PSoC® 3 and PSoC 5 – SFF-8485 Serial GPIO (SGPIO) Initiator Interface

AN66019

Author: Jason Konstas

Associated Project: Yes
Associated Part Family: All PSoC 3 and PSoC 5 parts
Software Version: PSoC Creator™ 1.0
Associated Application Notes: AN64315

Application Note Abstract

The SGPIO Initiator component enables designers using PSoC 3 or PSoC 5 to add a Serial General Purpose Input/Output (SGPIO) Initiator interface compliant with the Small Form Factor (SFF) Committee SFF-8485 industry standard specification. The SGPIO interface is used in Serial Attached Small Computer System Interface (SAS) and Serial Advanced Technology Attachment (SATA) hard drive enclosure systems. SGPIO is a unique, custom, 4-wire protocol that can not be found in any off-the-shelf microcontroller product. PSoC 3's ability to implement custom interface logic in programmable universal digital blocks makes it an ideal choice for this application.

Introduction

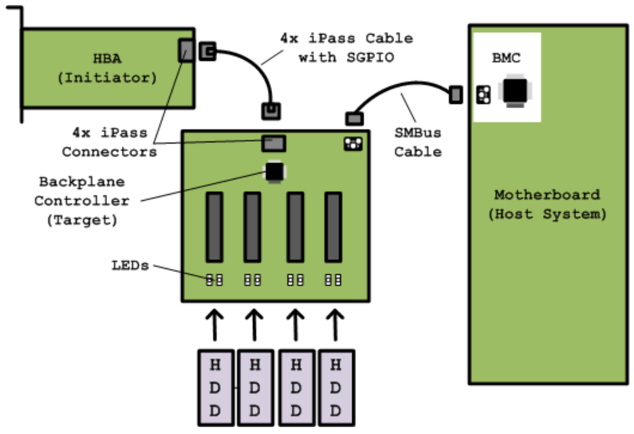
In storage systems that employ a multitude of hard disk drives using high-speed serial data communication interfaces such as SAS or SATA, there is a need for many general purpose IOs (GPIO) in the system to communicate drive status and to drive LED indicators. To reduce cable bulk, the SFF committee defined an industry standard protocol to serialize those GPIOs in these applications. While the SFF-8485 specification does not limit the number of hard drives supported by a single SGPIO bus link, most implementations support four drives through a 12-bit protocol (3 bits in both directions per drive).

The SGPIO_Initiator component makes it extremely easy for designers developing HBAs with PSoC 3 or 5 to add multiple SPGIO Initiator interfaces to a single device, enabling a PSoC based HBA supporting many hard drives. The SGPIO_Initiator implementation is completely hardware based, freeing the CPU in PSoC to implement management protocols such as Intelligent Platform Management Bus/Bridge (IPMB) and the International Blinking Pattern Interpretation (IBPI) defined in SFF-8489 required in these applications.

Typical System Storage Application

A typical system storage application block diagram is shown in Figure 1. The backplane PCB contains the backplane controller, connectors for plugging hard drives and communications cables to the HBAs and the baseboard management controller (BMC), the system host.

Figure 1 Typical System Storage Architecture
(image source: http://en.wikipedia.org/wiki/IBPI)



In this example system, four hard drives connect to the backplane. The backplane controller is able to detect the presence of these drives and send rudimentary status information over the SGPIO bus to the HBA responsible for managing the attached drives. The SGPIO Initiator on the HBA receives that information and sends back LED indicator information (on, off, and blinking rates) in accordance with the IBPI standard to drive the LEDs on the backplane. The BMC controller is the system-wide host performing management tasks and is located on the host motherboard connected to the backplane via an I^{2}C/System Management Bus (SMBus) physical link.

The PSoC 3 family is an ideal choice for the HBA as it has the programmable digital resources available to implement multiple SGPIO Initiator interfaces in hardware, as well as providing additional I^{2}C or other industry standard interfaces to hard drive controllers together with a high-performance 8051 (PSoC 3) or ARM Cortex M3 (PSoC 5) CPU core for implementing management protocols.

This application note will focus on how to use the SGPIO_Initiator component in PSoC by working through some example projects on the CY8CKIT-001 PSoC Development Kit (DVK). The examples will show how to support the SFF-8485 specification and get data on and off the SGPIO link reliably. Supporting management protocols and higher level functionality is beyond the scope of this application note.

