Application Note

Using the Z8 Encore! XP® MCUs as I²C Slaves

AN013905-0508

Z8 Encore! XP® Flash Microcontrollers

Abstract

This application note discusses a method of implementing I²C slave functionality on Zilog's Z8 Encore! XP® microcontrollers. The demonstration software for this application note monitors and controls two general-purpose input/output (GPIO) port pins that are connected to the Serial Data (SDA) and Serial Clock (SCL) lines of the I²C bus.

Note: *The source code files associated with this application note,* `AN0139-SC01.zip`*,* `AN0139-SC02.zip`*, and* `AN0139-SC03.zip` *are available for download at* www.zilog.com.

Figure 3 on page 5 displays the I²C implementation details for the Z8 Encore! XP 8K Series microcontrollers. Figure 13 on page 17 displays the implementation details for the Z8 Encore! XP 4K Series microcontrollers. For detailed information about the Z8 Encore! XP MCU, refer to the *Z8 Encore! XP 8K and 4K Series Product Specification (PS0228)*.

Z8 Encore! XP Flash Microcontrollers

Z8 Encore! XP line of microcontrollers expands upon the feature set of the Z8 Encore! line with added peripheral blocks. The Z8 Encore! XP 4K Series devices feature up to eight single-ended/differential channels of sigma-delta 10-bit A/D conversion with 1x or 20x differential input gain, transimpedance amplifier for current gain, and die temperature measurement over a range of –40 ºC to +105 ºC.

Two enhanced 16-bit timer blocks featuring Pulse Width Modulators (PWMs) and Capture/Compare allows to measure the system events. The devices support up to 18 vectored interrupts with programmable priorities. This effectively increases CPU bandwidth and increases application flexibility. The single-pin debugger and programming interface simplifies code development and allows easy in-circuit programming, making design creation faster and easier.

Discussion

I²C is a byte-oriented, serial data transfer protocol that consists of two signals—SCL and SDA. The protocol uses a master/slave transfer convention, in which the master is a device that initiates an I²C transfer, and the slave is a responder. The protocol allows multiple slaves to share a bus, and a particular slave is addressed by a 7-bit address value that is sent as the first transfer byte. The eighth bit of the first transfer (address) byte conveys the type of (Read/Write) operation that the master device performs on the slave device.

SCL and SDA are open-drain outputs with external pull-up resistors. Consequently, when any device connected to the I²C bus pulls a line Low, no other device can pull it High.

A slave device acts according to the instruction(s) sent by the master addressing it. A slave does not generate the clock, but uses the clock generated by the master for all data transfers. A slave can be a receiver or a transmitter. As a receiver, the slave acknowledges the address/data byte received. As a transmitter, the slave receives an acknowledgement from the master after the master receives the data transmitted by the slave.

Data transfer occurs on the SDA line and is phase-synchronized with the clock on the SCL line. There are two unique conditions to consider:

1. SDA changes state when SCL is High. This instance of a state change is treated as a control signal. When the SDA changes from a High to a Low, it indicates a START or a RESTART condition. When the SDA changes from a Low to a High, it indicates a STOP condition.
2. SDA changes state when SCL is Low. This instance of a state change is treated as data transfer condition (address or a data bit). The data is deemed valid only when the SCL line is High. Therefore, data is read on the SDA line when the SCL line is High and not otherwise.

Figure 1 displays the unique START and STOP conditions on the I2C bus.

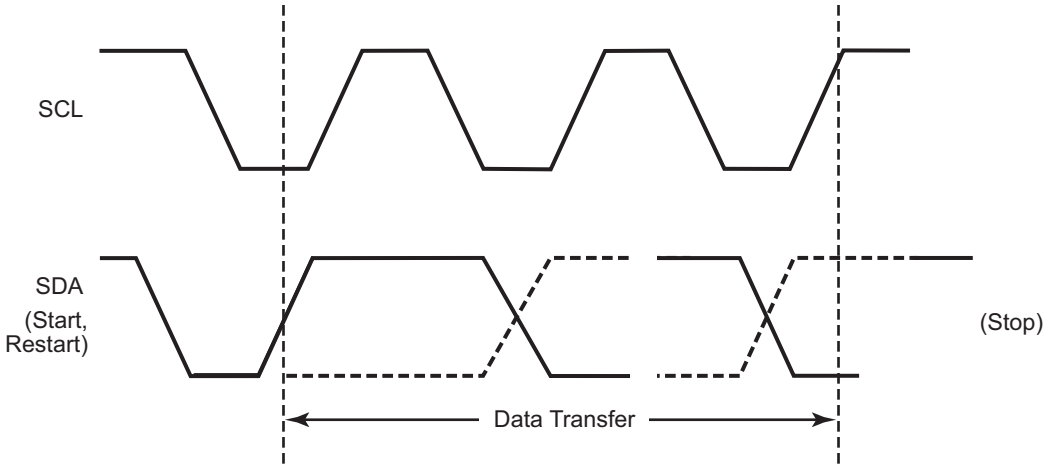


Figure 1. Start and Stop Conditions on the I2C Bus

Developing an I2C Slave Implementation for Z8 Encore! XP®

The Z8 Encore! XP microcontroller features an on-chip I2C peripheral that functions as an I2C master only. The slave functionality (receive/transmit on request), is implemented in software by using two GPIO pins.

The Z8 Encore! XP GPIO port C pin 0 (PC0) and pin 1 (PC1) are used for I2C slave emulation because these pins are configured to generate interrupts on both the rising and falling edges of the input signal. PC0 is configured as SDA and PC1 is configured as SCL.

