Application Note

Direct Memory Access on the Z8 Encore! XP® UART

AN014206-0608

Z8 Encore! XP® Flash Microcontrollers

Abstract

This application note displays data transfer operations that communicate via the Universal Asynchronous Receiver and Transmitter (UART) block of the Zilog's Z8 Encore! XP® MCU. This data transfer is with or without Direct Memory Access (DMA). The DMA is used for data transfer between the UART and a user-specified RAM space without CPU intervention. Data flow is regulated by setting the hardware flow control using the Clear to Send (CTS) and Request to Send (RTS) control signals.

Note: *The source code files associated with this application note,* `AN0142-SC01.zip` *and* `AN0142-SC02.zip` *are available for download at* www.zilog.com.

Z8 Encore! XP Flash MCU Overview

Zilog's Z8 Encore! XP products are based on the new eZ8™ CPU and introduce Flash memory to Zilog's extensive line of 8-bit microcontrollers. Flash memory in-circuit programming capability allows 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 suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Discussion

This section contains a brief overview of Universal Asynchronous Receiver and Transmitter (UART) communication in general, including the explanation of a few commonly used UART-related terms. An overview of the Z8 Encore! XP UART and Direct Memory Access (DMA) blocks is also included.

Overview of UART Communication

UART communication involves converting a parallel data stream into a serial data stream before transmission over a serial link. UART communication also converts the received serial data to parallel data for further processing by the CPU. UART functionality involves the following tasks:

- Adding start and stop bits to the transmitted data
- Generating a hardware interrupt (IRQ) when a character is received and when the UART is ready to transmit another character
- Performing flow control

Some of the terms commonly used in UART communication are explained briefly in the following sections.

Baud Rate

Baud rate is the UART speed calculated as the number of bits transferred in one second. The selected speed must not be greater than the capabilities of the UART that controls the serial port.

Parity Bit

The parity bit is used to check data errors in UART communication, and its use is optional. There are two types of parity checking—even and odd. Both the transmitting and receiving devices must be configured to use the same type of parity checking for an error-free communication.

The serial port sets the parity bit (the last bit after the data bits) to a value that ensures that the transmission contains either an even or odd number of logic High bits.

For example, assume that a data unit is `011`. For even parity checking, the parity bit is set to 0 to keep the number of logic High bits even. For odd parity checking, the parity bit at the end of the data unit is set to 1, which results in an odd number of logic High bits.

Flow Control

When a transmitting device sends data faster than a receiving device can process, a flow control mechanism is used to cope the situation. The receiving device asks the transmitting device to pause while it catches up.

For example, if a printer's buffer is full of data, and it still must print, then it tells the CPU to stop sending data until it is ready to accept more.

There are many flow control mechanisms. Flow control can be implemented in hardware or software, or a combination of both.

Hardware flow control—This is achieved by using two control signal lines between two communicating devices. The two control signal lines transmit the Request to Send (RTS) and the Clear to Send (CTS) signals that control the flow of data between the devices.

Software flow control—This is achieved by using special control characters embedded in the data stream, such as XON/XOFF, that deliver the message to start the data flow (XON) or stop the data flow (XOFF)

Overrun

A data overrun occurs when the CPU fails to release an empty receive buffer to receive an incoming message. The incoming message is lost, indicating that the CPU is extremely overloaded. A data overrun can be prevented by monitoring the status of the receive buffer and the flow control.

Z8 Encore! XP® UART Overview

The Z8 Encore! XP MCU contains two fully-independent UARTs. Each UART is a full-duplex communication channel capable of handling asynchronous data transfers. The key features of the UART include:

- 8-bit asynchronous data transfer
- Selectable even- and odd-parity generation and checking
- Option of one or two stop bits
- Separate transmit and receive interrupts
- Framing, parity, overrun, and break detection
- Separate transmit and receive enables
- 16-bit Baud Rate Generator (BRG)
- Selectable 9-bit multiprocessor mode

The Z8 Encore! XP UART consists of three primary functional blocks:

1. Transmitter
2. Receiver
3. Baud Rate Generator

The UART's transmitter and receiver function independently but employ the same baud rate generator and data format. Figure 1 on page 3 displays the UART architecture.

