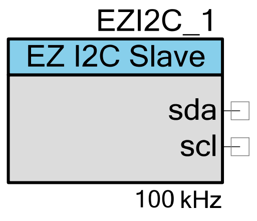
EZ I^{2}C Slave

1.50



Features

- Industry standard Philips I^{2}C bus compatible interface
- Emulates common I^{2}C EEPROM interface
- Only two pins (SDA and SCL) required to interface to I^{2}C bus
- Standard data rate up to 1 Mbps
- High level API requires minimal user programming
- Support one or two address decoding

General Description

The EZ I^{2}C Slave component implements an I^{2}C register-based slave device. The I^{2}C bus is an industry standard, two wire hardware interface developed by Philips®. The master initiates all communication on the I^{2}C bus and supplies the clock for all slave devices. The EZ I^{2}C Slave supports the standard mode with speeds up to 1000 kbps and is compatible with multiple devices on the same bus.

The EZ I^{2}C Slave is a unique implementation of an I^{2}C slave in that all communication between the master and slave is handled in the ISR (Interrupt Service Routine) and requires no interaction with the main program flow. The interface appears as shared memory between the master and slave. Once the Start() function is executed, there is little need to interact with the API.

When to use a EZ I^{2}C Slave

This component is best used when a shared memory model between the I^{2}C Slave and I^{2}C Master is desired. The EZ I^{2}C Slave buffer/s may be defined as any variable, array, or structure in the user's code without any thought of the I^{2}C protocol. The I^{2}C master may view any of the variables in this buffer and modify the variables defined by the SetBuffer1 or SetBuffer2 function.

PRELIMINARY

Input/Output Connections

This section describes the various input and output connections for EZ I^{2}C Slave.

SDA – In/Out

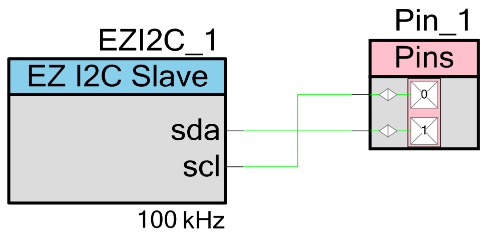
This is the I^{2}C data signal. It is a bi-directional data signal used to transmit or receive all bus data. This pin should be configured as Open-Drain, Drive Low.

SCL – In/Out

The SCL signal is the master generated I^{2}C clock. Although the slave never generates the clock signal, it may hold it low until it is ready to NAK or ACK the latest data or address. This pin should be configured as Open-Drain, Drive Low.

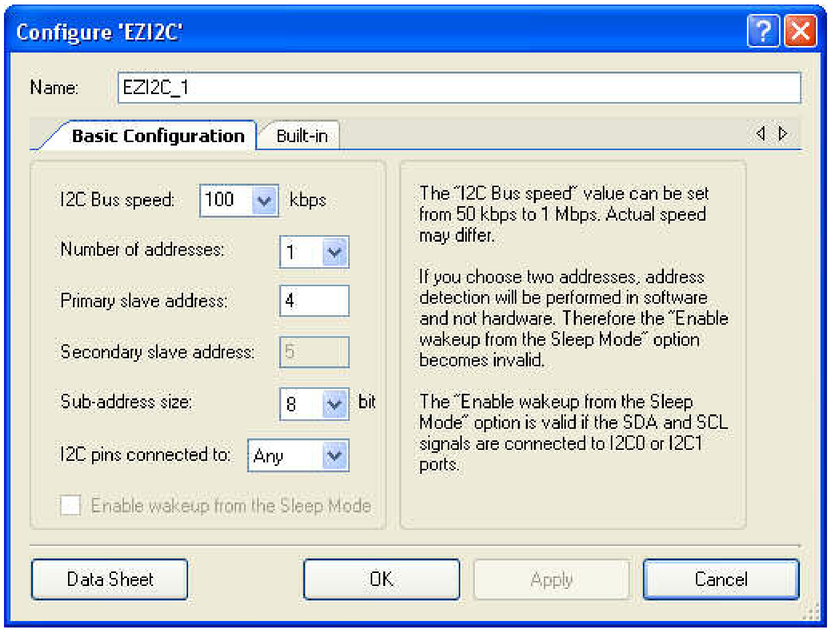
Schematic Macro Information

The default EZ I^{2}C Slave in the Component Catalog is a schematic macro using an EZ I^{2}C Slave component with default settings. The EZ I^{2}C Slave component is connected to a Pins component, which is configured as an SIO pair.