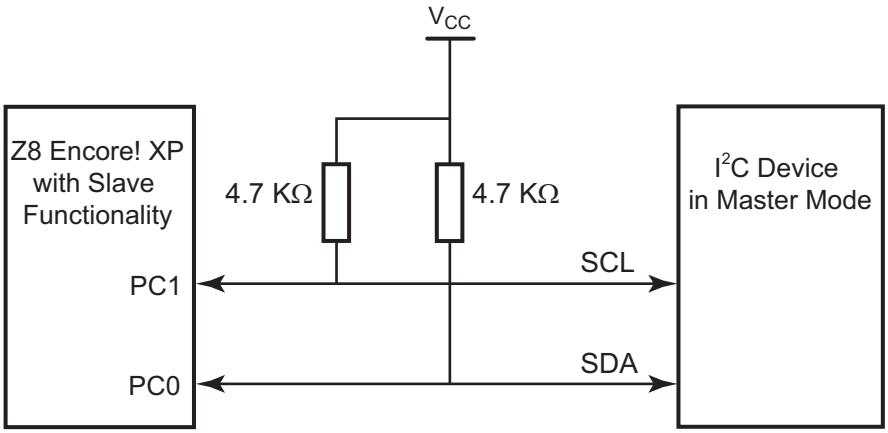


Figure 2. Connecting Slave and Master Z8 Encore! XP® MCUs

PC1 (SCL) is always set as an input line, while PC0 (SDA) is an input or output line, depending on whether it is used for receiving or transmitting data.

There are two methods to emulate I2C slave functionality for any microcontroller:

1. Polling
2. Generating interrupts.

The interrupt method is preferred to the polling method.

The main disadvantage of the polling method is that the processor is compromised for the entire transaction on the I2C bus. The processor continually polls the GPIO pin status for I2C activity and waits for Acknowledgement (ACK), No Acknowledgement (NACK), complete data transmit, data receive, and so on.

In the interrupt method, the processor's attention is drawn to the I2C only when an I2C START condition is detected. The processor's interrupt service routine monitors the status of the SCL and SDA lines. This is on a clock-by-clock basis at the appropriate time and phase within the clock, as detailed in the following section.

Software Implementation

The PC0 and PC1 port pins on the Z8 Encore! XP are capable of detecting and generating interrupts on the rising and falling edges of the SDA and SCL signals. The software performs specific operations during the rising and falling edges of the SCL. The external I2C master generates the SCL and all data transfers on the I2C bus are synchronized with respect to this clock.

I2C Slave Implementation Algorithm

The I2C slave implementation algorithm flows as follows:

1. The Z8 Encore! XP GPIO vectors corresponding to PC0 and PC1 are initialized with the I2C protocol handler service routines.
2. The system remains dormant, waiting for activity on the I2C; while the processor continues to perform its normal activities.
3. Upon detecting a transition on the SDA, the software checks for an I2C START condition.
4. After a START condition is detected, the I2C slave address is read when the SCL is High.
5. If the slave address matches, additional transaction information (Read/Write) is detected.

Otherwise, the system returns to monitoring a REPEAT START or a STOP condition.

The implementation of the I2C slave device presented in this application note is divided into three main routines, as listed below:

1. Main routine
2. SDA interrupt service routine
3. SCL interrupt service routine

Main Routine

In the `main()` routine, the GPIO ports are initialized. The direction of PC0 and PC1 is set to INPUT mode. I2C_Flag is initialized to ADDRESS RECEIVE mode and the I2C bus status is set to FREE. The interrupt priority for PC0 and PC1 is set as high. These interrupts are enabled.

The `main()` routine begins executing user code and waits for a START condition.

Note: *If a user program employs any other interrupts, they must be of a lower priority than the PC0 (SDA) and PC1 (SCL) interrupts. These interrupts must be nested. For a discussion of interrupt nesting, refer to Nesting Interrupts in Z8 Encore! XP® MCUs Application Note (AN0141).*

SDA Interrupt Service Routine

The program enters this routine when an SDA interrupt is generated. This routine identifies the START, normal data transfer, and the STOP conditions on the I2C bus. The SDA interrupt is disabled upon identifying a START/RESTART condition.

SCL Interrupt Service Routine

The program enters this routine when an SCL interrupt is generated. This interrupt service routine performs the following operations:

- Keeps a count of the number of SCL rising edges.
- Re-enables the SDA interrupt at the tenth SCL.
- Performs the appropriate function call to perform an address read, data read, or data transmit. It reads the valid data on SDA when SCL is High.
- Because the first bit to be transmitted/received is the msb of a byte, the bit is shifted according to the msb position. Every bit is shifted, and finally an 8-bit data byte is formed.

On every falling edge of the SCL, the SCL ISR calls one of the following functions based on the I2C_Flag:

- `Slave_address_receive()`
- `Slave_data_receive()`
- `Slave data transmit()`
- `Address_mismatch()`

When an address mismatch condition is identified, the program ignores the incoming data and waits for a STOP condition.

See the flowcharts in Appendix A—Flowcharts on page 10 for details.

Testing the I²C Slave Implementation

To test the I²C slave implementation, the I²C master code is used. The Z8 Encore! XP® master uses the following two routines to write and read from the Z8 Encore! XP slave:

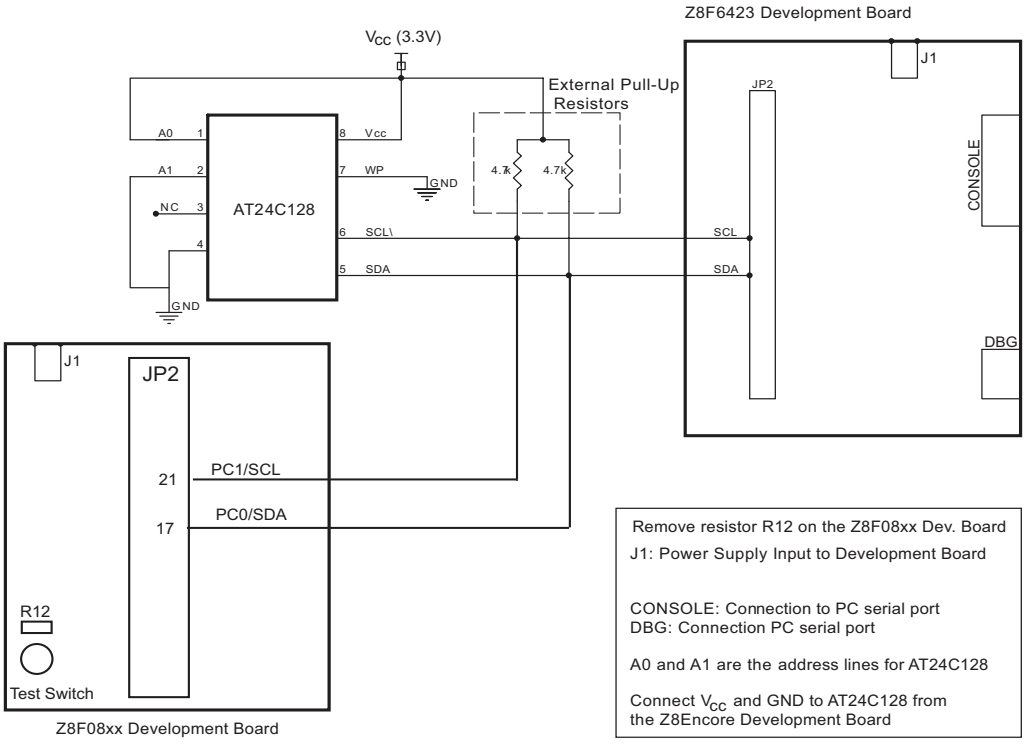
```
void Write_Z8Encore_slave
(unsigned char addr, unsigned char
data)

char Read_Z8Encore_slave (unsigned
char addr)
```

