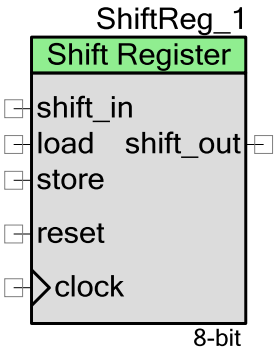
Shift Register (ShiftReg)

1.50

Features



- Adjustable shift register size: 2 to 32 bits
- Simultaneous shift in and shift out
- Right shift or left shift
- Reset input forces shift register to all 0s
- Shift register value readable by CPU or DMA
- Shift register value writable by CPU or DMA

General Description

The Shift Register (ShiftReg) component provides synchronous shifting of data into and out of a parallel register. The parallel register can be read or written to by the CPU or DMA. The Shift Register component provides universal functionality similar to standard 74xxx series logic shift registers including: 74164, 74165, 74166, 74194, 74299, 74595 and 74597. In most applications the Shift Register component will be used in conjunction with other components and logic to create higher level application-specific functionality, such as a counter to count the number of bits shifted.

In general usage, the Shift Register component functions as a 2- to 32-bit shift register that shifts data on the rising edge of the clock input. The shift direction is configurable and allows a right shift where the MSB shifts in the input and the LSB shifts out the output, or a left shift where the LSB shifts in the input and the MSB shifts out the output.

The Shift Register value may be written by the CPU or DMA at any time. A rising edge on the optional load input transfers pending FIFO data (previously written by the CPU or DMA) to the Shift Register. A rising edge on the optional store input transfers the current Shift Register value to the FIFO where it can later be read by the CPU.

The Shift Register component may generate an interrupt signal on any combination of the following signals: load, store or reset.

When to use a Shift Register

One of the most common uses of a shift register is to convert between serial and parallel interfaces. This is useful as many circuits work on groups of bits in parallel, but serial interfaces are simpler to construct.

The shift register can also be used as a simple delay circuit. In most cases the shift register will require additional application specific circuitry to function as desired. An example is a counter or state machine to store the shifted data after a number of events has occurred.

A common use of shift registers is to shift in or out eight bits of data based on a clock, as is done in the SPI protocol. If you are building a communication protocol, check to see if there is an existing higher level component for that communication protocol already.

Input/Output Connections

This section describes the various input and output connections for Shift Register. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

shift_in – Input *

Serial data input to the Shift Register MSB or LSB depending on shift direction. This terminal is displayed if the **Use Shift In** check box is selected.

load – Input *

The load input signal triggers the transfer of pending input FIFO data (previously written to the FIFO by CPU or DMA) to the Shift Register. A transfer occurs on the first rising edge of the component clock when the load signal is set. The load input is asynchronous to the clock input. This terminal is displayed if the **Use Load** check box is selected.

store – Input *

The store input signal triggers the transfer of the current shift register value into the output FIFO. A transfer occurs on the first rising edge of the component clock after the store signal rising edge has been detected. The _ReadData API routine can then be used to read the data from the FIFO. The store input is asynchronous to the clock input (still synchronous to the system). This terminal is displayed if the **Use Store** check box is selected.

reset – Input

The reset input (active high) causes the entire Shift Register to be set to zeros. This input does not affect the contents of the FIFOs. The reset input is synchronous to the clock input.

clock – Input

Clock source for the component. In some configurations this signal acts as an enable rather than a clock.