Parameters and Setup

Drag an EZ I^{2}C component onto your design and double-click it to open the Configure dialog.



Name: EZI2C_1

I2C Bus speed: 100 kbps

Number of addresses: 1

Primary slave address: 4

Secondary slave address: 5

Sub-address size: 8 bit

I2C pins connected to: Any

Enable wakeup from the Sleep Mode

The "I2C Bus speed" value can be set from 50 kbps to 1 Mbps. Actual speed may differ.

If you choose two addresses, address detection will be performed in software and not hardware. Therefore the "Enable wakeup from the Sleep Mode" option becomes invalid.

The "Enable wakeup from the Sleep Mode" option is valid if the SDA and SCL signals are connected to I2C0 or I2C1 ports.

The EZ I^{2}C component provides the following parameters.

I^{2}C Bus Speed

This parameter is used to set the I^{2}C bus speed value from 50 kbps to 1 Mbps; the actual speed may differ. The standard speeds are 50 kbps, 100 kbps (default), 400 kbps, and 1000 kbps. This speed is referenced from the system bus clock.

Number of Addresses

This option determines if 1 (default) or 2 independent I^{2}C slave addresses are recognized. If two addresses are recognized, address detection will be performed in software and not hardware, therefore the "Wake up from the low power modes" option becomes invalid.

Primary Slave Address

This is the primary I^{2}C slave address (default is 4). This value can be entered in decimal and hexadecimal format (the "0x" should be typed before the number).

Secondary Slave Address

This is the secondary I^{2}C slave address (default is 5). This value can be entered in decimal and hexadecimal format (the "0x" should be typed before the number). This second address is only valid when the **Number of Addresses** parameter is set to 2. The primary and secondary slave addresses must be different.

Sub-address Size

This option determines what range of data can be accessed. A sub-address of 8 (default) or 16 bits may be selected. If an address size of 8 bits is used, the master may only access data offsets between 0 and 254. You may also select a sub-address size of 16 bits. That will allow the I^2C master to access data arrays of up to 65,535 bytes at each slave address.

I2C pins connected to

This option allows configuring what pins (Any, I2C0 or I2C1) I2C bus is connected to. The EZ I^2C component is able to wake up device from the Sleep mode on slave address match only if bus pins are connected to I2C0 or I2C1 ports.

Enable wakeup from the Sleep mode

This parameter allows the device to be woken up from Sleep mode on slave address match. This option is disabled by default. Wake up on address match option is valid if a single I^2C address is selected and the SDA and SCL signals are connected to SIO ports (I2C0 or I2C1).

Consult device errata for this feature availability. The possibility of EZ I^2C to wake up device on slave address match should be enabled while switching to the sleep mode, refer to the "Power Management APIs" section of the *System Reference Guide*.

Clock Selection

The clock is tied to the system bus clock and cannot be changed by the user. The I^2C block supports 3, 6, 12, 24, 48, 96-102 MHz system bus clock for 100 kHz or 400 kHz operation in PSoC 3 ES2, ES3 and PSoC 5 ES1. Later device's revisions support all system bus clock frequencies. Consult device errata for more information.

Resources

The fixed I^2C block is used for this component.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "EZI2C_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "EZI2C".

Function	Description
void EZI2C_Init(void)	Initializes/restores default EZ I^2C configuration provided with customizer.
void EZI2C_Enable(void)	Enables component operation.
void EZI2C_Start(void)	Starts responding to I^2C traffic. (Enables interrupt)
void EZI2C_Stop(void)	Stops responding to I^2C traffic (Disables interrupt)
void EZI2C_EnableInt(void)	Enables interrupt, Start does this automatically.
void EZI2C_DisableInt(void)	Disables interrupt, Stop does this automatically.
void EZI2C_SetAddress1(uint8 addr)	Sets the I^2C primary address that it should respond.
uint8 EZI2C_GetAddress1(void)	Returns the I^2C address for the primary device.
void EZI2C_SetBuffer1(uint16 bufSize, uint16 rwBoundry, void * dataPtr);	Sets the buffer pointer for the primary address for both reads and writes.
uint8 EZI2C_GetActivity(void)	Checks status on device activity.
void EZI2C_SaveConfig(void)	Saves the current user configuration of the EZ I^2C component.
void EZI2C_RestoreConfig(void)	Restores the previously saved by EZI2C_SaveConfig() or EZI2C_Sleep() configuration of the EZI2C component.
void EZI2C_Sleep(void)	Saves component enable state and configuration. Stops component operation.
void EZI2C_Wakeup(void)	Restores component enable state and configuration. Should be called just after awaking from sleep.

