zilog

Application Note

IrDA Framer Implementation on the Z8 Encore! XP® MCU

AN015202-1207

Abstract

This Application Note describes an Infrared Data Association (IrDA) Framer implementation for Zilog's Z8 Encore! XP® Flash microcontroller using the IrDA encoder and decoder (endec) and Universal Asynchronous Receiver and Transmitter (UART) that are integrated with the Z8 Encore! XP MCU.

The IrDA Framer implements data transmission speeds between 9600 bps and 115.2 kbps (slow infrared speeds). The Framer implementation includes a service abstraction layer to transmit and receive the IrDA frames through the IrDA physical media, along with routines to initialize the UART device and set the transmission rate. Slow infrared (SIR-IrLAP) implementations can take advantage of the Framer services provided in this Application Note.

Note: *The source code file associated with this Application Note,* `AN0152-SC01.zip` *is available for download on* www.zilog.com.

Z8 Encore! XP® Flash Microcontrollers

Zilog's Z8 Encore! XP products are based on the new eZ8 CPU to introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows for faster development time and program changes in the field. The high-performance register-to-register based architecture of the eZ8 core maintains backward compatibility with Zilog's popular Z8® MCU.

The new Z8 Encore! XP microcontrollers combine a 20 MHz core with Flash memory, linear-register SRAM, and an extensive array of on-chip peripherals. These peripherals make the Z8 Encore! XP MCU suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Zilog IrDA Transceivers

Designed for applications in which space is at a premium, Zilog IrDA transceivers are the best choice for a wide range of applications such as inventory control, vending, portable scanners, portable medical, diagnostic and measurement products, as well as mobile phones, digital cameras, notebooks, personal digital assistants (PDAs), and handheld computers.

Zilog IrDA transceivers interact with IrDA-ready appliances and offer the lowest-power options available in the market. The same transceiver can be used with a variety of power supplies. As a result, lower power consumption and the ability to multisource power supply components saves component costs and development time.

Discussion

IrDA is a robust, proximity-based, line-of-sight infrared protocol defined by the Infrared Data Association for general use. IrDA communication is a widely adopted, short-range, wireless technology allowing secured, reliable, and low cost point-to-point communication among devices such as PDAs, digital cameras, pagers, set-top boxes, machinery, and computer peripherals.

The IrDA protocol defines multiple standards for use in data communication, control applications, local area network (LAN) applications, etc. The standard supports a number of communications types, including *slow infrared*, or SIR, for speeds in the range of

9600 bps to 115.2 kbps. IrDA supports faster speeds with the MIR and FIR types for medium and fast infrared communications, which operate in the Mbps range.

IrDA Protocol Layers

An IrDA protocol stack is a layered set of protocols particularly aimed at point-to-point infrared communications, and packaged with the applications required for the infrared communications environment.

Figure 1 displays an example of different layers (starting from the Physical Layer of the OSI Model) that comprise the IrDA protocol stack. This application note focuses on the Physical Layer and the IrLAP layer that sandwich the Framer implementation. For more details on other standard IrDA stacks, see References on page 11.

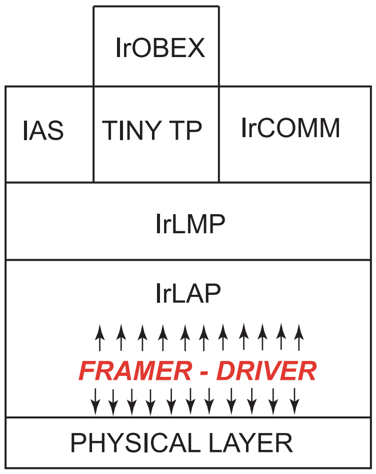

Figure 1. IrDA Stack Organization

Physical Layer

The Physical Layer includes a UART and an optical transceiver. In this Framer implementation, the UART-associated IrDA endec and the IR transceiver constitute the IrDA Physical Layer. These devices beam and receive IR signals. The typical distance for IrDA communications is from 5 cm to 60 cm away from the transceiver.

IrDA data links always operate in HALF-DUPLEX mode for the simple reason that while transmitting, a device's receiver is blinded by its transmitter.

Zilog® manufactures and sells a wide range of IR transceivers. For details, refer to www.zilog.com.

IrLAP Layer

The IrLAP layer implements the Link Access Protocol that establishes a basic, reliable connection. IrLAP services include the following:

- Device discovery
- Address conflict resolution
- Connectionless data transfer
- Connection establishment
- Connection-oriented data transfer

IrLAP Frame

All data and control transmissions on an IrLAP data link are organized in a specific format called a *frame*. This format carries control information and user data between a transmitting station and a receiving station, and it allows a receiving station to:

- Determine where the frame begins and ends
- Determine whether the frame is intended for a the receiving station
- Determine what actions to perform with the information received
- Detect the occurrence of transmission errors in received frames
- Acknowledge its receipt of frames to the transmitting station

Each IrLAP frame (see Figure 2 on page 3) is preceded and succeeded by fields which constitute the wrapping layer. The wrapping layer implements a Physical Layer scheme that serves to reliably transmit the payload data.

The wrapping layer fields serve to mark the beginning and end of the frame and to check for the reliable transmission of data. The format of the wrapper fields vary according to the Physical Layer scheme used, but every frame wrapper includes at least three components:

- A start Flag, BOF, that marks the beginning of the frame
- A frame check sequence field that allows the receiving station to check the transmission accuracy of the frame
- A stop Flag, EOF, that signals the end of the frame

A wrapper's design and function are independent of the payload frame's function.

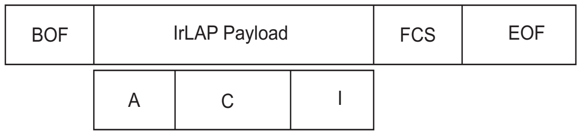

Figure 2. IrLAP Frame-Wrapper Added by Framer Around IrLAP Payload Data

These three parts of the IrLAP Frame are described below.

