Shadow Registers Datasheet ShadowRegs V 1.1

Copyright © 2007-2013 Cypress Semiconductor Corporation. All Rights Reserved.

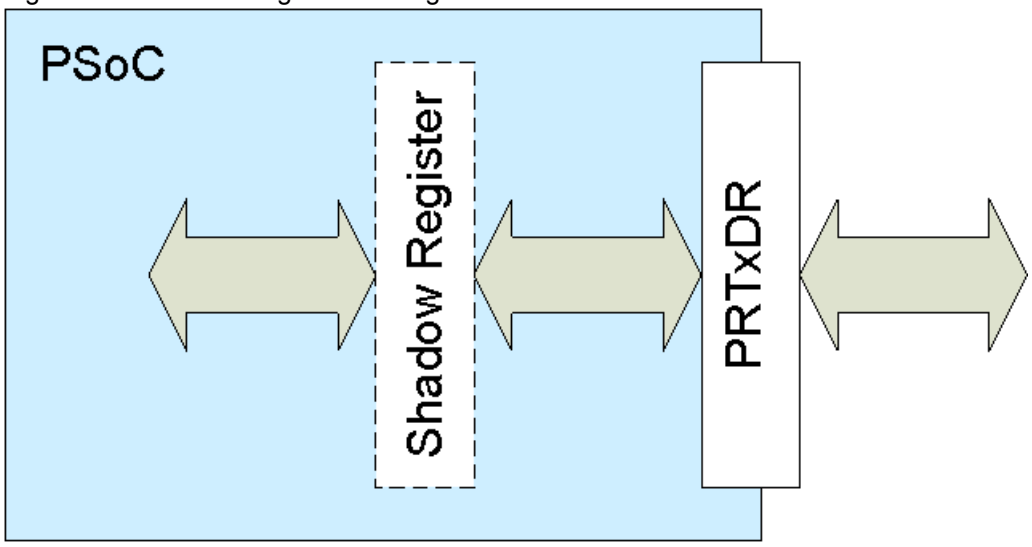
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C20x34, CY8C21x12, CY8C29/27/24/22/21xxx, CY8C20x36, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20xx6AS, CY8C20XX6L, CY8C20x46, CY8C20x96, CY8C20045, CY8C20055, CY7C64215, CY7C64343, CY7C60413, CY7C603xx, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTST120, CY8CTST200, CY8CTMG110, CY8CTMG120, CY8CTMG2xx, CY8CTMA120, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CYONS2010, CYONS2011, CYONSFN2051, CYONSFN2053, CYONSFN2061, CYONSFN2151, CYONSFN2161, CYONSFN2162, CYONSFN2010-BFXC, CYONSCN2024-BFXC, CYONSCN2028-BFXC, CYONSCN2020-BFXC, CYONSKN2033-BFXC, CYONSKN2035-BFXC, CYONSKN2030-BFXC, CYONSTN2040, CY8CTMA140, CY8C20xx7/7S, CY8C20045, CY8C20055, CYRF89x35, CY8C20065, CY8C24x93, CY7C69xxx						
	0	0	0	0	1	0

Features and Overview

- Provides a global shadow register for a selected port data register
- Generates a set of macros for port pin manipulation
- Prevents corruption of GPIO pin settings during CPU control of GPIO
- Cooperates with other user modules that allocate shadow registers.

The ShadowRegs User Module creates a RAM variable (the shadow register) that caches values written to a port data register (PRTxDR). Using a shadow register enables CPU control of an individual GPIO output pin without the risk of corrupting the settings of other GPIO pins sharing the same port.

Figure 1. ShadowRegs Block Diagram



Functional Description

The ShadowRegs User Module creates a shadow register for the selected port data register.

Note Some other user modules create shadow registers. If the selected port data register coincides with a port data register for which another user module created a shadow register, the user modules cooperate so that there is only one shadow register that needs to be accessed by your code.

Theory of Operation

Data written to a port data register may differ from data read from the port data register because the read data represents actual pin voltages while the written data controls the pin output setting (transistor switching). This difference between the read and write data introduces the risk of inadvertently corrupting a pin's output setting when performing a logical operation directly on the port data register to affect the setting of a different pin sharing the same port. This situation commonly occurs with pins operating in resistively pulled up/pulled down, or open drain drive modes.

Data from a shadow register does not immediately write to a physical port. An ISR reflects the incorrect physical port values if this ISR calls before writing to the physical port but after updating shadow register. This happens because the physical port is not updated. It is recommended to disable interrupts before writing to a shadow register and enable interrupts only after writing data to a physical port.

Example

An application uses P0[0] as a sourcing LED output and P0[1] as a pulled up input for a normally open switch connected to ground. After setting the drive mode registers for port 0, the firmware initializes the LED to off and enables the input with the following C statement.

```
PRT0DR = 0x02;
```

Without a shadow, register, the firmware toggles the LED by the following C statement.

```
PRT0DR ^= 0x01;
```

On the initial LED toggle operation, the value read from PRT0DR is 0x00 if the switch is closed, and 0x02 if the switch is open. Performing the XOR operation to toggle the LED results in 0x01 (0x00 XOR 0x01) if the switch is closed but 0x03 (0x02 XOR 0x01) if the switch is open when the initial toggle occurs. Toggling the LED when the switch is closed causes P0[1] to be driven to 0 V internally. When the switch is opened, the value read from PRT0DR is still 0x01 because the voltage on P0[1] is still 0 V. The switch input was inadvertently disabled.