Optional Second Address API

These commands are present only if two I^2C addresses are enabled.

Function	Description
void EZI2C_SetAddress2(uint8 addr)	Sets the I^2C secondary address that it should respond.
uint8 EZI2C_GetAddress2(void)	Returns the I^2C address for the secondary device.
void EZI2C_SetBuffer2(uint16 bufSize, uint16 rwBoundry, void * dataPtr);	Sets the buffer pointer for the secondary address for both reads and writes.

Optional Sleep/Wake modes

These functions are only available if a single address is used and the SCL and SDA signals are routed to the I^2C ports.

Function	Description
void EZI2C_SlaveSetSleepMode(void)	Disables the run time I^2C regulator and enables the sleep Slave I2C.
void EZI2C_SlaveSetWakeMode(void)	Disables the sleep EZ I^2C slave and re-enables the run time I^2C.

PRELIMINARY

Global Variables

Function	Description
EZI2C_initVar	Indicates whether the EZ I^2C has been initialized. The variable is initialized to 0 and set to 1 the first time EZI2C_Start() is called. This allows the component to restart without reinitialization in after the first call to the EZI2C_Start() routine. If reinitialization of the component is required the variable should be set to 0 before the EZI2C_Start() routine is called. Alternately, the EZ I^2C can be reinitialized by calling the EZI2C_Init() and EZI2C_Enable() functions.
EZI2C_dataPtrS1	Stores pointer to the data exposed to an I^2C master for the first slave address.
EZI2C_rwOffsetS1	Stores offset for read and write operations, is set at each write sequence of the first slave address.
EZI2C_rwIndexS1	Stores pointer to the next value to be read or written for the first slave address.
EZI2C_wrProtectS1	Stores offset where data is read only for the first slave address.
EZI2C_bufSizeS1	Stores size of data array exposed to an I^2C master for the first slave address.
EZI2C_dataPtrS2	Stores pointer to the data exposed to an I^2C master for the second slave address.
EZI2C_rwOffsetS2	Stores offset for read and write operations, is set at each write sequence of the second slave device.
EZI2C_rwIndexS2	Stores pointer to the next value to be read or written for the second slave address.
EZI2C_wrProtectS2	Stores offset where data is read only for the second slave address.
EZI2C_bufSizeS2	Stores size of data array exposed to an I^2C master for the second slave address.
EZI2C_curState	Stores current state of an I^2C state machine.
EZI2C_curStatus	Stores current status of the component.

void EZI2C_Init(void)

Description: Initializes/restores default EZI^2C configuration provided with customizer. Usually called in EZI2C_Start(). All changes applied by API to the component's configuration will be reset.

Parameters: None.

Return Value: None.

Side Effects: None.

void EZI2C_Enable(void)

Description: Enables the I2C block operation, sets interrupt priority, sets interrupt vector, clears ending interrupts and enables interrupts. Clears status variables and reset state machine variable.

Parameters: None.

Return Value: None.

Side Effects: None.

void EZI2C_Start(void)

Description:	Starts the component and enables the interrupt. If this function is called at first (or EZI2C_initVar was cleared, then EZI2C_Init() function is called and all offsets and pointers are reset. Anyway, the state machine state is set to IDLE, status variable is cleared and the interrupt is enabled.
Parameters:	None.
Return Value:	None.
Side Effects:	This component automatically enables its interrupt. If I^{2}C is enabled without the interrupt enabled, it could lock up the I^{2}C bus.

void EZI2C_Stop(void)

Description:	Disables the I^{2}C block's slave operation and the corresponding interrupt.
Parameters:	None.
Return Value:	None.
Side Effects:	None.

void EZI2C_EnableInt(void)

Description:	Enables the interrupt service routine This is normally handled with the Start() command.
Parameters:	None.
Return Value:	None.
Side Effects:	None.

void EZI2C_DisableInt(void)

Description:	Disables I^{2}C interrupts. Normally this function is not required since the Stop function disables the interrupt. If the I^{2}C interrupt is disabled while the I^{2}C master is still running, it may cause the I^{2}C bus to lock up.
Parameters:	None.
Return Value:	None.
Side Effects:	If the I^{2}C interrupt is disabled and the master is addressing the current slave, the bus will be locked until the interrupt is re-enabled.

void EZI2C_SetAddress1(uint8 address)

Description:	This function sets the main address of this I^{2}C slave device. This value may be any value between 0 and 127.
Parameters:	address: The 7-bit slave address between 0 and 127.
Return Value:	None.
Side Effects:	None.

uint8 EZI2C_GetAddress1(void)

Description: Returns the I^{2}C slave address for the primary device.