BOF—The Beginning of Frame (BOF) character identified by the hexadecimal character, `0xC0`.

IrLAP Payload—The actual data to be transmitted. The format for the IrLAP payload data is provided below:

- An address (A) field that identifies a secondary station connection address
- A control (C) field that specifies the function of the particular frame
- An optional information (I) field that contains the information data

Each of these fields contains 8 bits or multiple of 8 bits.

FCS—The frame check sequence (FCS) is a 16-bit sequence that is specified according to the CRC-CCITT algorithm on IrLAP payload data. It contains a 16-bit cyclic redundancy check that follows the CRC-CCITT standard. The FCS is valid only for IrLAP payload data comprising the A, C, and I fields. The FCS inevitably follows the IrLAP payload data sequence. The purpose of the FCS is to check for errors in the received frame produced during frame transmission.

CRC computation is performed on the IrDA payload data. For details on the algorithm to calculate CRC, refer to www.irda.org.

The CRC computation is performed prior to the transparency check performed by the transmitting and receiving stations. For information on transparency checking, see Transparency Character Check by Framer on page 4.

EOF—The End of Frame (EOF) character, identified by the `0xC1` hexadecimal character.

Framer

The Framer is a software function that acts as the data frame formatter (wrapper), and it can optionally function as the device driver for the system's transceiver controller. The Framer performs the following tasks:

- Initializes the infrared hardware device
- Sets the transmission speed
- Adds a wrapper around the IrLAP payload data
- Transmits the data to the Physical Layer

While receiving the data from the IR transceiver, the Framer removes the wrapper from the IrLAP payload data and sends the received data to the IrLAP layer for further processing.

The Framer generates a wrapper for the IrLAP frames, by adding a BOF character before the IrLAP data followed by the frame check sequence and the EOF.

Frame Transmission

All the bytes of an IrLAP Frame are transmitted as *low order bit first*. The two-byte FCS is transmitted as *least-significant byte first*. Every byte transmitted in the asynchronous/SIR mode consists of exactly one start bit, 8 data bits, and one stop bit; there is no parity bit.

Note: *The characters used to indicate the BOF (`0xC0`), EOF (`0xC1`), and Control Escape (`0x7D`) are designated as control characters. The Control Escape character (`0x7D`) is used for character transparency, which is discussed in the next section.*

Transparency Character Check by Framer

Because there are no restrictions on the content of an IrLAP payload, problems arise during frame reception when the payload data contains control characters as actual data. To workaround this problem, the Framer implementation defines a character transparency check that transforms information/data bytes, which would otherwise be interpreted as control characters, into noncontrol characters prior to transmission.

Prior to transmitting the IrLAP payload data, the Framer examines each byte present in the IrLAP payload data. The Framer performs the following tasks when it encounters a data byte that is a designated control character (`0xCO`, `0xC1`, or `0x7D`):

1. Inserts a Control Escape(`0x7D`)byte before the control character.
2. Complements bit 5 of the byte, which means that it performs an exclusive `OR` operation (XOR) on the byte with `0x20`.

Example

Assume that the IrLAP payload data comprises of the following bytes:

```
0x20, 0xAB, 0x89, 0xC0, 0x98
```

In this example, the first three characters following the BOF character are transmitted without any modification because they are not control characters. The fourth character, `0xC0`, which also represents the control character for BOF, is transmitted as two characters—`0x7D`, followed by `0xE0` (`0xC0 XOR 0x20=0xE0`). The next character, `0x98`, is transmitted as it is.

The Framer performs the transparency check for the FCS in a similar way. The resulting transmission sequence is represented as:

```
(BOF)0xC0, 0x20, 0xAB, 0x89, 0x7D,
0xE0, 0x98, FCS, (EOF)0xC1
```

The Framer performs the following tasks, to receive the IrLAP Frame:

1. First receives the BOF.
2. Discards `0x7D` (the first Control Escape character).
3. The character (`0xE0`)following the Control Escape character is XORed with `0x20` to obtain the actual character, `0xC0`.
4. The Framer then receives other data until it encounters an EOF.

The resulting reception sequence is represented as:

```
(BOF)0xC0, 0x20, 0xAB, 0x89, 0xC0,
0x98, FCS, (EOF)0xC1
```

Developing the Framer Application for the Z8 Encore! XP® MCU

The Z8 Encore! XP® Framer implementation specifications are listed below.

- The Framer implementation is based on the IrDA Lite protocol (see Appendix C—IrDA Lite on page 24 for more details on this protocol)
- The Framer implementation forms an abstraction layer between the IrLAP layer and the Physical Layer
- The Physical Layer for this implementation utilize the following Z8 Encore! XP MCU peripherals:
  - UART0
  - IrDA endec0
- The Framer performs the usual wrapper function on IrLAP frames. adding the BOF, FCS, and EOF characters.
- The Framer implementation includes APIs that initialize the hardware devices, set the transmission rate, and provide a time-out functionality using the Z8 Encore! XP timer.

The block diagram in Figure 3 displays the connections between the Z8 Encore! XP MCU peripheral, UART0, the IrDA endec0, and the IR Slim SIR transceiver.