Introduction to the SGPIO Protocol

SGPIO uses a 4-wire protocol as shown in Figure 2. SCLOCK, SLOAD and SDATAOUT are all driven by the SGPIO Initiator. SDATAIN is driven by the SGPIO Target.

SCLOCK is a reference clock provided by the Initiator up to a maximum frequency of 100 kHz.

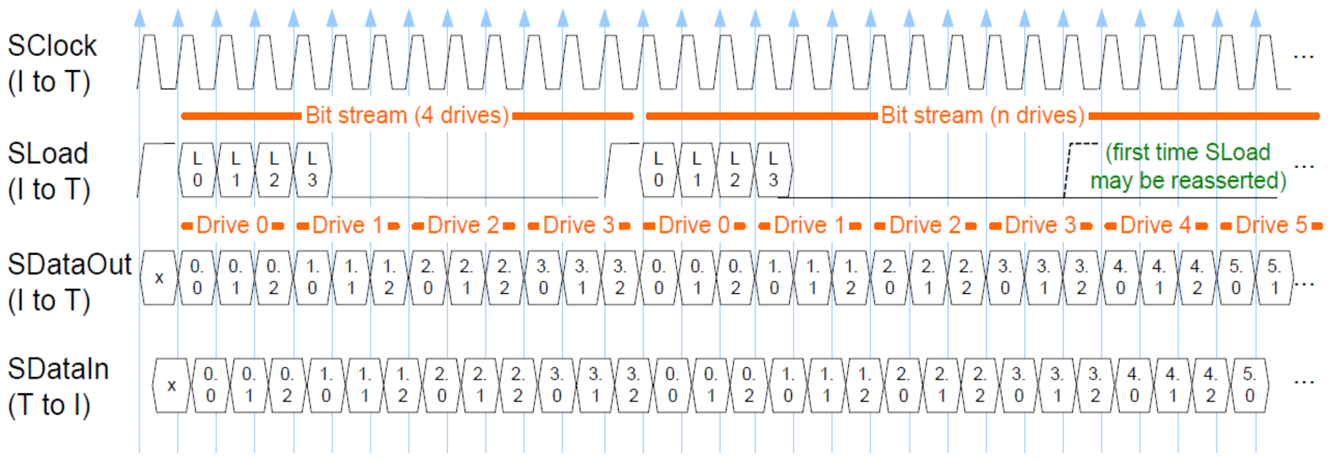
SLOAD is primarily used as a reference synchronization pulse to indicate the start of each serial frame. Since each frame carries 3 bits per drive, the typical frame length is 12 bits long supporting four drives. Therefore, the SLOAD pulse is driven high for 1 bit period every 12 bits concurrent with the last bit period of the current frame.

SLOAD is also optionally used to communicate four bits of vendor specific information (L3:0), immediately following the synchronization pulse. The SGPIO Target should look for a pattern of at least five successive logic zero levels on the SLOAD pulse, followed immediately by a logic one level to guarantee synchronization to the SLOAD pulse and to avoid false synchronization on the L3:0 vendor specific information.

SDATAOUT is the LED indicator information (3 bits per drive) sent by the Initiator. It is clocked out of the Initiator on the rising edges of SCLOCK and the target should capture the information on the falling edges of SCLOCK.

SDATAIN is the drive status information (3 bits per drive) sent by the Target. It is also clocked out of the Target on the rising edges of SCLOCK and captured by the Initiator on the falling edges of SCLOCK.