Note: *In this implementation, the slave address of the Z8 Encore! XP is defined as* `0xAE` *for the* `WRITE` *mode (to change this device address, change the* `#definition DEVICE_ADDRESS` *in the* `scl_interrupt.h` *file). To make the device operate in* `READ` *mode, use the slave address* `0xAF`.

Setup

Figure 3 displays the setup to test I²C transactions between a Z8 Encore! XP I²C Z8F642xx master device and a Z8 Encore! XP Z8F08xx slave device. An I²C EEPROM is connected to the I²C bus as a second I²C slave.



Interfacing AT24C128 I²C EEPROM and Z8F08xx I²C Slave to Z8F6423 Development Board

Figure 3. Test Setup for the Z8 Encore! XP Z8F08xx I²C Slave

To use the Z8 Encore! XP MCU as an I2C slave, make the connections as shown in Figure 13 on page 17. Download the source code from www.zilog.com onto the Z8 Encore! XP MCU Development Board and follow the remaining steps (after step 1) listed in the Procedure section.

Equipment Used

The equipment used for testing the I2C slave implementation are shown below:

- One Z8 Encore! XP® Development Kit (Z8F64200100KITG) with a Z8 Encore! XP 64K Series (Z8F6423x) MCU used as the I2C master for testing the I2C slave
- One Z8 Encore! XP Z8F08xx Development Kit (Z8F08200100KITG) with a Z8 Encore! XP 8K Series (Z8F08xx) MCU used as the I2C slave
- I2C EEPROM (AT24C128) with the required pull-up resistors (see Figure 3 on page 5)
- ZDS II—Z8 Encore! IDE with ANSI C-Compiler
- Logic analyzer to observe I2C waveforms
- PC equipped with a HyperTerminal application as a user interface for testing

Procedure

To test the I2C slave implementation, perform the following procedure.

1. Using ZDS II, download the slave implementation code contained in the `AN0139-SC02.zip` file onto a Z8 Encore! XP Z8F08xx MCU Development Board. This board now functions as the I2C slave.
2. Download the I2C master code listed in the `AN0139-SC01.zip` file to another 64KB Z8 Encore! XP MCU (Z8F6423x) Development Board using ZDS II. This board now functions as the I2C master.
3. The SDA and SCL lines of the three devices (master, Z8 Encore! XP, and slave) are connected according to the setup displayed in Figure 3 on page 5.
4. Launch the HyperTerminal application with the following settings:

   Baud rate = 9600 bps
   Data bit = 8 bits
   Parity = None
   Stop bits = 1
   Flow control = None
   Port = COM2

5. Execute the master program from the ZDS II IDE.
6. The `TEST_PROGRAM` prompt appears in the **HyperTerminal** window along with the Write and Read menus for both of the slave devices.
7. Enter `W` for Write, and enter `0xAE` as the Write address for the Z8 Encore! XP slave device.
8. Enter the data to be written to the Z8 Encore! XP slave device. A displayed message, `data written successfully`, appears in the **HyperTerminal** window, as shown in Figure 4 on page 7.
9. Enter `R` for Read, and enter `0xAF` as the read address for the Z8 Encore! XP slave device.
10. The same data that was previously written to the Z8 Encore! XP slave device is displayed in the **HyperTerminal** window, as shown in Figure 4 on page 7.
11. Enter `A` to write to the I2C EEPROM slave device, and enter the address of the location into which you want the data to be written.
12. Enter the data to be written to the I2C EEPROM device. A displayed message, `data written successfully`, appears in the HyperTerminal window, as shown in Figure 5 on page 8.
13. Enter `B` to read from the I2C EEPROM slave device, and enter the address of the location into which you want the data to be read.

14. The data available at the specified I2C EEPROM address is displayed in the HyperTerminal window, as shown in Figure 5 on page 8.

Results

Testing was performed at data transfer rates up to 70 KHz, and the results were found to be compliant with I2C slave functionality.