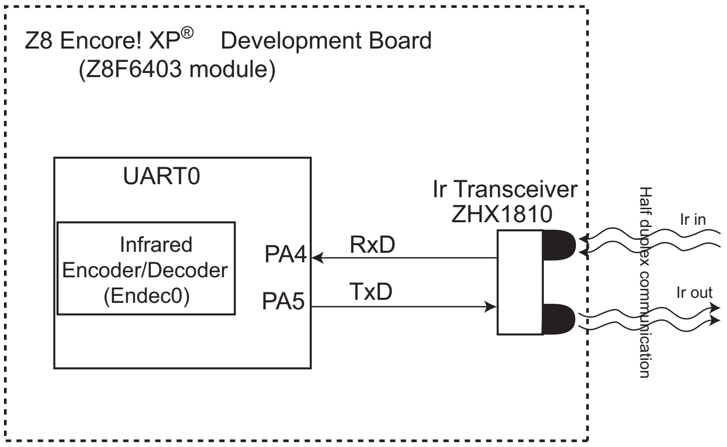
Z8 Encore! XP® Development Board
(Z8F6403 module)
UART0
Infrared Encoder/Decoder (Endec0)
PA4
PA5
RxD
TxD
Ir Transceiver ZHX1810
Half duplex communication
Ir in
Ir out

Figure 3. UART0 Connected to an IR Transceiver

In Figure 3, the Port A GPIO pins of the Z8F6403 MCU, PA4 and PA5, are connected to the IrDA transceiver, Zilog® part number ZHX1810. Setting the IREN (IR enable) bit in the UART0's Control 1 Register enables the IrDA endec0.

For details on the UART registers, refer to *Z8 Encore! XP® 64K Series Flash Microcontrollers Product Specification (PS0199)*. For details on the IR transceiver, refer to *Slim Series SIR Transceiver Product Specification (PS0093)*.

Framer Operation

An important design feature of the Framer implementation is buffer management because the RAM size is limited when using IrDA Lite. The IrDA Lite protocol specifies that the maximum data size is 64 bytes, implying that the optional information field (I) of IrLAP must be between 0 to 64 bytes, and the IrLAP must form the frame length with a maximum of 64 bytes. The Address field (A) and Control field (C) are one byte each. Therefore, the transmit buffer in this application uses 66 bytes of RAM space, which is shared between the IrLAP layer and the Framer.

The IrLAP layer initiates and controls the functions of the Framer. The IrLAP layer forms the IrLAP payload data and writes the payload data into the transmit buffer. The transmit buffer is a shared resource between the IrLAP layer and the Framer, therefore, an inter-locking mechanism is implemented restricting both IrLAP and Framer from using the transmit buffer at the same time.

The Framer operation can be viewed as functioning in three distinct modes, as described below.

IDLE Mode—The Framer operates in IDLE mode under two circumstances—when it is initialized by the IrLAP layer and operates neither in RECEIVE mode nor TRANSMIT mode; or when the Framer is issued a shutdown command by the IrLAP layer.

TRANSMIT Mode—The Framer operates in TRANSMIT mode when it receives an indication from the IrLAP layer to transmit data. In TRANSMIT mode, the Framer transitions through different states to transmit the IrLAP payload data.

RECEIVE Mode—The Framer operates in RECEIVE mode when it receives an indication from the IrLAP layer to receive data. The Framer transitions through different states to receive the data from the UART0 peripheral.

Note: *The Framer does not operate in* RECEIVE *mode by default.*

The following sections explain these modes in detail.

IDLE Mode

During initialization by the IrLAP layer, the Framer exists in IDLE mode. In this mode, the RECEIVE and TRANSMIT modes are disabled. When RECEIVE mode is enabled, the Framer enters the IN_BOF RECEIVE state, which indicates that it is ready to receive data. Similarly, when TRANSMIT mode is enabled, the Framer enters the BOF TRANSMIT state and transmits the BOF character.

TRANSMIT Mode

In TRANSMIT mode, the Framer transitions from one state to the next in a sequence starting from the BOF state, as described below.

1. BOF—The Framer transmits the BOF character.
2. DATA state—The Framer transmits the IrLAP payload data. If the Framer encounters a control character while transmitting the payload data, it transmits a Control Escape character (`0x7D`). The control character is XORed with `0x20` and then transmitted. The FCS is calculated for each byte of data transmitted.
3. FCS state —The resulting two bytes of the Frame Check Sequence (FCS) are transmitted.
4. EOF state—The Framer transmits the EOF character.

The Framer then resets back to the BOF state.

RECEIVE Mode

In RECEIVE mode, the Framer transitions from one state to another, starting from the IN_BOF state, as described below.

1. IN_BOF state—When the Framer receives the BOF character from the UART0 device, it is in the IN_BOF state. While in this state, it can receive other BOF characters. If the Framer receives an EOF character after the BOF character, it remains in the IN_BOF state. If the Framer receives any other character it transitions to the IN_DATA state.
2. IN_DATA state—While in this state, the Framer performs CRC check as characters are received. When the Control Escape character is received, the Framer performs the transparency check on the following character. When an EOF character is received, the Framer checks

the validity of the data by comparing the FCS calculated with the FCS received. Upon verifying the validity of the data, the Framer invokes a callback function to inform the IrLAP layer that the data reception was successful. The Framer then transitions to the IN_BOF state to wait for any incoming BOF character. If a BOF character is received while in the IN_DATA state, the Framer transitions to the RECEIVE_ERROR state.

Note: *The FCS data is also considered as normal data sequence.*

3. RECEIVE_ERROR state—If a BOF character is received in the IN_DATA state, the Framer transitions to the RECEIVE_ERROR state. In this state, all previously-received data is discarded and the Framer transitions to the IN_BOF state.

In the state machine diagram displayed in Figure 4, the columns indicate the events and the rows indicate the receive states.

EVENTS (CHARACTERS RECEIVED) →

STATES ↓

	BOF Character	Control Escape "CE" (0x7D)	Any other Character	EOF Character
IN_BOF	Remain in IN_BOF	Transit to IN_DATA	Transit to IN_DATA Read and Store data byte	Remain in IN_BOF
IN_DATA	Transit to RECEIVE_ ERROR	IN_DATA Transparency Character Check done. Read and store data byte	IN_DATA Read and Store data byte	Transit to IN_BOF Intimate IrLAP about data reception
RECEIVE_ERROR	Flush the Received data and transit to IN_BOF state			

Figure 4. IrDA Framer Receiver State Machine

Software Implementation

Based on the description of the Framer Operation on page 5, the software is implemented as a set of APIs and a callback function that the IrLAP layer can use to send and receive data. The functionality of these APIs and callbacks are described in this section. For other details, such as parameters and return values, see Appendix B—API Descriptions on page 17.

