Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 001-58771

Spec Title: SELF TEST WITH PSOC(R) - AN58771

Sunset Owner: Sampath Selvaraj (SAMP)

Replaced by: None

AN58771

Self Test with PSoC®

Author: Jay Jedinak
Associated Project: Yes
Associated Part Family: CY8C21x34
Software Version: PSoC Designer™ 5.0
Related Application Notes: AN17582

IEC60730 is a safety standard requiring self check routines for processors to ensure reliable and safe operation. These features help in safety critical applications and are consequently becoming increasingly popular in non hazardous designs. This application note discusses how to conduct simple self test processes for the PSoC® 1. The tests covered are CPU, RAM, ROM, Program Counter, and Interrupts for Class B environments.

OBSOLETE

Introduction

The International Electrotechnical Commission (IEC) has developed safety standards intended to ensure proper functionality and safety of critical systems. This application note focuses on the IEC specification IEC60730, and specifically Annex H Class B. The Annex H section of IEC60730 outlines necessary requirements for microcontroller based systems. It provides specifications for two levels of hazard potential: Class B and Class C. Class B is intended to maintain the safe functioning of most kitchen appliances, washers/dryers, and door locks. Class C is for systems of greater hazard than Class B such as systems using open flames.

There are several key features of MCU, the incorrect functioning of which can cause the microcontroller to operate unpredictably. The CPU, RAM, ROM, Program Counter, and Interrupts are examples of necessary and vital elements of a microcontroller. The following sections outline each of these elements and provide a code snippet demonstrating how to self test them.

Software Implementation

Embedded designers prefer coding in C to Assembly. However, there are instances where the terse and efficient nature of Assembly coding adds flexibility to a design. By minimizing the diagnostic self test code size, the ability to maximize the flexibility of PSoC is the most enhanced. And although the self test functions are written in Assembly, they can be called either from C or Assembly. For more information regarding writing Assembly in PSoC 1, reference the Assembly Language Guide at www.cypress.com or the documentation directory within PSoC Designer™.

The code provided incorporates a prefix alias which aligns itself with coding from the Cypress application note, AN17582 *Failure Mode and Effect Analysis (FMEA) Implementation for CSD*. This application note presents diagnostic checks for CapSense® projects with CSD using a CY8C21x34. The same coding naming scheme has been retained.

CPU Self Test

The PSoC 1 relies on five registers for program execution. These registers are:

Accumulator (A)

Index (X)

Program Counter (PC)

Stack Pointer (SP)

Flags (F)

To check for stuck bits within these registers, a checkerboard test is implemented. A checkerboard test writes '1's to a register and verifies the value; then it writes '0's and verifies again. This is done by first writing 0xAA and then 0x55 for each byte. The switching of these values resembles the differing colored squares on a checkerboard. The checkerboard method is implemented for all CPU registers except the Program Counter. See the following Program Counter for more details.

Function:

BYTE FMEA_bCPU_Test(void)

returns: 0 No error

halt Error detected

If an error is detected, then the PSoC should not continue to function because its behavior can be unpredictable and therefore, potentially unsafe.

RAM Self Test

The RAM self test is very similar in principle to the CPU test in that a checkerboard test is implemented to check for stuck bits. Because the values in RAM may be useful to the operating program, its values are stored on the stack while its RAM location is tested.

Function:

BYTE FMEA_bRAM_Test(void)

returns:

Table 1. Error Code Descriptions for FMEA_CheckBaselines

Error Code	Value	Description
FMEA_RAM_OK	0x00	No errors detected
FMEA_RAM_TOTAL_OK	0x01	No errors detected on total
FMEA_ERR_RAM_FAILURE	0x02	Error detected

FMEA_bRAM() must be called 512 times to cover all bytes in RAM. The FMEA_RAM_OK return allows you to check a section of RAM one byte at a time. This provides greater time management flexibility. When the function returns a 0x01, you know all the pages of RAM have been checked.