shift_out – Output *

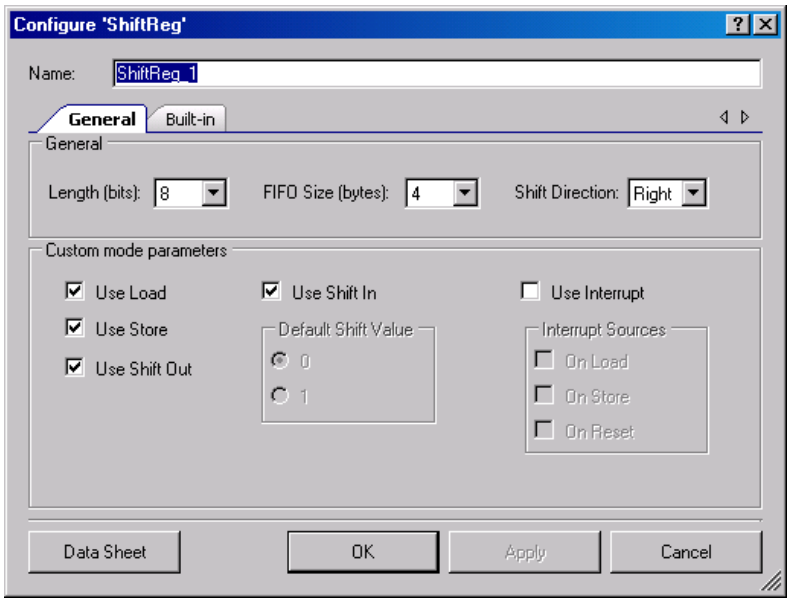
Outputs serial data from the Shift Register MSB or LSB based on shift direction. This terminal is displayed if the **Use Shift Out** check box is selected.

interrupt – Output *

Interrupt signal generated by the shift register component. Interrupts are generated based on the specified parameters. This terminal is displayed if the **Use Interrupt** check box is selected.

Parameters and Setup

Configure parameters by double-clicking the component to open the Configure ShiftReg dialog.



Length (bits)

This parameter determines the length of the shift register in bits. Valid values are 2 through 32 bits. The default is 8.

FIFO Size (bytes)

This parameter defines the number of shift register words the input and output FIFOs can hold. Choose either 1 or 4.

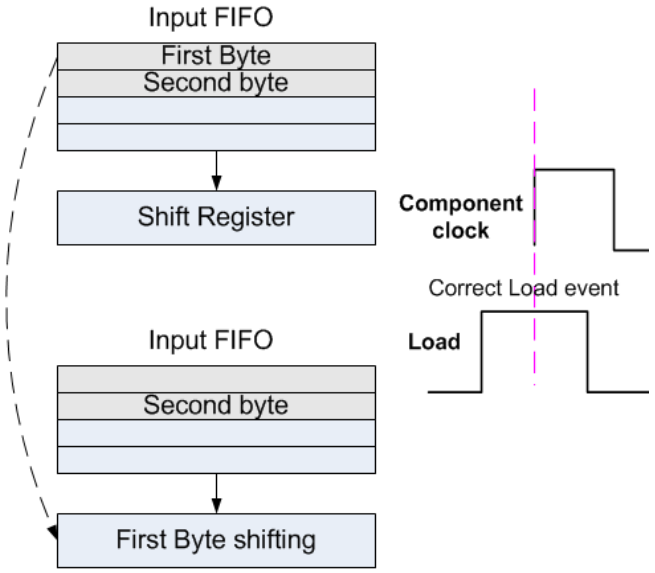
Shift Direction

This parameter determines the shift direction; Right or Left. The default is Right (LSB first).

Use Load

When this option is selected, the load input terminal is included on the Shift Register symbol. The load signal is internally routed to the control logic such that a word from the input FIFO is transferred to the Shift Register on a rising edge of the component clock and high level of the load signal.

Be careful with the load signal duration. When the load signal is asserted, the next FIFO word loads into the Shift Register on each rising edge of the component clock. If the load signal is held high for too long, multiple load events can occur. If the FIFO is empty when a load event occurs, arbitrary data is loaded into the Shift Register.



If **Use Load** is selected, the ShiftReg_WriteRegValue, ShiftReg_ReadRegValue, and ShiftReg_GetIntStatus APIs are generated for working with the output FIFO. The *component .h* file has the necessary API prototypes and #define constants.

