8-Bit Successive Approximation ADC Datasheet SAR8 V 1.0

Copyright © 2005-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O and Clock)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C24x33, CY8C23x33	0	0	0	147	0	1

See application note “Analog - ADC Selection” AN2239 for other converters.

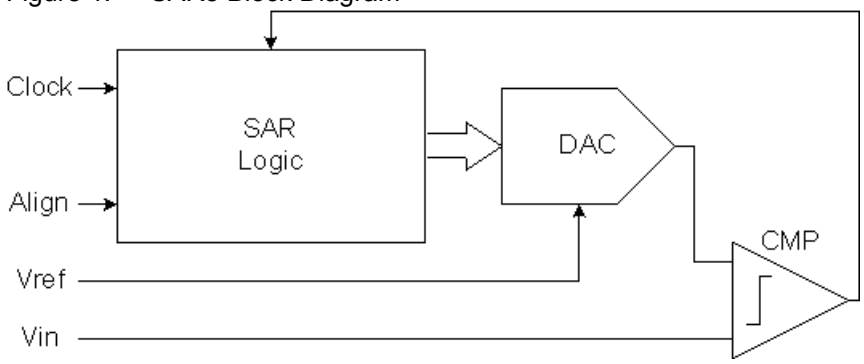
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- Best analog-to-digital conversion results on CY8C24x33 and CY8C23x33 devices
- Fast (typical 2.7 µs) conversion time
- Successive approximation functionality support
- 8-bit resolution
- Eight primary input analog channels and two internal analog channels from CT blocks
- Single conversion
- Free running conversion
- Selectable conversion trigger
- Programmable sample time
- Programmable clock divider
- Cancel/restart feature for running conversions
- Support to autoalign/trigger at any point of PWM, timer, or counter cycle
- Automatic entry into low power mode after every conversion in single conversion mode

The SAR8 User Module is an 8-bit successive approximation register (SAR) ADC converter that converts an input voltage to a digital code using a dedicated analog PSoC block. It features typical conversion times of 2.7 µs (limited with firmware processing) and produces an 8-bit unsigned value for each sample. In motor control applications you may need to trigger the ADC to execute one analog-to-digital conversion at a certain time during the PWM period in order to obtain best performance.

Figure 1. SAR8 Block Diagram



Functional Description

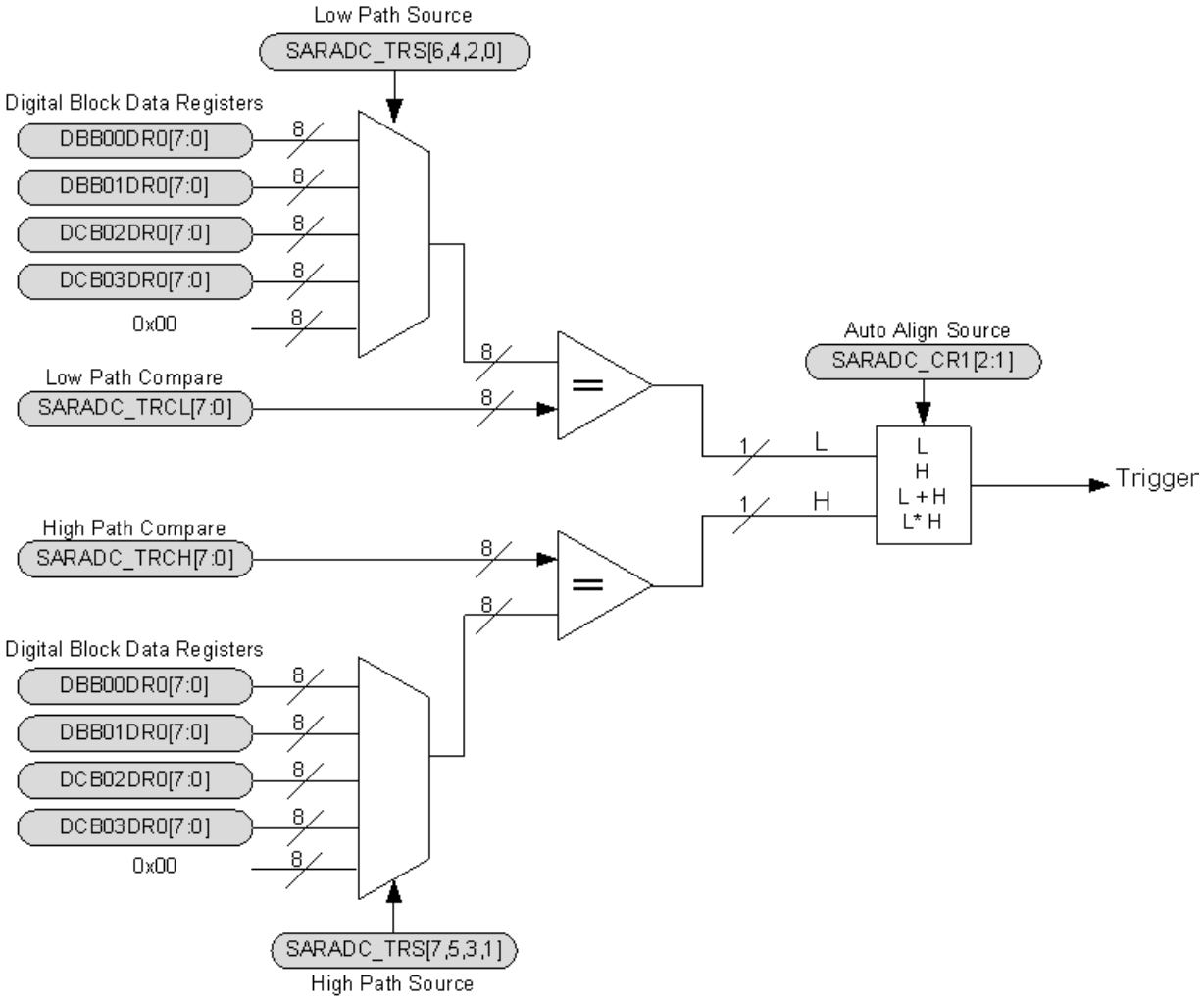
The successive approximation register is the PSoC SARADC_DL register. This register holds the result of the conversion. The SAR logic forms a series of approximations by scaling the reference voltage and comparing it to the input voltage. This performs a binary search to find the DAC setting that is closest to the input voltage.