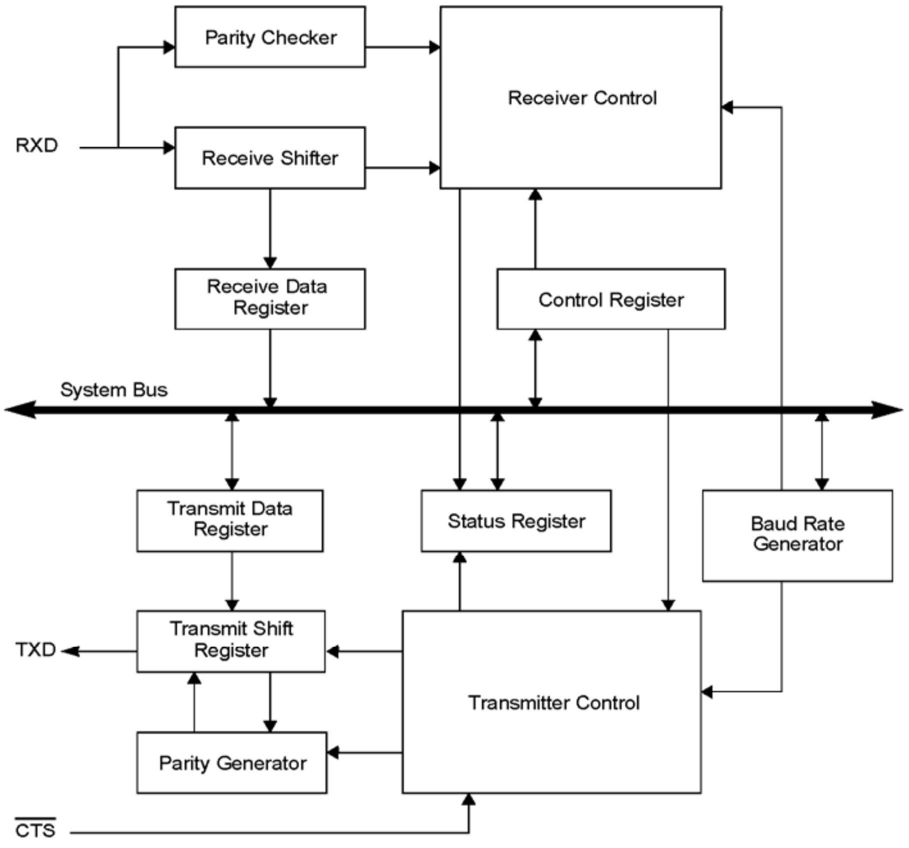

Figure 1. UART Block Diagram

Data Format

The UART always transmits and receives data in an 8 bit data format, with the least significant bit (lsb) expected first. Optionally, an even or odd parity bit can be added to the data stream. Each character begins with an active Low Start bit and ends with either 1 or 2 active High Stop bits. See Figure 2 and Figure 3 on page 4.