ROM Self Test

The ROM self test calculates a running checksum 64 bytes at a time. Similar to the FMEA_bRAM_Test(), this function can be called over time to allow time sensitive control loops. To complete a full diagnostic of the ROM, this function must be called 128 times. When the final checksum is calculated, FMEA_bROM_Test() compares the running checksum total with a checksum preloaded into the last two bytes of Flash.

A supplementary program is used to calculate the checksum of the project. This checksum is placed into the project's hex file, which is then programmed into the PSoC. The FMEA_bROM_Test() compares its running checksum to this hardcoded checksum.

Programming Steps

1. Compile project in PSoC Designer.
2. Edit *ChecksumCalc.bat* such that all hex files have the same prefix as your actual hex file.
3. Run *Checksumcalc.exe*.
4. Program the part.

See Appendix A for a detailed description.

Function:

BYTE FMEA_bRAM_Test(void)

returns:

Table 2. Error Code Descriptions for FMEA_CheckBaselines

Error Code	Value	Description
FMEA_ROM_OK	0x00	No errors detected
FMEA_ROM_TOTAL_OK	0x01	No errors detected on total
FMEA_ERR_ROM_FAILURE	0x02	Error detected

Program Counter Self Test

The program counter register is part of the CPU register set. To test these registers explicitly using a checkerboard test, the addresses of 0x5555 and 0xAAAA must be allocated only for this test. Because it is only four bytes of Flash, it appears to be a good investment of resources. However, this breaks the Flash into segments of memory which are now smaller in size. This condition is not favorable for efficient compilation and can inhibit some programs from fitting into Flash although their absolute size is smaller than the actually Flash size.

A functional test is favored in this instance and allowed according to Annex H Table H.11.12.7 of IEC60730. To successfully run the program and other diagnostics, the Program Counter must be functioning.

Interrupt Self Test

Interrupts are an integral part of most embedded designs and require attention to ensure proper operation. For safety and reliability, having an interrupt occur too often or too infrequently, such as hanging in an infinite loop, may cause a system to perform in an unsafe manner.

Interrupts that occur frequently can be handled by enabling the Watch Dog Timer (WDT). If the WDT is not refreshed, as it is stuck in an interrupt, then it causes a reset.

For interrupts that occur infrequently or those that cause a system to lag and perform out of specification, using a routine that exchanges data via digital communications can be used. Exchanges of data can continually alert a co-processor of your status. For example, a system may exchange timing data read from a counter, and transmit this data to a co-processor. The co-processor evaluates the counter value and determines if the value is within tolerance.

Associated Project

The project accompanying this document provides an example of simple calls to verify RAM, CPU, and ROM. All CapSense blocks are left vacant for ease of integration of a pre-existing project, as demonstrated in AN17582. The current project uses 750 bytes of Flash, and can be modified according to your memory needs.

PROOF DRAFT

Summary

IEC60730 stipulates methods for a microcontroller to run diagnostics to verify its reliable functionality and ensure the safe operation of a system. However, safety is not the only inspiration. There are many non hazardous systems which benefit from reliable operation of their microcontroller. This application note provides a starting point to ensure a design with PSoC embraces reliability and also retains its flexibility for robust and demanding applications.

About the Author

Name: Jay Jedinak

Title: Applications Engineer Sr.

Background: Jay Jedinak is a graduate of Seattle Pacific University with a degree in Electrical Engineering.

Contact: yji@cypress.com

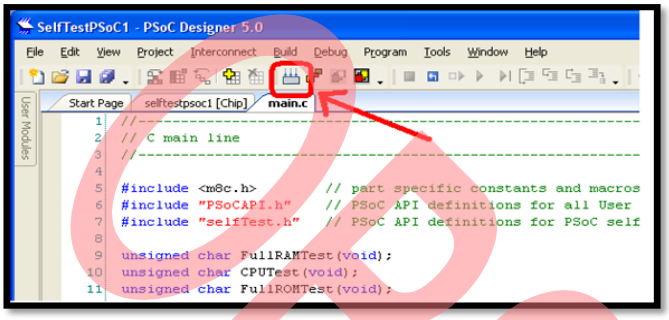
OBSOLETE

Appendix A

The following instructions helps to program your part for proper Flash/ROM diagnostic testing.