Figure 2. SGPIO Protocol Summary



Getting Started with the SGPIO_Initiator Component

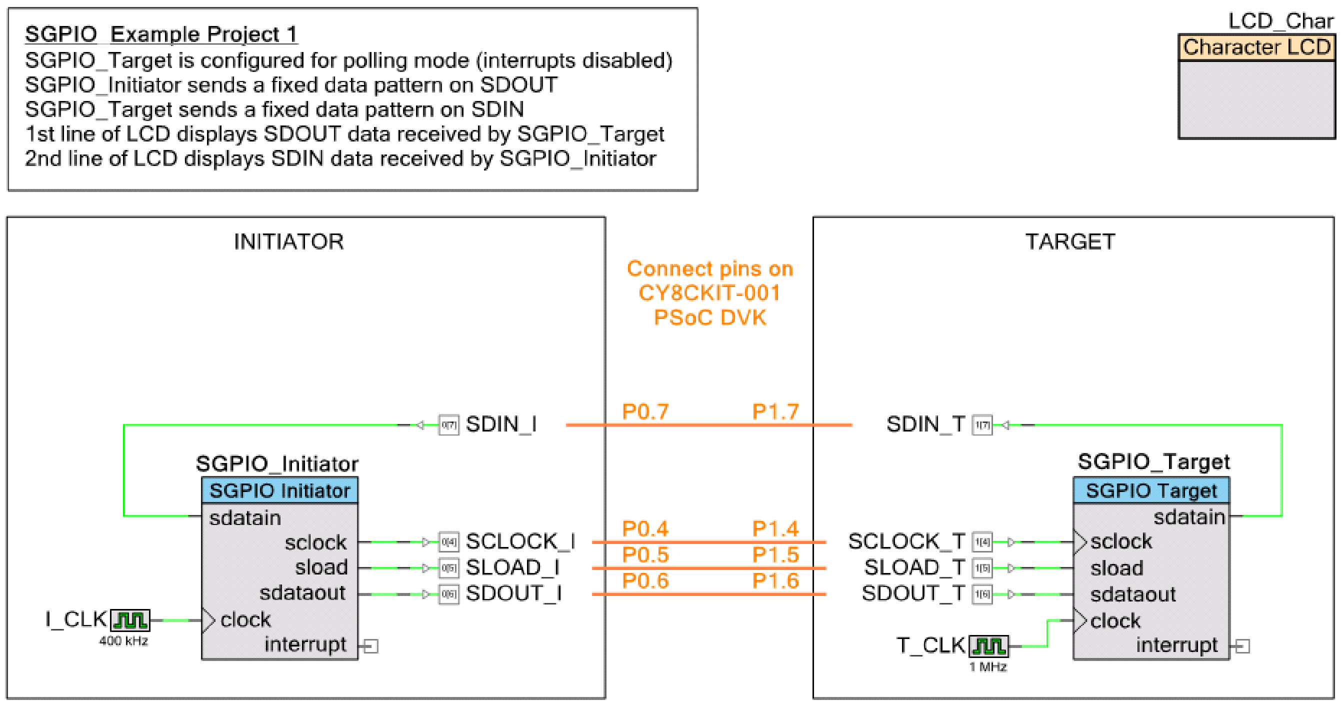
Distributed with this application note is a PSoC Creator™ bundled project ZIP file that contains three example design projects and a library project that contains the **SGPIO_Initiator** component and the **SGPIO_Target** companion component for validation purposes. Save the ZIP file to a convenient location on your hard drive and extract the contents to a local folder.

To get started with the example projects, double-click the *AN66019.cywrk* PSoC Creator workspace file.

1. In the **Workspace Explorer** tab to the left of the screen, expand project **Example1** by clicking on the small '+' icon to the left of it.
2. Double-click *TopDesign.cysch* to open the top-level design schematic for the hardware blocks inside PSoC.

Figure 3 shows the top-level design schematic for Example Project 1.

Figure 3. Top-level Design Schematic for Example Project 1



SGPIO Example Project 1
SGPIO_Target is configured for polling mode (interrupts disabled)
SGPIO_Initiator sends a fixed data pattern on SDOUT
SGPIO_Target sends a fixed data pattern on SDIN
1st line of LCD displays SDOUT data received by SGPIO_Target
2nd line of LCD displays SDIN data received by SGPIO_Initiator

To the right of the schematic is the PSoC Creator **Component Catalog**. The tab labeled **Cypress** contains the default library of Cypress-provided content. The tab labeled **Reference** contains the **SGPIO_Initiator** and the **SGPIO_Target** components provided with this application note as part of the bundled example projects. The **Solutions** library can be reused and imported into any PSoC Creator design project that requires the **SGPIO_Initiator** component functionality.

All of the example projects provided with this application note are designed to run on the PSoC DVK. Each example project contains the SGPIO_Target component used to exercise and verify the functionality of the SGPIO_Initiator component in loopback mode.

To replicate this environment properly, some wire connections are required on the PSoC DVK as shown in Figure 3. Using header **P19** on the DVK, make the four connections shown in orange color in Figure 3.