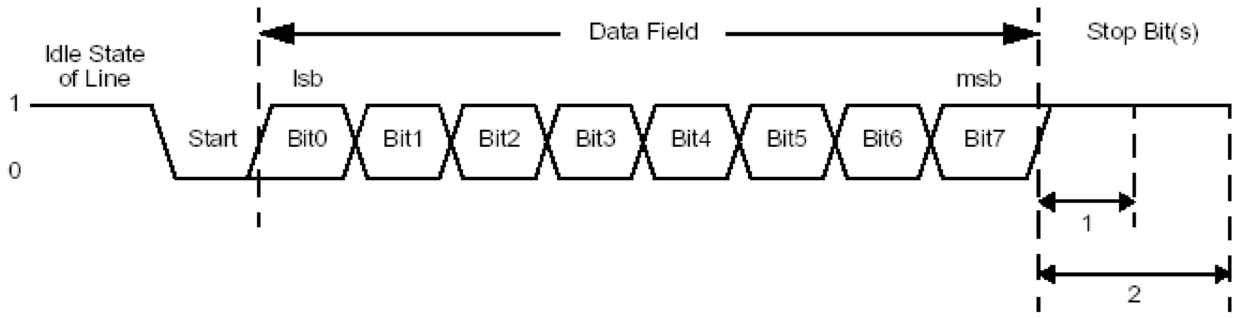

Figure 2. UART Asynchronous Data Format without Parity

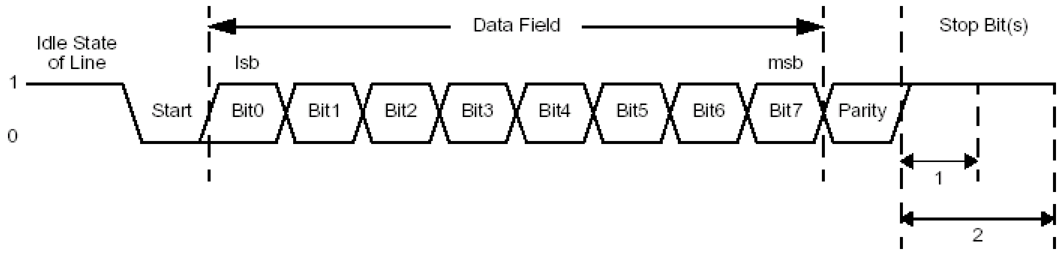

Figure 3. UART Asynchronous Data Format with Parity

UART Baud Rate Generator

The UART Baud Rate Generator creates a lower frequency clock for data transmission. The input to the Baud Rate Generator is the system clock. The UART*x* Baud Rate High and Low Byte registers combine to create a 16-bit baud rate divisor value (BRG[15:0]) that sets the UART data transmission rate (baud rate). The UART data rate is calculated using the following equation:

$$\text{UART Data Rate (bits/s)} = \frac{\text{System Clock Frequency (Hz)}}{16 \times \text{UART Baud Rate Divisor Value}}$$

For any UART data rate, the integer baud rate divisor value is calculated using the following equation:

$$\text{UART Baud Rate Divisor Value (BRG)} = \frac{\text{Round(System Clock Frequency (Hz))}}{16 \times \text{UART Data Rate (bits/s)}}$$

The baud rate error, relative to the appropriate data rate, is calculated using the following equation:

$$\text{UART Baud Rate Error (\%)} = \frac{100\,\text{(Actual Data Rate – Appropriate Data Rate)}}{\text{Appropriate Data Rate}}$$

For reliable communication, the UART baud rate error must never exceed 5 percent. Table 1 on page 5 lists UART baud rates using a 18.432 MHz system clock.

Table 1. UART Baud Rates (with 18.432 MHz System Clock)

Appropriate Data Rate (KHz)	BRG Divisor (Decimal)	Actual Rate (KHz)	Error (%)
1250.0	1	1152.0	–7.84%
625.0	2	576.0	–7.84%
250.0	5	230.4	–7.84%
115.2	10	115.2	0.0
57.6	20	57.6	0.0
38.4	30	38.4	0.0
19.2	60	19.2	0.0
9.60	120	9.60	0.0
4.80	240	4.80	0.0
2.40	480	2.40	0.0
1.20	960	1.20	0.0
0.60	1920	0.60	0.0
0.30	3840	0.30	0.0

Z8 Encore! XP® DMA Overview

The Z8 Encore! XP DMA Controller provides three independent DMA channels. Two of the channels (DMA0 and DMA1) transfer data between the on-chip peripherals and memory. The third DMA channel is dedicated for the A/D converter.

The DMA transfers data between the on-chip peripheral Control Registers and the RAM. The DMA and UART can coordinate to achieve automatic data transfer from the UART Data Receive Register to the DMA Buffer Register. This coordination reduces the CPU processing overhead required to support UART data reception.