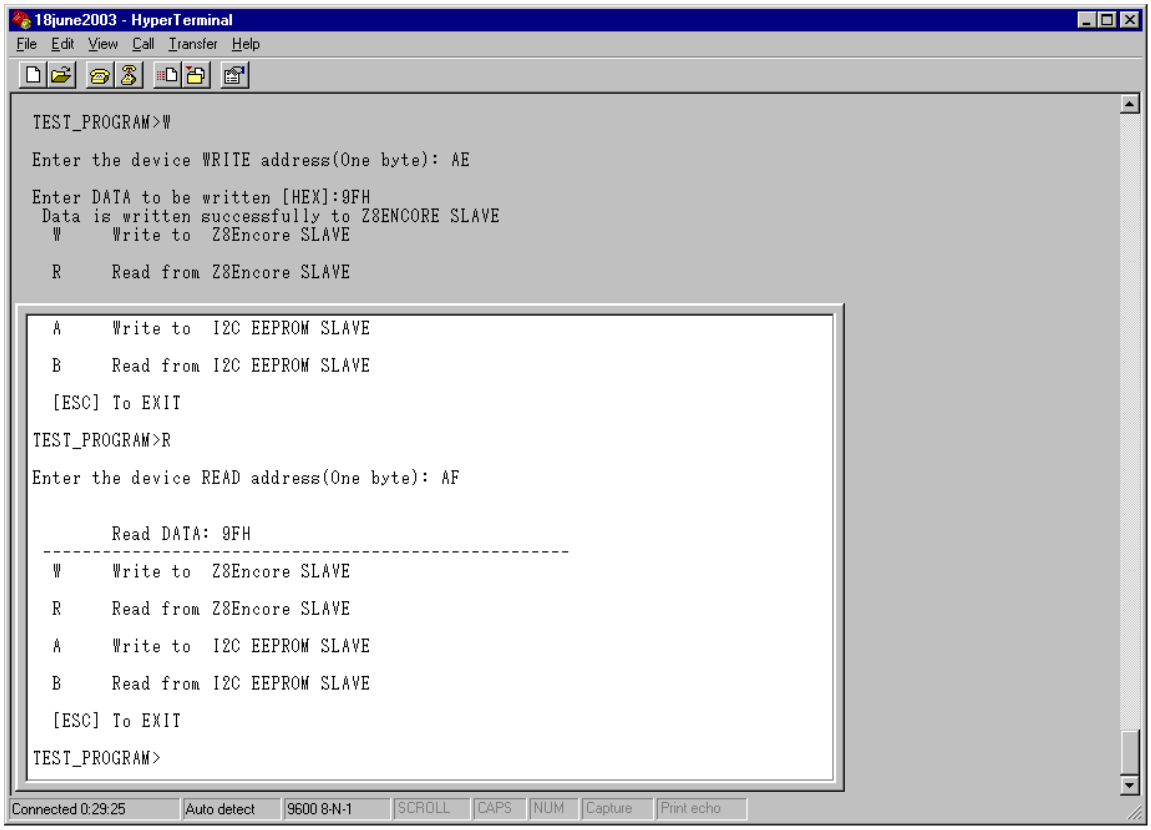


Figure 4. HyperTerminal Display, displaying a Read/Write Operation on the Z8 Encore! XP® Z8F08xx I2C Slave

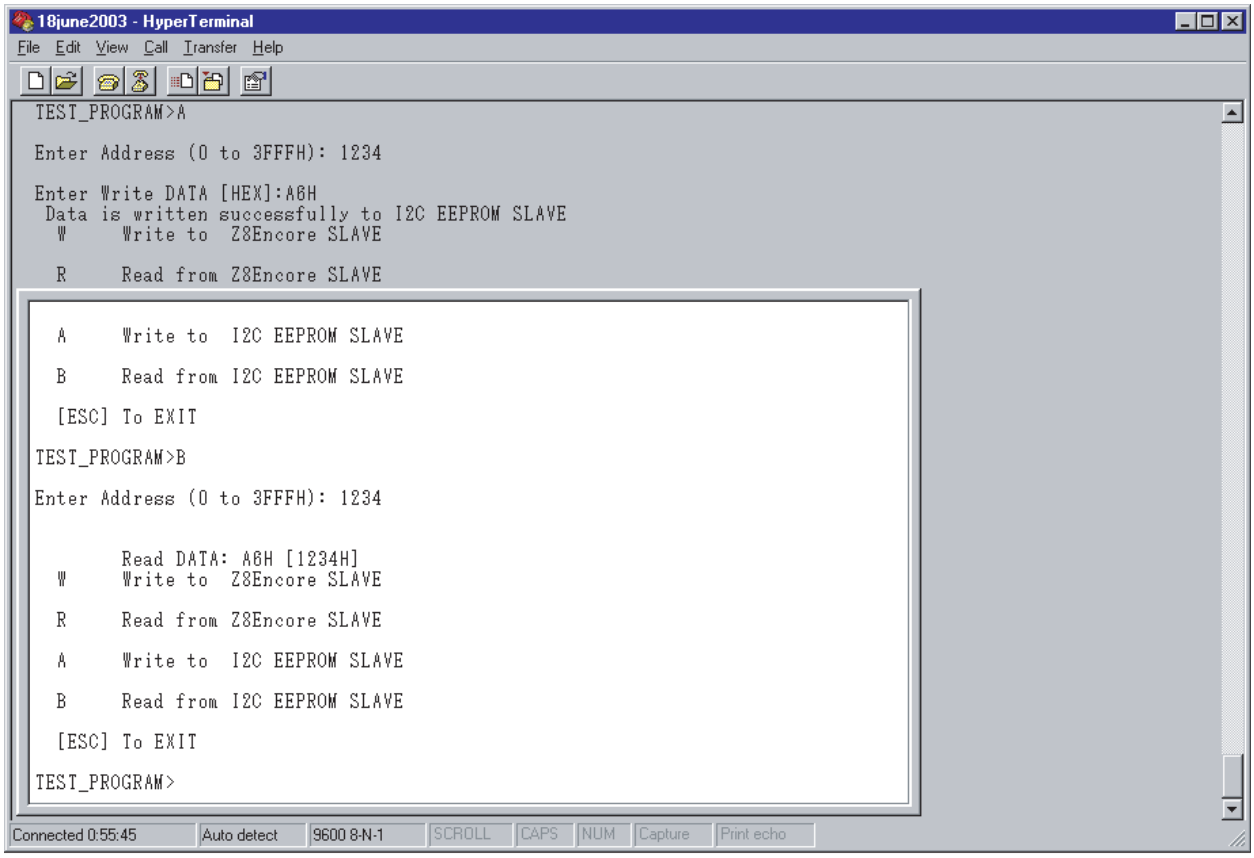


Figure 5. HyperTerminal Display, displaying the Read/Write Operation on the I2C EEPROM

Summary

This application note presents a method for implementing slave functionality on the Z8 Encore! XP MCU using GPIO pins to emulate SCL and SDA lines. The software supports transactions on the I2C bus at data transfer rates of up to 70 KHz.

The Zilog application note titled *Using the Z8 Encore! XP MCU as an* I2C *Bus Master (AN0126)* provides master code containing APIs that can be used for communication between an I2C master and an I2C EEPROM slave device. This code can be downloaded onto a Z8 Encore! XP MCU. The slave implementation provided with this application note can be downloaded to the same Z8 Encore! XP MCU. Thus, a single Z8 Encore! XP MCU, with both master and slave implementations, can be configured to work as a slave or as a master.

