	debugl l, '!'	w	3	3
Debugf	Output the given register	debugf 1, reg	w, z	3	3

The first parameter is a constant and must be 1 for the macro to take affect.

Debug Information

The debug control registers in 'PROJECT.SRC' can be changed to show more debug information.

6.0 Hardware

The hardware consists of the following main aspects:

- Scenix SX communications controller
- IR transceiver interface
- RS232 serial interface
- Port C break-out interface
- Indication LEDs
- Power supply

The IR transceiver circuit is based on the HP HSDL1001 transceiver. The current limiting resistor for the transceiver LED was chosen to maximize the LED current and thus the effective range of the link while operating at 115200bps. The LED is capable of handling an average current of 100mA and a peak current of 1000mA. The pulse will always be 2.16us (108 clock cycles @ 50MHz) in length regardless of the baud-rate and the minimum bit time is 8.64us (115200bps = 4*108 clock cycles @ 50MHz). The worst case character '0' consists of 1 start pulse, 8 data pulses, and one stop no-pulse thus the worst case average current percentage of peak current is 22.5% (2.16us / 8.64us * 9/10bits). The voltage drop across the LED varies with the LED current and as such the resistor value is hard to calculate for a desired current greater than the typical application information given in the data sheet. It was chosen to use four 22Ω 0.125W resistors in parallel to give an effective resistance of 5.5Ω with a maximum power dissipation of 0.125W. In one of the prototypes the peak LED current was measured to be 454mA (2.52V / 5.556Ω) thus making the worst-case average current 102.15mA (454mA * 22.5%) and the worst-case average power dissipation of 257.42mW (2.52V * 102.15mA). The LED current is acceptable however the current will be susceptible to component tolerances and care must be taken. The resistor power dissipation is not a cause for concern due to the half-duplex nature of the communications reducing the longer-term average power dissipation.

It should also be noted that the IR transceiver LED timing is completely software controlled. If the SX device leaves the LED on continually then the IR transceiver may be permanently damaged as it will be operating at 4.5 times its absolute maximum average current rating.

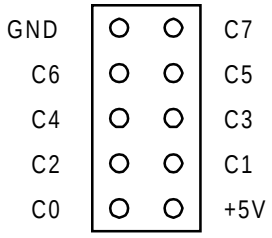
The RS232 interface provides a 115200bps full-duplex serial interface with RTS/CTS hardware hand-shaking. It is based on a MAX232 compatible charge-pump voltage converter to provide the voltage conversion and is controller by bits 0 to 3 of port B.

The port C break-out interface provides access to port C of the SX device (8-bit bi-directional) and well as the power supply rails.

The power supply consists of a bridge rectifier and a linear voltage regulator to allow the circuit to operate of a range of power sources. The regulator has a typical drop-out voltage of 2V thus for a DC source the voltage should be greater than 8.4V (5V rail + 2V regulator + 1.4V bridge).

The power supply de-coupling capacitors are of high importance as the HP transceiver is highly susceptible to power supply noise. With insufficient power supply de-coupling the transceiver will give falsely indicate that a pulse has been received when the RS232 charge pump spikes the power supply rail. The transceiver is protected by a 150uF low ESR tantalum capacitor (C2) and a 100nF X7R ceramic capacitor (C9). C2 is shown in the circuit diagram as being part of the power supply circuit however it is intended primarily to protect the IR transceiver and must be placed as close as practicable to the transceiver.

The port C breakout has the following pinout (looking down on the pins) where the +5V pin is pin 1:



7.0 Appendix A - Frame Check Sequence (FCS)

The FCS is a 16-bit CCITT CRC covering the payload data before transparency is applied (i.e. FCS must also be transparent) as described on pages 113-116 of IrLAP.

- The FCS is initialized to $FFFF.
- For transmission the complement of the FCS is sent.
- For reception the FCS of the data including the FCS bytes will result in $F0B8 for a valid frame.