Parameters: None.

Return Value: The same I^{2}C slave address set by SetAddress1 or the default I^{2}C address.

Side Effects: None.

void EZI2C_SetBuffer1(uint16 bufSize, uint16 rwBoundry, void * dataPtr)

Description: This function sets the buffer pointer, size and read/write area for the slave data. This is the data that is exposed to the I^{2}C Master.

Parameters: bufSize: Size of the buffer in bytes.

rwBoundry: Sets how many bytes are writable in the beginning of the buffer. This value must be less than or equal to the buffer size.

dataPtr: Pointer to the data buffer.

Return Value: None

Side Effects: None.

uint8 EZI2C_GetActivity(void)

Description: This function returns a nonzero value if the I2C read or write cycle occurred since the last time this function was called. The activity flag resets to zero at the end of this function call. The Read and Write busy flags are cleared when read, but the "BUSY" flag is only cleared by an I2C Stop.

Parameters: A non-zero value is returned if activity is detected.

Return Value: Status of I^{2}C activity.

Constant	Description
EZI2C_STATUS_READ1	Set if Read sequence is detected for first address. Cleared when status read.
EZI2C_STATUS_WRITE1	Set if Write sequence is detected for first address. Cleared when status read.
EZI2C_STATUS_READ2	Set if Read sequence is detected for second address (if enabled). Cleared when status read.
EZI2C_STATUS_WRITE2	Set if Write sequence is detected for second address (if enabled). Cleared when status read.
EZI2C_STATUS_BUSY	Set if Start detected, cleared when stop detected.
EZI2C_STATUS_ERR	Set when I^{2}C hardware detected, cleared when status read.

Side Effects: None.

PRELIMINARY

void EZI2C_SetAddress2(uint8 address)

Description: Sets the I^2C slave address for the second device. This value may be any value between 0 and 127. This function is only provided if two I^2C addresses have been selected in the user parameters.

Parameters: address: The 7-bit slave address between 0 and 127.

Return Value: None.

Side Effects: None.

uint8 EZI2C_GetAddress2(void)

Description: Returns the I^2C slave address for the second device. This function is only provided if two I^2C addresses have been selected in the user parameters.

Parameters: None.

Return Value: The same I^2C slave address set by SetAddress2 or the default I^2C address.

Side Effects: None.

void EZI2C_SetBuffer2(uint16 bufSize, uint16 rwBoundry, void * dataPtr)

Description: This function sets the buffer pointer, size and read/write area for the second slave data. This is the data that is exposed to the I^2C Master for the second I^2C address. This function is only provided if two I^2C addresses have been selected in the user parameters.

Parameters: bufSize: Size of the buffer exposed to the I^2C master.

rwBoundry: Sets how many bytes are readable and writable by the the I^2C master.
This value must be less than or equal to the buffer size. Data located at offset rwBoundry and above are read only.

dataPtr: This is a pointer to the data array or structure that is used for the I^2C data buffer.

Return Value: None.

Side Effects: None.

void EZI2C_SlaveSetSleepMode(void)

Description: Disables the run time I^2C regulator and enables the sleep Slave I^2C. Should be called just prior to entering sleep. This function is only provided if a single I^2C address is used.

Parameters: None.

Return Value: None.

Side Effects: None.

void EZI2C_SlaveSetWakeMode(void)

Description:	Disables the sleep Ez I^2C slave and re-enables the run time I^2C. Should be called just after awaking from sleep. Must preserve address to continue. This function is only provided if a single I^2C address is used.
Parameters:	None.
Return Value:	None.
Side Effects:	None.

void EZI2C_Sleep(void)

Description:	Saves component enable state and configuration. Stops component operation. Should be called just prior to entering sleep. If "Enable wakeup from the Sleep mode" is properly configured and enabled, this function should not be called.
Parameters:	None.
Return Value:	None.
Side Effects:	None.

void EZI2C_Wakeup(void)

Description:	Restores component enable state and configuration. Should be called just after awaking from sleep.
Parameters:	None.
Return Value:	None.
Side Effects:	Calling this function before EZI2C_SaveConfig() or EZI2C_Sleep() will lead to unpredictable results.

void EZI2C_SaveConfig(void)

Description:	Saves the current user configuration of the EZ I^2C component.
Parameters:	None.
Return Value:	None.
Side Effects:	None.

void EZI2C_RestoreConfig(void)

Description:	Restores the previously saved by EZI2C_SaveConfig() or EZI2C_Sleep() configuration of the EZ I^2C component.
Parameters:	None.
Return Value:	None.
Side Effects:	Calling this function before EZI2C_SaveConfig() or EZI2C_Sleep() will lead to unpredictable results.