However, other I2C devices in the network must be able to identify whether the Z8 Encore! XP MCU is functioning as a slave or as a master. To allow other I2C devices to identify the Z8 Encore! XP MCU as a slave or a master, set the Z8 Encore! XP MCU as a slave by default. A GPIO pin on the Z8 Encore! XP MCU (other than PC0 or PC1) is configured as an output. A High/Low level on this GPIO pin indicates a master and a Low/High indicates a slave. The other I2C devices in the network must poll this GPIO line to know the status of the Z8 Encore! XP MCU. The software presented in this application note does not provide this feature.

Additionally, this application note highlights the interface details for the I2C implementation on the Z8 Encore! XP® Series microcontrollers. For a detailed information about the Z8 Encore! XP MCU, refer to the *Z8 Encore! XP 8K and 4K Series Product Specification (PS0228)*.

References

The documents associated with Z8 Encore! XP MCU and the ZDS II available on www.zilog.com are provided below:

- eZ8™ CPU User Manual (UM0128)
- Z8 Encore! XP F64xx Series Product Specification (PS0199)
- Z8 Encore! XP F0822 Series Flash Microcontrollers Product Specification (PS0225)
- Z8 Encore! Flash Microcontroller Development Kit User Manual (UM0146)
- Z8 Encore! XP® 8K and 4K Series Product Specification (PS0228)
- Z8 Encore! XP® F042A Series Development Kit User Manual (UM0166)
- Zilog Developer Studio II–Z8 Encore! User Manual (UM0130)

Appendix A—Flowcharts

The flow of the `main` routine for the I2C slave implementation is displayed in Figure 6.

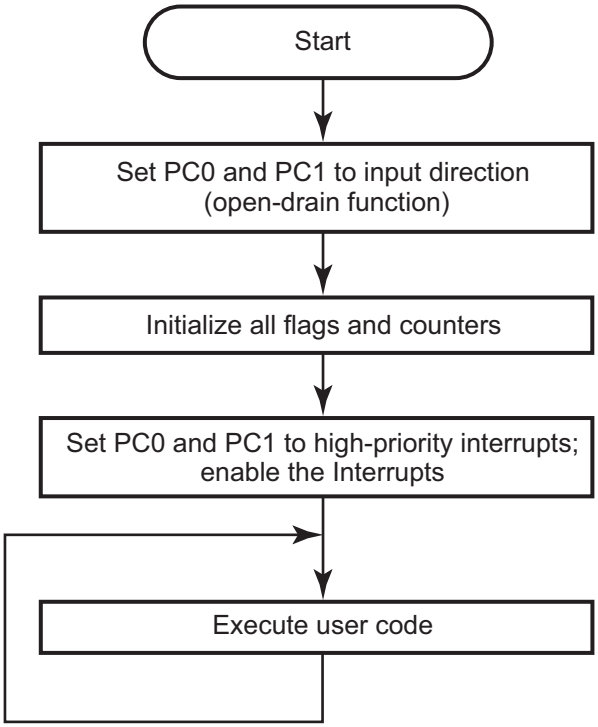


Figure 6. Flowchart of the Main Routine

The flow of the SDA interrupt service routine is displayed in Figure 7.

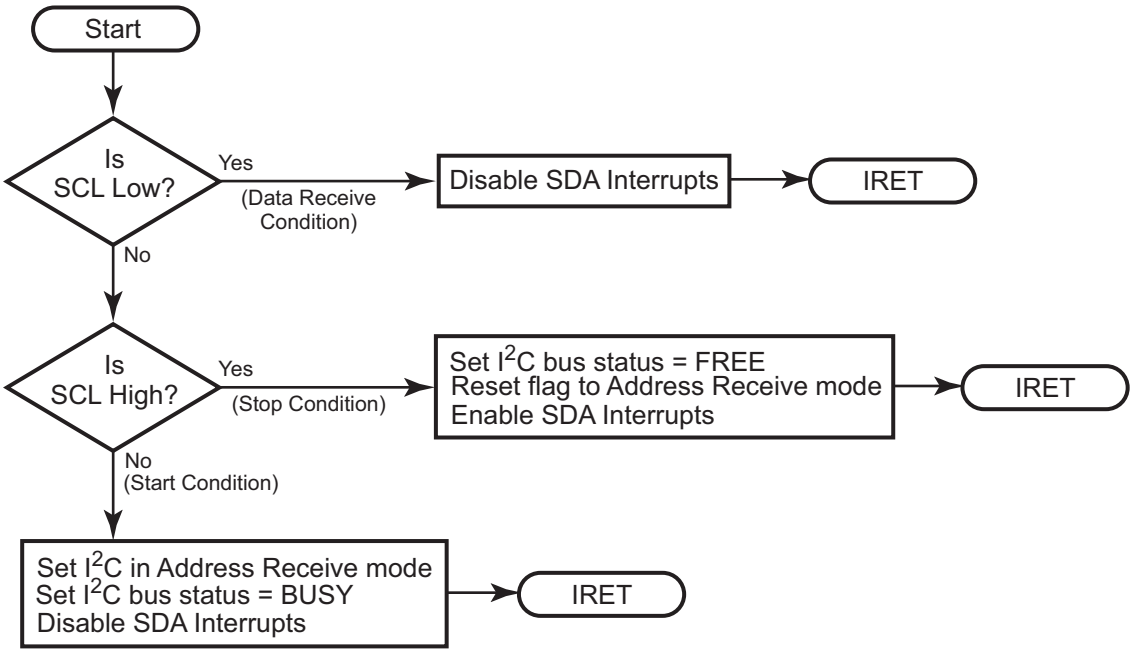


Figure 7. Flowchart of the SDA Interrupt Service Routine

The flow of the SCL interrupt service routine is displayed in Figure 8.