Initialization

The IrLAP layer invokes the Framer by calling the `Z_irda_lite_init_phy()` API. This API initializes the IrDA Physical Layer device—in this case, the UART0. This API performs the following tasks:

- Sets the GPIO pins for alternate function of UART0
- Sets the UART baud rate to 9600bps
- Initializes the status flags for the receive and transmit modes
- Initializes TIMER0
- Enables the IrDA endec
- Initializes the transmit buffer

- Disables the transmit and receive interrupts

Note: *IrDA communication always occurs in half-duplex mode. The receive and transmit interrupts are not concurrently enabled. The transmit interrupt is enabled only when the Framer is initialized to* TRANSMIT *mode and the receive interrupt is enabled only when the Framer is initialized to* RECEIVE *mode.*

Data Transmission

The IrLAP layer calls the `Z_irda_lite_get_tx_buffer()`, before sending the IrLAP payload data to the transmit buffer. Because the transmit buffer is shared between IrLAP layer and the Framer, this API is used by the IrLAP layer to determine the status of the transmit buffer. Upon receiving an indication that the transmit buffer is free, IrLAP fills the payload data into the transmit buffer. When the IrLAP receives an indication that the transmit buffer is not free (implying that the Framer is transmitting data from the transmit buffer it waits until the buffer is free.

Then, IrLAP calls the `Z_irda_lite_transmit()` API, to start data transmission. This API initializes the Framer transmit mode states, enables TRANSMIT mode and disables RECEIVE mode. The UART0 transmit interrupts are also enabled. The data transmission is managed by the transmit interrupt handler.

Transmit Interrupt Handler—The Framer uses the interrupt method to transmit the IrLAP payload data. The transmit interrupt handler takes care of loading the data to the UART0 transmit data register, sets the status flags according to the data transmitted, checks for transparency characters, and calculates the FCS for each byte transmitted. The `generate_crc()` function calculates the CRC value.

Data Reception—The IrLAP layer calls the `Z_irda_lite_receive()` API to receive data. This API disables the transmit interrupt, enables the Framer receive mode, and enables the receiver interrupt.

Receive Interrupt handler—The Framer uses the interrupt method to receive the IrLAP payload data. The receive interrupt handler reads the data from the UART0 receive data register, sets the status flags according to the data bytes received, checks for transparency characters, and calculates and compares the FCS. The `generate_crc()` function calculates the CRC value.

Callback Functions—The IrLAP layer calls the `Z_init_callback()` API, which initializes a Framer `receive_callback()` function.

The Framer invokes the `receive_callback()` function within its `do_rx_upcall()` function to provide information (address of the receive buffer and the length of the data received) to the IrLAP layer upon successfully receiving data. The IrLAP layer uses this information to process the data received.

Receive Time-Out—Timer0 is used to provide the time-out feature for the Framer. At initialization, TIMER0 is loaded with an initial value to generate interrupts every 10ms. When a character is received, TIMER0 is enabled and begins counting. If the next character is received within 10 ms, TIMER0 is reset to zero.

If the next character is not received within 10ms, then TIMER0 generates an interrupt and the program jumps to the TIMER0 ISR. This ISR sets the `Receive_Timeout` Flag. The Framer ignores the entire packet of data in the receive buffer and sets the receiver status Flag to the IN_BOF state.

Shutdown—When the IrLAP layer receives a shutdown command from the higher IrDA stack layers, it calls the `Z_irda_lite_shutdown()` API to disable the TRANSMIT and RECEIVE modes of the Framer. The Framer enters IDLE mode, wherein no transaction is possible. In IDLE mode, the status Flag for RECEIVE mode is set to IDLE, while the status

Flag for TRANSMIT mode is set to BOF. To exit IDLE (shutdown) mode, the `Z_irda_lite_init_phy()` API is invoked by the IrLAP layer.

Testing

This section describes the setup, equipment, and procedures to test the IrDA Framer implementation on the Z8 Encore! XP® MCU.

Test Setup

Two Z8 Encore! XP development kits and two PCs are used for testing the Framer. Figure 5 displays the test setup. The power supply connection is not included in the block diagram.

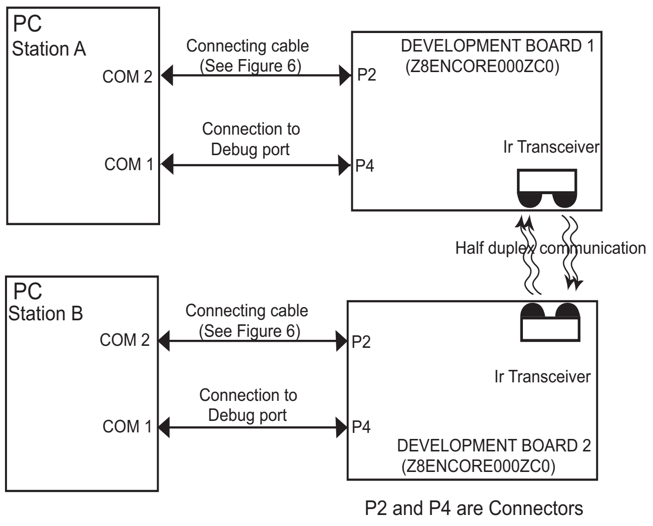

Figure 5. Test Setup for the Framer Implementation

The Z8 Encore! XP Development Board features a console port (P1, not displayed in Figure 5), which is normally used to connect to the PC's COM port to communicate with the HyperTerminal application. However, this console port cannot be used if an IrDA endec is being used. Therefore, for this test setup, the MODEM port (P2) is used to connect to HyperTerminal by using a connecting cable as displayed in Figure 6 on page 10.