PRELIMINARY

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the EZ I^{2}C component. This example assumes the component has been placed in a design with the default name "EZI2C_1".

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
/*******************************************************************************
*  Example code to demonstrate the use of the EZ I2C
*
*  This example enables two Slave addresses. The buffer for
*  the first is set to the structure MyI2C_Regs and the
*  buffer for the second address is set to the constant
*  string DESC. The slave addresses for buffer1 and buffer2
*  are set to 6 and 7 respectively.
*
*  Parameter Settings:
*       BusSpeed_kHz:  400
*       EnableWakeup:  false
*       I2C_Address1:  4  (Does not matter since program resets to 6)
*       I2C_Address2:  5  (Does not matter since program resets to 7)
*       I2C_Addresses: 2
*       Sub_Address_Size:  Width_8_bits
*
*******************************************************************************/

#include <device.h> /* Part specific constants and macros */

typedef struct _EZ2C_REGS
{
    uint8 stat;             /* R/W variable */
    uint8 cmd;              /* R/W variable */
    int16 volts;            /* R/W variable */
    uint8 str[6];     /* Read only to I2C */
} EZ2C_REGS;

EZ2C_REGS myRegs;

const char desc[] = "Hello I2C Master";

void main()
{
    /* Enable global interrupts */
    CYGlobalIntEnable;

    /* Turn on EZI2C */
    EZI2C_1_Start();

    /* Set up Buffer1 */
    EZI2C_1_SetBuffer1(sizeof(myRegs), 4, (void *) &myRegs);

    /* Set up buffer2 */
    EZI2C_1_SetBuffer2(sizeof(desc), 10, (void *) &desc);
```

PRELIMINARY

```
        /* Change address1 to 6 */
        EZI2C_1_SetAddress1(6);

        /* Change address2 to 7 */
        EZI2C_1_SetAddress2(7);

