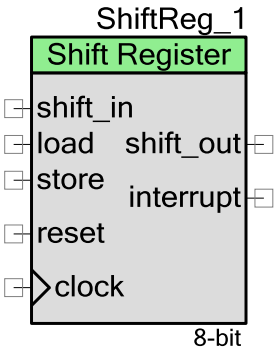
Shift Register

1.20

Features



- Adjustable shift register size: 1 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 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 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 functions as a 1-32 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 shift out the output, or a left shift were the LSB shifts in the input and the MSB shifts out the output.

The reset input (active high) causes the entire shift register contents to be set to all zeros. The reset input is synchronous to the clock input.

The shift register value may be read by the CPU or DMA at any time. A rising edge on the optional store input transfers the current shift register value to the FIFO from where it can later be read by the CPU. The store input is asynchronous to the clock input.

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 (already written by CPU or DMA) to the shift register. The load input is asynchronous to the clock input.

The Shift Register component may generate an interrupt signal on any combination of the following signals; Load, Store or Reset.

Various component API routines are generated based on the configuration specified for the component. APIs which support functionality that is not specified will not be generated.

PRELIMINARY

When to use a Shift Register

The shift register is intended to be a very low level building block with which to create more advanced functionality. 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 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. For SPI type uses, the separate SPI component is likely more useful. The SPI component already incorporates a bit counter and higher level functionality than that of the shift register component.

Input/Output Connections

This section describes the various input and output connections for Shift Register. An asterisk (*) in the list of I/O's states 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 rising edge of the load signal. The load input is synchronous 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 rising edge of the Store signal. The FIFO read API routine can 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

Resets shift register state to 0. 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 *

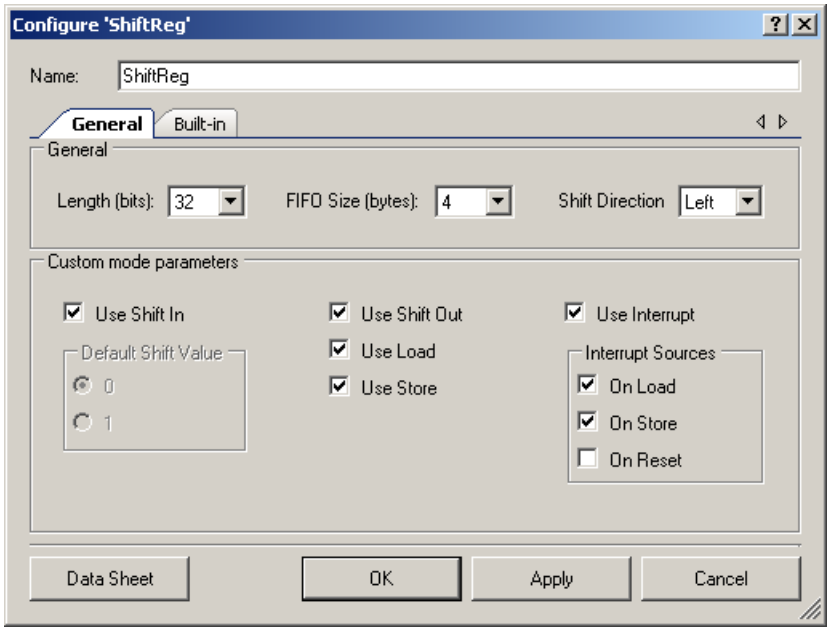
Outputs serial data from the Shift Register MSB or LSB based on shift direction. This terminal is displayed if the Use Shift Out check box is selected.

interrupt – Output *

Interrupt signal generated by the shift register component. Interrupts are generated based on the internal state machine meeting specified conditions. 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.



Hardware Configuration Options

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 set to false. 0 and 1 are valid values for this parameter.

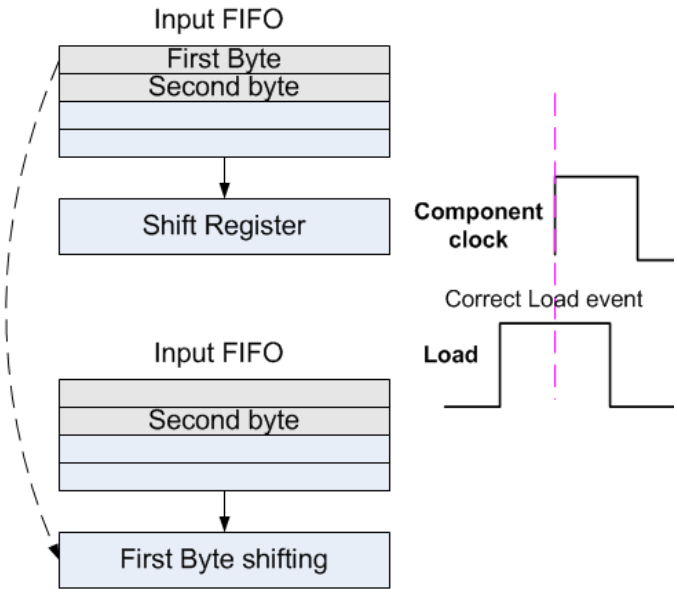
Use Shift Out

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

Use Load

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

Be careful with the load signal duration. If the load signal will be asserted more than one component clock period, on the next clock rising edge the next FIFO word loads into the shift register. Incorrect load signal duration can result in multiple load events. If the FIFO was already empty, some arbitrary data loads into the shift register.