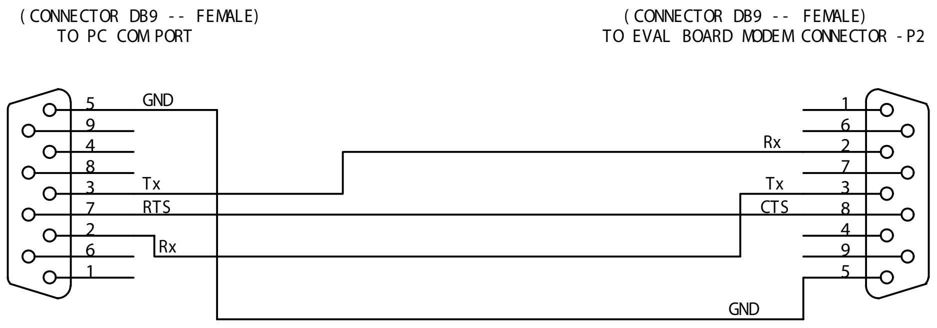
(CONNECTOR DB9 -- FEMALE)
TO PC COM PORT
(CONNECTOR DB9 -- FEMALE)
TO EVAL BOARD MODEM CONNECTOR - P2
5
9
4
8
3
7
2
6
1
GND
Tx
RTS
Rx
1
6
2
7
3
8
4
9
5
Rx
Tx
CTS
GND

Figure 6. Connection Cable between the COM2 Port and the P2 Connector

The HyperTerminal settings are as follows:

- 57600 bps baud rate
- 8 data bits
- No parity
- 1 stop bit
- No flow control

Note: *To enable the IR transceivers for IrDA communication, short jumper J10 on the Z8 Encore! XP® Development Board.*

Equipment Used

The following equipments are used to test the IrDA Framer implementation:

- Two Z8 Encore! XP® Development Kits (Z8ENCORE000ZC0) featuring the Z8F6403 MCU
- ZDSII IDE for Z8 Encore!® v4.2.1
- HyperTerminal

Test Procedure

Follow the steps below to test the IrDA Framer implementation on Z8 Encore! XP MCU.

1. Using the ZDSII IDE, download the object files onto both of the Z8 Encore! XP Development Boards.
2. On the two PCs—Station A and Station B, launch the HyperTerminal application with the settings provided in the Test Setup on page 9.
3. Execute the program using ZDSII on Station A and Station B.
4. Follow the HyperTerminal menu to set Station A as the receiver and Station B as the transmitter.
5. Enter the length of the data to be transmitted from the transmitting station (Station B). Length of data is specified as a hexadecimal value; the maximum value is `0x40` (for 64 bytes).
6. Station B transmits five packets of data, sequentially, with each packet length equal to the number of bytes specified. Station A receives these packets and displays them in the HyperTerminal window.
7. Set Station B as receiver and Station A as transmitter and repeat the test.

Note: *The* `test.c` *file (available in the* `AN0152-SC01.zip` *file) contains hard-coded data that can be used to simulate IrLAP payload data. This data is stored in the form of a two-dimensional array. The first element of the array in each row indicates the packet number (0, 1, 2, ...) and the remainder are data bytes. When the Framer receives the data, it is* *compared with the transmitted data and is displayed in the HyperTerminal window, only if both of them match.*

8. The data bytes in the array `const unsigned char transmit_array [ROW][COLUMN]`, defined in the `test.c` file, can be modified (if required) to test for the transmission and reception of different data. After this change, the project must be compiled and rebuilt to generate new object files. The same object file is loaded into both of the target boards under test.

Test Results

The testing of the IrDA Framer between the two Z8 Encore! XP® Development Boards was found to be satisfactory. The data transmitted from the transmitting station was displayed correctly in the receiving station's HyperTerminal window.

Testing on Other Target Platforms

The Framer implementation was additionally tested on the following devices:

- A Z8 Encore! XP Board and an eZ80® Development Platform equipped with the eZ80L92 Module, which features a full IrDA stack
- A Z8 Encore! XP Board and a laptop computer with the IrDA port enabled
- A Z8 Encore! XP Board and an IR dongle (IrDA COM port serial adaptors)

With the eZ80L92 Module and a laptop computer, the Framer is caused to respond to one of the discovery frames and receive the remainder of the discovery frames transmitted by the sender. Because the Z8 Encore! XP Board features only a Framer, these tests illustrate that the Framer is able to communicate with IrDA-compatible devices, assuming that appropriate modifications are made.

The IR dongle receives the data sent by the Framer. This data is captured in a file, using HyperTerminal, and compared with the transmitted data.

Summary

This Application Note describes a Z8 Encore! XP® MCU-based Framer implementation for the IrDA Lite stack. This Framer implementation can be used to develop other protocol layers for IrDA Lite with little or no modification. The Z8 Encore! XP MCU features on-chip integration of 64 KB of Flash memory, 4 KB of RAM, UARTs, and an IrDA endec. These features make the Z8 Encore! XP MCU an ideal choice to implement the IrDA Lite protocol stack because it can help to avoid the use of external components—thereby reducing cost and design time.

References

The documents associated with IrDA specifications, protocols, Z8 Encore! XP Flash Microcontrollers, the eZ8 CPU, and the ZDS II Development Tool for Z8 Encore! available on www.zilog.com are provided below:

- IrLAP—IrDA Serial Infrared Link Access Protocol, version 1.1 (IrLAP); www.irda.org
- Z8 Encore!® Flash Microcontroller Development Kit User Manual (UM0146)
- Slim Series SIR Transceiver Product Specification (PS0093)
- eZ8 CPU Core User Manual (UM0128)
- Zilog Developer Studio II (ZDS II)–Z8 Encore!® User Manual (UM0130)

Appendix A—Flowcharts

This Appendix displays the flowcharts for the IrDA Framer implementation (see Figure 7 through Figure 12).

