Digital Inverter Datasheet DigInv V 1.60

Copyright © 2002-2015 Cypress Semiconductor Corporation. All Rights Reserved.

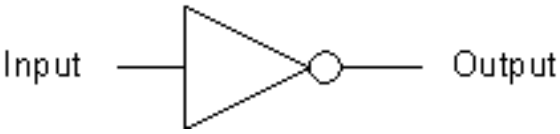
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY8C23x33, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C21x45, CY8C22x45, CY8CTMA140, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx						
	1	0	0	57	0	1
CYWUSB6953	1	0	0	57	0	1

Features and Overview

- Output is digital inverted input
- Requires only one digital block
- Can be used to generate an interrupt on the falling edge of the input

The DigInv User Module is a simple digital inverter. The output is a logical NOT of the input signal.

Figure 1. DigInv Block Diagram



Functional Description

DigInv is a single input digital inverter. It can be mapped onto any digital PSoC block. The API provides functions to start and stop the DigInv User Module and to enable or disable its interrupt capability. When the DigInv is stopped, the output is held low.

DC and AC Electrical Characteristics

Table 1. DigInv DC and AC Electrical Characteristics

Parameter	Conditions and Notes	Typical	Limit	Units
Input F_{max} from Global Bus			12	MHz

Parameter	Conditions and Notes	Typical	Limit	Units
Input F_{max} from internal connections			48	MHz
Input to Output Transition		< 25		ns
Output F_{max} to Global Bus			12	MHz
Output F_{max} to internal connections			48	MHz

Timing

The DigInv User Module is limited to 12 MHz transition speeds when the input or output is connected to a global bus. To achieve higher transition speeds, place the DigInv User Module next to a user module providing the input or receiving the output, so that the Previous connectivity option can be chosen. Placing the DigInv in PSoC block DBA03 enables the output to be connected to any other digital PSoC block as an input.

Placement

The DigInv may be placed in any digital PSoC block.

Parameters and Resources

Input

The input is selected from one of 16 sources. These sources include the 48 MHz oscillator output, lower frequencies (24V1 and 24V2) divided down from the 24 MHz system clock, and other PSoC blocks and external inputs routed through global inputs and outputs.

Output

The output may be routed to one of four global output signals.

ClockSync

If the output signal of the DigInv User Modules is to be used as an input signal or a clock for other blocks within the PSoC it is recommended that the input is synchronized with one internal system clock. The choice of clocks depends on where the output is routed internally, which is further explained in the following table.

ClockSync Value	Use
Sync to SysClk	Use this setting when routing the output to blocks using the 24 MHz (SysClk) or a SysClk derived clock source that is divided by two or more. Examples include VC1, VC2, VC3 (when VC3 is driven by SysClk), 32KHz.
Sync to SysClk*2	Use this setting when routing the output to blocks using the 48 MHz (SysClk*2) or a SysClk*2 based clock.
Unsynchronized	Use when unsynchronized inputs are desired. In general this use is advisable only when planning to feed the output directly to a pin or for interrupt generation.

Interrupt Generation Control

There are two additional parameters that become available when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays:

- Interrupt API
- IntDispatchMode

InterruptAPI

The InterruptAPI parameter allows conditional generation of a user module's interrupt handler and interrupt vector table entry. Select "Enable" to generate the interrupt handler and interrupt vector table entry. Select "Disable" to bypass the generation of the interupt handler and interrupt vector table entry. Properly selecting whether an Interrupt API is to be generated is recommended particularly with projects that have multiple overlays where a single block resource is used by the different overlays. By selecting only Interrupt API generation when it is necessary the need to generate an interrupt dispatch code might be eliminated, thereby reducing overhead.

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting "ActiveStatus" causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting "OffsetPreCalc" causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

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.

Following are the API programming routines provided for DigInv.

DigInv_Start

Description:

Starts the DigInv operation.

C Prototype:

```
void  DigInv_Start(void);
```

Assembly:

```
lcall  DigInv_Start
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function. Before calling the DigInv_Start function this flag would be set to FALSE and then checked in the ISR. If the ISR finds the value of the flag to be TRUE then it executes the ISR code, if the value of the flag is FALSE then it sets the flag to TRUE and exits without executing the rest of the ISR code.

DigInv_Stop

Description:

Stops the DigInv operation. The output will be held low.

C Prototype:

```
void  DigInv_Stop(void);
```

Assembly:

```
lcall  DigInv_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

DigInv_EnableInt

Description:

Enables interrupt mode operation.

C Prototype:

```
void DigInv_EnableInt(void);
```

Assembly:

```
lcall DigInv_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

DigInv_DisableInt

Description:

Disables interrupt mode operation.

C Prototype:

```
void DigInv_DisableInt(void);
```

Assembly:

```
lcall DigInv_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

You can alter the A and X registers by this function.

Sample Firmware Source Code

The following is assembly language source that illustrates the use of APIs.

```
;-----------------------------------------------------
; Example assembly program using DigInv User Module
;-----------------------------------------------------

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:

    lcall  DigInv_EnableInt  ; Use if interrupts desired
    lcall  DigInv_Start      ; Enable inverter

; Place user code here.

.terminate:
    jmp .terminate
```

The same code in C is:

```
//*****************************************
//  Example C program using DigInv User Module
//
//*****************************************
#include    "M8C.h"
#include    "PSoCAPI.h"

void main(void)
{
DigInv_EnableInt();  // Use if interrupts desired
   DigInv_Start();      // Enable Inverter

// Rest of User code
}
```

Configuration Registers

The digital PSoC block registers used to configure the DigInv User Module are described in the following tables. Only the parameterized symbols are explained.

Table 2. Block DigInv: Register Function

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

Table 3. Block DigInv: Register Input

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

Input selects the input from 1 of 16 sources and is set in the Device Editor.

Table 4. Block DigInv: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	Out Enable	Out Sel	

Out Enable is a flag that indicates the output is enabled. Out Sel is a flag that indicates where the output of the DigInv will be routed. Both parameters are set in the Device Editor.

Table 5. Block DigInv: Counter Register DR0

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

Table 6. Block DigInv: Period Register DR1

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

Table 7. Block DigInv: CompareValue Register DR2

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

Table 8. Block DigInv: Control Register CR0

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

Enable, when set, indicates that the DigInv is enabled. It is modified by using the DigInv API.

Version History

Version	Originator	Description
1.5	DHA	Added Version History
1.60	MYKZ	Corrected method of clearing posted interrupts.

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 © 2002-2015 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.