7.0.1 Basic FCS Algorithm

```
X         =       FCS-Low xor DATA
FCS-Low   =       F(X)-Low xor FCS-High
FCS-High  =       F(X)-High
```

Table 7-1. Virtual Table Algorithm F(X)

Bit	Algorithm
15	X_7 xor X_3
14	X_6 xor X_2
13	X_5 xor X_1
12	X_4 xor X_0
11	X_3
10	X_7 xor X_3 xor X_2
9	X_6 xor X_2 xor X_1
8	X_5 xor X_1 xor X_0
7	X_4 xor X_0
6	X_3
5	X_2
4	X_1
3	X_7 xor X_3 xor X_0
2	X_6 xor X_2
1	X_5 xor X_1
0	X_4 xor X_0

```
X         =       X xor (X << 4)
F(X)-High =       X xor (X >> 5)
F(X)-Low  =       (F(X)-High >> 4) xor (X << 3)

<< N      =       Shift left N bits (does not wrap around)
>> N      =       Shift right N bits (does not wrap around)
```

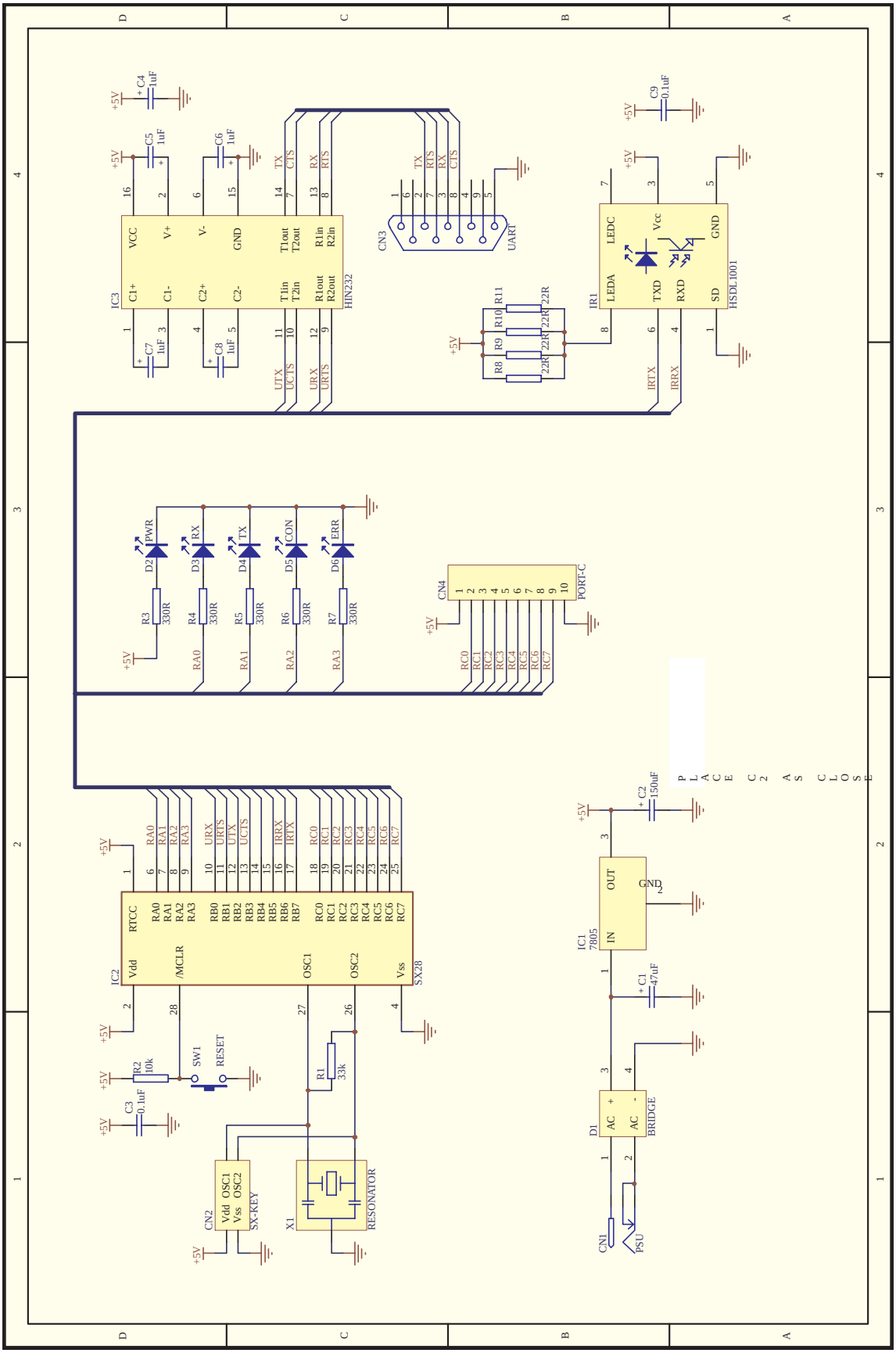
7.0.2 Final FCS Algorithm

```
X         =       FCS-Low xor DATA
X         =       X xor (X << 4)
A         =       X xor (X >> 5)
FCS-Low   =       (A >> 4) xor (X << 3) xor FCS-High
FCS-High  =       A
```