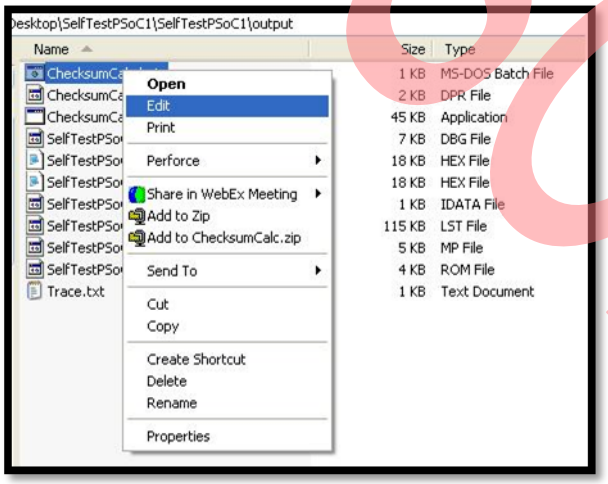
1. Compile your project by clicking the **Compile** button.

Figure 1. Compile Project



2. Edit the *ChecksumCalc.bat* file located in the 'output' folder of your projects root directory. To do this, right-click the *ChecksumCalc.bat* file and select **Edit**.

Figure 2. Edit *ChecksumCalc.bat*



3. Modify the hex files such that the prefix to each hex file has the same name as your project. Save the file and close it.

Figure 3. Prefixes of hex Files

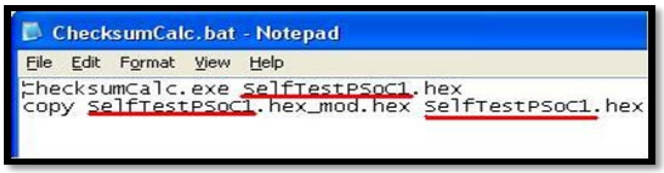
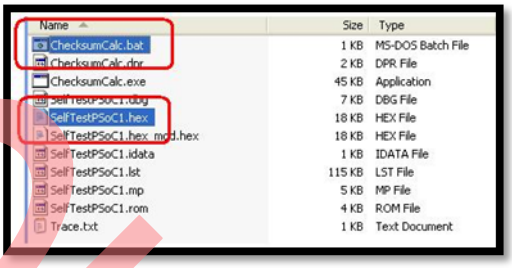
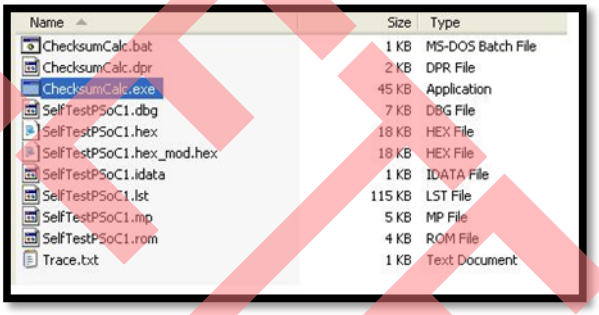


Figure 4. Observe *ChecksumCalc.bat* and *SelfTestPSoC1.hex*



4. Run the *ChecksumCalc.exe* file in the same output folder.

Figure 5. Run the *ChecksumCalc.exe*



After you run *ChecksumCalc.exe*, you are ready to program your part.

If you modify the program, redo the complete process.

PSOC IT

Document History

Document Title: Self Test with PSoC®

Document Number: 001-58771

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2842959	YJI	01/08/2010	New application note
*A	3683677	YJI	07/23/2012	Updated template
*B	3874135	SAMP	01/18/2013	Obsolete document.

OBSOLETE

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

OBSOLETE

PSoc® is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2010-2013. 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.
This 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.