OneWire Datasheet OneWire V 1.1

Copyright © 2008-2014 Cypress Semiconductor Corporation. All Rights Reserved.

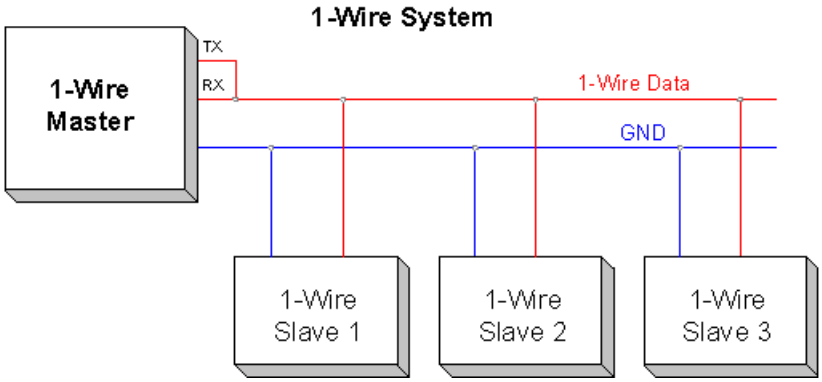
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/23/21xxx, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx						
Main UM	2	0	0	168	1	2
+ Optional CRC16	0	0	0	43	2	0
+ Optional Search	0	0	0	618	11	0
+ Optional Overdrive Speed	0	0	0	45	1	0
+ Optional Parasite Power	0	0	0	42	1	0

Features and Overview

- Requires only two I/O pins to interface multiple slave devices
- Functions provided support reading and writing of both bits and bytes
- Function provided for CRC-8 data integrity checking
- Optional CRC-16 function for iButton® data integrity checking
- Optional functions provided for performing One-Wire search for handling multiple devices
- Optional functions provided for overdrive speed supported by some One-Wire devices
- Optional functions provided for parasite power supported by some One-Wire devices

The OneWire User Module is a set of library routines that read and write data as a master using the Maxim Integrated Products 1-Wire® protocol. A One-Wire master may communicate with one or many slave devices using only one signal wire and a ground. The master initiates all data transfers.

Figure 1. OneWire Block Diagram



Functional Description

The OneWire User Module uses a single I/O pin to interface with One-Wire components over twisted-pair cable. A One-Wire network consists of a master, the wiring, and one or more slave devices. Both master and slave devices are open drain with external pull up. The PSoC I/O should be configured to Resistive Pull up. One-Wire is a proprietary interface developed by Dallas Semiconductor that uses a single wire for both data and power. 1-Wire is a registered trademark of Dallas Semiconductor Corp., a wholly owned subsidiary of Maxim Integrated Products, Inc. For more information on the One-Wire specification, see http://www.maxim-ic.com/.

The user module consists of two digital blocks. The BitClk block provides clock for the XCVR block. The clock is different for different working situations. All BitClk control is performed under user module API control. The XCVR block provides all communication at the receive and transmit level.