Read, write and status query APIs for working with the input FIFO are generated when this selection is made. 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 functionality is selected the store input terminal is included on the Shift Register symbol. The store signal is internally routed to the datapath control logic such that on a rising edge of the store signal, the current Shift Register word in the shift register (datapath ALU) is transferred to the output FIFO. Note, that store signal should be asserted to low level at least on one component clock period before next store event; otherwise store edge will not be detected. Read, write and status query APIs for working with the output FIFO are generated when this selection is made. The component .h file has the necessary API prototypes and #define constants.

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

Caution: You need to be careful if you use the ReadRegValue API routine in conjunction with the UseStore output FIFO functionality. The ReadRegValue API implementation uses the datapath software capture mechanism to transfer 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.

Use Interrupt

This parameter enables the use of interrupts generated by the Shift Register. When this parameter is set to true (default), the interrupt output is included on the symbol. When this functionality is selected the interrupt output terminal is included on the Shift Register symbol. The interrupt signal is used to indicate that one of the specified Interrupt Source conditions has occurred. The user can enable or disable interrupt generation and specify the events that will trigger an interrupt: Load, Store or Reset. The interrupt functionality can be configured in the configuration GUI or at runtime using generated APIs.

If UseInterrupt is not selected, the interrupt terminal is not shown on the component symbol and the associated API routines are not generated.

Software Configuration Options

Length

This parameter determines the width/length of the shift register in bits. Valid values are 1 through 32 b. The default is 8.

Direction

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

FIFOSize

This parameter defines the number of shift register words the input and output FIFOs can hold. Size options are 1 and 4.

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 Registers	Control Registers	Counter7	Flash	RAM	
8-Bits	1	5	2	1	0	2422	487	?
16-Bits	2	5	2	1	0	2403	490	?
24-Bits	3	5	2	1	0	2486	496	?
32-Bits	4	5	2	1	0	2581	496	?

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_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 Shift Register interrupt. Multiple sources may be ORed together

Function	Description
ShiftReg_GetIntStatus	Gets the Shift Register Interrupt status.
ShiftReg_WriteRegValue	Write a value directly to the shift register
ShiftReg_ReadRegValue	Read the current value from the shift register.
ShiftReg_WriteData	Write data to the shift register input FIFO
ShiftReg_ReadData	Read data from the shift register output FIFO.
ShiftReg_GetFIFOStatus	Returns current status of input or output FIFO.

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.

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_DisableInt(void)

Description: Disables the Shift Register interrupt

Parameters: None

Return Value: None

Side Effects: None

void ShiftReg_SetIntMode(uint8 InterruptSource)

Description: Sets the Interrupt Source for the Shift Register 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 Load interrupt.
ShiftReg_STORE_INT_EN	Enables Store interrupt.
ShiftReg_RESET_INT_EN	Enables 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 Interrupt Status register.

void ShiftReg_WriteRegValue (uint8/16/32 TxDataByte)

Description: Writes a value directly to shift register

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

Return Value: None

Side Effects: Component must be stopped to use this API function.
Note: written value will be available for reading after at least 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 Shift Register Length parameter

Side Effects: Clears the shift register output FIFO. Function should be used at least after one component clock period after calling ShiftReg_WriteRegValue().

cystatus ShiftReg_WriteData(uint8/16/32 TxDataByte)

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) TxDataByte: 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 input FIFO is full then data will not be written to 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 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 fo 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

Defines

ShiftReg_SR_SIZE

Defines Shift Register length in bits. Specified by the user in the configuration GUI.

ShiftReg_USE_IN_FIFO

Indicates that an input FIFO is defined in the project.

ShiftReg_FIFOSize

Defines the size of the Input FIFO in Shift Register words. Shift Register word size is determined by the Shift Register Length (in bytes) parameter value.

ShiftReg_DIRECTION

Defines the direction of the 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."

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

```
#include <device.h>

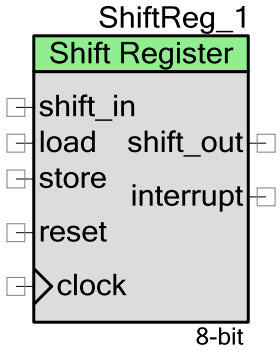
void main()
{
    ShiftReg_1_Start();
}
```

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.