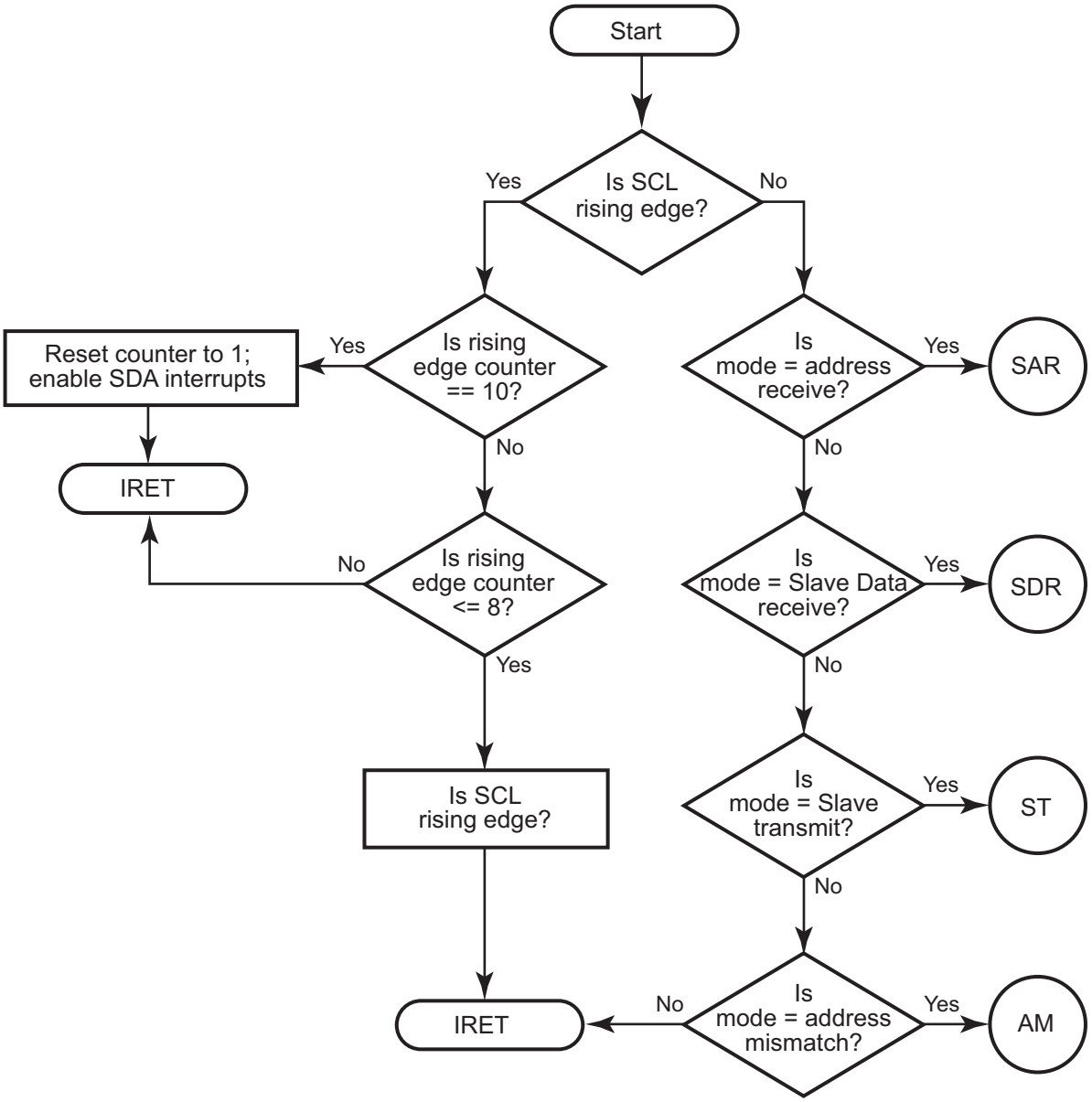


Figure 8. Flowchart of SCL Interrupt Service Routine

The flow of the Slave Address Receive function is displayed in Figure 9.