Timing

Figure 2. OneWire Timing Diagram

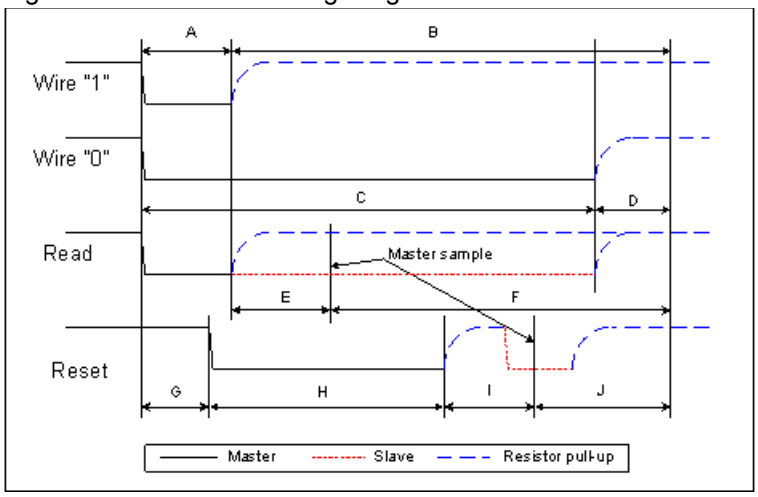


Table 1. Timing Value Table

Parameter	Standard Speed			Overdrive Speed		
	Min (µs)	Typ (µs)	Max (µs)	Min (µs)	Typ (µs)	Max (µs)
A	5	6	15	1	1.5	1.85
B	59	64	-	7.5	7.5	-
C	60	60	120	7	7.5	14
D	8	10	-	2.5	2.5	-
E	5	9	12	0.5	0.75	0.85
F	50	55	-	6.75	7	-
G	0	0	0	2.5	2.5	-
H	480	480	640	68	70	80
I	63	70	78	7.2	8.5	8.8
J	410	410	-	39.5	40	-

Placement

The OneWire User Module is composed of two digital blocks XCVR and BitClk. The XCVR block must be placed in the PSoC digital communication block and the BitClk block may be placed in any digital PSoC block to the adjacent left position of the XCVR block.

Parameters and Resources

Clock

The input clock to the user module. It must be 3 MHz for proper interface timing. The Global I/O buses may be used to connect this 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. One of the divided clocks, VC1, VC2, or VC3, can also be specified as the clock input.

RX

As a general rule, input through the desired bus option to a source of input data. The input can be connected to one of the external pins using a global bus. The PSoC pins connected to user module RX input and TX output should be connected together externally.

TX

The output of the transmitter can be routed to the Global Output Bus. The Global Output Bus can then be connected to an external pin. The pin connected to TX should be configured to pull up mode in order to provide correct user module functioning. The PSoC pins connected to user module RX input and TX output should be externally connected together.

CRC

Enables or disables the optional CRC-16 functions. If disabled, the code for CRC-16 calculations is not generated to save ROM and RAM space.

Search

Enables or disables the optional One-Wire search functions. If disabled, some code is eliminated to save ROM and RAM space.

OverDrive

Enables or disables the optional overdrive speed functions. If disabled, some code is eliminated to save flash and RAM space. Note that not all One-Wire devices support overdrive speed.

ParasitePower

Enables or disables the optional parasite power functions. If disabled, the code is not generated to save ROM and RAM space. Not all One-Wire devices support parasite power. Note: The global interrupt must be enabled to support parasite power. The interrupt for the digital block is automatically enabled by the user module.

ParasitePowerTXPort, ParasitePowerTXPin

These parameters select the TX port and pin if the ParasitePower feature is enabled. Set both to None if the ParasitePower feature is not used.

ClockSync

In 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. 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. 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.

InvertRX

This parameter determines the sense of the input signal. When "Normal" is selected, the input is active-high. Selecting "Invert" causes the sense to be interpreted as active-low.

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:

InterruptAPI

The InterruptAPI parameter allows conditional generation of a user module's interrupt service routine (ISR) and interrupt vector table entry. Select "Enable" to generate the ISR and interrupt vector table entry. Select "Disable" to bypass the generation of the ISR and interrupt vector table entry. Properly selecting this parameter is particularly important for projects that have multiple overlays where a single block resource is shared by the different overlays. By enabling Interrupt API generation only when it is necessary, the need to generate interrupt dispatch code is usually eliminated. This reduces code 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 small amount 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 before 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 may 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.

The API routines allow programmatic control of the OneShot User Module.

OneWire_Start

Description:

Initializes the digital blocks for the OneWire User Module. This function should be called once before the other OneWire functions.

