Application Note

Using ZSL with the Z8 Encore!® MCU UART

AN028101-0608

Z8 Encore!®

Abstract

Zilog Developer Studio (ZDS II) for Zilog's Z8 Encore!® v4.9.0 introduces a software component called the Zilog Standard Library (ZSL). ZSL is a set of library files that provides an interface between the user application and the on-chip peripheral set of all Z8 Encore! MCU. The ZSL version included in this ZDS II distribution supports General-Purpose Input/Output (GPIO) and Universal Asynchronous Receiver/Transmitter (UART) on-chip peripherals.

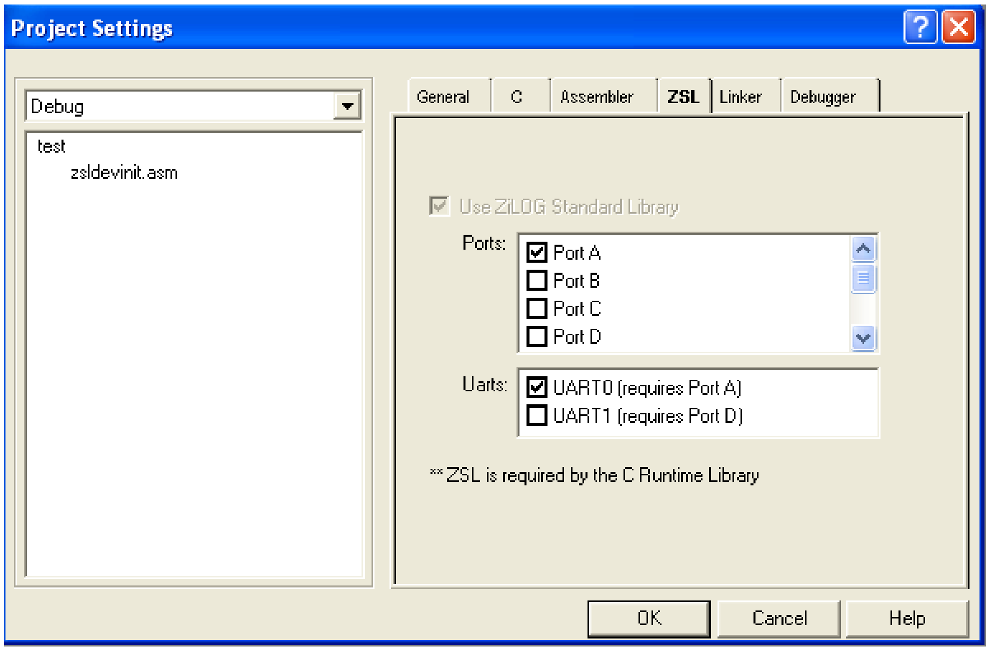
This Application Note describes how to use ZSL for the Z8 Encore! MCU's UART peripheral.

Note: *ZDS II is available for free download at* www.zilog.com.

Using ZSL for UART

To integrate the new ZSL software component, Zilog® has modified the C-language Run-Time Library (RTL) `sio.c` and `sio.h` files. These files originally contains UART-related implementations that now resides in ZSL.

To migrate to ZDS II v4.9.0, these two files exist in the RTL as a wrapper for corresponding ZSL functions. The functions called from `sio.c` file call ZSL functions, in turn, to access the UART. You can select the UART device in ZDS II by navigating to **Project** → **Settings** → **ZSL** → **UART0/UART1**, as displayed in the **Project Settings** dialog box in Figure 1.