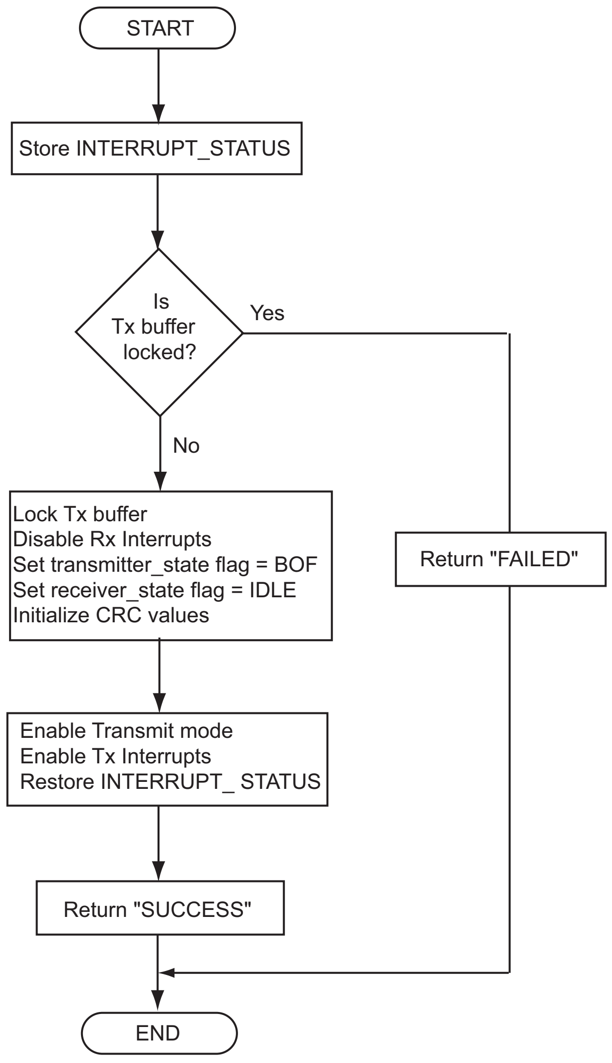

Figure 7. The START-TRANSMIT Routine

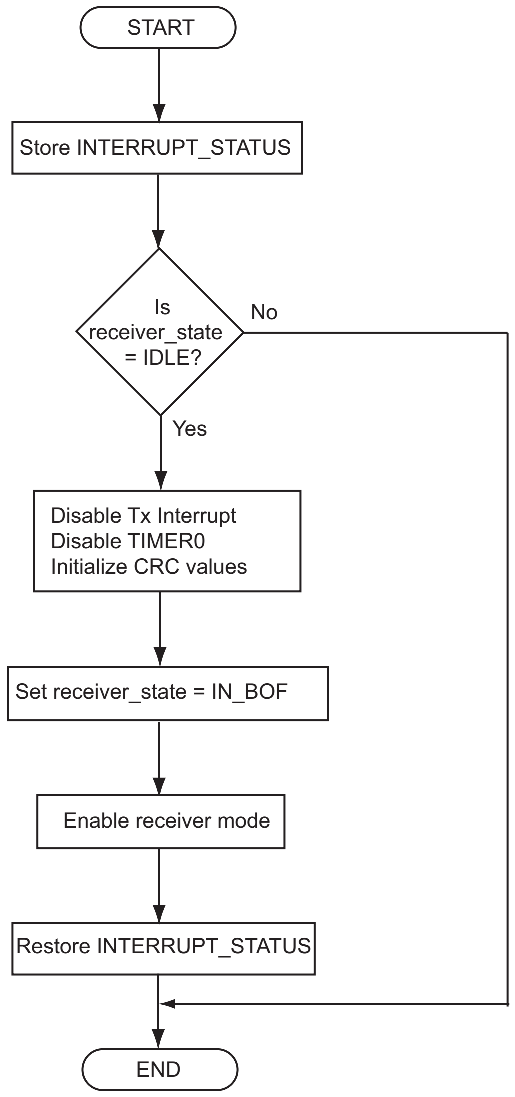

Figure 8. The START-RECEIVER Routine

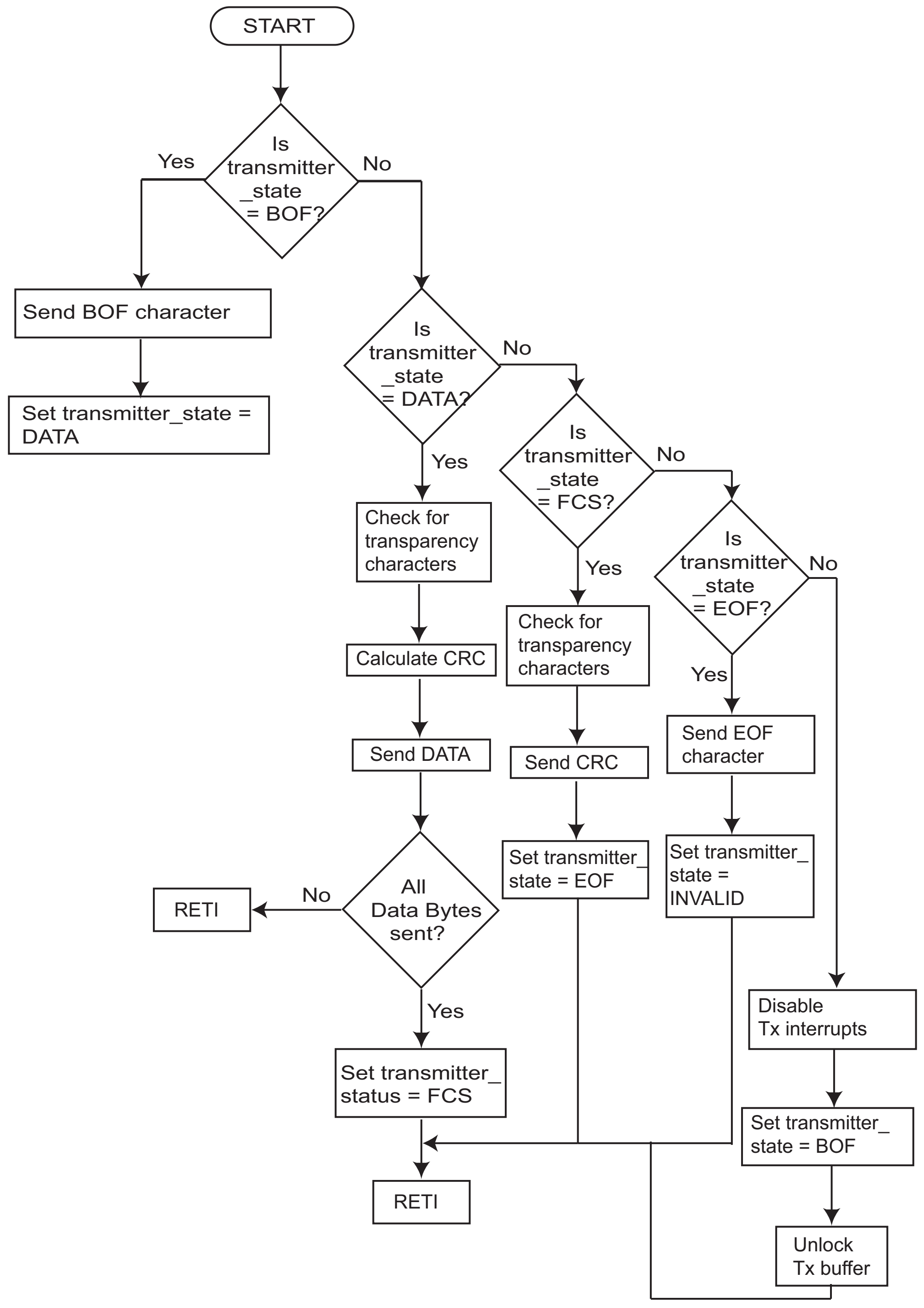
START
Is transmitter _state = BOF?
Yes
No
Send BOF character
Set transmitter_state = DATA
Is transmitter _state = DATA?
No
Yes
Check for transparency characters
Calculate CRC
Send DATA
All Data Bytes sent?
No
RETI
Yes
Set transmitter_ status = FCS
RETI
Is transmitter _state = FCS?
No
Yes
Check for transparency characters
Send CRC
Set transmitter_ state = EOF
Is transmitter _state = EOF?
No
Yes
Send EOF character
Set transmitter_ state = INVALID
Disable Tx interrupts
Set transmitter_ state = BOF
Unlock Tx buffer