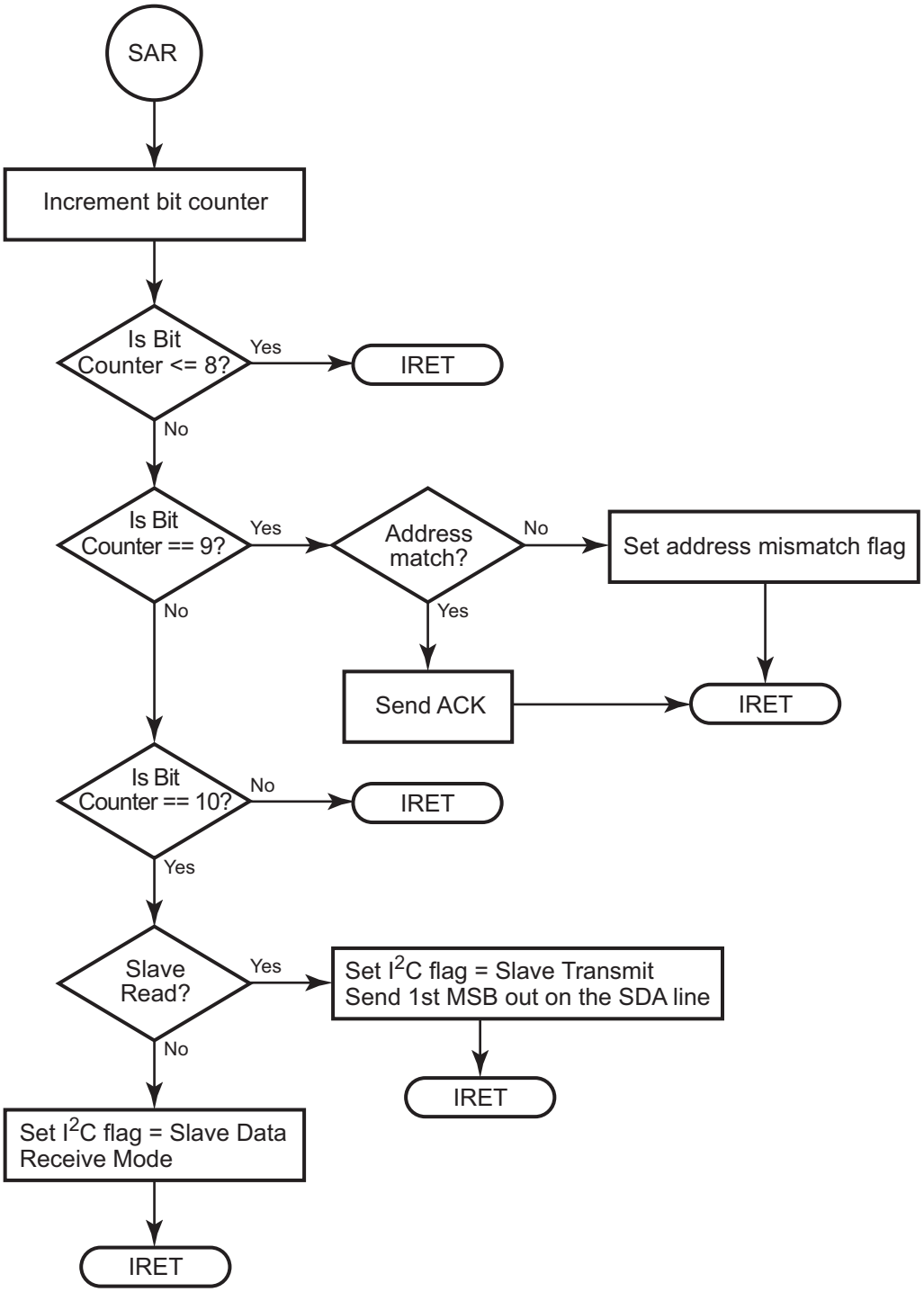


Figure 9. Flowchart of the Slave Address Receive Function

The flow of the Slave Data Receive function is displayed in Figure 10.

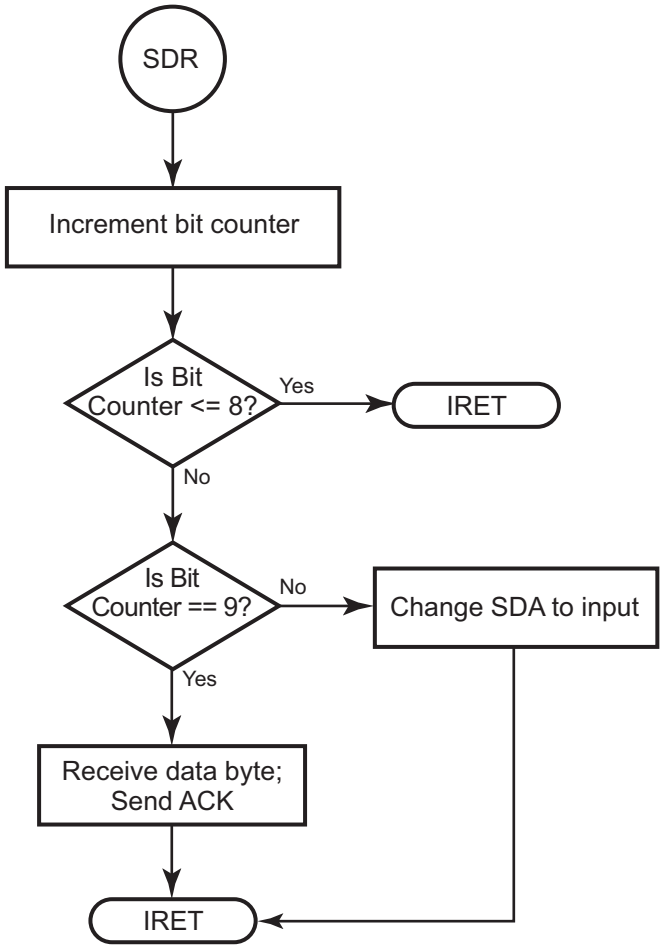


Figure 10. Flowchart of the Slave Data Receive Function

The flow of the Slave Transmit function is displayed in Figure 11.

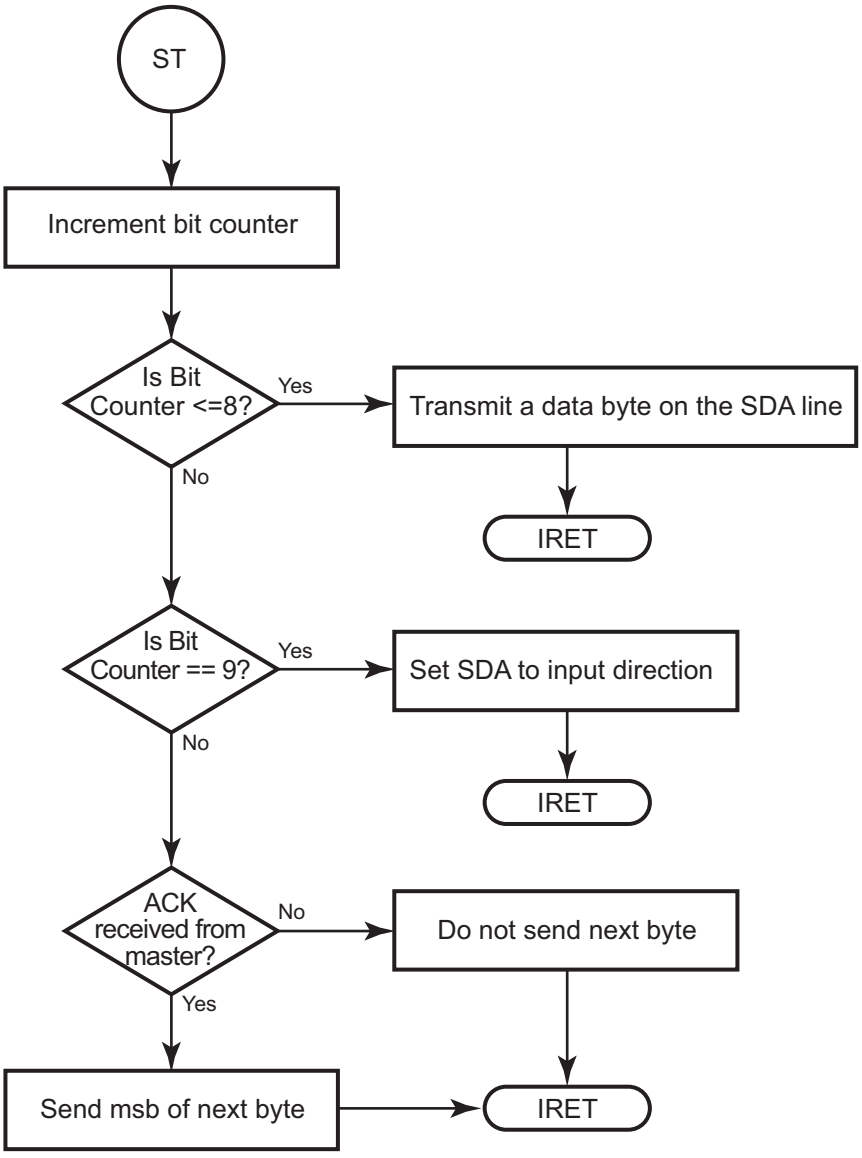


Figure 11. Flowchart of the Slave Transmitter Function

The flow of the Address Mismatch function is displayed in Figure 12.