        while(1)
        {
          /* Place user code here to update and read structure data. */
        }
    }
```

Functional Description

This component supports only an I^{2}C slave configuration with one or two I^{2}C addresses. The addresses are right justified.

This component requires that you enable global interrupts since the I^{2}C hardware is interrupt driven. Even though this component requires interrupts, you do not need to add any code to the ISR (Interrupt Service Routine). The module services all interrupts (data transfers) independent of your code. The memory buffers allocated for this interface look like simple dual port memory between your application and the I^{2}C Master.

If required, you can create a higher level interface between a master and this slave by defining semaphores and command locations in the data structure.

Memory Interface

To an I^{2}C master the interface looks very similar to a common I^{2}C EEPROM. The EZ I^{2}C API is treated as RAM or FLASH that can be configured as simple variables, arrays, or structures. In a sense it acts as a shared memory interface between your program and an I^{2}C master on the I^{2}C bus. The API allows the user to expose any data structure to an I^{2}C Master. The component only allows the I^{2}C master to access the specified area of memory and prevents any reads or writes outside that area. The data exposed to the I^{2}C interface can be a single variable, an array of values, or a structure. All that is required is a pointer to the start of the variable or data structure when initialized. The interface to the internal processor or I^{2}C master is identical for both slave addresses. For example, if the buffer for the primary slave address is configured as follows:

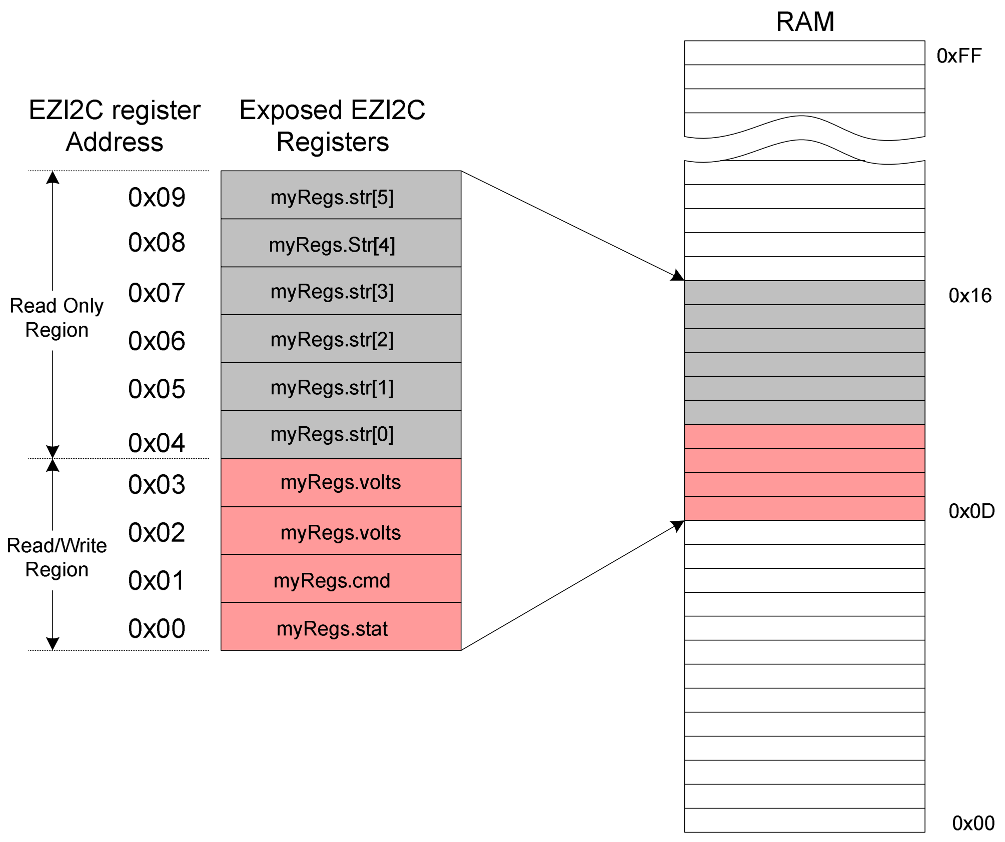
```
typedef struct _EZ2C_REGS
{
    uint8 stat;            /* R/W variable */
    uint8 cmd;             /* R/W variable */
    int16 volts;           /* R/W variable */
    uint8 str[6];          /* Read only to I2C */
} EZ2C_REGS;

EZ2C_REGS myRegs;
EZI2C_SetBuffer1(sizeof(myRegs), 4, (void *) &myRegs);
```

PRELIMINARY

The buffer representation in memory could be represented as shown in the following diagram:

Figure 1 Memory representation of the EZI2C buffer exposed to an I2C Master



This structure may contain any group of variables with any name as long as it is contiguous in memory and referenced by a pointer. The interface (I^2C Master) only sees it as an array of bytes, and cannot access any memory outside the defined area. Using the example structure above, a supplied API is used to expose the data structure to the I^2C interface.

```
EZI2C_SetBuffer1(sizeof(myRegs), 4, (void *) &myRegs);
```

The first parameter sets the size of the exposed memory to the I^2C interface. In this case, it is the entire structure. The second parameter sets the boundary between the read/write and read only areas by setting the number of bytes in the read/write area. The read/write area is first, followed by the read only area. In this case, only the first 4 bytes may be written, but all bytes may be read by the I^2C master. The third parameter is a pointer to the data.

In the following example a 15-byte array is created and exposed to the I^2C interface. The first 8 bytes of the array are read/write, and the remaining 7 (15-8) bytes are read only.

```
char theArray[15];
EZI2C_SetBuffer2(15, 8, (void *) theArray);
```

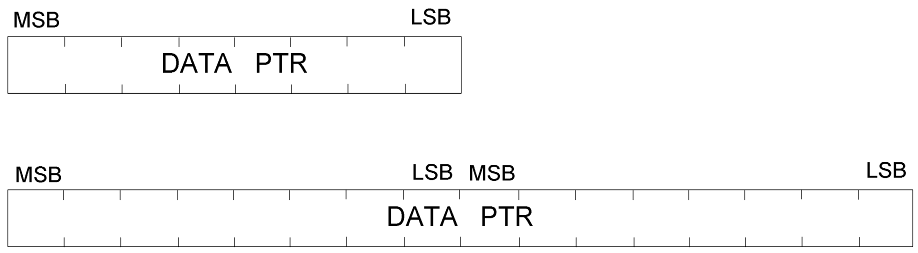
The following simple example shows only a single integer (2 bytes) is exposed. Both bytes are readable and writable by the I^{2}C master.