If **Use Load** is not selected, the load terminal is not shown on the component symbol and the associated API routines are not generated.

Use Store

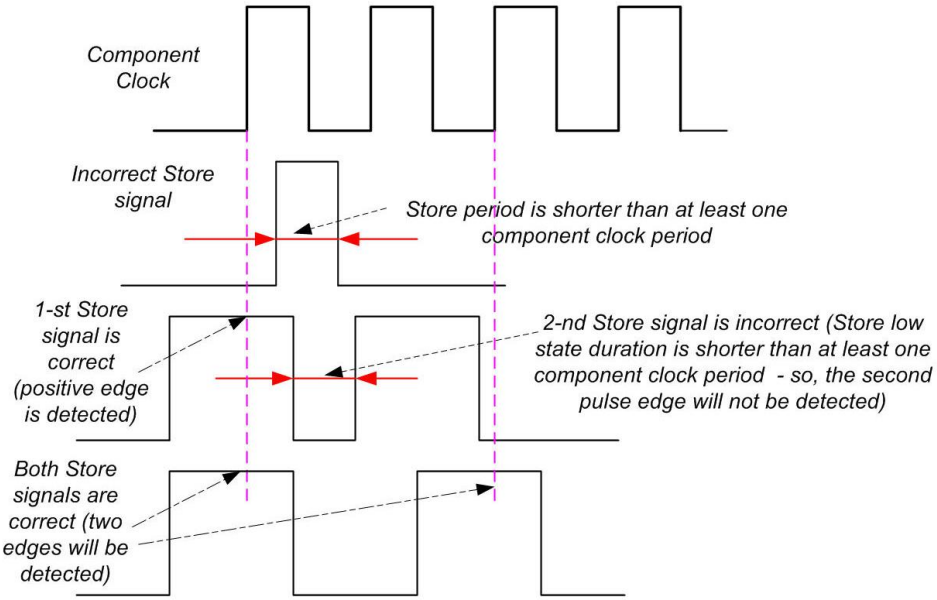
When this option is selected, the store input terminal is included on the Shift Register symbol. The store signal is internally routed to the control logic such that on a rising edge of the component clock after the store signal rising edge has been detected, the current word in the Shift Register is transferred to the output FIFO.

Notes

The store signal pulse duration should be longer than at least one component clock period.

The store signal should be low for at least one component clock period before the next store event.

Otherwise the store edge will not be detected.



If **Use Store** is selected, the ShiftReg_WriteRegValue, ShiftReg_ReadRegValue, and ShiftReg_GetIntStatus APIs are generated for working with the output FIFO. The *component .h* file has the necessary API prototypes and #define constants.

Caution Be careful if you use the ReadRegValue API routine in conjunction with the **Use Store** output FIFO functionality. The ReadRegValue API implementation transfers the current Shift Register ALU value into the output FIFO and then reads this data from the FIFO. Any data previously captured in the output FIFO using the Store signal, but not yet read by the application, will be lost.

If **Use Store** is not selected, the store terminal is not shown on the component symbol and the associated API routines are not generated.

Use Shift Out

This parameter determines if the shift_out output of the Shift Register symbol is provided. The default is true.

Use Shift In

This parameter determines if the shift_in input of the Shift Register symbol is provided. The default is true.

Default Shift Value

This parameter allows you to define a default value for the input to the Shift Register. This parameter is only used if the **Use Shift In** parameter is not checked. The valid values for the **Default Shift Value** parameter are 0 and 1.

Use Interrupt

If this parameter is selected, the interrupt output terminal displays on the symbol. This enables the use of interrupts generated by the Shift Register.

If **Use Interrupt** is not selected, the interrupt terminal is not shown on the symbol and the associated API routines will not be generated.

Interrupt Sources

This parameter becomes enabled if you select **Use Interrupt**. The interrupt signal is used to indicate that one of the specified conditions has occurred. You can enable or disable interrupt generation and specify the events that will trigger an interrupt: On Load, On Store or On Reset.