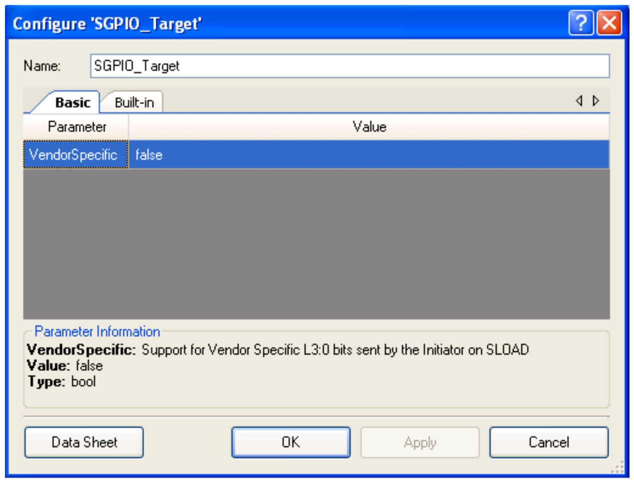
Pay close attention to the clock sources chosen for both components.

- The SGPIO_Initiator input clock (labeled **I_CLK**) must be set to 4x the desired SGPIO SCLOCK frequency. To run SCLOCK at the maximum 100 kHz, set I_CLK to 400 kHz.
- The SGPIO_Target input clock (labeled **T_CLK**) should be set to at least 8x the SCLOCK frequency. 1 MHz was chosen to satisfy this requirement.

Example 1

Double-click the **SGPIO_Target** component to open the component customizer as shown in Figure 4.

Figure 4. SGPIO_Target Component Customizer



Configure 'SGPIO_Target'

Name: SGPIO_Target

Parameter	Value
VendorSpecific	false

Parameter Information
VendorSpecific: Support for Vendor Specific L3:0 bits sent by the Initiator on SLOAD
Value: false
Type: bool

For the first example project, ensure that the **VendorSpecific** parameter is set to **false**. This will disable support for the L3:0 vendor specific information optionally transmitted over the SLOAD signal by the Initiator. Click **OK** to close the customizer.

Program the Example1 project into the DVK by selecting **Debug > Program** from the pulldown menus. If the project is running correctly, the text displayed on the debug LCD should read:

```
Tgt: SDOUT=029C
Int:  SDIN=08D1
```

In the **Workspace Explorer**, double-click *main.c* to examine the firmware used in this example project. The code is shown in Code 1.

Code 1. Example1 *main.c* Firmware Listing

```
#define DUMMY_SDOUT 0x029C      /* Drive0=4, Drive1=3, Drive2=2, Drive3=1 */
#define DUMMY_SDIN  0x08D1      /* Drive0=1, Drive1=2, Drive2=3, Drive3=4 */

uint16  sdataIn, sdataOut;

void main()
{
    /* Initialize the Debug LCD display */
    LCD_Char_Start();
    LCD_Char_Position(0,0);
    LCD_Char_PrintString("Tgt: SDOUT=");
    LCD_Char_Position(1,0);
    LCD_Char_PrintString("Int:  SDIN=");

    /* Initialize the SGPIO Components */
    SGPIO_Target_Start();
    SGPIO_Initiator_Start();
    SGPIO_Initiator_WriteTxData(DUMMY_SDOUT);

    for(;;)
    {
        /* SGPIO_Initiator Processing */
        while(!SGPIO_Initiator_GetIntStatus());
        sdataIn = SGPIO_Initiator_ReadRxData();              /* Read new SDIN data from Target */
        SGPIO_Initiator_WriteTxData(DUMMY_SDOUT);            /* Reload dummy Initiator SDOUT test data */

        /* SGPIO_Target Processing */
        while(!SGPIO_Target_GetIntStatus());
        sdataOut = SGPIO_Target_ReadRxData();        /* Read new SDOUT from the Initiator */
        SGPIO_Target_WriteTxData(DUMMY_SDIN);        /* Reload dummy Target SDIN test data */

        /* Display results on LCD */
        LCD_Char_Position(0,11);
        LCD_Char_PrintHexUint16(sdataOut);
        LCD_Char_Position(1,11);
        LCD_Char_PrintHexUint16(sdataIn);

        /* Reset the Component Buffers */
        /* The slowness of LCD updates will have caused FIFO buffer overflows */
        SGPIO_Initiator_ClearRxBuffer();
        SGPIO_Target_ClearRxBuffer();
    }
}
```

In the Code 1 excerpt, all lines of code that start with the prefix **SGPIO_Initiator** reference APIs provided by Cypress for that component. Refer to the component datasheet for a full list of the provided APIs.