[✓] Use ZiLOG Standard Library

Ports: [✓] Port A, [ ] Port B, [ ] Port C, [ ] Port D

Uarts: [✓] UART0 (requires Port A), [ ] UART1 (requires Port D)

** ZSL is required by the C Runtime Library

Figure 1. The ZSL UART Selection

Selecting one or both the UARTs in the ZSL settings initializes them at their default values. If both UARTs are selected in the **UARTs** panel of the **Project Settings** dialog box, they can be used simultaneously with the following considerations:

- Both UART0 and UART1 can access the UART APIs (`open_UARTx()` and `setbaud_UARTx()`).
- Only the default UART can access the generic UART APIs `getch()`, `putch()`, `kbhit()`, and standard I/O functions such as `printf()`. This default is set in the `uart-control.h` header file, which is located in the below filepath in the installed directory:

  `ZDSII_Z8Encore!_<version>\include\Zilog`

To use the ZSL UART APIs, the `ez8.h` file must be included in the application program. Alternatively, the application can include the `uart.h` file located in the below filepath in the installed directory:

`ZDSII_Z8Encore!_<version>\include\Zilog`

The selected UART is initialized with the following default values:

- Polling mode
- 38400 bps baud rate
- 8 data bits
- 1 stop bit
- No parity
- Hardware flow control disabled

ZDS II copies the ZSL device initialization file, `zsldevinit.asm`, into the user project when ZSL is selected from within ZDS II. The initialization file contains the `_open_periphdevice()` function which calls the initialization routines for all the devices required for the application.

The `_open_periphdevice()` routine is invoked from the startup routine before the `main()` function is called. ZDS II defines specific macros for each device; this definition is dependent on the UART device selected. The selected device(s) is/are initialized with default parameters and can be used starting from the `main()` function. This feature is shown in the code below:

```
#include<stdio.h>
#include<ez8.h>
main()
{
  // no need to initialize UART. The
  default values are considered
  printf("\n ZSL Settings For UART Of
  The Z8 Encore! MCU\n");
  //routed through the default UART
  set in uartcontrol.h
}
```

You can start using the UART with the standard APIs available to perform read and write to the UART with the default parameters listed above. To use different set of parameters, ZSL provides the appropriate APIs.

Only the following parameters are set using the ZSL APIs at run-time:

- Baud rate
- Parity
- Stop bit(s)

In addition to these three parameters, the other parameters, such as *hardware flow control enabling*, *error checking,* and *setting interrupt priority*, can be modified in the `uartcontrol.h` file.

Note: *The* `uartcontrol.h` *header file contains compile time configurations. Any modification to the parameters will be effective only on rebuilding the library.*

The baud rate, parity, and stop bits can be changed during run time using the appropriate APIs.

For details on all the APIs and procedure to rebuild the library, refer to the *Zilog Standard Library API Reference Manual (RM0038)*.

Following sections describe how to use ZSL to perform the below tasks:

- Initialize the UART for custom settings.
- Change the UART settings during run time.
- Read from the UART in the POLLING and INTERRUPT mode.
- Write to the UART in the POLLING and INTERRUPT mode.

Initializing the UART

UART0 is the default device selected. UART1 can be made the default device by changing the value in the `uartcontrol.h` file as shown below:

```
#define DEFAULT_UART      UART1
//Default device control macro
```

The library must be rebuilt for the settings to take effect. The UART can be initialized either to its default settings using the `open_UARTx()` calls, or to custom settings using a structure of type UART (defined in `uart.h`) and `control_UARTx()` calls.

The following code displays how to initialize the UART for custom settings using a structure. The `init_uart()` is a user-defined function. `serial` is a structure of type UART. The UART structure is defined in the `uart.h` header file. The `status` variable contains a return value from the API that indicates success or failure.

A number of error message examples are shown below. For a complete list of error messages, refer to `uart.h` header file.

```
#define UART_ERR_NONE
//The error code for success.
#define UART_ERR_KBHIT
//The error code for keyboard hit.
#define UART_ERR_BUSY
//Definition for 'UART busy'.
#define UART_ERR_INVBAUDRATE
//The error code returned when baud
//rate specified is invalid.
void init_uart (void)
{
  unsigned char status;
  UART serial;
  //serial is structure of type UART.
  serial.baudRate = 9600UL;
  //or BAUD_9600
  serial.stopBits = 1;
  //or STOPBITS_1
  serial.parity = 0;
  //or PAR_NOPARITY
  status = control_UART0(&serial);
  //status contains the return value
  if (status! = UART_ERR_NONE)
  //If error occurs close the device
  //and return
  {
    close_UART0();
    return;
  }
}
```

The above code initializes UART0 with a baud rate of 9600 bps, one stop bit, and no parity. If an error occurs during initialization, as indicated by `status`, the `close_UART0()` API renders the UART0 non-functional.