```
uint16 myVar;
EZI2C_SetBuffer1(2, 2, (void *) (&myVar));
```

Interface as Seen by External Master

The EZ I^{2}C Slave component supports basic read and write operations for the RAM area and read only operations for the FLASH area. The two buffer area interfaces contain separate data pointers that are set with the first one or two data bytes of a write operation, depending on the sub address size (Sub_Address_Size parameter). For the rest of this discussion, we will concentrate on an 8-bit sub address size. For the 16-bit sub address size bus communication will the same, but the data address field will have 16-bit length, not 8-bit.

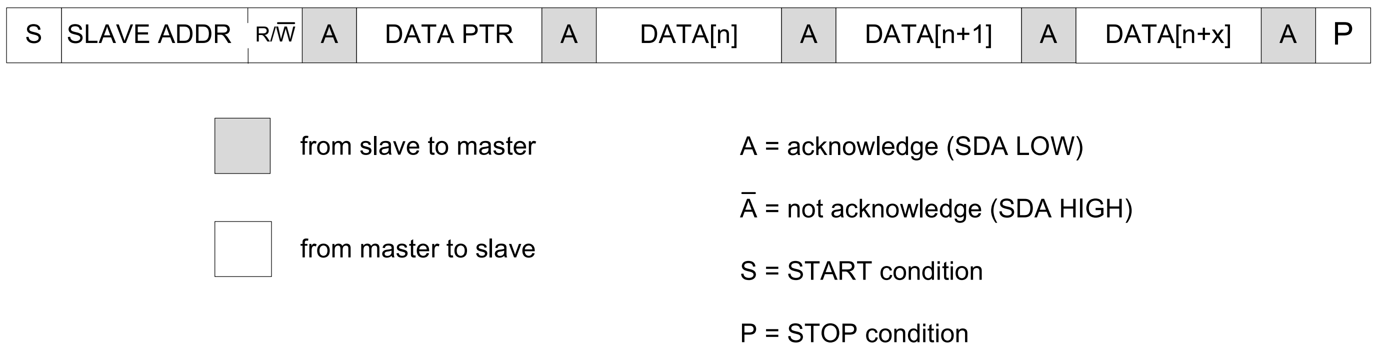
Figure 2 The 8 bits and 16 bits sub address size (from top to bottom)



When writing one or more bytes, the first data byte is always the data pointer. The byte after the data pointer is written into the location pointed to by the data pointer byte. The second data byte is written to the data pointer plus one and so on. This data pointer increments for each byte read or written, but is reset to the first value written at the beginning of each new read operation. A new read operation begins to read data at the location pointed to by the data pointer.

The following diagram illustrates the bus communication for an 8-bit data write, data pointer write, and a data read operation. Remember that a data write operation always rewrites the data pointer.

Figure 3 Write x bytes to I^{2}C slave



PRELIMINARY

For example, if the data pointer is set to four, a read operation begins to read data at location four and continue sequentially until the end of the data or the host completes the read operation. For example, if the data pointer is set to four, each read operation resets the data pointer to four and reads sequentially from that location. This is true whether a single or multiple read operations are performed. The data pointer is not changed until a new write operation is initiated.

If the I^{2}C master attempts to write data past the area specified by the SetBuffer1() function, the data is discarded and does not affect any RAM inside or outside the designated RAM area. Data cannot be read outside the allowed range. Any read requests by the master, outside the allowed range results in the return of invalid data.

Figure 4 Illustrates the data pointer write for and 8 Bit data Pointer.

Figure 4 Set slave data pointer

S	SLAVE ADDR	R/$\overline{W}$	A	DATA PTR	A	P

Figure 5 illustrates the read operation for 8 Bit data pointer. Remember that a data write operation always rewrites the data pointer.

Figure 5 Read x bytes from I^{2}C slave

S	SLAVE ADDR	R/$\overline{W}$	A	DATA[n]	A	DATA[n+1]	A	DATA[n+x]	$\overline{A}$	P

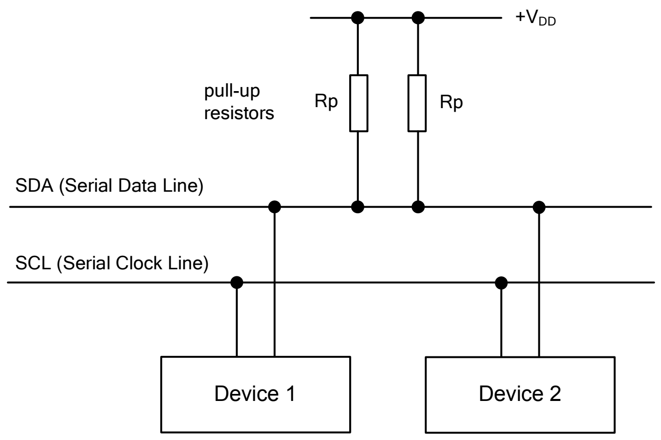
At reset, or power on, the EZ I^{2}C Slave component is configured and APIs are supplied, but the resource must be explicitly turned on using the EZI2C_Start() function.

Detailed descriptions of the I^{2}C bus and the implementation here are available in the complete I^{2}C specification available on the Philips web site, and by referring to the device data sheet supplied with PSoC Creator.

External Electrical Connections

As the block diagram illustrates, the I^{2}C bus requires external pull up resistors. The pull up resistors (RP) are determined by the supply voltage, clock speed, and bus capacitance. Make the minimum sink current for any device (master or slave) no less than 3 mA at VOLmax = 0.4V for the output stage. This limits the minimum pull up resistor value for a 5V system to about 1.5 kΩ. The maximum value for RP depends upon the bus capacitance and clock speed. For a 5V system with a bus capacitance of 150 pF, the pull-up resistors are no larger than 6 kΩ. For more information on "The I^{2}C -Bus Specification", see the Philips web site at www.philips.com.