In Example 1, the CPU polls the interrupt status of the SGPIO_Initiator component waiting for new SDIN data to arrive from the Target. When this event is detected, the SGPIO_Initiator reads the SDIN data and loads the transmitter FIFO with dummy data to send to the Target during the next frame. The same procedure is used to read/write data to/from the SGPIO_Target component.

Since SGPIO is a full duplex communication link, transmitting and receiving data simultaneously, it is always necessary to write new transmit data to the SGPIO components whenever new receive data is read.

The LCD displays the SDOUT data received by the Target on the top line and displays the SDIN data received by the Initiator on the second line.

Congratulations! You just completed your first SGPIO Initiator design with PSoC. If the data displayed on your LCD reads `0000` for both Target and Initiator, check the DVK external wiring connections you made earlier. Refer to Figure 3 for the correct connection scheme.

Example2 – Using Interrupts

While the SGPIO_Initiator component supports the polling methodology used in Example1, it is recommended to use interrupts to service the SGPIO_Initiator interface due to the real-time sensitivity of handling the interface. This is particularly true if the CPU is burdened with supporting higher level management protocols such as IPMB or IBPI.

Close *main.c* and *TopDesign.cysch* for Example1. Go back to the Workspace Explorer and right click on **Project 'Example2'.** Select **Set As Active Project**.

Open *TopDesign.cysch* for Example2 to start working on this project. The only change made to the top level design schematic is the addition of interrupt components, retrieved from the standard **Cypress Component Catalog**. For the Example2 project, double click the SGPIO_Target component to open the Component Customizer and ensure that the **VendorSpecific** parameter is set to **true**.

Right click on **Project 'Example2'** and **choose Set As Active Project** then program the Example2 project into the DVK by selecting **Debug > Program** from the pull-down menus.

If the project is running correctly, the text displayed on the debug LCD should read:

```
Tgt: D=0123 V=0A
Int: D=0EDC
```

In the **Workspace Explorer**, double click on *main.c, INITIATOR_INT.c* and *TARGET_INT.c* to examine the firmware used in this example project. The code is shown in code excerpts Code 2, Code 3, and Code 4.

The first thing to notice in the second example is that the code in *main.c* only handles the LCD display. Handling of the SGPIO_Initiator component data transfers is done entirely in the **INITIATOR_INT** interrupt service routine. Similarly, handling of the SGPIO_Target component data transfers is done entirely in the **TARGET_INT** interrupt service routine.

Code 2. Example2 *main.c* Firmware Listing

```
#include <device.h>

/* Dummy Test Patterns */
#define DUMMY_SDOUT    0x0123
#define DUMMY_VENDOR   0x0A
uint16 sdataIn, sdataOut, vendorL30;

void main()
{
    /* Initialize the Debug LCD Display */
    LCD_Char_Start();
    LCD_Char_Position(0,0);
    LCD_Char_PrintString("Tgt: D=     V=");
    LCD_Char_Position(1,0);
    LCD_Char_PrintString("Int: D=");

    /* Enable Interrupts in the System */
    CYGlobalIntEnable;
    TARGET_INT_Start();
    INITIATOR_INT_Start();

    /* Initialize the SGPIO Components */
    SGPIO_Target_Start();
    SGPIO_Target_EnableInt();
    SGPIO_Initiator_Start();
    SGPIO_Initiator_EnableInt();
    SGPIO_Initiator_WriteVendorSpecific(vendorL30);
    SGPIO_Initiator_WriteTxData(sdataOut);

    for(;;)
    {
        /* Display results on LCD */
        LCD_Char_Position(0,7);
        LCD_Char_PrintHexUint16(sdataOut);
        LCD_Char_Position(0,14);
        LCD_Char_PrintHexUint8(vendorL30);
        LCD_Char_Position(1,7);
        LCD_Char_PrintHexUint16(sdataIn);
    }
}
```

Code 3. Example2 INITIATOR_INT.c Firmware Listing

```
CY_ISR(INITIATOR_INT_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START INITIATOR_INT_Interrupt` */

    // Read SDIN data from the Rx FIFO //
    sdataIn = SGPIO_Initiator_ReadRxData();
    /* Load SDOUT data for transmission on the next frame */
    SGPIO_Initiator_WriteTxData(sdataOut);
/* `#END` */
```