Alternatively, the UART can be initialized to its default settings by using only APIs. The following code displays this feature. You can define the parameters as listed below, or can use the standard definitions from the `uart.h` header file.

```
#define BAUD 9600UL
//valid values = 9600, 19200, 38400,
57600, 115200
#define FALSE 0
#define TRUE 1
void init_uart(void)
{
  unsigned char flag = TRUE,status;
  open_UART0();
  //opens UART0. Configures the
  //appropriate port
  //registers for alternate func
  //tions and is set
  //for default settings 38400 baud, 8
  //databits,
  //1 stop bit, no parity,and for Poll
  //mode.
  printf("\nUART0 initialized for
  default settings\n");
  //The default baud, data bits, stop
  //bits and parity can be changed
  //at runtime by using appropriate
  //APIs as shown below.
  do
  {
    status = setbaud_UART0(BAUD);
    // user can instead pass BAUD_9600
    // as
    // defined in uart.h
    if(status! = UART_ERR_NONE)
    {
      flag = FALSE;
    }
    status = setparity_UART0(PAR_ODPA
    RITY); // odd parity
    if(status! = UART_ERR_NONE)
    {
      flag = FALSE;
    }
    status = setstopbits_UART0(STOPBI
    TS_2);// set 2 stop bit
    if(status! = UART_ERR_NONE)
    {
      flag = FALSE;
    }
    if (flag == TRUE)
    //come out of the loop on success
    break;
  } while(flag == FALSE);
  // repeat until successful
  // initialization
printf("\n UART0 initialized for cus-
tom settings");
}
```

The `setbaud_UART0()`, `setparity_UART0()`, and `setstopbits_UART0()` APIs each set UART0 to a baud rate of 9600 bps, odd parity, and two stop bits. The code explained above checks the return value of each API and completes its routine only when all the APIs return successfully.

Changing the UART Settings

The settings for the selected UART can be changed in the code by different methods. After the UART is initialized, either for custom settings or default settings, it can be modified by following methods:

- Using the ZSL UART structure and the `control_UARTx()` API.
- Using APIs such as `setbaud_UARTx()`, `setparity_UARTx()`, etc.
- By explicitly writing into the UART registers.

The following code displays the above feature. The UART0 is first initialized with a baud rate of 9600 bps using a structure `UART` of type UART. At runtime, the baud rate is interactively changed to 38400 bps. The change in baud rate will be evident from the proper display of characters in the HyperTerminal window.

```
#define BAUD 38400UL
int init_uart(void)
{
  int ch;
  unsigned char status, flag;
  UART uart;
  uart.baudRate = 9600UL;
  //BAUD_9600;
  uart.stopBits = 1;
  //STOPBITS_1
  uart.parity = 0;
  //PAR_NOPARITY
  status = control_UART0(&uart);
  //Initialized with 9600 baud rate
  if(status! = UART_ERR_NONE)
  {
    return -1;
  }
  printf("\n Change the terminal set-
  ting to %u and press Z\n",BAUD);
  flag = setbaud_UART0(BAUD);
  // change the baud rate to 38400
```

```
    while(ch! = 'Z')
    {
      ch = getch(); //Wait for Z to be
      detected by UART
    }
    printf("\n UART0 baud rate changed
    to %u\n",BAUD);
    printf("\n Enter characters. Press
    $ to exit. \n");
    while((ch = getch())! = '$')
    {
      putch(ch); //display the entered
      characters until terminated by $
    }
  }
```

Using the Generic UART APIs

ZSL provides the following generic APIs:

- `getch()`
- `putch()`
- `kbhit()`

The `getch()` API reads a data byte from the default UART device. If there is no data in the UART device, the API blocks until the data is available. If the API is compiled with UART0/1ERRORCHECKING enabled, any error in communication is reported as a return value. If there is any error in the received data byte, an error code is set in the `g_recverr0` global variable. The application can determine the error by updating the `g_recverr0` global variable with a known value before calling the API, and then reading the `g_recverr0` global variable a second time to determine whether that value changed.