Clock Selection

Any signal can be used as the Shift Register component clock input. Data is shifted on the rising edge of the clock input signal.

Placement

The Shift Register component is implemented using UDB array resources. The necessary UDB resources are allocated by the tool placement algorithms.

Resources

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registe	Control Register	Counter 7	Flash	RAM	
8-Bits	1	2	1	1	0	498	4	6
16-Bits	2	2	1	1	0	561	6	6
24-Bits	3	2	1	1	0	631	8	6

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macrocells	Status Register	Control Register	Counter 7	Flash	RAM	
32-Bits	4	2	1	1	0	631	10	6

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 "ShiftReg_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 "ShiftReg".

Function	Description
ShiftReg_Init	Initializes or restores default Shift Register configuration
ShiftReg_Enable	Enables the Shift Register
ShiftReg_Start	Starts the Shift Register and enables all selected interrupts
ShiftReg_Stop	Disables the Shift Register
ShiftReg_EnableInt	Enables the Shift Register interrupt
ShiftReg_DisableInt	Disables the Shift Register interrupt
ShiftReg_SetIntMode	Sets the interrupt source for the interrupt.
ShiftReg_GetIntStatus	Gets the Shift Register interrupt status.
ShiftReg_WriteRegValue	Writes a value directly to the shift register
ShiftReg_ReadRegValue	Reads the current value from the shift register.
ShiftReg_WriteData	Writes data to the shift register input FIFO

ShiftReg_ReadData	Reads data from the shift register output FIFO.
ShiftReg_GetFIFOStatus	Returns current status of input or output FIFO.
ShiftReg_Sleep	Stops the component and saves all non-retention registers.
ShiftReg_Wakeup	Restores all non-retention registers and starts component
ShiftReg_SaveConfig	Saves configuration of Shift Register
ShiftReg_RestoreConfig	Restores configuration of Shift Register

Global Variables

Variable	Description
ShiftReg_initVar	Indicates whether the Shift Register has been initialized. The variable is initialized to 0 and set to 1 the first time ShiftReg_Start() is called. This allows the component to restart without reinitialization in after the first call to the ShiftReg_Start() routine. If reinitialization of the component is required, then the ShiftReg_Init() function can be called before the ShiftReg_Start() or ShiftReg_Enable() function.

void ShiftReg_Init(void)

Description: Initializes/restores default Shift Register interrupt sources.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_Enable(void)

Description: Enables the Shift Register.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_Start(void)

Description: Starts the Shift Register and enables all selected interrupts.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_Stop(void)

Description: Disables the Shift Register.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_EnableInt(void)

Description: Enables the Shift Register interrupts.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_DisableInt(void)

Description: Disables the Shift Register interrupts.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_SetIntMode(uint8 interruptSource)

Description: Sets the interrupt source for the interrupt. Multiple sources may be ORed together.

Parameters: (uint8)InterruptSource: Bitfield containing the constant for the selected interrupt sources. Multiple sources can be ORed together to select multiple interrupts.

Return Value	Description
ShiftReg_LOAD_INT_EN	Enables the Load interrupt.
ShiftReg_STORE_INT_EN	Enables the Store interrupt.
ShiftReg_RESET_INT_EN	Enables the Reset interrupt.

Return Value: None

Side Effects: None

uint8 ShiftReg_GetIntStatus (void)

Description: Gets the interrupt status for the Shift Register interrupts.

Parameters: None

Return Value: Bitfield containing the status for the selected interrupt source/s.

Return Value	Description
ShiftReg_LOAD	Load interrupt occurred.
ShiftReg_STORE	Store interrupt occurred.
ShiftReg_RESET	Reset interrupt occurred.

Side Effects: Clears the Interrupt Status register.

void ShiftReg_WriteRegValue (uint8/16/32 shiftData)

Description: Writes a value directly to the Shift Register.