A unique feature of the SAR8 is advanced conversion trigger logic that allows conversions to be synchronized with one or two digital blocks.

The SAR8 ADC can work in one-shot mode or in free-run mode. In both cases API functions allow you to initiate a new measurement even when the ADC is performing a conversion. When in one-shot mode, the ADC automatically goes into low power mode after conversion. When in free-run mode the ADC goes into low power mode if it is stopped.

The autoalign option allows you to trigger conversion requests with two 8-bit digital comparators (Low and High). These comparators can be driven by any of the PSoC device's digital blocks. The comparators compare the actual value of the block's DR0 register with the data stored in SAR8_COMPARE_LO_REG and SAR8_COMPARE_HI_REG registers. Each comparator can trigger the ADC independently or they can be combined to form a 16-bit trigger source.

Figure 2. Schematic of the SAR8 Autoalign



Another feature of the SAR8 ADC is data scaling. It allows you to right shift data that is read out from the result register. The scale factor is adjustable from 1 to 2^{-N}, N=0..6.

DC and AC Electrical Characteristics

The following table lists guaranteed maximum and minimum specifications for the voltage and temperature ranges: 4.75V to 5.25V and -40°C . TA . 85°C, or 3.0V to 3.6V and -40°C . TA . 85°C, respectively. Typical parameters apply to 5V and 3.3V at 25°C and are for design guidance only

Table 1. SAR8 DC and AC Electrical Characteristics

Parameter	Min	Typ	Max	Units	Conditions and Notes
$V_{ADCVREF}$	3.0[a]	-	Vdd	V	Reference voltage at pin P3[0] when configured as ADC reference voltage. $V_{ADCVREF} \leq$Vdd.
$I_{ADCVREF}$	-	-	3	mA	Current supplied to the SAR8 ADC when P3[0] is configured as the ADC V_{REF}
C_{IN}	-	-	60	pF	Input capacitance on ADC input
R_{IN}	-	-	1	kΩ	Input resistance on ADC input
INL[b]	-1.2	-	+1.2	LSB	R-2R Integral Non-linearity. The maximum LSB is over a sub-range not exceeding 1/16 of the full-scale range.
DNL[b]	-1	-	+1	LSB	R-2R Differential Non-linearity. Output is monotonic.
Freq	0.375	-	24	MHz	Input clock frequency

a. A lower $V_{ADCVREF}$ voltage can be used with a reduction in accuracy.

b. Applies to ADC clock frequencies of 1.5 Mhz or less.

Placement

There is a single dedicated resource to support the SAR8. It is the only placement choice.