Code 4. Example2 *TARGET_INT.c* Firmware Listing

```
CY_ISR(TARGET_INT_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START TARGET_INT_Interrupt` */

    /* Read SDOUT data and vendor specific L3:0 data from the Rx FIFO */
    sdataOut  = SGPIO_Target_ReadRxData();
    vendorL30 = SGPIO_Target_ReadRxData();

    /* Load the inverse of what was received for transmission out on SDIN next frame */
    SGPIO_Target_WriteTxData(~sdataOut);
/* `#END` */
```

Example 3 – Add Support for SGPIO Bus Timeout

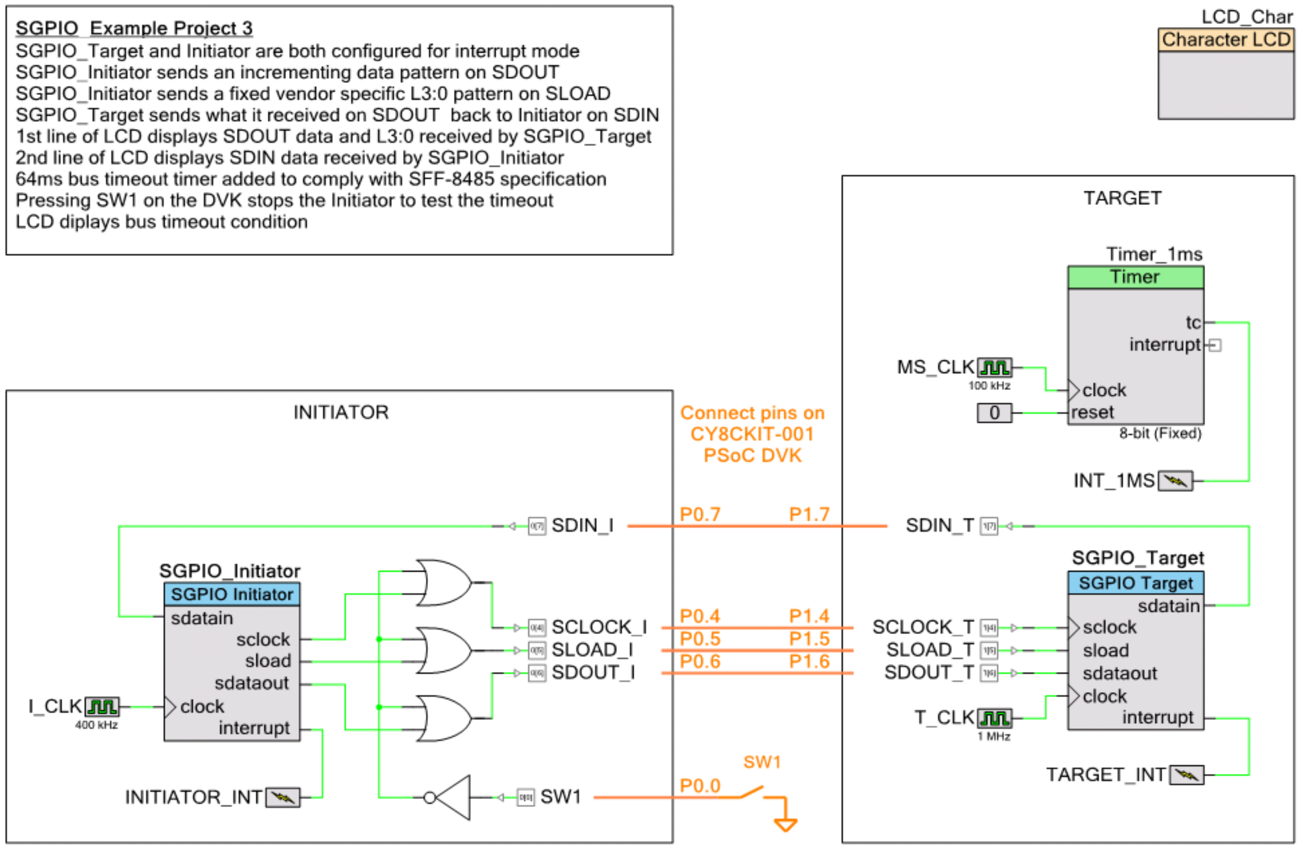
The SFF-8485 SGPIO specification requires that the SGPIO Target detect a bus timeout condition typically caused by the removal of the cable that connects to the HBA. In that case, SCLOCK, SLOAD, and SDATAOUT will all be pulled high using 2 kΩ pull-up resistors on the backplane PCB. If the Target detects all three inputs are high for 64 msec, then all LED indicators should be turned off.