C Prototype:

```
void OneWire_Start(void);
```

Assembler:

```
lcall OneWire_Start
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_Stop

Description:

Stops the functioning of the digital blocks for the OneWire User Module.

C Prototype:

```
void OneWire_Stop(void);
```

Assembler:

```
lcall OneWire_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_fReset

Description:

Reset any attached One-Wire devices and indicates if any devices are present.

C Prototype:

```
BYTE OneWire_fReset(void);
```

Assembler:

```
lcall OneWire_fReset
```

Parameters:

None

Return Value:

A nonzero value is returned if one or more devices are present.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_WriteBit

Description:

Writes a bit to the One-Wire device.

C Prototype:

```
void OneWire_WriteBit(BYTE bDataBit);
```

Assembler:

```
mov  A, [bDataBit]    ; the least significant bit of the byte is written
lcall OneWire_WriteBit
```

Parameters:

bDataBit: The least significant bit of this byte is written to the One-Wire device. The value is passed in Accumulator.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_bReadBit

Description:

Reads a bit from a One-Wire device.

C Prototype:

```
BYTE OneWire_bReadBit(void);
```

Assembler:

```
lcall OneWire_bReadBit
mov  [bDataBit], A
```

Parameters:

None

Return Value:

The LSB of this byte is the value read from a One-Wire device. The result returns through the Accumulator.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_WriteByte

Description:

Writes a byte to the One-Wire devices.

C Prototype:

```
void OneWire_WriteByte( BYTE bData);
```

Assembler:

```
mov  A, [bData]          ; Load byte
lcall OneWire_WriteByte
```

Parameters:

bData: Byte to be written to the One-Wire devices. Passed through Accumulator.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_bReadByte

Description:

Reads a byte from a One-Wire device.

C Prototype:

```
BYTE OneWire_bReadByte(void);
```

Assembler:

```
lcall OneWire_bReadByte
mov  [bData], A
```

Parameters:

None

Return Value:

bData: Byte read from a One-Wire device. Retuned value passed through Accumulator.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_ClearCRC8

Description:

Clear the byte in RAM which holds the current CRC8 value. This function should be called before calling OneWire_bCRC8 when computing a new CRC value.

C Prototype:

```
void OneWire_ClearCRC8(void);
```

Assembler:

```
lcall OneWire_ClearCRC8
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_bCRC8

Description:

Compute a CRC for One-Wire devices. This function should be called for each byte that is part of the CRC value. The current CRC value is returned. Note that the returned CRC value is all zeros when all of the bytes including the CRC from the device are correct and loaded into this function.

C Prototype:

```
BYTE OneWire_bCRC8(BYTE bData);
```

Assembler:

```
mov  A, [bData] ; Load byte to be included in the CRC value.
lcall OneWire_bCRC8
mov  [bCRC],  A ; use returned CRC.
```

Parameters:

bData: Data byte to be part of the CRC value. Passed through accumulator.

Return Value:

bCRC: The current CRC value. Returned through accumulator.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_ClearCRC16

Description:

Clear the bytes in RAM which holds the current CRC-16 value. This function should be called before calling OneWire_wCRC16Value when computing a new CRC value. Note that this function is available only if CRC is set to CRC16 in user module parameters.

C Prototype:

```
void OneWire_ClearCRC16(void);
```

Assembler:

```
lcall OneWire_ClearCRC16 ; lcall function
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_wCRC16Value

Description:

Compute a CRC-16 for One-Wire devices. This function should be called for each byte that is part of the CRC-16 value. The current CRC value is returned. Note that the returned CRC value is zero when all of the bytes including the CRC from the device are correct and loaded into this function. Note that this function is available only if CRC is set to CRC16 in user module parameters.

C Prototype:

```
WORD OneWire_wCRC16Value(BYTE bData);
```

Assembler:

```
mov  A, [bData] ; Load byte to be included in the CRC value.
lcall OneWire_wCRC16Value
mov  [wCRC_LSB], A
mov  [wCRC_MSB], X
```

Parameters:

bData: Data byte to be part of the CRC value. Is passed through accumulator.