Parameters and Resources

ADCInput

Any Port 0 pin can be selected as the signal source for the ADC. Analog blocks ACB00 and ACB01 can also be used as the signal source.

ADCClock

The clock rate can be selected from:

- SYSCLK
- SYSCLK/2
- SYSCLK/4
- SYSCLK/8
- SYSCLK/16
- SYSCLK/32
- SYSCLK/64

The ADC sample rate is the ADC clock divided by 8, and the conversion time is ADC clock period multiplied by 8.

The ADC automatically goes into low power mode after conversion, so for one-shot mode, using a lower speed ADC clock consumes more power than a higher speed ADC clock. In free run mode, using a lower speed ADC clock consumes less power than using a higher speed ADC clock.

Scale

Conversion results are automatically integer divided by the Scale factor when they are read out. Possible values for Scale are: 1, 2, 4, 8, 16, 32, 64.

Run

This parameter selects the One-Shot or Free-Run conversion mode.

ADCPower

The SAR8 block can obtain power from internal VPWR (Vdd) or from P3[0].

VPWRADC comparator block obtains power supply from internal VPWR (Vdd).**P3[0]**ADC comparator block obtains power supply from P3[0]. P3[0] must be less than or equal to VPWR (Vdd).

R2RPower

There are two possible selections.

VPWRADC DAC reference generation block obtains power supply from internal VPWR (Vdd). The ADC DAC power (R2RPower) must be less than or equal to that of the ADC comparator block (ADCPower). **P3[0]**ADC R2R reference generation block obtains power supply from P3[0]. P3[0] must be less than or equal to VPWR.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include" files.

Note

** In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Entry points are provided to initialize the SAR8 User Module, perform and read conversions, and disable the SAR8 function.

SAR8_Start

Description:

Enables ADC operation. The SAR8 automatically enters low power mode right after it finishes the conversion. If the Run parameter is set to Free-Run the ADC will begin collecting data after SAR8_Start is called. If the Run parameter is set to One-Shot then the SAR8_Trigger function must be called or a trigger signal from the Autoalign circuit must occur before a sample will be collected.

C Prototype:

```
void  SAR8_Start(void)
```

Assembler:

```
lcall  SAR8_Start
```

Parameters:

None

Returns:

None

Side Effects:

See Note ** at the beginning of the API section.

SAR8_Stop

Description:

Disables ADC operation. After Stop is called the ADC will go into low power mode.

C Prototype:

```
void  SAR8_Stop(void)
```

Assembler:

```
lcall  SAR8_Stop
```

Parameters:

None

Returns:

None

Side Effects:

See Note ** at the beginning of the API section.

SAR8_EnableInt

Description:

Enables interrupts. Interrupts must be enabled globally using the M8C_EnableGInt macro before the SAR8 interrupts can be used.

C Prototype:

```
void  SAR8_EnableInt(void)
```

Assembly:

```
lcall  SAR8_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SAR8_DisableInt

Description:

Disables interrupts.

C Prototype:

```
void SAR8_DisableInt(void)
```

Assembler:

```
lcall SAR8_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

See Note ** at the beginning of the API section.

SAR8_Trigger

Description:

Triggers the ADC to perform the sample and conversion from the next system clock cycle. If the ADC is already in converting a sample, a SAR8_Trigger forces the ADC to cancel the ongoing conversion and restart one new conversion on the next system clock cycle. If the ADC is working in free-run mode, this function will not influence the ADC sampling.

C Prototypes:

```
void  SAR8_Trigger(void)
```

Assembler:

```
lcall  SAR8_Trigger
```

Parameters:

None

Returns:

None

Side Effects:

See Note ** at the beginning of the API section.

SAR8_fIsDataAvailable

Description:

Checks the availability of sampled data.

C Prototype:

```
BYTE  SAR8_fIsDataAvailable(void)
```

Assembler:

```
lcall  SAR8_fIsDataAvailable
```

Parameters:

None

Returns:

Returns a non-zero value if data has been converted and is ready to read.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_bGetData

Description:

Returns converted data as an unsigned byte value. SAR8_fIsDataAvailable() should be called to verify that the data sample is ready.

C Prototype:

```
BYTE  SAR8_bGetData(void)
```

Assembler:

```
lcall  SAR8_bGetData
```

Parameters:

None

Returns:

Returns converted data as a unsigned byte value.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SelectADCChannel

Description:

Select the ADC input from 10 optional input channels.