Figure 9. The Transmit Interrupt Handler

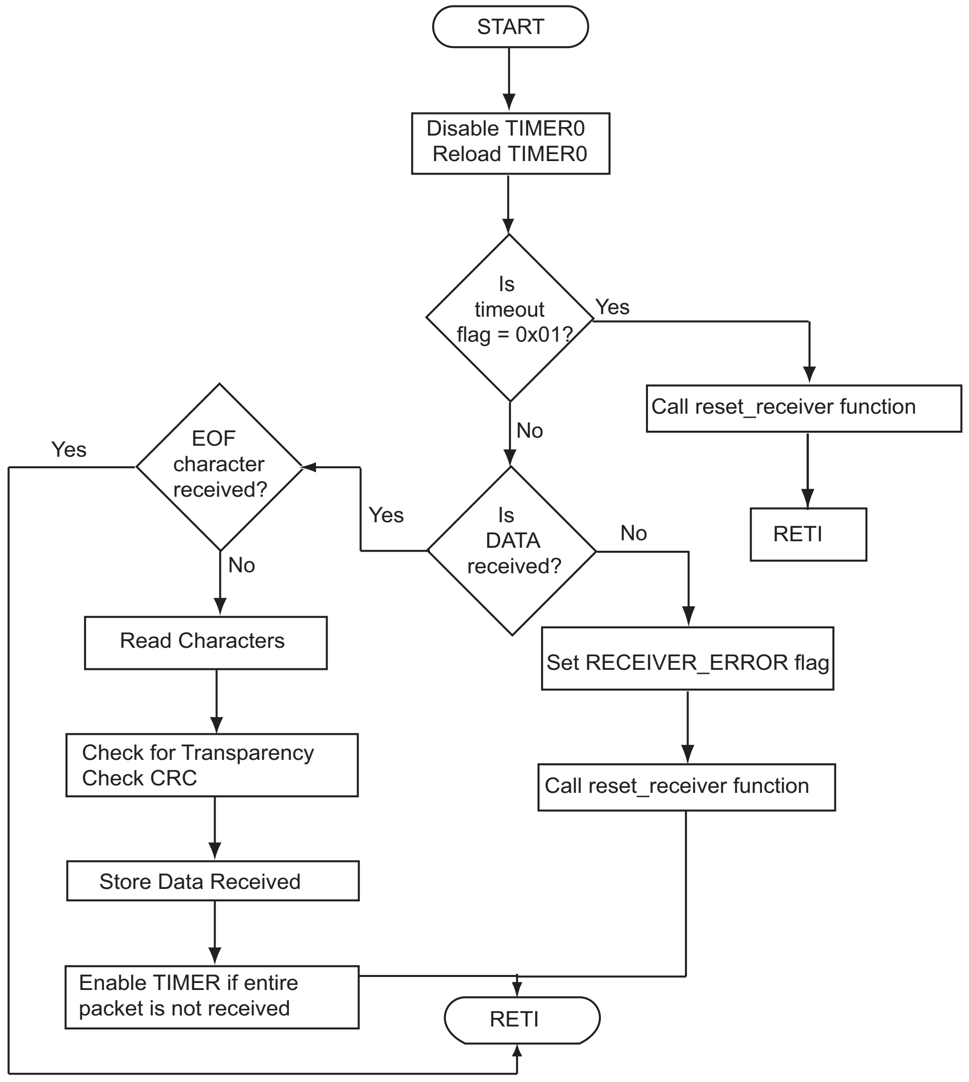

Figure 10. The Receive Interrupt Handler

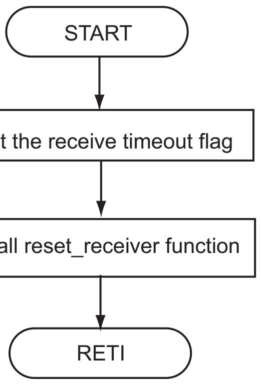

Figure 11. The Receive Time-Out Interrupt Routine

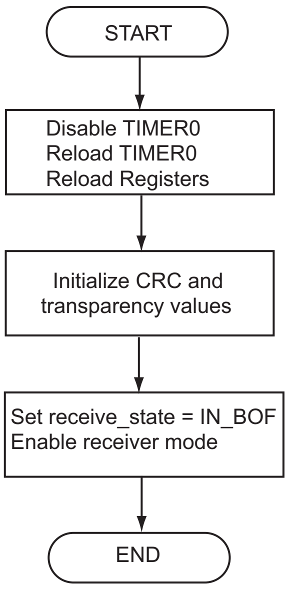

Figure 12. The Reset Receiver Function

Appendix B—API Descriptions

This Appendix describes the following APIs provided for the IrDA Framer implementation on the Z8 Encore! XP® MCU.

- void Z_irda_lite_init_phy (void)
- void Z_init_callback (PHY_RX_CALLBACK rx_callback)
- char Z_irda_lite_get_tx_buffer (void)
- void Z_irda_lite_shutdown (void)
- BYTE Z_irda_lite_transmit (BYTE length)
- void z_irda_lite_receive (void);

void Z_irda_lite_init_phy (void)

Prototype

```
Z_irda_lite_init_phy (void)
```

Description

This API executes the following tasks:

- Selects the GPIO pins for the alternate function of UART0
- Sets the UART baud rate to 9600 bps
- Disables the Transmit and Receive interrupts
- Initializes the receiver to the IDLE state
- Initializes the transmitter to the BOF state
- Initializes TIMER0

Argument (s)

None

Return Value (s)

None

void Z_init_callback (PHY_RX_CALLBACK rx_callback)

Prototype

```
Z_init_callback (&test_callback_function)
```

The address of `test_callback_function` is passed as an argument.

Description

This API initializes the `Rx` (Receive) callback function.

Argument (s)

`PHY_RX_CALLBACK`	Type definition of a function pointer.

Return Value (s)

None

char Z_irda_lite_get_tx_buffer (void)

Prototype

```
Z_irda_lite_get_tx_buffer (void)
```

Description

This API checks the status of the transmit buffer. When the Framer is transmitting data from the transmit buffer, this API returns BUSY, signifying that the buffer is not available to IrLAP. Otherwise, this API returns FREE, signifying that the transmit buffer is free for the IrLAP to load the buffer with new IrLAP payload data.

Argument (s)

None

Return Value (s)

`char`	The status of the transmit buffer is BUSY or FREE.

void Z_irda_lite_shutdown (void)

Prototype

```
Z_irda_lite_get_tx_buffer(void)
```

Description

This API disables all of the UART interrupts (`Rx` and `Tx`). The Framer is set to the IDLE state. Specifically, this API sets the `receive_state` Flag to IDLE and the `transmit_state` Flag to the BOF state.