Return Value:

The current CRC value. The MSB of the value is located in X and LSB in A registers.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_GetROM

Description:

Each One-Wire device has an 8-byte value called a ROM buffer that contain information about the device. This information includes the device family ID, part ID, and part serial number. All search functions (fFindFirst, fFindNext, fVerify, FamilyTarget setup, FamilySkipSetup) use an 8-byte buffer that to compare with the device ROM buffer. This function copies the search function's ROM buffer into an array of 8 bytes. Note that this function is available only if Search is enabled in user module parameters.

C Prototype:

```
void OneWire_GetROM(BYTE * pbArray);
```

Assembler:

```
mov  A, >pbArray ; Load MSB part of pointer to RAM based array of 8 bytes.
mov  X, <pbArray ; Load LSB part of pointer to RAM based array of 8 bytes.
lcall OneWire_GetROM
```

Parameters:

pbArray: Pointer to an array of an 8 bytes to copy data from the search function.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_SetROM

Description:

Each One-Wire device has an 8-byte value called a ROM buffer that contain information about the device. This information includes the device family ID, part ID, and part serial number. All search functions (fFindFirst, fFindNext, fVerify, FamilyTarget setup, FamilySkipSetup) use an 8-byte buffer that to compare with the device ROM buffer. This function copies an array of 8 bytes into the search function's buffer. Note that this function is available only if Search is Enabled in user module parameters.

C Prototype:

```
void OneWire_SetROM(char * pbArray);
```

Assembler:

```
mov  A, >pbArray ; Load MSB part of pointer to RAM based array of 8 bytes.
mov  X, <pbArray ; Load LSB part of pointer to RAM based array of 8 bytes.
lcall OneWire_SetROM
```

Parameters:

pbArray: Pointer to an array of an 8 bytes to provide data for the search function.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_fFindFirst

Description:

Search for the first One-Wire device. The function initializes the search state and then performs the search. The device ROM number can then be read by the OneWire_GetROM function. Note that this function is available only if Search is Enabled in user module parameters.

C Prototype:

```
BYTE OneWire_fFindFirst(void);
```

Assembler:

```
lcall OneWire_fFindFirst
mov  [fFinded], A
```

Parameters:

None

Return Value:

A nonzero value is returned if a device was found.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_fFindNext

Description:

Search for the next One-Wire device. This function is usually called after a OneWire_fFindFirst or a OneWire_fFindNext. The function performs the search without changing the search state. The device ROM number can then be read by the OneWire_GetROM function. If no additional device is found the function returns a 0 and the search state is set to find the first device the next time a search is performed. Note that this function is available only if Search is Enabled in user module parameters.

C Prototype:

```
BYTE OneWire_fFindNext(void);
```

Assembler:

```
lcall OneWire_fFindNext
mov  [fFinded], A
```

Parameters:

None

Return Value:

A nonzero value is returned if a device was found.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_fVerify

Description:

This function determines if a device with a given ROM number is present. The ROM number to verify is input by the OneWire_SetROM function before calling OneWire_fVerify. If the device is present, the function returns a non zero value. Note that this function is available only if Search is Enabled in user module parameters.

C Prototype:

```
BYTE OneWire_fVerify(void);
```

Assembler:

```
lcall OneWire_fVerify
```

Parameters:

None:

Return Value:

A nonzero value is returned if the device is present.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_FamilyTargetSetup

Description:

This function presets the search state to first find a particular family type. The ROM number of One-Wire devices begins with a one byte family code. The family code lets the One-Wire master determine the capabilities of the connected devices. This function makes it possible to limit searches to a desired family. When OneWire_fFindNext is called, the first device of the family type is found. The device ROM number can then be read by the OneWire_GetROM function. If no devices of the indicated family are present, then another family type is found. Therefore, the family code of the found ROM number must be examined after the search. Note that this function is available only if Search is Enabled in user module parameters.