Data Transfer Using the Z8 Encore! XP UART

The Z8 Encore! XP UART transfers data as a single byte or as two bytes. The eZ8™ CPU must constantly keep track of these transfers either by polling on the control registers or by being informed of it by the UART interrupts. Routing data through the DMA reduces CPU intervention, thereby efficiently utilizing CPU time.

A device driver has been developed to achieve UART data transfer with DMA. The following sections explain the software implementation of UART data transfer with and without the DMA.

Software Implementation

The software implementation using only a UART involves initialization of the UART, receiving the data, and sending the data to the specified device. These tasks are accomplished by three APIs:

UART Initialization API—To initialize the UART port for data communication in an interrupt mode.

Get Character API—To get a received byte of data from the UART when an interrupt is generated.

Put Character API—To transmit a byte of data via UART by writing to the Transmit Data Registers.

The software implementation using DMA for UART communication involves initializing both the UART and the DMA. The DMA channel is first initialized to receive data from the specified UART, and the DMA buffer's START and END addresses are written to the appropriate control registers.

Receiving the data using DMA is implemented as follows:

1. Two 4-byte buffers—`buffer0` and `buffer1`—are used.
2. The DMA is initialized with `buffer0` as the DMA buffer. The UART receives data from the external device and fills this DMA buffer with the received data.
3. An interrupt is generated when the DMA buffer is full. The Interrupt Service Routine (ISR) for the DMA interrupt performs the following operations:
   a. Checks if `buffer0` = READ_COMPLETE.
   b. If yes, points to `buffer1` by initializing the DMA start and end address dynamically.
   c. Marks `buffer0` as READ_PENDING.
   d. If no, checks if `buffer1` = READ_COMPLETE.
   e. If yes, points to `buffer0` by initializing the DMA start and end address dynamically.
   f. Marks `buffer1` as READ_PENDING.
   g. If no, exits from the ISR.
4. When this interrupt is generated, the DMA is pointed to receive data into `buffer1` by writing to the appropriate control register.
5. While the UART receives data into `buffer1`, the processor retrieves data from `buffer0` and sets a flag indicating that `buffer0` is empty and ready to receive more data.
6. If `buffer0` and `buffer1` are both full but the processor has not retrieved data from either of them, the Flow Control API is called to set the RTS signal. This signal informs the transmitting UART device to stop the transmission.

The advantage of switching between two DMA buffers is that it allows the processor flexibility to read the received data at its convenience.

The APIs used to accomplish the UART-DMA data transfer are listed below:

DMA initialization API—To initialize the DMA channel for automatic data transfer from UART to RAM.

Get Data API—To initiate data transfer from UART to DMA buffers.

Put Data API—To initiate data transfer from DMA buffers to UART.

Read Buffer API—To read the data from the DMA data buffer to a user-defined array and to set a flag to indicate that the data was read.

Flow Control API—To set the RTS signal using a GPIO pin. The Flow Control API is used in situations when the Z8 Encore! XP MCU receives data at a rate faster than it can handle.

All of the APIs mentioned in this section are explained in Appendix B—API Description on page 11.

Refer to the *Z8 Encore! XP® F64XX Series Product Specification (PS0199)* for a detailed description of the UART and DMA control registers.

Testing

The basic setup for testing the UART communication with or without DMA is displayed in Figure 4 on page 7. This setup displays the connection between HyperTerminal and the Z8 Encore! XP® MCU.

UART and DMA are Z8 Encore! XP on-chip peripherals.

