16-Bit CRC Generator Datasheet CRC16 V 3.2

Copyright © 2002-2014 Cypress Semiconductor Corporation. All Rights Reserved.

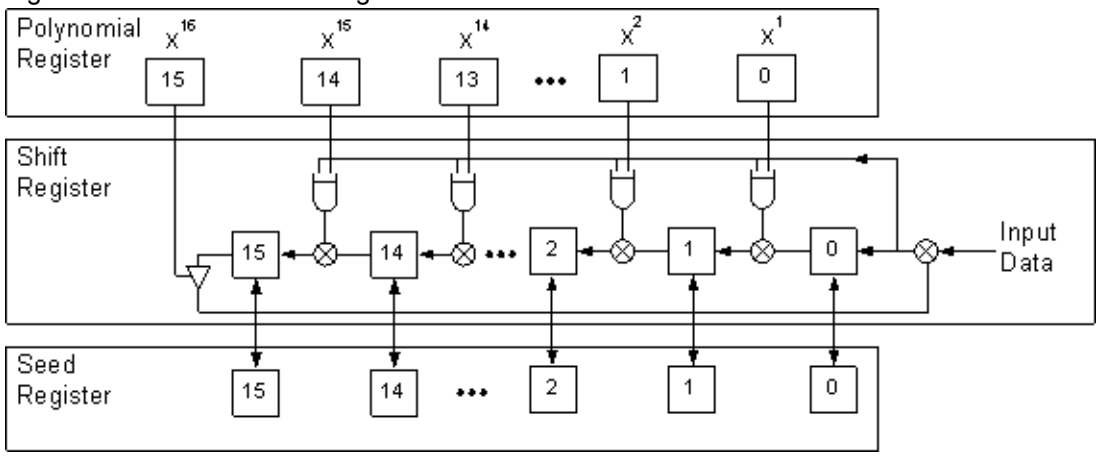
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMG300, CY8CTST300, CY8CTMA300, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52						
	2	0	0	54	0	1
CYWUSB6953	2	0	0	54	0	1

Features and Overview

- 2 to 16-bit CRC generator
- Data input clocking up to 48 MHz
- Programmable polynomial
- Programmable seed value
- Serial data in, parallel result out

The CRC16 User Module computes a 2 to 16-bit cyclical redundancy check (CRC) algorithm on an input serial data stream. The polynomial can be defined to implement CRC functions, such as the CRC-16 or CCITT algorithm. A seed value can be specified to initialize the starting data value.

Figure 1. CRC16 Block Diagram



Functional Description

The CRC16 User Module computes a 16-bit CRC algorithm with two consecutive digital blocks named CRC16_LSB and CRC16_MSB. The Polynomial, Shift, Seed, and Control registers of each CRC16 block correspond to the digital PSoC block registers.

The configuration of the underlying connective hardware of the CRC16 digital PSoC blocks coordinate the operation of the two PSoC blocks as a single 16-bit CRC User Module. The Polynomial, Shift, and Seed registers refer to the combined registers from the CRC16_LSB and CRC16_MSB blocks (unless specifically noted). The CRC16_MSB registers form the most significant byte of the register set and the CRC16_LSB registers form the least significant byte. For example, the Polynomial register is composed of the CRC16_MSB Polynomial register and the CRC16_LSB Polynomial register.

The CRC16 User Module is implemented as a linear feedback shift register (LFSR). The Shift register computes the LFSR function; the Polynomial register holds the polynomial that defines the LFSR polynomial; and the Seed register enables initialization of the starting data.

This module requires that the Seed and Polynomial registers are initialized prior to setting the start bit in the CRC16_LSB's Control register.

Writing the seed value into the Seed register, while the CRC16 is not started, causes the seed value to be latched into the Shift register, initializing the starting data. Writing the seed value, after the CRC16 is started, has no effect.

Computation of an N-bit LFSR result is specified by a polynomial with N+1 terms, the last of which is the X^0 term where X^0=1. For example, the widely used CRC-CCITT 16-bit polynomial is $X^{16}+X^{12}+X^5+1$. The CRC algorithm assumes the presence of the X^0 term, so that the polynomial for an N-bit result can be expressed by an N bit rather than N+1-bit specification.

To specify the polynomial specification, write an N+1 bit binary number corresponding to the full polynomial, with 1's for each term present. The CRC-CCITT polynomial would be 10001000000100001b. Then, drop the right-most bit (the X^0 term) to obtain the CRC16 polynomial value. To implement the CRC-CCITT example, the Polynomial register is loaded with the value of 8810h.

When the seed value and polynomial are initialized, the CRC16 User Module is started. A rising edge of the input clock shifts each bit, MSB first, of the input data stream through the Shift register, computing the specified CRC algorithm. Eight clocks are required to compute the CRC for each byte of input data.