Parameters: (unit8/16/32) shiftData: Data to be written. Data type is determined by the Shift Register Length parameter.

Return Value: None

Side Effects: The component must be stopped to use this API function.

Note The written value is available for reading after one component clock period.

uint8/16/32 ShiftReg_ReadRegValue(void)

Description: Returns the current value from the shift register.

Parameters: None

Return Value: (unit8/16/32) Shift Register value. Data type is determined by the Length parameter

Side Effects: Clears the shift register output FIFO. Wait at least one component clock period after calling WriteRegValue before calling this function.

Caution Be careful if you use the ReadRegValue API routine in conjunction with the **Use Store** output FIFO functionality. The ReadRegValue API implementation transfers the current Shift Register ALU value into the output FIFO and then reads this data from the FIFO. Any data previously captured in the output FIFO using the Store signal, but not yet read by the application, will be lost.

cystatus ShiftReg_WriteData(uint8/16/32 shiftData)

Description: Write data to the shift register input FIFO. A data word is transferred to the shift register on a rising edge of the load input

Parameters: (unit8/16/32) shiftData: Data to be written. Data type is determined by the Shift Register Length parameter.

Return Value: (cystatus) Returns an error if the FIFO is full or CYRET_SUCCESS on successful operation. If the input FIFO is full then the data will not be written to the FIFO.

Return Value	Description
CYRET_SUCCESS	Successful operation
CYRET_INVALID_STATE	Input FIFO is full

Side Effects: None

uint8/16/32 ShiftReg_ReadData(void)

Description: Read data from the shift register output FIFO. A data word is transferred to the output FIFO on a rising edge of the store input.

Parameters: None

Return Value: (unit8/16/32) next available data word. Data type is determined by the Shift Register Length parameter.

Side Effects: None

uint8 ShiftReg_GetFIFOStatus (uint8 fifoId)

Description: Returns the current status of the input or output FIFO.

Parameters: (uint8) FifoID: identifies which FIFO status is read.

FifoID Value	Description
ShiftReg_IN_FIFO	Used to read status of the input FIFO
ShiftReg_OUT_FIFO	Used to read status of the output FIFO

Return Value: (unit8) FIFO Status of one of defined values.

Return Value	Description
ShiftReg_RET_FIFO_FULL	FIFO is full
ShiftReg_RET_FIFO_NOT_FULL	FIFO is not full
ShiftReg_RET_FIFO_EMPTY	FIFO is empty

Side Effects: None

void ShiftReg_Sleep(void)

Description: Calls ShiftReg_Stop() and ShiftReg_SaveConfig() functions before entering sleep mode.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_Wakeup(void)

Description: Calls ShiftReg_Start() and ShiftReg_RestoreConfig() functions to return to the previous actual state after chip has been returned from sleep mode.

Parameters: None

Return Value: None

Side Effects: None

void I2S_SaveConfig(void)

Description: Saves the user configuration of Shift Register non-retention registers. Called by ShiftReg_Sleep routines to save the component configuration before entering sleep mode.

Parameters: None

Return Value: None

Side Effects: None

void I2S_RestoreConfig(void)

Description: Restores the user configuration of Shift Register non-retention registers. This routines is called by ShiftReg_Wakeup() to restore the component when exiting sleep mode.

Parameters: None

Return Value: None

Side Effects: Call this routine only after calling the ShiftReg_SaveConfig() function. Calling it independently of the ShiftReg_SaveConfig() function will overwrite the current settings with the initial settings.

Defines

- ShiftReg_SR_SIZE – Defines Shift Register length in bits.
- ShiftReg_USE_INPUT_FIFO – Indicates that an input FIFO is defined in the project.

  Note The output FIFO is always defined because it is used for Software Capture.
- ShiftReg_FIFOSize – Defines the size of the Input FIFO in Shift Register words. The Shift Register word size is determined by the **Length** (in bytes) parameter value.