C Prototype:

```
void  SAR8_SelectADCChannel(BYTE bChannel)
```

Assembler:

```
mov  A, bChannel
lcall  SAR8_SelectADCChannel
```

Parameters:

bChannel: One byte that specifies the ADC input channel. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Channel
SAR8_P0_0	0x00	P0[0]
SAR8_P0_1	0x08	P0[1]
SAR8_P0_2	0x10	P0[2]
SAR8_P0_3	0x18	P0[3]
SAR8_P0_4	0x20	P0[4]
SAR8_P0_5	0x28	P0[5]
SAR8_P0_6	0x30	P0[6]
SAR8_P0_7	0x38	P0[7]
SAR8_ACB00	0x40	ACB00
SAR8_ACB01	0x48	ACB01

Returns:

None

Side Effects:

See Note ** at the beginning of the API section.

SAR8_AutoAlign

Description:

Enables/disables ADC sampling with selected digital block(s).

C Prototype:

```
void  SAR8_AutoAlign(BYTE bAlignMode)
```

Assembler:

```
mov  A, bAlignMode
lcall  SAR8_AutoAlign
```

Parameters:

bAlignMode: One byte that specifies the align mode.

Symbolic Name	Value
SAR8_NOAUTOALIGN	0
SAR8_AUTOALIGN	1

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetAlignPath

Description:

Selects ADC autoalign trigger logic.

C Prototype:

```
void  SAR8_SetAlignPath(BYTE bAlignPath)
```

Assembler:

```
mov  A, bAlignPath
lcall  SAR8_SetAlignPath
```

Parameters:

bAlignPath: One byte that specifies the ADC autoalign trigger logic. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Description
SAR8_LowHighIndependent	0x00	Low and high channels are completely independent. Both can trigger ADC.
SAR8_LowOnly	0x02	Only low channel triggers ADC.
SAR8_HighOnly	0x04	Only high channel triggers ADC.
SAR8_LowHighCombined	0x06	High and low channels combine together to form a 16-bit trigger source.

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetHighAlignSrc

Description:

Selects one of the digital blocks as the alignment source for the high path of the autoalign trigger logic.

C Prototype:

```
void  SAR8_SetHighAlignSrc(BYTE bAlignSrc)
```

Assembler:

```
mov  A, bAlignSrc
lcall  SAR8_SetHighAlignSrc
```

Parameters:

bAlignSrc: One byte that specifies one of the digital blocks as the alignment source for the high path. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Digital Block
SAR8_None	0x00	Not connected
SAR8_DBB00	0x01	DBB00
SAR8_DBB01	0x02	DBB01
SAR8_DCB02	0x03	DCB02
SAR8_DCB03	0x04	DCB03

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetLowAlignSrc

Description:

Allows you to select one of the digital blocks as the alignment source for the low path of the autoalign trigger logic.

C Prototype:

```
void  SAR8_SetLowAlignSrc(BYTE bAlignSrc)
```

Assembler:

```
mov  A, bAlignSrc
lcall  SAR8_SetLowAlignSrc
```

Parameters:

bAlignSrc: One byte that specifies any of digital blocks as alignment source for the low path. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Digital Block
SAR8_None	0x00	Not connected
SAR8_DBB00	0x01	DBB00
SAR8_DBB01	0x02	DBB01
SAR8_DCB02	0x03	DCB02
SAR8_DCB03	0x04	DCB03

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetCmpH

Description:

Set comparator value for high path of autoalign.

C Prototype:

```
void SAR8_SetCmpH(BYTE bValue)
```

Assembler:

```
mov  A, bValue
lcall  SAR8_SetCmpH
```

Parameters:

bValue: One byte that specifies the comparator data for high path of autoalign.

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetCmpL

Description:

Set comparator value for low path of autoalign.

C Prototype:

```
void SAR8_SetCmpL(BYTE bValue)
```

Assembler:

```
mov  A, bValue
lcall  SAR8_SetCmpL
```

Parameters:

bValue: One byte that specifies the comparator data for low path of autoalign.

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetClock

Description:

Sets the SAR8 ADC clock rate. If you change the clock speed of the clock, you must stop the ADC conversion first, change the clock speed, then restart the conversion.