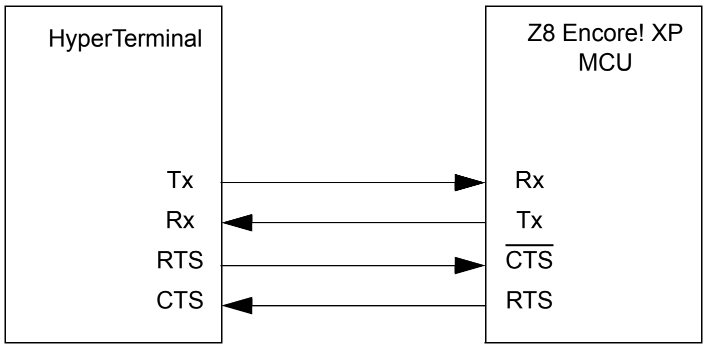

Figure 4. Test Setup to Demonstrate UART Communication

System Configuration

The HyperTerminal settings are as follows:

- Baud rate: 9600 bits/sec
- Parity: None
- Data bit: 8
- Stop bit: 1
- Flow control: Hardware (optional)

Note: *Ensure that the RTS and CTS pins are connected or choose None for the flow control option.*

Equipment Used

The equipment used for testing UART communication with or without DMA are as follows:

- Z8 Encore! XP Development Kit (Z8ENCORE000ZCOG) featuring the Z8F640x MCU
- Z8 Encore! XP Evaluation Kit (Z8F64200100KITG) featuring the Z8F642x MCU
- ZDS II IDE for Z8 Encore!®
- HyperTerminal (PC-based)

Test Procedure

The test procedure listed below enumerates the steps for data transfer with UART with and without DMA (when the UART receive buffer is full, DMA is disabled automatically, and the communication becomes a UART communication without DMA).

1. Configure **HyperTerminal** as mentioned in the System Configuration section.
2. Download the code to on-chip Flash memory.
3. When the HyperTerminal prompt appears, enter any combination of eight keyboard characters to fill the DMA buffers (there are two 4-byte DMA buffers). Buffer status is displayed in the **HyperTerminal** window.
4. To read the buffer data in HyperTerminal, press *b*. Buffer data is displayed in the **HyperTerminal** window.
5. To read the array data in HyperTerminal, press *a*. The array data is displayed in the **HyperTerminal** window.
6. After the UART receive buffer is full, the DMA is disabled automatically.
7. Enter any character continuously on the keyboard. The character is echoed back to the

HyperTerminal, indicating that UART communication without DMA is occurring.

Note: *Flow control RTS pin (Port F, bit 0) goes active Low immediately after filling the buffer and goes High after reading from the buffer.*

Test Results

The performance of the Z8 Encore! XP® XP UART DMA was observed with the aid of the HyperTerminal and found to be as expected.

Summary

The Z8 Encore! XP architecture offers two UART ports with full-duplex communication. The API is implemented such that data transfer can occur with or without the DMA.

In DMA data transfer mode, the program searches for an empty buffer to fill the data. If both buffers are full, the communication is stopped by setting the hardware flow control. This is implemented using the CTS and RTS signals. The API-based software implementation can be used directly or modified to suit the requirements of the user application with minimum effort.

References

The documents associated with Z8 Encore! XP Flash Microcontrollers available on www.zilog.com are provided below:

- Z8 Encore! XP® F64XX Series Product Specification (PS0199)
- Z8 Encore! Flash Microcontroller Development Kit User Manual (UM0146)
- eZ8™ CPU User Manual (UM0128)
- Zilog Developer Studio II—Z8 Encore! User Manual (UM0130)

Appendix A—Flowcharts

Figure 5 displays the UART0 initialization to receive or transmit data.