Figure 6 Connection of devices to the I^{2}C-bus



Note Purchase of I^{2}C components from Cypress or one of its sublicensed Associated Companies, conveys a license under the Philips I^{2}C Patent Rights to use these components in an I^{2}C system, provided that the system conforms to the I^{2}C Standard Specification as defined by Philips.

Interrupt Service Routine

The interrupt service routine is used by the component code itself and should not be modified by the user.

Component Debug Window

The EZ I^{2}C component supports the PSoC Creator component debug window. Refer to the appropriate device data sheet for a detailed description of each register. The following registers are displayed in the EZ I^{2}C component debug window.

Register:	EZI2C_XCFG
Name:	Extended Configuration Register
Description:	Used to configure some of the advanced configuration options of the fixed function block.

Register:	EZI2C_ADDR
Name:	Slave Address Register
Description:	Used to indicate the 7-bit slave address for hardware address match detection.

Register:	EZI2C_CFG
Name:	Configuration Register
Description:	Used to configure the standard configuration options.

PRELIMINARY

Register:	EZI2C_CSR
Name:	Status Register
Description:	For the Fixed Function block this register is the status feedback register from hardware and includes some run-time control bits as a shared register.

Register:	EZI2C_DATA
Name:	Transmit and Receive Data Register
Description:	Used to load transmit data and read received data.

References

Not applicable

DC and AC Electrical Characteristics

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		67	MHz	

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.50.a	Moved component into subfolders of the component catalog	
1.50	Standard data rate has been updated to support up to 1 Mbps.	Allows setting up I^2C bus speed up to 1 Mbps.
	Keil reentrancy support was added.	Support for PSoC 3 with the Keil compiler the capability for functions to be called from multiple flows of control.

Version	Description of Changes	Reason for Changes / Impact
	Added Sleep/Wakeup and Init/Enable APIs.	To support low power modes, as well as to provide common interfaces to separate control of initialization and enabling of most components.
	The XML description of the component has been added.	This allows for the PSoC Creator to provide a mechanism for creating new debugger tool windows for this component.
	Added support for the PSoC 3 ES3 devices.	The required changes have been applied to support hardware changes between PSoC 3 ES2 and ES3 devices.
	The default schematic template has been added to the component catalog.	Every component should have a schematic template.
	The EZ I^{2}C's bus speed generation was fixed. Previously it was x4 greater than should be. Added more comments in the source code to describe bus speed calculation.	The proper I^{2}C bus speed calculation and generation.
	Optimized form height for Microsoft Windows 7.	In Windows 7 scrollbar appeared just after customizer start.
	Added tooltips for address input boxes with 'Use 0x prefix for hexadecimals' text.	To inform user about possibility of hexadecimal input.
1.20.a	Moved component into subfolders of the component catalog.	
	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device
1.20	The Configure dialog was updated.	
	Digital Port was changed to Pins component in the schematic	