The SGPIO_Target component does not handle higher-level functions such as turning LEDs on or off. However, it does provide the mechanism for detecting the bus timeout condition.

Close *main.c* and *TopDesign.cysch* for Example2. Go back to the Workspace Explorer and right click on **Project 'Example3'.** Select **Set As Active Project**.

Open *TopDesign.cysch* for Example3 as shown in Figure 5 to start working on this project.

Figure 5. Top-level Design Schematic for Example Project 3


SGPIO Example Project 3
SGPIO_Target and Initiator are both configured for interrupt mode
SGPIO_Initiator sends an incrementing data pattern on SDOUT
SGPIO_Initiator sends a fixed vendor specific L3:0 pattern on SLOAD
SGPIO_Target sends what it received on SDOUT back to Initiator on SDIN
1st line of LCD displays SDOUT data and L3:0 received by SGPIO_Target
2nd line of LCD displays SDIN data received by SGPIO_Initiator
64ms bus timeout timer added to comply with SFF-8485 specification
Pressing SW1 on the DVK stops the Initiator to test the timeout
LCD diplays bus timeout condition
LCD_Char
Character LCD
TARGET
Timer_1ms
Timer
tc
interrupt
MS_CLK
100 kHz
clock
reset
0
8-bit (Fixed)
INT_1MS
INITIATOR
Connect pins on CY8CKIT-001 PSoC DVK
P0.7
P1.7
SDIN_I
SDIN_T
SGPIO_Initiator
SGPIO Initiator
sdatain
sclock
sload
sdataout
clock
interrupt
I_CLK
400 kHz
SCLOCK_I
SLOAD_I
SDOUT_I
P0.4
P0.5
P0.6
P1.4
P1.5
P1.6
SCLOCK_T
SLOAD_T
SDOUT_T
SGPIO_Target
SGPIO Target
sdatain
sclock
sload
sdataout
clock
interrupt
T_CLK
1 MHz
TARGET_INT
SW1
P0.0
SW1
INITIATOR_INT

Two major changes have been made to the top-level design schematic for this example. First, the SGPIO_Initiator circuit has been modified such that pressing the switch labeled **SW1** on the PSoC DVK will force all three output signals from the SGPIO_Initiator high. This will be used to test the Target's ability to detect the bus timeout condition. To support this example, add another wire on the DVK from **P19 P0_0** to **P14 SW1**.

The second change made to the schematic is the addition of a 1 ms Timer component from the standard Cypress Component Catalog. The Timer used in this example is clocked from a 100 kHz clock source (provided by PSoC's versatile internal clock divider subsystem) and has been configured with a period of 100 to generate a terminal count interrupt every 1 ms. Support for the 64 ms bus timeout will be handled in firmware using provided APIs.

Program the Example3 project into the DVK by selecting **Debug > Program** from the pull-down menus. If the project is running correctly, the text displayed on the debug LCD should look similar to Example2 with the exception that the data packets sent and received over the SGPIO bus will increment periodically.

Press and hold the switch labeled **SW1** on the CY8CKIT-001 DVK to force SCLOCK, SLOAD, and SDATAOUT high to confirm that the Target can detect the fault.

When the bus timeout fault is detected for 64 msec, the text displayed on the debug LCD should read:

```
SGPIO TIMEOUT
```

Release the switch to resume normal operation.

In the Workspace Explorer, double click on *main.c* and *INT_1MS.c* to examine the firmware used in this example project. Those excerpts are shown below in Code 5 and Code 6. The code in **TARGET_INT.c** has been modified to return the same data received (no longer inverted) and is not shown in this application note.