Argument (s)

```
void
```

Return Value (s)

```
void
```

BYTE Z_irda_lite_transmit (BYTE length)

Prototype

```
Z_irda_lite_transmit (length)
```

Description

IrLAP calls this API when data is required to be transmitted. This API locks the `Tx` buffers, disables the receive interrupt and enables the transmit interrupt. It also checks the Framer state which is set to transmit mode. This API returns '`SUCCESS`' if setup for transmission takes place properly.

Argument (s)

`BYTE (type def for char)` The length of the buffer to be transmitted.

Return Value (s)

`char`	SUCCESS when the transmission is successful.

void z_irda_lite_receive (void);

Prototype

```
z_irda_lite_receive (void)
```

Description

IrLAP calls this API to receive data. The API checks the Framer state and sets it to RECEIVE mode. This API also checks the receiver status Flag. If the receiver status Flag is set to IDLE, the receiver is enabled. This API disables the `Tx` interrupts and enables the `Rx` interrupts.

Argument (s)

```
void
```

Return Value (s)

```
void
```

Appendix C—IrDA Lite

This Appendix describes the IrDA Lite protocol.

IrDA Lite is a minimal implementation of the complete IrDA stack. However, it interoperates with the full-featured IrDA stacks by sacrificing speed and throughput, among other attributes. A majority of the devices incorporating IrDA Lite are embedded devices. Most of these devices provide less memory than laptop or desktop PCs.

The Framer implementation design for the Z8 Encore! XP® MCU is based on the following IrDA Lite specifications and is compliant to the IrDA Physical Specification version 1.3.

Baud Rate—The communication speed between two IrDA-compatible devices is always fixed at 9600 bps. This speed is not negotiable during communication.

Data Size—The maximum length of the IrLAP payload data that can be transmitted or received in one frame is 64 bytes.

Window Size—At any given time, IrLAP can form only one IrLAP payload data packet to send to the Framer for transmission.

Additional BOFs—The number of BOF characters allowed ranges from 1 character to a maximum of 11 characters.

Warning: DO NOT USE IN LIFE SUPPORT