Reading the CRC computed result is a two-step process. First, the Shift register is read. This causes the result data to be latched into the Seed register. Then, the result is read directly from the Seed register.

Note that the initial seed value is lost. This is usually of no consequence since the seed value is only used to initialize the Shift register once, per data set.

It is advisable to stop the CRC16 User Module before reading the CRC value, to ensure that the data is not inadvertently clocked in while performing a read.

DC and AC Electrical Characteristics

Table 1. CRC16 DC and AC Electrical Characteristics

Parameter	Conditions and Notes	Typical	Limit	Units
Max Input Data Clocking		12[1]	48	MHz

Electrical Characteristics Notes

1. If the input data stream or clock is routed using a global bus, then the maximum clock input rate is 12 MHz.

Placement

The CRC16 User Module can be placed in any two consecutive digital PSoC blocks.

Parameters and Resources

InputDataStream

The input data stream can be connected to a low, high, neighboring PSoC block, the analog comparator output bus, or one of the global busses. Using a global bus, the input can be connected to one of the external pins.

Clock

The CRC16 is clocked by one of 16 possible sources. The Global I/O busses can be used to connect the clock input to an external pin or a clock function generated by a different PSoC block. When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation.The 48 MHz clock, the CPU_32kHz clock, one of the divided clocks, 24V1 or 24V2, or another PSoC block output can be specified as the clock input.

ClockSync

In the PSoC devices, digital blocks may provide clock sources in addition to the system clocks. Digital clock sources may even be chained in ripple fashion. This introduces skew with respect to the system clocks. These skews are more critical in the CY8C29/27/24/22/21xxx PSoC device families because of various data-path optimizations, particularly those applied to the system busses. This parameter may be used to control clock skew and ensure proper operation when reading and writing PSoC block register values. Appropriate values for this parameter should be determined from the following table.

ClockSync Value	Use
Sync to SysClk	Use this setting for any 24 MHz (SysClk) derived clock source that is divided by two or more. Examples include VC1, VC2, VC3 (when VC3 is driven by SysClk), 32 kHz, and digital PSoC blocks with SysClk-based sources. Externally generated clock sources should also use this value to ensure that proper synchronization occurs.
Sync to SysClk*2	Use this setting for any 48 MHz (SysClk*2) based clock unless the resulting frequency is 48 MHz (in other words, when the product of all divisors is 1).
Use SysClk Direct	Use when a 24 MHz (SysClk/1) clock is desired. This does not actually perform synchronization but provides low-skew access to the system clock itself. If selected, this option overrides the setting of the Clock parameter, above. It should always be used instead of VC1, VC2, VC3 or Digital Blocks where the net result of all dividers in combination produces a 24 Mhz output.
Unsynchronized	Use when the 48 MHz (SysClk*2) input is selected. Use when unsynchronized inputs are desired. In general this use is advisable only when interrupt generation is the sole application of the Counter.

Invert InputDataStream

This parameter allows you to invert the InputDataStream.

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.

Calls to the API function can be made in both assembly and C. The following is the list of CRC16 supplied API functions.

CRC16_Start

Description:

Enables the CRC16 User Module for operation. Before the CRC16 is started, the polynomial and seed values should be initialized.

C Prototype:

```
void  CRC16_Start(void)
```

Assembler:

```
lcall  CRC16_Start
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

CRC16_Stop

Description:

Disables the CRC16 User Module.

C Prototype:

```
void  CRC16_Stop(void)
```

Assembler:

```
lcall  CRC16_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

CRC16_WriteSeed

Description:

Initializes the CRC16 **Shift** register with an initial seed value. The CRC16 User Module is stopped while updating the **Shift** register with the new seed value. Upon exit, the start state is restored.

C Prototype:

```
void  CRC16_WriteSeed(WORD wSeed)
```

Assembler:

```
mov  X, [wSeed]
mov  A, [wSeed+1]
lcall CRC16_WriteSeed
```

Parameters:

WORD wSeed: 16-bit seed value. MSB passed in the X register. LSB passed in the A register. For the CCITT polynomial, use the defined constant CRC_CCITT_SEED, which is set to 0xFFFF.

Return Value:

None

Side Effects:

Writing a value into the Seed register also latches it into the Shift register. The A and X registers may be altered by this function.

CRC16_WritePolynomial

Description:

Loads the **Polynomial** register with the CRC function polynomial. The CRC16 User Module is stopped while the **Polynomial** register is updated. Upon exit, the start state is restored.