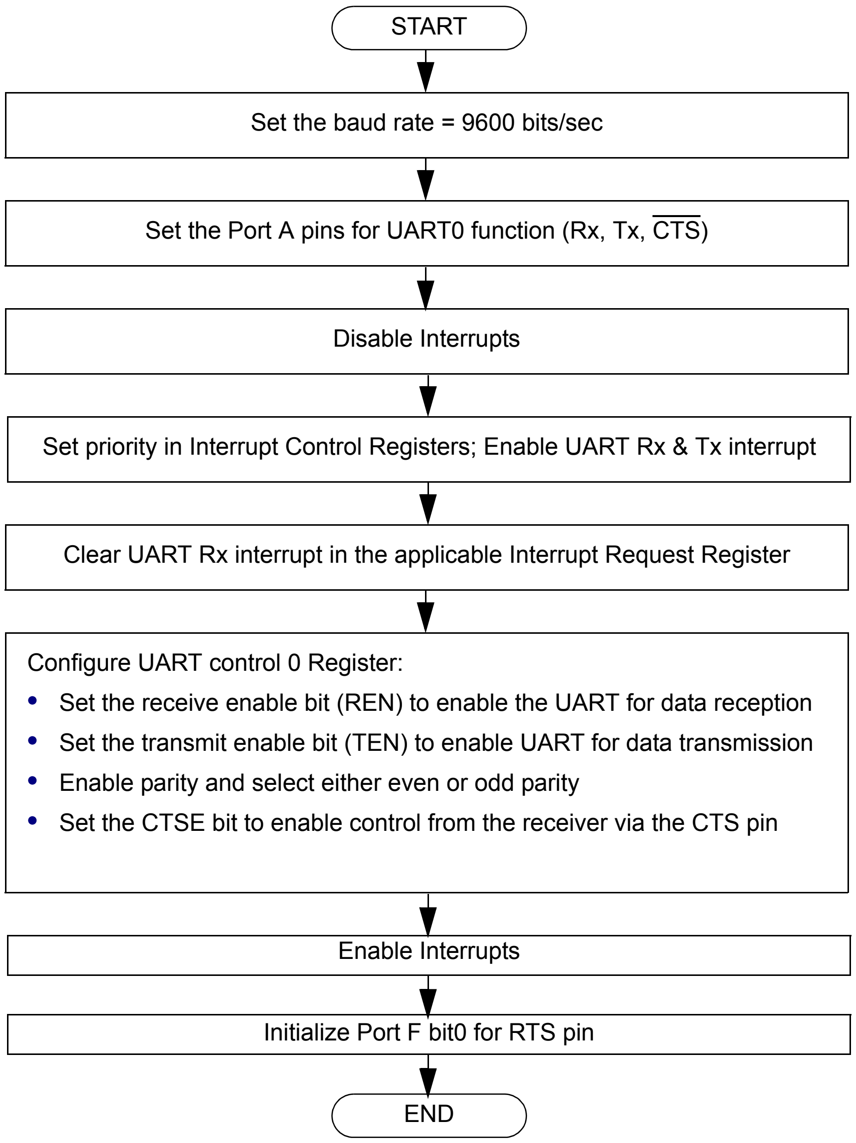
START
Set the baud rate = 9600 bits/sec
Set the Port A pins for UART0 function (Rx, Tx, $\overline{\text{CTS}}$)
Disable Interrupts
Set priority in Interrupt Control Registers; Enable UART Rx & Tx interrupt
Clear UART Rx interrupt in the applicable Interrupt Request Register
Configure UART control 0 Register:
• Set the receive enable bit (REN) to enable the UART for data reception
• Set the transmit enable bit (TEN) to enable UART for data transmission
• Enable parity and select either even or odd parity
• Set the CTSE bit to enable control from the receiver via the CTS pin
Enable Interrupts
Initialize Port F bit0 for RTS pin
END

Figure 5. UART0 Initialization to Receive or Transmit Data

Figure 6 displays the DMA initialization for UART0 data transfer.