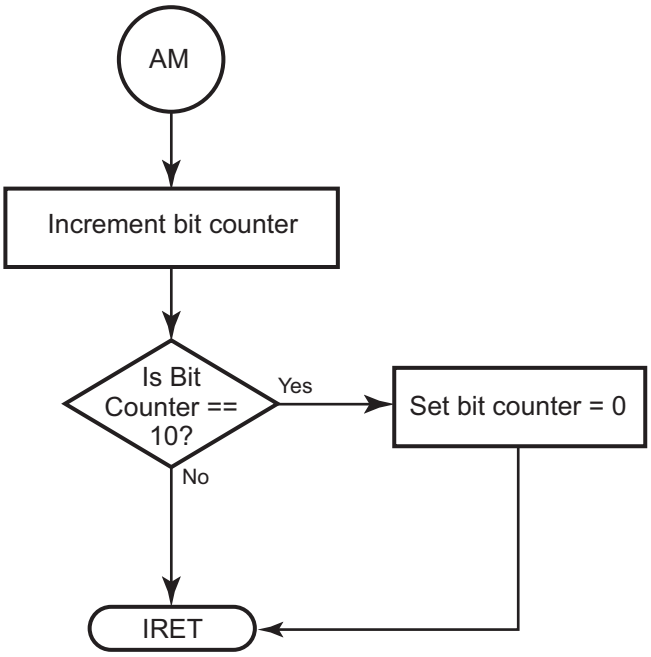


Figure 12. Flowchart of the Address Mismatch Function

Appendix B—Schematic Diagram for the Z8 Encore! XP® I²C Slave

Figure 13 is a schematic displaying the test setup for the Z8 Encore! XP I²C slave.

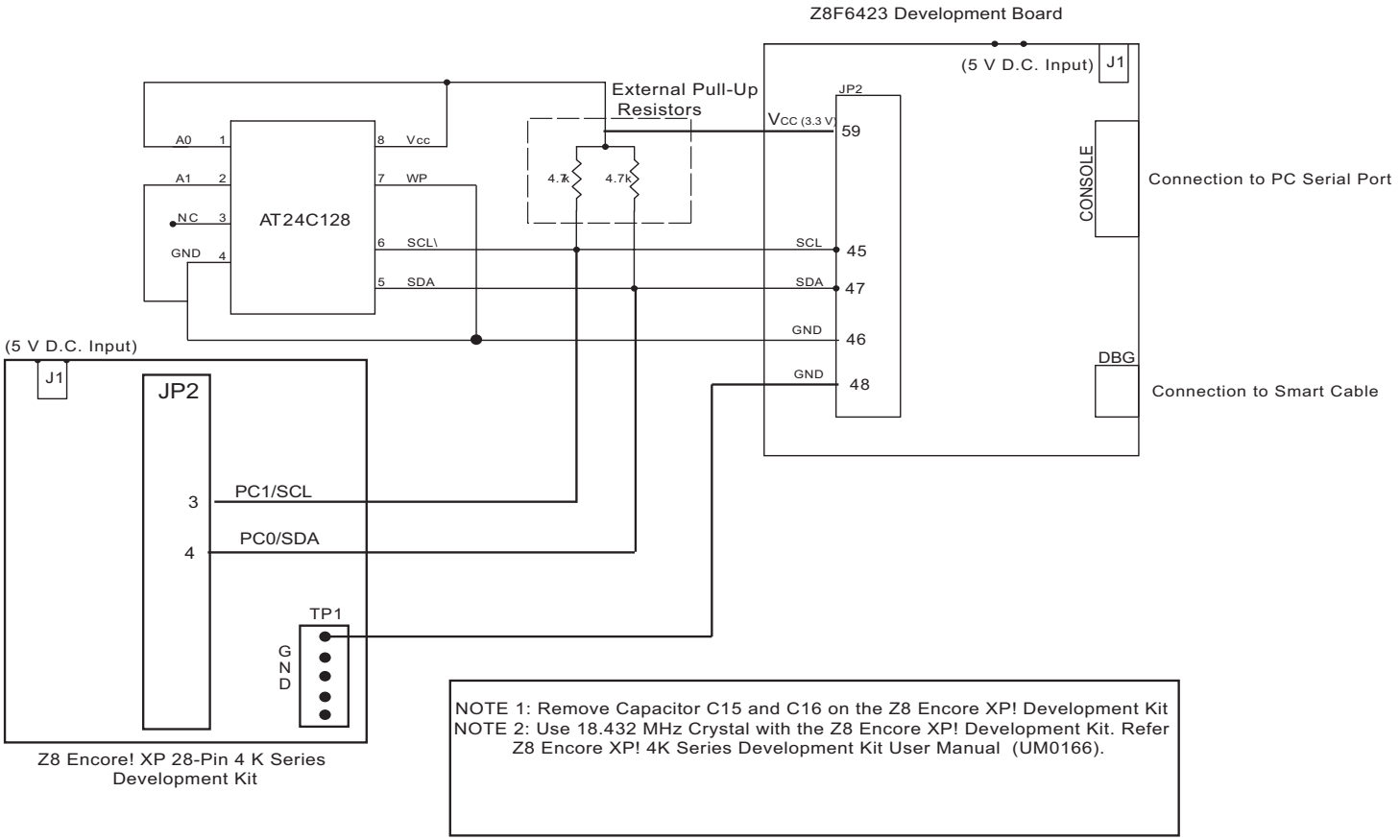


Figure 13. Test Setup for the Z8 Encore! XP I²C Slave

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