C Prototype:

```
void  CRC16_WritePolynomial(WORD wPolynomial)
```

Assembler:

```
mov  X, [wPolynomial]
mov  A, [wPolynomial+1]
lcall CRC16_Stop
```

Parameters:

WORD wPolynomial: 16-bit polynomial value. Reference the CRC16 User Module Functional Description section for a discussion on how to set the polynomial value. For the CCITT polynomial, use the defined constant CRC_CCITT_POLYNOMIAL, which is set to 0x8810. MSB is passed in the X register and LSB is passed in the A register.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

CRC16_ReadCRC

Description:

Reads the computed CRC resultant data. Calling this function, while data is currently being clocked in, gives inaccurate results.

C Prototype:

```
WORD CRC16_wReadCRC(void)
```

Assembler:

```
lcall CRC16_wReadCRC
mov  reg[wCrcValue+1], A
mov  A, X
mov  reg[wCrcValue], A
```

Parameters:

None

Return Value:

WORD wCrcValue: Value read from the Shift register.

Side Effects:

The Seed register is overwritten with the computed CRC value. The A and X registers may be altered by this function.

Sample Firmware Source Code

```
;*****************************************************************************
; Setup CCITT CRC16
;
; This function initializes a CRC16 user module to compute a CCITT CRC
; algorithm.
;
;*****************************************************************************
include    "CRC16.inc"
export    SetupCCITT

SetupCCITT:
    ; stop the CRC16 user module
    call   CRC16_Stop

    ; load the CCITT polynomial
    mov    A, <CRC_CCITT_POLYNOMIAL     ;LSB
    mov    X, >CRC_CCITT_POLYNOMIAL     ;MSB
    call   CRC16_WritePolynomial

    ; load the CRC16 seed
    mov    A, <CRC_CCITT_SEED           ;LSB
    mov    X, >CRC_CCITT_SEED           ;MSB
    call   CRC16_WriteSeed

    ;start the CRC16
    call   CRC16_Start
```

The same code in C is:

```
#include "CRC16.h"

void SetupCCITT(void)
{
    // stop the CRC16 user module
    CRC16_Stop();

    // load the CCITT polynomial
    CRC16_WritePolynomial(CRC_CCITT_POLYNOMIAL);

  // load the CCITT seed
    CRC16_WriteSeed(CRC_CCITT_SEED);

  // start the CRC16
    CRC16_Start();
}
```

Configuration Registers

The PSoC Digital block registers used to configure a user module are:

Table 2. CRC16_MSB: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	1	0

Table 3. CRC16_LSB: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	1	0

Table 4. CRC16_MSB: Register Input

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

Clock selects the input clock from one of 16 sources. This parameter is set in the Device Editor.

Table 5. CRC16_LSB: Register Input

Bit	7	6	5	4	3	2	1	0
Value	InputDataStream				Clock			

InputDataStream selects the data input from one of 16 sources. Clock selects the input clock from one of 16 sources. These parameters are set in the Device Editor.

Table 6. CRC16_MSB: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 7. CRC16_LSB: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 8. CRC16_MSB: Shift Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Shift Register (MSB)							

Shift Register is the CRC16 MSB Shift register. It is read and configured using the CRC16 API.

Table 9. CRC16_LSB: Shift Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Shift Register (LSB)							

Shift Register is the CRC16 LSB Shift register. It is read and configured using the CRC16 API.

Table 10. CRC16_MSB: Polynomial Register DR1

Bit	7	6	5	4	3	2	1	0
Value	Polynomial Register (MSB)							

Polynomial Register is the CRC16 MSB Polynomial register. It is modified using the CRC16 API.

Table 11. CRC16_LSB: Polynomial Register DR1

Bit	7	6	5	4	3	2	1	0
Value	Polynomial Register (LSB)							

Polynomial Register is the CRC16 LSB Polynomial register. It is modified using the CRC16 API.

Table 12. CRC16_MSB: Seed Register DR2

Bit	7	6	5	4	3	2	1	0
Value	Seed Register (MSB)							

Seed Register is the CRC16 MSB Seed register. It is modified using the CRC16 API.

Table 13. CRC16_LSB: Seed Register DR2

Bit	7	6	5	4	3	2	1	0
Value	Seed Register (LSB)							

Seed Register is the CRC16 LSB Seed register. It is modified using the CRC16 API.

Table 14. CRC16_MSB: Control Register CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 15. CRC16_LSB: Control Register CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Start/Stop

Start/Stop indicates that the CRC16 is enabled when set. It is modified using the CRC16 API.

Version History

Version	Originator	Description
3.2	TDU	1. In APIs CRC16_WriteSeed and CRC16_WritePolynomial, changed CRC16_CCITT_SEED to CRC_CCITT_SEED and CRC16_CCITT_POLYNOMIAL to CRC_CCITT_POLYNOMIAL. 2. Updated Clock description to include: When using an external digital clock for the block, the row input synchronization should be turned off for best accuracy, and sleep operation.

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

Copyright © 2002-2014 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.