C Prototype:

```
void OneWire_FamilyTargetSetup(BYTE bFamilyType);
```

Assembler:

```
mov  A, 10h ; Family code to find
lcall OneWire_FamilyTargetSetup
```

Parameters:

bFamilyType: The family code byte to be used in the next search.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_FamilySkipSetup

Description:

This function changes the search state to skip all devices of the family code found from the previous search. This function is only useful after a search. It should be called after a search returns a family code that is to be skipped. The searches that follow find devices that come after the current family code. If there are no other family codes that follow, then the search returns the same family code again. The next call to this function sets the search state so that the next search returns a 0. Therefore, the family code should be examined after each search followed by a call to this function if it is a code to be skipped. Note that this function is available only if Search is Enabled in user module parameters.

C Prototype:

```
void OneWire_FamilySkipSetup(void);
```

Assembler:

```
lcall OneWire_FamilySetup
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_SetOverdrive

Description:

This function changes the bit timing to overdrive speed. Note that this function is available only if Overdrive is Enabled in user module parameters.

C Prototype:

```
void OneWire_SetOverdrive(void);
```

Assembler:

```
lcall OneWire_SetOverdrive
```

Parameters:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_ClearOverdrive

Description:

This function changes the bit timing to normal speed. Call OneWire_Reset after this function to return the One-Wire device(s) to normal speed. Note that this function is available only if Overdrive is Enabled in user module parameters.

C Prototype:

```
void OneWire_ClearOverdrive(void);
```

Assembler:

```
lcall OneWire_ClearOverdrive
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_WriteByteStrong

Description:

This function writes a byte to the One-Wire devices, then immediately sets the output pin to strong mode. This function should be used when devices using parasite power need additional supply current. For example, One-Wire thermometers require additional supply current when performing a temperature conversion. This function switches to strong mode within 10µs of writing the last bit when the CPU clock is set to at least 6 MHz. Note that this function is available only if Parasite Power feature is enabled in user module parameters.

C Prototype:

```
void OneWire_WriteByteStrong( BYTE bData);
```

Assembler:

```
mov  A, 0CCh ; Load byte
lcall OneWire_WriteByteStrong
```

Parameters:

bData: Byte to be written to the One-Wire devices.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

OneWire_SetResistive

Description:

This functions changes the output pin back to resistive Pull Up mode after call OneWire_WriteByteStrong. Note that this function is available only if Parasite Power feature is enabled in user module parameters.

C Prototype:

```
void OneWire_SetResistive(void);
```

Assembler:

```
lcall OneWire_SetResistive
```

Parameters:

None

Return Value:

None

Sample Firmware Source Code

Here is a simple assembly and C example for communicating with a One-Wire device:

```
;;-------------------------------------------------------------------
;; Sample asm OneWire Code
;;
;; Write commands to do a temperature conversion and read the
;; temperature for a single DS18S20 device.
;;-------------------------------------------------------------------

include "m8c.inc"
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

_main:
call OneWire_Start          ; Initialize 1-Wire pin
Loop:
call OneWire_fReset         ; reset the 1-Wire device
mov A, CCh                  ; "skip ROM command"
call OneWire_WriteByte
mov A, 44h                  ; "convert temperature" command
call OneWire_WriteByte

; Wait at least 750 ms for temperature conversion finish

call OneWire_fReset         ; reset the 1-Wire device
mov A, CCh                  ; "skip ROM" command
call OneWire_WriteByte
```

```
mov A, BEh                  ; "read scratchpad" command
call OneWire_WriteByte
call OneWire_bReadByte      ; read temperature LSB
mov REG[PRT1DR], A          ; write temperature to Port 1
jmp Loop

ret
```