The FCS-Low variable is used to hold temporary variable X resulting in:

```
FCS-Low   =       FCS-Low xor DATA
FCS-Low   =       FCS-Low xor (FCS-Low << 4)
A         =       FCS-Low xor (FCS-Low >>       5)
FCS-Low   =       (A >> 4) xor (FCS-Low << 3) xor FCS-High
FCS-High  =       A
```

8.0 Appendix B – Circuit Diagram



9.0 Appendix C – Bill of Materials

Qty	Ref	Description
Resistors		
4	R8-R11	22R, 1%, 0.125W, 1206 Package
5	R3-R7	330R, 1%, 0.125W, 1206 Package
1	R2	10K, 1%, 0.125W, 1206 Package
1	R1	33K, 1%, 0.125W, 1206 Package
Capacitors		
2	C3,C9	100nF, 50V, X7R Ceramic, 1206 Package
5	C4-C8	2.2uF, 16V, Tantalum, 3216 Package
1	C1	47uF, 16V, Low ESR, Tantalum, 7343 Package
1	C2	150uF, 6.3V, Low ESR, Tantalum, 7343 Package
Semiconductors		
1	D1	1B01S 1A Bridge Rectifier (Surface Mount)
3	D3,D4,D6	HP HSMS-T400 Red LED (3528 Package)
1	D5	HP HSMY-T400 Yellow LED (3528 Package)
1	D2	HP HSMG-T400 Green LED (3528 Package)
1	IC1	MC7805CD2T 5V 1.5A Regulator (D^2PAK Package)
1	IC2	SX28AC/SO (SOIC Package)
1	IC3	HIN232CB Dual RS232 Transceiver (SOIC Package)
Other		
1	X1	50Mhz Resonator (Surface Mount)
1	SW1	4mm SPNO Push Button (Surface Mount)
1	CN3	9 pin Female D Connector, Right angle PCB mount
1	CN1	2.5mm PCB mount DC Socket
1	CN2	4 pin SIL Header
1	CN4	2x5 pin DIL Header

Lit #: SXL-AN16-02

Sales and Tech Support Contact Information

For the latest contact and support information on SX devices, please visit the Scenix Semiconductor website at www.scenix.com. The site contains technical literature, local sales contacts, tech support and many other features.

SCENIX

1330 Charleston Road
Mountain View, CA 94043
E-Mail: sales@scenix.com
Web Site: scenix.com
Tel.: (650) 210-1500
Fax: (650) 210-8715