- ShiftReg_DIRECTION – Defines the direction of the shift (0 – Left Shift , 1 – Right Shift).

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the Shift Register component. This example assumes the component has been placed in a design with the default name "ShiftReg_1" and “Length” parameter is set to 32.

The Character LCD (named “LCD”) is used to display Shift Register read data on 20x4 LCD glass. The control register (named “Control”) is used to generate custom component clock (for demonstration only. In real projects, the “Clock” component should be used along with additional control logic [counter etc.] to initiate correct data loading, shifting and capturing).

Note If you rename components which are used in the project you must also edit the example code as appropriate to match the component names you specify.

```
#include <device.h>

void main()
{
    uint8 i = 0;
    uint8 j = 0;
    uint8 control_value = 0;
    uint32 read_value   = 0;

    LCD_Start();
    ShiftReg_1_Start();
    ShiftReg_1_WriteRegValue(0x80808080);

    for(j=1;j<4;j++)
    {
        for(i=0;i<16;i+=9)
        {
            Control_Write(control_value);
            CyDelay(100);
            control_value|=(~control_value);
            Control_Write(control_value);

            read_value = ShiftReg_1_ReadRegValue();

            LCD_Position(j,i);
            LCD_PrintInt16((HI16(read_value)));
            LCD_Position(j,i+4);
            LCD_PrintInt16(LO16(read_value));

            control_value=(~control_value);
        }

        i=0;

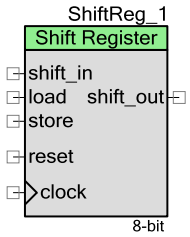
    }
}
```

Functional Description

The Shift Register has a number of parameters which allow for considerable flexibility in the configuration of the component. This section provides additional explanation of the Shift Register operation and how the parameters can be used to customize the component for your application. The Shift Register can be used stand alone, or in conjunction with other components to create application specific functionality.

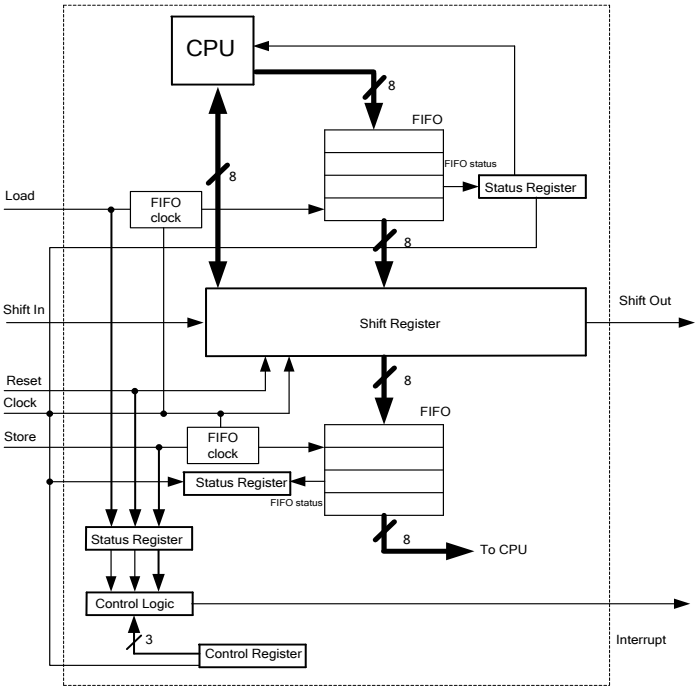
Default Configuration

The default configuration of the Shift Register component provides basic parallel shift register functionality similar to standard 7400 series logic shift registers. This functionality includes synchronous shifting of data into and out off a parallel register on the rising edge of the clock input. Serial bitstream data is shifted into the shift_in terminal and shifted from the shift_out output terminal.



Block Diagram and Configuration

The following is the Shift Register block diagram.



The Shift Register is a UDB-based component that consists of an input FIFO (F0), a direct shift register (A0 and A1 with duplicated value for providing the software capture), an output FIFO (F1), as well as control and status registers.