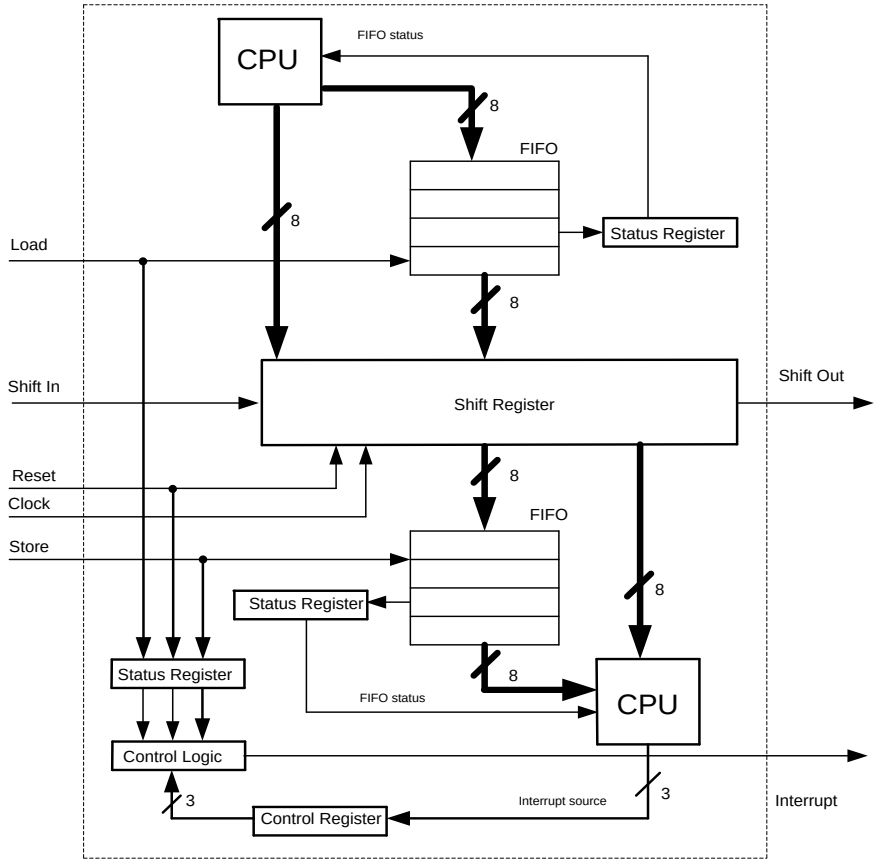
Default Configuration

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 which consists of input FIFO (F0), directly shift register (A0 and A1 with duplicated value for providing the Software Capture), output FIFO (F1) and also control and status registers.

Input FIFO F0 is configured to input mode. It means that this FIFO can be written by CPU (using WriteData() API function) and this value can be loaded into the A0 register for shifting. This function is checking current FIFO status before each writing cycle using GetFIFOStatus() function. Value to be shifted can be also written directly to A0 register by calling WriteRegValue(). Because of internal hardware implementation it's strongly recommended to stop component operation (by using Stop() function or stopping input clock) when you using WriteRegValue() function. Otherwise writing A0 register during the shift operation will lead to incorrect data to be written. Load operation has the hardware restriction (load event can be provided only when input FIFO is not empty).

To provide this functionality 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 datapath control store. Shift operation 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 to changing the control store address immediately and on the positive edge of datapath clock (component clock - in this case) correspond operation will be executed. As we can see, load causes to loading value from F0 to A0. Reset command causes the clearing A0. In this case load value is ignored.

To read shifted value two mechanisms are used. There are Software and Hardware Capture. Hardware Capture event happens on each positive edge on Store component input. It causes to writing Shift Register value to the output FIFO. This value can be read by the ReadData() API function. Store input has a hardware restriction (Store input will be active only if output FIFO is not full).

Software Capture happens on each calling the ReadRegValue() function. This function reads the A1 value where duplicated value of A0 is consisted. 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 ReadRegData() function clears the output FIFO so you should be very careful when using it. Note: Using this function actual value in the A1 will be available in the next clock cycle after writing Shift Register.

Interrupt generation mechanism is implemented using statusi 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 interrupt pulse on appropriate status register output is automatically generated. These three bits are in “Clear on read” mode.

Second status register is used for storing current input and output FIFOs status. All status bits are in “Sticky” mode (are not cleared after reading).

When Shift Register size is more than 8, datapaths chaining connectivity is provided to connect 2,3 or 4 datapaths between each other to implement component size 16, 24 or 32. To implement Shift Register size which is not coincides with datapath(s) measures 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	reserved					reset	store	load

reset: Reset status bit.

store: Store status bit.

load: Load status bit.

ShiftReg_SR_FIFO_STATUS

Bits	7	6	5	4	3	2	1	0
Value	Reserved				F1_not_empty	F1_full	F0_not_full	F0_empty

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	

Parameter	Typical	Min	Max	Units	Conditions and Notes
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
1.20.b	Minor datasheet edit.	
1.20.a	Minor datasheet edit.	
1.20	Option of selecting FIFO size is disabled when load and store are not used. Updated the Configure dialog. Removed generated code for unused parameters.	Various changes were made to fix issues with version 1.10, which was not fully functional.

© 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.