C Prototype:

```
void SAR8_SetClock(BYTE bClock)
```

Assembler:

```
mov  A, bClock
lcall  SAR8_SetClock
```

Parameters:

bClock: One byte that specifies clock rate. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Clock Rate
SAR8_SYSCLK	0x00	SysClk
SAR8_SYSCLK_2	0x01	SysClk/2
SAR8_SYSCLK_4	0x02	SysClk/4
SAR8_SYSCLK_8	0x03	SysClk/8
SAR8_SYSCLK_16	0x04	SysClk/16
SAR8_SYSCLK_32	0x05	SysClk/32
SAR8_SYSCLK_64	0x06	SysClk/64

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetScale

Description:

Sets the scale factor when reading conversion result data.

C Prototype:

```
void SAR8_SetScale(BYTE bScaleMode)
```

Assembler:

```
mov  A, bScaleMode
lcall  SAR8_SetScale
```

Parameters:

bScaleMode: One byte that specifies right size scale. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Scale Factor
SAR8_1_1	0x00	1
SAR8_1_2	0x08	1/2
SAR8_1_4	0x10	1/4
SAR8_1_8	0x18	1/8
SAR8_1_16	0x20	1/16
SAR8_1_32	0x28	1/32
SAR8_1_64	0x30	1/64

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

SAR8_SetRunMode

Description:

Sets running mode for ADC to single conversion or free-run. This parameter is set by the user module Run parameter. It is not necessary to call this function unless the run mode needs to be changed.

C Prototype:

```
void SAR8_SetRunMode(BYTE bRunMode)
```

Assembler:

```
mov  A, bRunMode
lcall  SAR8_SetRunMode
```

Parameters:

bRunMode: One byte that specifies the conversion mode

Symbolic Name	Value
SAR8_OneShot	0x00
SAR8_FreeRun	0x01

Returns:

None.

Side Effects:

See Note ** at the beginning of the API section.

Sample Firmware Source Code

This sample code starts a continuous conversion, polls the data available flag, and sends the converted byte to a user function.

```
;-------------------------------------------------------------------------
; Sample Asm Code for the SAR8
;-------------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all user modules

export _main

_main:
    M8C_EnableGInt
    mov    A, SAR8_P0_0
    call   SAR8_SelectADCChannel
    mov    A, SAR8_FreeRun
    call   SAR8_SetRunMode
    call   SAR8_Start
.loop:
    call   SAR8_fIsDataAvailable
    cmp    A, 0
    jz     .loop
    call   SAR8_bGetData
    mov    [_bResult], A
    ; call User function here
    jmp    .loop
```

The same program in C:

```
//-------------------------------------------------------------------------
// Sample C Code for the SAR8
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

BYTE bResult;

void main(void)
{
  M8C_EnableGInt;

SAR8_SelectADCChannel(SAR8_P0_0);
  SAR8_SetRunMode(SAR8_FreeRun);
  SAR8_Start();

while(1) {
    while (0 == SAR8_fIsDataAvailable());  // wait for result
    bResult = SAR8_bGetData();
    // Add user code here
  }
```

```
}
```

Configuration Registers

The following registers are used for the SAR8 block.

Table 2. Block SAR8, Register: SAR8_CONTROL_0_REG (SARADC_CR0; 0, 69h)

Bit	7	6	5	4	3	2	1	0
Value	Reserved	ADC Channel				Data ready	Start/Busy	ADCEN

Table 3. Block SAR8, Register: SAR8_CONTROL_1_REG (SARADC_CR1; 0, 6Ah)

Bit	7	6	5	4	3	2	1	0
Value	ADCPower	R2RPower	Reserved			Align Source		AutoAlign

Table 4. Block SAR8, Register: SAR8_CONTROL_2_REG (SARADC_CR2; 1, ABh)

Bit	7	6	5	4	3	2	1	0
Value	0	FreeRun	Scale			Clock		

Table 5. Block SAR8, Register: SAR8_TRIGGER_SRC_REG (SARADC_TRS; 1, A8h)

Bit	7	6	5	4	3	2	1	0
Value	DCB03		DCB02		DBB01		DBB00	

Table 6. Block SAR8, Register: SAR8_COMPARE_LO_REG (SARADC_TRCL; 1, A9h)

Bit	7	6	5	4	3	2	1	0
Value	bValue							

Table 7. Block SAR8, Register: SAR8_COMPARE_HI_REG (SARADC_TRCH; 1, AAh)

Bit	7	6	5	4	3	2	1	0
Value	bValue							

Table 8. Block SAR8, Register: SAR8_DATA_LO_REG (SARADC_DL; 0, 67h)

Bit	7	6	5	4	3	2	1	0
Value	Data							

Version History

Version	Originator	Description
1.0	DHA	Initial version.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2005-2015 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.