The solution is to always manipulate the shadow register first, then copy the shadow register value into the port data register. The following C statements initialize the LED output and the switch input:

```
//Disable interrupts by clearing corresponding bits in the INT_MSKx registers
Port_0_Data_SHADE = 0x02;
PRT0DR = Port_0_Data_SHADE;
//Enable interrupts by setting the corresponding bits in the INT_MSKx registers
```

The following code toggles the LED without affecting the switch input

```
//Disable interrupts by clearing corresponding bits in the INT_MSKx registers
Port_0_Data_SHADE ^= 0x01;
PRT0DR = Port_0_Data_SHADE;
//Enable interrupts by setting the corresponding bits in the INT_MSKx registers
```

Regardless of whether the switch is open or closed, 0x03 (0x02 XOR 0x01) is written to the port data register on the initial toggle operation. This is because the logical operation is performed using the shadow register (0x02 regardless of whether the button is open or closed) as input rather than the port data register (0x00 when the switch is closed, and 0x02 when the switch is open).

Interrupts are disabled or enabled to avoid incorrect ISR execution when the ISR calls after writing data to a shadow register, but before writing data to a physical port.

Placement

The ShadowRegs User Module is software only and does not consume any PSoC blocks. Multiple ShadowRegs User Modules can be used as necessary.

Parameters and Resources

ShadowPort

This parameter selects the PRTxDR register for which a shadow register is created. The ShadowPort parameter contains a list of all available ports.

Application Programming Interface

There is no API for this user module.

PSoC Designer generates pin manipulation macros for all named pins of the selected port. The pin manipulation macros are contained in the psocgpioint.inc file. It contains the following macros (*ShadowRegs* is replaced by the instance name of the user module):

- macro Get*ShadowRegs*Pin_Data;
- macro Set*ShadowRegs*Pin_Data;
- macro Clear*ShadowRegs*Pin_Data;

PSoC Designer automatically generates C language constants and masks in psocgpioint.h, but you need to create:

```
MyPin_DataShadow &= ~MyPin_MASK;
MyPin_Data_ADDR = MyPin_DataShadow;
```

Sample Firmware Source Code

The following is the Sample C code for this user module:

```
//
//  This sample shows how to creates a shadow register for the Port_0 data register.
//  The Port_0_Data_SHADE variable is defined in PSoCConfig.asm file as 'extern'.
//
// OVERVIEW:
//
//  The SHADOWREGS UM creates a shadow register for the selected port data register.
//  In this example the SHADOWREGS creates a shadow register for the Port_0 data
//   register.
//
//The following changes need to be made to the default settings in the Device Editor:
//
//   1. Select SHADOWREGS user module.
//   2. Rename User Module's instance name to SHADOWREGS.
```

```
//   3. Set User Module's ShadowPort Parameter to Port_0.
//
// CONFIGURATION DETAILS:
//
//  1. The UM's instance name must be shortened to SHADOWREGS.
//
// PROJECT SETTINGS:
//
//     Default
//
// USER MODULE PARAMETER SETTINGS:
//
//  ------------------------------------------------------------------------------
//  UM             Parameter         Value                Comments
//  ------------------------------------------------------------------------------
//  SHADOWREGS     Name              SHADOWREGS           UM's instance name
//                 ShadowPort        Port_0
//  ------------------------------------------------------------------------------

/* Code begins here */

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

void main(void)
{
    // M8C_EnableGInt ; // Uncomment this line to enable Global Interrupts

    Port_0_Data_SHADE = 0x00;

    while(1)
    {
        if(PRT0DR & 0x02)
        {
            Port_0_Data_SHADE |= 0x01;
            PRT0DR = Port_0_Data_SHADE;
        }
        else
        {
            Port_0_Data_SHADE &= ~0x01;
            PRT0DR = Port_0_Data_SHADE;
        }
        // Insert your main routine code here.
    }
}
```

Here is the same code in Assembly:

```
;
;  This sample shows how to creates a shadow register for the Port_0 data register.
;  The Port_0_Data_SHADE variable is defined in PSoCConfig.asm file as 'extern'.
;
; OVERVIEW:
;
;  The SHADOWREGS UM creates a shadow register for the selected port data register.
;  In this example the SHADOWREGS creates a shadow register for the Port_0 data register.
```

```
;
;The following changes need to be made to the default settings in the Device Editor:
;
;   1. Select SHADOWREGS user module.
;   2. Rename User Module's instance name to SHADOWREGS.
;   3. Set User Module's ShadowPort Parameter to Port_0.
;
; CONFIGURATION DETAILS:
;
;  1. The UM's instance name must be shortened to SHADOWREGS.
;
; PROJECT SETTINGS:
;
;     Default
;
; USER MODULE PARAMETER SETTINGS:
;
;  ------------------------------------------------------------------------------
;  UM             Parameter         Value                 Comments
;  ------------------------------------------------------------------------------
;  SHADOWREGS     Name              SHADOWREGS            UM's instance name
;                 ShadowPort        Port_0
;  ------------------------------------------------------------------------------
; Code begins here

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 ; Uncomment this line to enable Global Interrupts
    mov [Port_0_Data_SHADE], 0x00
    ; Insert your main assembly code here.

.terminate:
    and reg[PRT0DR], 0x02
    jz  .pinIsZero
    or  [Port_0_Data_SHADE], 0x01
    mov A, [Port_0_Data_SHADE]
    mov reg[PRT0DR], A
    jmp .terminate
.pinIsZero:
    and  [Port_0_Data_SHADE], ~0x01
    mov A, [Port_0_Data_SHADE]
    mov reg[PRT0DR], A
    jmp .terminate
```

Version History

Version	Originator	Description
1.1	DHA	Added Version History
1.1.b	DHA	Updated description for interrupt service routines in user module datasheet.

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 © 2007-2013 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.