The same functionality using C language:

```
//--------------------------------------------------------------------
// Sample C code for OneWire
//
// Write commands to do a temperature conversion and read the
// temperature for a single DS18S20 device.
//--------------------------------------------------------------------
#include "OneWire.h"
#include "PSoCAPI.h"

void main(void) {
OneWire_Start();           // Initialize 1-Wire pin
for (;;) {
OneWire_fReset();        // reset the 1-Wire device
OneWire_WriteByte(0xCC); // "skip ROM" command
OneWire_WriteByte(0x44); // "convert temperature" command

// Wait at least 750 ms for temperature conversion finish


OneWire_fReset();        // reset the 1-Wire device
OneWire_WriteByte(0xCC); // "skip ROM" command
OneWire_WriteByte(0xBE); // "read scratchpad" command
PRT1DR = OneWire_bReadByte();  // read temeperature LSBa and send to P1
  }
}
```

Configuration Registers

The One-Wire uses two digital PSoC block named BitClk and XCVR. Each block is personalized and parameterized through 7 registers. The following tables give the “personality? values as constants and the parameters as named bit-fields with brief descriptions. Symbolic names for these registers are defined in the user module instance’s C and assembly language interface files (the *.h* and *.inc* files).

Table 2. Block BitClk, Function Register (DxBxxFN), Bank1

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

BCEN gates the output onto the row broadcast bus line. This bit field is set in the Device Editor by directly configuring the broadcast line

Table 3. Block BitClk, Input Register (DxBxxIN), Bank1

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

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

Table 4. Block BitClk, Output Register (DxBxxOU), Bank1

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

The user module “ClockSync” parameter in the Device Editor determines the value of ClockSynk bits.

Table 5. Block BitClk, Control Register (DxBxxCR0), Bank0

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

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

Table 6. Block BitClk, Counter Register (DxBxxDR0), Bank0

Bit	7	6	5	4	3	2	1	0
Value	Down Counter Register							

Counter register holds current Counter value and is not directly readable or writeble.

Table 7. Block BitClk, Period Register (DxBxxDR1), Bank0

Bit	7	6	5	4	3	2	1	0
Value	Period							

Period register is controlled over User Module API according to current operation type.

Table 8. Block BitClk, Compare Register (DxBxxDR2), Bank0

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

Compare value. Is not used in current implementation.

Table 9. Block XCVR, Function Register (DCBxxFN), Bank1

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

The InvertRX flag, set through a user module parameter displayed in the Device Editor, controls the sense of the RX input signal. BCEN gates the output onto the row broadcast bus line. This bit field is set in the Device Editor by directly configuring the broadcast line.

Table 10. Block XCVR, Input Register (DCBxxIN), Bank1

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

RX selects the input signal of the same name from one of 16 sources. The user module “RX” parameter setting in the Device Editor determines its value.

Table 11. Block XCVR, Output Register (DCBxxOU), Bank1

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

The user module “ClockSync” parameter in the Device Editor determines the value of ClockSynk bits. TX selects output from one of four global busses. This parameter is set in the Device Editor.

Table 12. Block XCVR, Control Register (DCBxxCR0), Bank0

Bit	7	6	5	4	3	2	1	0
Value	0	Overrun	SPIComplete	TXRegEmpty	RXRegFull	0	0	Start/Stop

Overrun flag indicates that RXBuffer register value was overwritten. SPIComplete flag indicates that last data byte was shifted out and all associated clocks was generated. TXRegEmpty flag indicates that previous data byte transferred to Shift register and TXBuffer register is empty. RXRegFull flag indicates that new data byte is received and stored in RXBuffer register. Start/Stop indicates that the OneWire is enabled when set. It is modified using the OneWire API.

Table 13. Block XCVR, SPI Shift Register (DCBxxDR0), Bank0

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

The SPI shift register. This register is not directly readable or writable.

Table 14. Block XCVR, TX Buffer Register (DCBxxDR1), Bank0

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

Transmitter Buffer register.

Table 15. Block XCVR, RX Buffer Register (DCBxxDR2), Bank0

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

Receiver Buffer register.

Version History

Version	Originator	Description
1.1	TDU	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. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2008-2014 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.