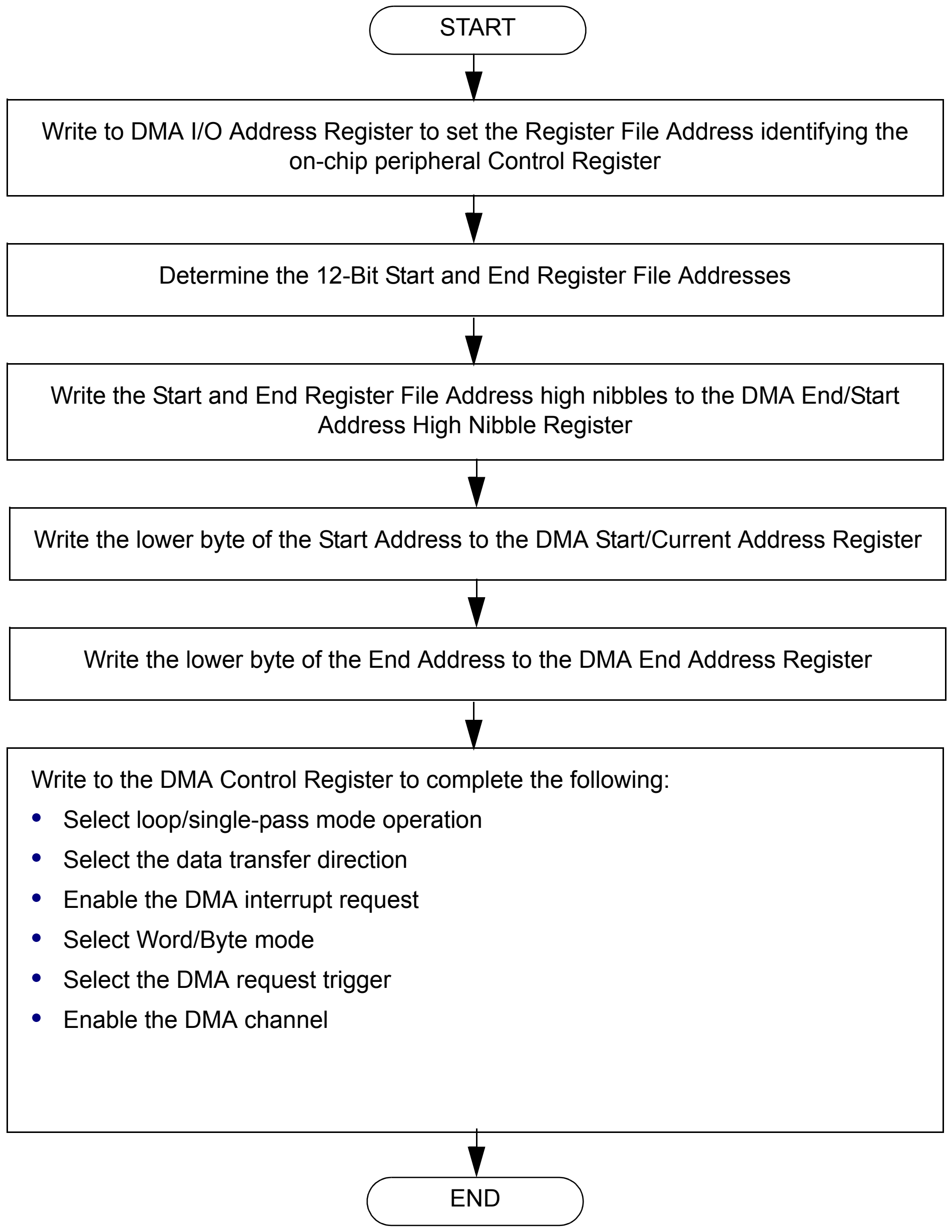
START
Write to DMA I/O Address Register to set the Register File Address identifying the on-chip peripheral Control Register
Determine the 12-Bit Start and End Register File Addresses
Write the Start and End Register File Address high nibbles to the DMA End/Start Address High Nibble Register
Write the lower byte of the Start Address to the DMA Start/Current Address Register
Write the lower byte of the End Address to the DMA End Address Register
Write to the DMA Control Register to complete the following:
• Select loop/single-pass mode operation
• Select the data transfer direction
• Enable the DMA interrupt request
• Select Word/Byte mode
• Select the DMA request trigger
• Enable the DMA channel
END

Figure 6. DMA Initialization for UART0 Data Transfer

Appendix B—API Description

The APIs to perform data transfer using UART, with or without the DMA, are described in this appendix.