The `putch()` API writes a data byte into the default UART transmit buffer. The API returns a value to indicate success (0) or failure (1 or non-zero character).

The `kbhit()` API checks for any keystrokes on the default UART device. If a keystroke is detected, the `kbhit()` function returns 1; otherwise, it returns 0. The API does not reads the data; instead, it only returns a status. The application can next call `getch()` to obtain the keystroke.

The following code displays the use of these three generic APIs. The `get_put_char()` is a user defined function. The API `open_UART0()` initializes `UART0()` with default settings. A keystroke is detected by waiting in an infinite loop. The `ch` variable contains the value of the entered key, which is subsequently displayed in the HyperTerminal window. The `flag` variable is set to 0 on successful execution, otherwise set to 1 on failure.

```
void get_put_char(void)
{
  int ch,flag;
  open_UART0();
  // Opens the UART with default
  values
  while (!kbhit());
  // wait for a keystroke
  ch = getch();
  // ch holds value of the entered key
  printf("\n The entered character is
  \n");
  flag = putch(ch);
  // display back the entered key.
}
```

Writing to and Reading from the UART

The following section discusses the APIs that read from and write to the UART, in POLLING and INTERRUPT modes.

Writing to the UART in POLLING Mode

ZSL provides the `write_UARTx()` API for writing data into the UART. This API accepts a pointer to the buffer containing data to be transmitted and the number of bytes to be transmitted. The default mode for writing is the POLLING mode. The API returns a value only after transmitting all bytes.

The following code displays the use of `write_UARTx()` API.

In this code, `uart_write()` is a user-defined function, `mssg` is the string to be displayed, and `init_uart()` initializes UART0, as described in the Initializing the UART on page 3.

```
void uart_write(void)
{
  unsigned char status;
  char mssg[] = "Welcome to the world
  of Encore! Microcontrollers from
  Zilog"; //Data to be written
  init_uart(); //Initialize the UART
  with appropriate parameters
  status = write_UART0(mssg,strlen(ms
  sg)); // writes data onto UART
  //which on successful writing is
  displayed on terminal
  if(status! = UART_ERR_NONE) //sta-
  tus has the return value
  {
    close_UART0();
    return;
  }
}
```

Reading from the UART in POLLING Mode

ZSL provides the `read_UARTx()` API for reading data from the UART. This API accepts a pointer to a buffer that stores received data bytes and a pointer to an integer that indicates the number of bytes to be read. When the API returns, this variable contains the actual number of bytes read. The default mode of reading is the POLLING mode. The `getch()` API is used to read characters and can terminate when it reads a specific character when the number of bytes to be read is unknown.

The following code displays the use of the `read_UARTx()` API. In this code, the `uart_read()` is a user-defined function, and `readdata[]` is a buffer that holds the read data. On successful return from the `read_UART0()` API, this buffer holds data that is to be displayed using the `write_UART0()` API.

```
void uart_read(void)
{
  unsigned int length,read_length;
  char readdata[15],flag;
  //define buffer to hold read data
  length = 15;
  //define the number of bytes of data
  to be read
  init_uart(); //Initialize the UART
  with appropriate parameters
  printf("\nEnter a string of %u char-
  acters\n",length);
  flag = read_UART0(readdata,&length)
  ; //read data entered through
  //keyboard
  if(flag! = UART_ERR_NONE)
  {
     close_UART0();
     return;
  }
  read_length = length;
  //read_length contains number of
  //bytes as read by read_UART0()
  printf("\nLength of read data is
  %u\n",read_length);
  printf("\nThe read data is\n\n");
  flag = write_UART0(readdata,read_le
  ngth);
  //write back the data
  //read on to the terminal
  if(flag! = UART_ERR_NONE)
  {
     close_UART0();
     return;
 }
}
```

Reading from and Writing to the UART in INTERRUPT Mode

POLLING mode is the default mode for transmission and reception. To use INTERRUPT mode, a macro value in the `uartcontrol.h` file is changed for the selected UART, as described below:

```
#define UART0_MODE
MODE_INTERRUPT
//UART0 mode control macro.
```

The `uartcontrol.h` header file is located in the below filepath in the installed directory:

```
  ZDSII_Z8Encore!_<ver-
  sion>\include\Zilog
```

In INTERRUPT mode, `write_UARTx()` and `read_UARTx()` are non-blocking functions. The functions `get_txstatus_UARTx()` and `get_rxstatus_UARTx()` determine the status of the transmit and receive operations, respectively.