The input FIFO F0 is configured to input mode. This means that this FIFO can be written by the CPU (using the WriteData() API function) and this value can be loaded into the A0 register for shifting. This function checks the current FIFO status before each cycle using the GetFIFOStatus() function.

The value to be shifted can be also written directly to the A0 register by calling WriteRegValue(). Because of internal hardware implementation it is strongly recommended to stop component operation (by using Stop() function or stopping input clock) when using WriteRegValue(). Otherwise writing the A0 register during the shift operation will lead to incorrect data being written.

The Load operation has the hardware restriction (load event can be provided only when input FIFO is not empty).

To provide the shift functionality, the UDB datapath(s) is (are) used in the following configuration:

State == 100 (4)	Shift Operation (Left or Right)
State == 101 (5)	Reset (XOR A0 A0)
State == 110 (6)	Load A0 <=F0
State == 111 (7)	Reset (XOR A0 A0)

All operations except store are controlled from the datapath control store. Shift is a default operation (cs_addr = “000”). Load input is connected to the cs_addr[1] line and reset input - to cs_addr[0]. If some of these lines change their level it causes the control store address to change immediately. On the positive edge of the datapath clock (component clock - in this case), the corresponding operation will be executed. The load causes to loading value to change from F0 to A0. The reset command causes the clearing of A0. In this case, the load value is ignored.

To read the shifted value, two mechanisms are used: hardware and software capture. The hardware capture event happens on each positive edge on the store input. It causes the Shift Register value to be written to the output FIFO. This value can be read by the ReadData() API function. The store input has a hardware restriction (store input will be active only if output FIFO is not full).

Software capture happens each time the ReadRegValue() function is called. This function reads the A1 value where it duplicates the value of A0. This operation reduces to A1 value to be automatically written to the output FIFO F1 (because F1 is configured to software capture from A1). Before providing the software capture, the ReadRegData() function clears the output FIFO. Therefore, you should be careful when using it.

Note Using this function, the actual value in the A1 will be available in the next clock cycle after writing the Shift Register.

The interrupt generation mechanism is implemented using the status register. It has three bits which represent three interrupt sources: Load, Store, Reset. When one of these bits changes its value from 0 to 1, the interrupt pulse on the appropriate status register output is automatically generated. These three bits are in “Clear on read” mode.

The second status register is used for storing the current input and output FIFO's status. All status bits are in “Sticky” mode (are not cleared after reading).

When the Shift Register size is more than 8, the datapath's chaining connectivity is provided to connect 2, 3 or 4 datapaths between each other to implement component size 16, 24 or 32. To implement a Shift Register size that does not coincide with the datapath(s) measures, a verilog-controlled MSB is used into the datapath(s) configuration(s).

Component start/stop is realized using CLK_EN bit of control register.

Registers

ShiftReg_SR_CONTROL

Bits	7	6	5	4	3	2	1	0
Value	Reserved							clk_en

- clk_en : Enables Shift Register operation.

ShiftReg_SR_STATUS

Bits	7	6	5	4	3	2	1	0
Value		F1_not_empty	F1_full	F0_not_full	F0_empty	reset	store	load

- load: Load status bit.
- store: Store status bit.
- reset: Reset status bit.
- F0_empty: Input FIFO is empty.
- F0_not_full: Input FIFO is not full.
- F1_full: Output FIFO full
- F1_not_empty: Output FIFO is not empty.

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

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.c	Minor datasheet edit.	
1.50.b	Minor datasheet edit.	
1.50	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.
	Update the Configure dialog.	Changed locations of ‘Use Shift Out’ and ‘Use Shift' and changed default value of ‘Use interrupt’ check box to improve functionality.
	Changed the ShiftReg_ReadRegValue() implementation.	This provides faster Software Capture execution.

© Cypress Semiconductor Corporation, 2009-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.