APIs for UART Without DMA

The APIs listed below are used for routine UART data transfer:

- void init_uart0(void)
- unsigned char getch(void)
- void putch(char Data)

UART-DMA APIs

To use DMA along with UART for data transfer, the UART-DMA APIs listed below are required:

- void init_UART0_RxD_DMA0(void)
- unsigned char Get_Data(void)
- unsigned char Put_Data(void)
- unsigned char Read_Buffer(void)
- void Flow_Cont(void)

void init_uart0(void)

The `init_uart0` API initializes UART0 for data communication in interrupt mode. It performs the following tasks:

- Sets the baud rate = 9600 bits/sec
- Sets the Port A pin for UART0 function (Rx, Tx, $\overline{\text{CTS}}$)
- Enables the UART0 Receiver and Transmitter interrupts and sets the priority to High
- Clears the UART0 Receiver interrupt in the Interrupt Request 0 Register (IRQ0)
- Writes to the UART Control 0 Register to:
  - Set the receive enable bit (REN) to enable UART0 for data reception
  - Set the transmit enable bit (TEN) to enable UART0 for data transmission
  - Enable parity and select even parity
  - Set the CTSE bit to enable control from the receiver via the CTS pin
- Initializes Port F bit 0 for RTS control

unsigned char getch(void)

When the UART receives data, a UART interrupt is generated. The user calls the `getch` API to get a received character from the UART. This API reads the character from the UART0 Receive Data Register.

void putch(char Data)

The `putch` API is called to transmit a byte via the UART. The data to be sent to the UART device is written into the UART0 Transmit Data Register.

void init_UART0_RxD_DMA0(void)

The `init_UART0_RxD_DMA0` API initializes the DMA0 channel for automatic data transfer from UART0 to RAM. It performs the following tasks:

- Sets the DMA0 I/O address register to communicate with UART0
- Sets the 12-bit Start and End addresses. The 12-bit Start address is given by {DMAx_H[3:0], DMA_START[7:0]}. The 12-bit End Address is given by {DMAx_H[7:4], DMA_END[7:0]}

  For example:

  To set the 12-bit buffer with Start address=0xD00 and End address=0xD03, the values to be placed in the appropriate registers are:

```
DMA0_H = 0xDD
DMA0_START = 0x00
DMA0_END = 0x03
```

- The DMA initialization API writes to the DMA0 Control register to:
  - Select the loop mode operation
  - Select the data transfer direction from the on-chip peripheral control register to the Register File RAM
  - Enable the DMA0 interrupt request
  - Select the Word mode
  - Select the DMA0 request trigger
  - Enable the DMA0 channel

unsigned char Get_Data(void)

The `Get_Data` API initiates data transfer from the UART to the DMA buffers. It fills the specified data buffers.

unsigned char Put_Data(void)

The `Put_Data` API initiates the data transfer from the DMA buffers to UART. It performs the following tasks:

- Calls the `putch` API
- Marks the read buffer as an empty buffer (READ_COMPLETE)

unsigned char Read_Buffer(void)

The `Read_Buffer` API is used to read data from the DMA data buffer to a user-defined array and to set the READ_COMPLETE flag. This flag is used to swap the active DMA buffer (READ_COMPLETE) with the other buffer (READ_PENDING) that is full of received data. The application reads the data for further processing from the user-defined array.

The `Read_Buffer` API performs the following tasks:

- Reads data from DMA buffer
- Marks the read buffer as an empty buffer (READ_COMPLETE)

void Flow_Cont(void)

The `Flow_Cont` API is used to set the RTS signal, using a GPIO pin, to manage instances when the Z8 Encore! XP MCU receives data at a rate faster than it can perform processing. Flow control is set in the hardware. The Flow Control API controls the RTS control signal generated by the microcontroller. It performs the following tasks:

- Checks buffer status for both `buffer0` and `buffer1`
- If the buffers are full and the flag is READ_PENDING, it stops further communication

Note: These APIs can be easily modified for implementing the UART1 and DMA1 data communication by changing the appropriate register names from 0 to 1.

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

As used herein

Document Disclaimer