The return values are UART_IO_COMPLETE to indicate the completion of the Read operation, or UART_IO_PENDING to indicate that reading is still in progress. If the API is compiled while the UARTx_ERRORCHECKING macro is enabled, any error in the received data byte is reported when a call to `get_rxstatus_UARTx()` is made.

In INTERRUPT mode, after a call to the `write_UARTx()` and/or `read_UARTx()` API, you can execute other instructions that do not interfere with UARTx operation. Zilog® recommends you to check the status of the Read and Write operations using the above APIs before proceeding with Read or Write operations.

The following code displays UART usage in the INTERRUPT mode. In this code, the `read_write()` is a user-defined function. After calling the `read_UART0()` API, you can execute other tasks, as explained. The result of the read operation is checked before the next call is made to the API. The data that is read is written using the `write_UART0()` API, also in INTERRUPT mode.

```
void read_write(void)
{
  unsigned int
  read_length,length = 16,length_1 =
  10;
//define the number of bytes of data
to be read
char readdata[],readdata_1[];//
define buffer to hold read data
int i;
open_UART0();
printf("\nEnter a string of %u char-
acters\n",length);
read_UART0(readdata,&length);//read
data entered through keyboard
//perform other tasks here
PBADDR = 0x01;
 PBCTL = 0x00;
 PBOUT = 0x00;
 for(i = 20000;i = 0;i--);
 PBOUT = 0xFF;
//check the result of previous oper-
ation before proceeding with a new
//one
while(UART_IO_PENDING = get_rxstatu
s_UART0());
    read_length = length;
    //read_length contains number
    of bytes
    //as calculated by read_UART0()
    printf("\nLength of read data
    is %u\n",read_length);
    printf("\nEnter a string of %u
    characters\n",length_1);
    read_UART0(readdata_1,&length_1
    ); //read data
    //perform other tasks here
    PBOUT = 0x00;
    //toggle port pin
    // check status
while(UART_IO_PENDING = get_rxstatu
s_UART0());
    read_length = length_1;
    //read_length contains number
    of bytes
    //as calculated by read_UART0()
    printf("\nLength of read data
    is %u\n",read_length);
    printf("\n The data most
    recently read:\n");
    write_UART0(readdata_1,read_len
    gth); //write back the data read
    on to
    //the terminal
    //toggle port pin again or per-
    form
```

```
      //other tasks here
      PBOUT = 0xFF;
      //check result of operation
  while(UART_IO_PENDING = get_txstatu
  s_UART0());
      printf("\nUART operation in
      interrupt mode tested\n");
}
```

Summary

ZSL provides a simple method to configure and use the UART peripheral of the Z8 Encore!® MCU. A number of APIs are used to access and modify a few basic parameters of the UART. Other parameters are changed by modifying the default values in the `uartcontrol.h` file or by directly writing into the UART control registers in the code. If changes are made to the `uartcontrol.h` file or any other ZSL file, the ZSL library must be rebuilt. For details on all the APIs and procedure for rebuilding the library, refer to *Zilog Standard Library API Reference Manual (RM0038)*.

The `uart.h` file provides definitions for the valid parameter values and the return values of the APIs. The UART structure is also defined in `uart.h` header file.

References

The documents associated with ZSL and Z8 Encore! products available for download at www.zilog.com are listed below:

- Zilog Standard Library API Reference Manual (RM0038)
- Z8 Encore!® Using Zilog Standard Library (ZSL) White Paper (WP0010)

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

Zilog'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF Zilog CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer