Application Note

Software I²C Master and Slave Mode Support for Z8 Encore!® F0830/F083A

AN026602-0308

Abstract

This Application Note demonstrates a method of implementing I²C Master functionality on Zilog's Z8 Encore!® microcontrollers. The software implemented in this Application Note controls the two GPIO port pins connected to the serial data (SDA) and serial clock (SCL) lines of the I²C bus.

Note: *The source code file associated with this application note, AN0266-SC01 is available for download at* www.zilog.com.

Discussion

I²C is a byte-oriented, serial data transfer protocol that consists of two signals:

1. Serial clock (SCL)
2. Serial data (SDA)

The protocol uses a Master-Slave transfer convention where 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 8th bit of the first transfer byte (address byte) describes the type of operation (Read/Write) 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 can only pull a line Low.

The Slave device acts according to the instruction(s) sent by the Master addressing it. The Slave does not generate the clock, but uses the clock generated by the Master for all the data transfers. The 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. The two unique conditions are:

1. *SDA changes the state when SCL is High*: This instance of a state change is considered as a control signal. When the SDA changes the state 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 the state when SCL is Low*: This instance of a state change is considered as data transfer condition (address or a data bit). The data is 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.

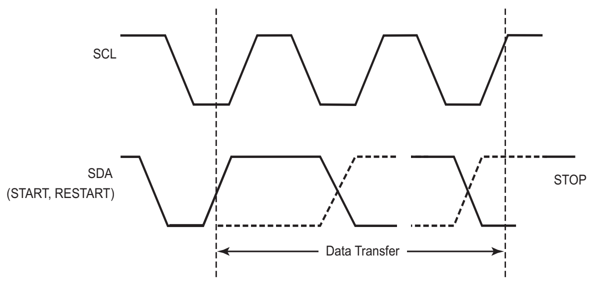

Figure 1. START and STOP Conditions on the I²C Bus

Developing Software Emulation

The Z8 Encore!® F0830 and Z8 Encore! F083A microcontrollers do not have a hardware peripheral for I²C support, however the protocol is very easy to implement in the software. For more information on software solution for I²C Slave support, refer to *Using the Z8 Encore!® and Z8 Encore! XP® MCUs as I²C Slaves Application Note (AN0139)*.

Note: *This application note uses the Slave support from AN0139 to add Master support for a full Master/Slave solution. For continuity with AN0139, port pin PC0 is used for the SDA and PC1 is used for SCL.*

PC0 and PC1 pins are configured for open-drain, so that they do not provide any source current. All the source current is provided by the external pull-ups, which allow the Slave devices to hold the SCL Low for clock stretching. Switching the port from an input to output configuration is not required because the port input register, PxIN, always returns the state of the input pin. Once the Master has released the pin High, the Slave can pull the pin Low or leave it High without the Master having to switch the pin from output to input.

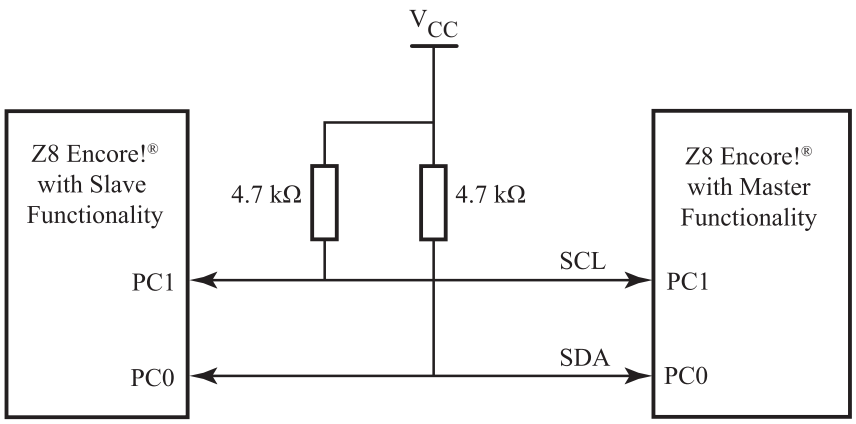
V_CC
Z8 Encore!® with Slave Functionality
4.7 kΩ
4.7 kΩ
Z8 Encore!® with Master Functionality
SCL
PC1
PC1
SDA
PC0
PC0

Figure 2. Z8 Encore!® with Slave Functionality

Hardware

The hardware required to implement the I²C bus is a pull-up resistor on the SCL and SDA lines.

Software

The Master emulation is serviced by the following seven API functions:

1. void InitI2C()
2. void Start()
3. void Stop()
4. void WriteByte (unsigned char data)
5. unsigned char ReadByte()
6. void PutAck(unsigned char)
7. unsigned char GetAck()

`void InitI2C()`

This function configures the I/O port for I²C operation. The selected port pins are configured as open-drain outputs, with High drive enable.

`void Start()`

This function generates a START condition.

`void Stop()`

This function generates a STOP condition.

`void WriteByte (unsigned char data)`

A character is transmitted on the bus. There is no acknowledge (ACK) or negative acknowledge (NACK); this is handled by `PutAck(unsigned char)`.

`unsigned char ReadByte()`

A character is returned from the bus. The Master receives a character from the bus by clocking 8 bits from a Slave device.

`void PutAck(unsigned char)`

If the argument is TRUE an acknowledge pulse, or ACK is generated. If the argument is FALSE, a NACK is generated.

`unsigned char GetAck()`

This function returns TRUE if the Slave acknowledges the transfer otherwise FALSE.

These seven API functions are driven by several macros and constants defined in the header file `I2Cmaster.h`. See Appendix A—Flowcharts on page 6 for API functions.

Note: *In the following code example,* `F0830` *is a code definition used for conditional compiling and not a reference to the product itself.*

```
#define F0830 FALSE//Set TRUE or
FALSE.

//Alter these definitions to move the
I2C bus to other pins

#define SDA 1
#define SCL 2
#define PORTOC PCOC
#define PORTHDE PCHDE
#define PORTDD PCDD
#define PortIn PCIN
#define PortOut PCOUT
```

The definition `F0830` defines whether `F0830` or `F083A` is used. The difference is that an Z8 Encore! F0830 runs on a 5.5 MHz internal oscillator and the F083A runs on a 20 MHz internal oscillator. The 20 MHz clock speed needs longer clock delays because of the higher speed, so that this switch accommodates the speed difference.

The next constants define the pins that are used for SCL and SDA. The next definitions; SDA, SCL, and Port OC define the port that is used for the I^2C interface. For simplicity, SDA and SCL must be on the same port, that is, Port A, Port B, or Port C.

Macros are used to set SDA and SCL High or Low. These macros use the previous port definitions to make it easy to change pin assignments. The `BitStretch` macro is an indefinite loop that waits for the Slave device to release the SCL line. The application may require a *trap* for this condition.

Testing the I^2C Master/Slave

Equipments Required

The equipments required for testing include:

- Z8 Encore! F083A 28-Pin Development Kit
- 4.7 K resistors

The Slave is measured by performing a write to the Slave while monitoring the SCL line with an oscilloscope and measuring the total write time. The total time for the write is 150 µs, 9 bits/150 µs = 60 kbps.

The Master is measured by performing a serial read to an external, 400 kHz EEPROM attached to the I^2C bus. The time required to perform the serial read is measured using an oscilloscope to monitor the activity on the SCL line. Using the minimum delay in `Clock()` the complete serial read time for 20 bytes is 923 µs. The number of bytes received is 20 x 9 bits for a total transfer of 180 bits that is, 180 bits/928 µs = 194 kbps.

Hardware Setup

Figure 3 on page 4 displays the hardware setup to test the I^2C Master/Slave.

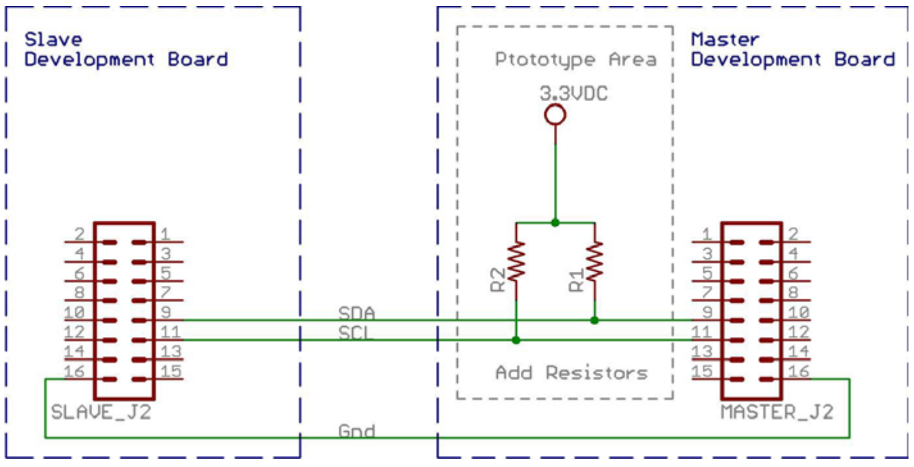

Figure 3. Test Setup for I^{2}C Master/Slave

Procedure

Follow the steps below to test the I^{2}C Master/Slave:

1. Solder two 4.7 K resistors to the 3.3 V DC test pad on the Master's development kit. The resistors are the required pull-ups for I^{2}C protocol.
2. Solder a wire to the free end of one resistor and secure the other end of the wire to the Master's pin J2 through J9. Wirewrap is preferred because it's easy to work on these headers; this will be the SDA.
3. Solder a wire to the free end of the other resistor and secure the other end of the wire to the Master's pin J2 through J11; this will be the SCL.
4. Connect a wire from the Master's J2 through J9 to the Slave's J2 through J9.
5. Connect a wire from the Master's J2 through J11 to the Slave's J2 through J11.
6. Connect a ground wire from the Master's J2 through J16 to the Slave's J2 through J16.
7. Connect the power supply to each development kit.
8. Build the Master ZDS II project and flash the code into the Master development board.
9. Build the Slave ZDS II project and flash the code into the Slave development board.
10. Reset the Slave and the Master development boards by pressing SW1 or power cycling the boards.

Slave and Master

Slave

The Slave software is derived from *Using the Z8 Encore!® and Z8 Encore! XP® MCUs as I^{2}C Slaves Application Note (AN0139)*. As the Z8 Encore!® F083A run at 20 MHz, the internal oscillator can be used and is not necessary to switch the clock source of the Slave to the external resonator. Also, the address of the Slave is changed. The Slave address is located in the header file `scl_interrupt_XP.h` in the definition `DEVICE_ADDRESS`. The address has to be changed so that it does not have the same address as EEPROMs. The only other modification is to include an IF statement to toggle LED D3 when a packet is received. This is done in the Main loop of `scl_interrupt_XP.c` file. If a packet is received LED D3 on the Slave development board will toggle state as ON or OFF.

Master

The Master software sends a packet to the Slave when SW2 is pressed. If the Slave receives the packet, the Slave toggles the state of LED D3 on the Slave development board. If the packet is not received, then the Master will light its LED D3.

At power-up, the Master configures the SDA and SCL pins as open-drain outputs. That is, the Master does not pull the SDA and SCL High; instead they are pulled High by the pull-up resistor. This allows a Slave device to pull SDA and SCL Low. That is, the Master need not reconfigure the I/O pins from output to input for a slight improvement in code size speed. The LED pin is also enabled and is configured for 13 mA drive.

The Main loop waits for 125 ms to help debounce the switch and then the switch is read. If the switch is closed, a packet transmission gets started. The packet transmission is a FOR loop with multiple retries. The Slave may not respond to a packet because of clock cycles being out of synchronous or the Slave not being ready. EEPROMs, for example, do not return an ACK until after a write cycle is completed, which can be 10 ms or more, depending upon the part. Addressing the Slave until an acknowledge is received is called **Acknowledge Polling** and is a common method to determine when a write cycle is complete. Write cycle times are highly variable, and without the acknowledge polling you must wait for the maximum write cycle time. With polling, the Master continuously addresses the Slave and will receive a negative acknowledge (NACK), while the Salve is busy with a write cycle. When the Slave completes its write cycle, it returns an ACK to the Master signaling that the Slave has completed its write cycle.

Addressing a Slave begins with a START condition, followed by the Slave's device address, and then the test for the Slave's acknowledge. If the Slave responds the next byte is sent, but if it does not acknowledge it is repeated until the maximum number of repeats and Flash the LED.

An additional portion of sample software includes EEPROM reading and writing to a 24C08 EEPROM and is similar to the process as communicating with the Slave development board.

EEPROM addressing is different from device to device, therefore, verify the data sheet for the EEPROM used in your design. Most EEPROMs include a Page Write, where multiple bytes can be written without readdressing the EEPROM and with only a single write cycle. This feature is chip specific. As a result, the sample software does not perform a Page Write.

A Sequential Read is a feature that allows a Master to read multiple bytes without addressing the Slave for each individual byte, and significantly increases the throughput of data transfers. The requirement of a Sequential Read is that the Master acknowledge each byte received, and not acknowledge the final byte. The sample software includes a Sequential Read filling a small buffer.

Summary

This Application Note presents a method for implementing Master/Slave functionality on the Z8 Encore!® 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 60 kbps for the Slave and 194 kbps for the Master.

References

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

- Z8 Encore!® F083A Series Product Specification (PS0263)
- eZ8 CPU Core User Manual (UM0128)
- Zilog Developer Studio II—Z8 Encore!® User Manual (UM0130)
- Using the Z8 Encore!® and Z8 Encore! XP® MCUs as I²C Slaves Application Note (AN0139)
- Z8 Encore!® F083A Series Development Kit User Manual (UM0206)

Appendix A—Flowcharts

This Appendix displays the flowcharts of the API functions (Figure 4 through Figure 10).

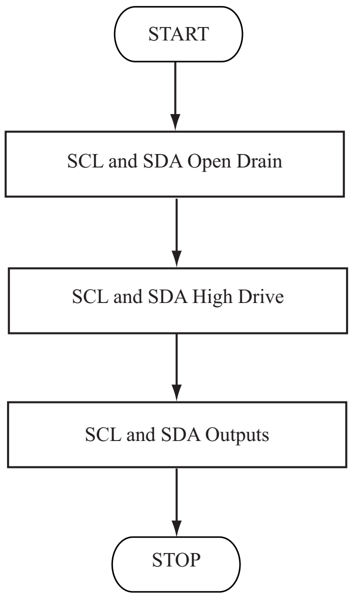

Figure 4. `InitI2c()` Function

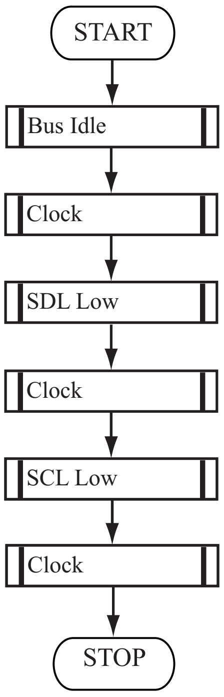

Figure 5. `Start()` Function

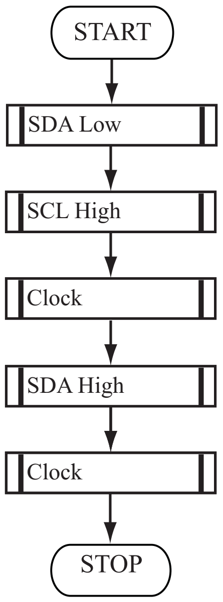

Figure 6. `Stop()` Function

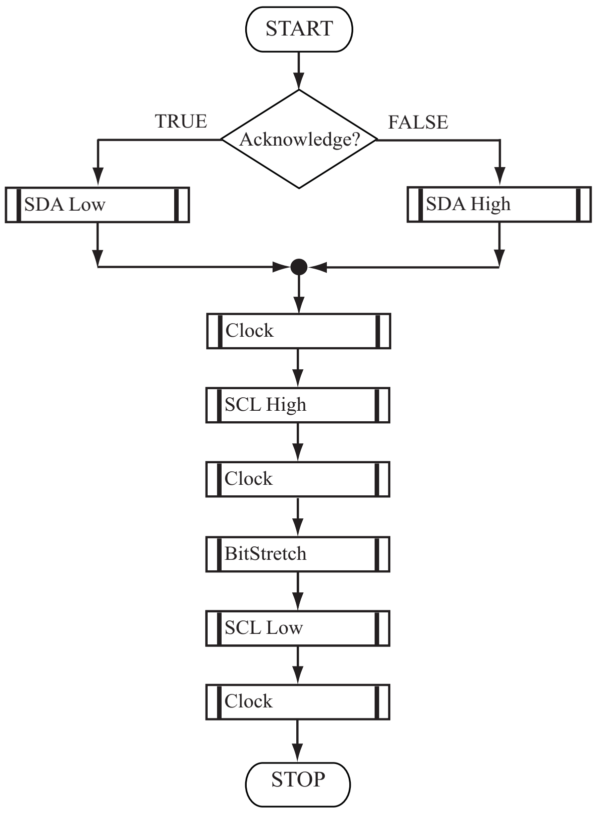

Figure 7. `PutAck()` Function

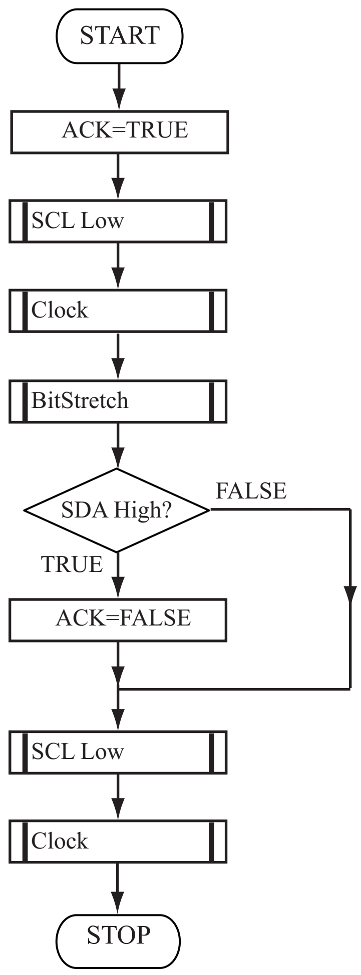

Figure 8. `GetAck()` Function

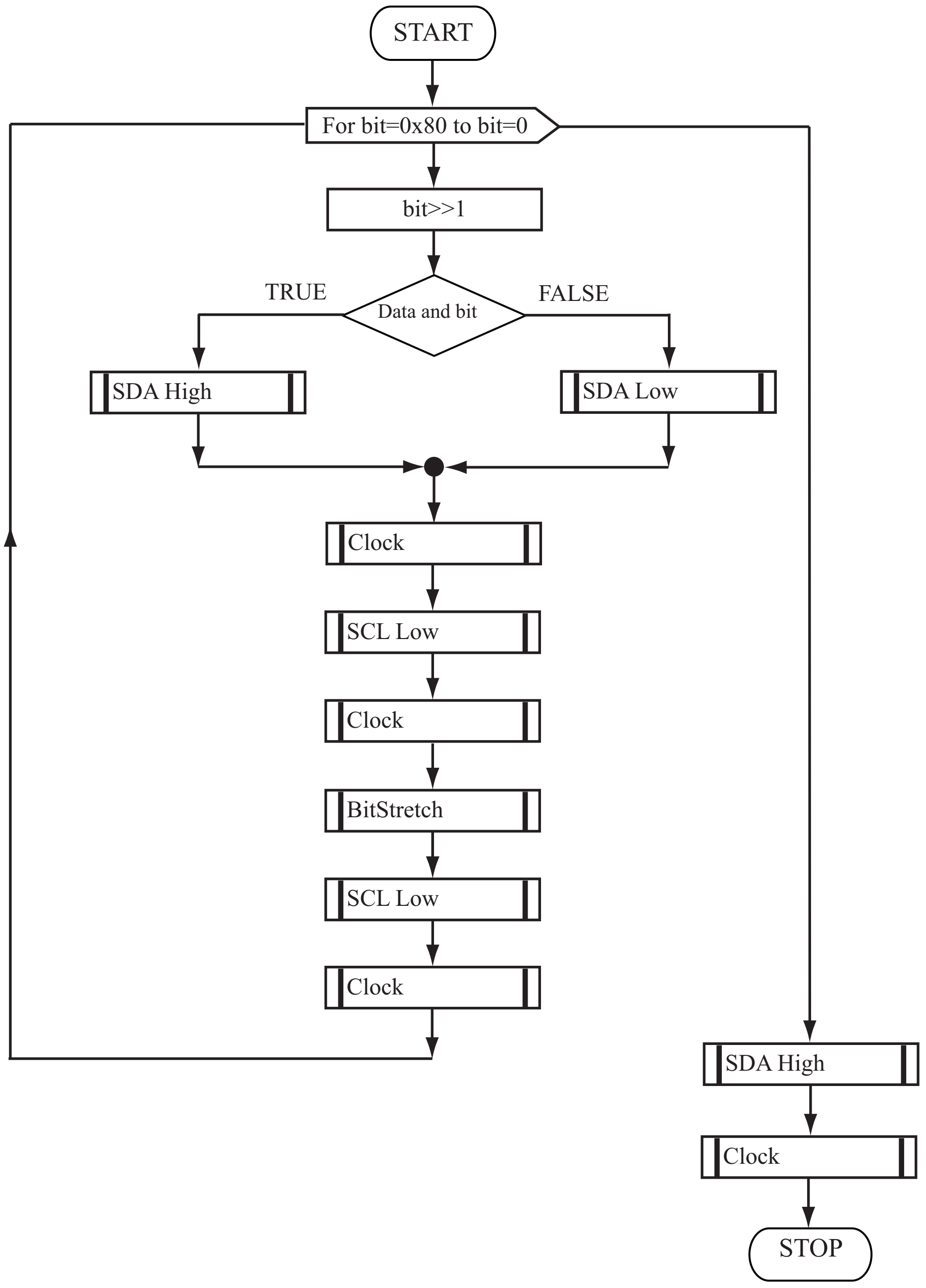

Figure 9. `WriteByte()` Function

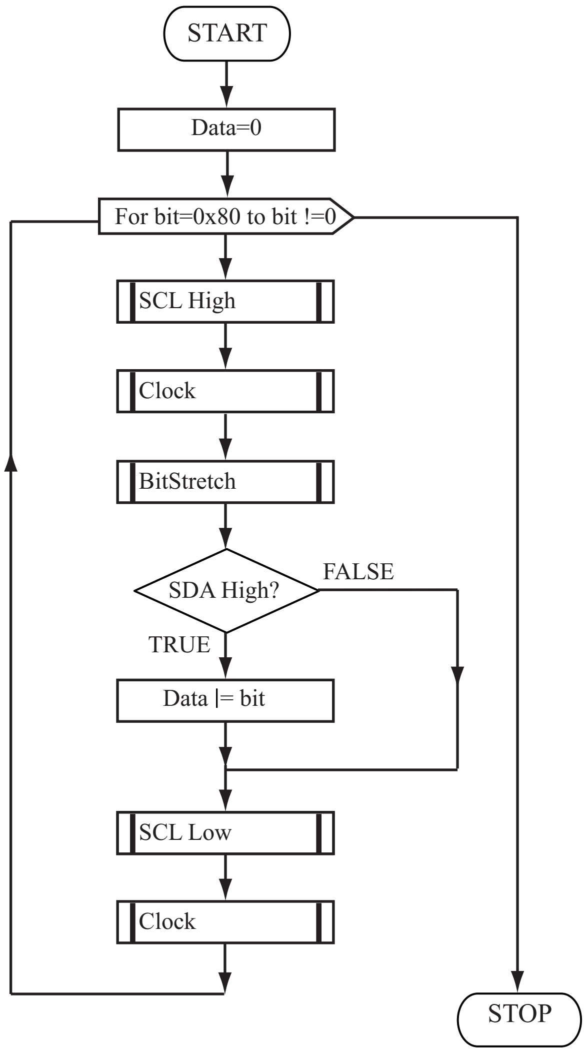

Figure 10. `ReadByte()` Function

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