Code 5. Example3 *main.c* Firmware Listing

```
#include <device.h>

#define DUMMY_SDOUT     0x0123
#define DUMMY_VENDOR    0x05

uint16  sdataIn;
uint16  sdataOut  = DUMMY_SDOUT;
uint16  vendorL30 = DUMMY_VENDOR;
uint8   timeout64Counter = 0;
uint8   timeout64Flag = 0;
uint16  testData = 0;

void main()
{
    uint8 delayLoop = 0;

    /* Initialize the Debug LCD Display */
    LCD_Char_Start();
    LCD_Char_Position(0,0);
    LCD_Char_PrintString("Tgt: D=     V=");
    LCD_Char_Position(1,0);
    LCD_Char_PrintString("Int: D=");

    /* Start the Millisecond Timer Component */
    Timer_1ms_Start();

    /* Enable Interrupts in the System */
    CYGlobalIntEnable;
    TARGET_INT_Start();
    INITIATOR_INT_Start();
    INT_1MS_Start();

    /* Initialize the SGPIO Components */
    SGPIO_Target_Start();
    SGPIO_Target_EnableInt();
    SGPIO_Initiator_Start();
    SGPIO_Initiator_EnableInt();
    SGPIO_Initiator_WriteVendorSpecific(vendorL30);
    SGPIO_Initiator_WriteTxData(testData);


        for(;;)
            {
                // Change the Dummy Test Data Periodically //
                if (++Delay_Loop == 255)
                    Test_Data++;

                // Display results on LCD //
                if (timeout64Flag)            {
                    // Display timeout message //
                    LCD_Char_Position(0,0);
                    LCD_Char_PrintString("SGPIO TIMEOUT   ");
                    LCD_Char_Position(1,0);
                    LCD_Char_PrintString("                ");
                    timeout64Flag = 0;

                    // Wait for bus activity to resume and then restore LCD display //
                    SGPIO_Target_ResetTimeout();
                    while (SGPIO_Target_GetTimeoutStatus()){};
                    LCD_Char_Position(0,0);
                    LCD_Char_PrintString("Tgt: D=     V=");
                    LCD_Char_Position(1,0);
                    LCD_Char_PrintString("Int: D=");
                    SGPIO_Target_Start();
                    SGPIO_Target_EnableInt();
                }
                else
                {
                    LCD_Char_Position(0,7);
                    LCD_Char_PrintHexUint16(sdataOut);
                    LCD_Char_Position(0,14);
                    LCD_Char_PrintHexUint8(vendorL30);
                    LCD_Char_Position(1,7);
                    LCD_Char_PrintHexUint16(sdataIn);
                }
            }
        }
```

Code 6. Example3 *INT_1MS.c* Firmware Listing

```
CY_ISR(INT_1MS_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START INT_1MS_Interrupt` */

    /* Check for SGPIO bus activity since the last 1ms interrupt */
    if (SGPIO_Target_GetTimeoutStatus())
    {
        /* No bus activity detected */
        if (++timeout64Counter > 63)
        {
            timeout64Counter = 63;
            timeout64Flag = 1;
            SGPIO_Target_Stop();
        }
    }
    else
    {
        /* Bus activity detected */
        timeout64Counter = 0;
    }

    /* Reset the SGPIO bus activity detection logic */
    SGPIO_Target_ResetTimeout();

    /* `#END` */
}
```

The INT_1MS code excerpt demonstrates how to use the SGPIO_Target APIs to detect the bus timeout condition. The custom hardware implementation of the SGPIO_Target component includes a sticky bit that goes to zero whenever bus activity is detected. This sticky bit is re-armed for activity detection by calling the `SGPIO_Target_ResetTimeout()` API. In this example, that API is called every millisecond in this interrupt service routine. On subsequent millisecond interrupts, the status of that sticky bit is checked using the `SGPIO_Target_GetTimeoutStatus()` API.

That API returns a zero if there has been activity detected on the SGPIO bus and returns a non-zero value if no activity has been detected.

The 1 ms interrupt service routine increments the `timeout64Counter` variable every time 1 ms if no activity has been detected. If that variable exceeds 63 counts, then 64 msec of no activity has been detected on the SGPIO bus. The example shown here stops the SGPIO_Target component and sets a flag called `timeout64Flag` for the main loop to handle.

Referring back to the Code 5 excerpt, the main processing loop detects the timeout flag and reuses the same API calls to wait until SGPIO bus activity resumes (when the **SW1** switch is released in this example project). When bus activity does resume, the SGPIO_Target component is restarted and normal operation resumes.

One benefit of implementing the 64 ms bus timeout using this firmware/hardware combination is that a single millisecond timer and interrupt service routine can be used to monitor many SGPIO Target interfaces in the same PSoC device. Each SGPIO_Target component consumes two of PSoC's universal digital blocks (UDBs), enabling one PSoC 3 device to support up to 12 SGPIO_Target interface components in a single chip.

Congratulations! You have just completed an SGPIO Initiator and Target loopback design using PSoC 3 that fully implements all features specified by the SFF-8485 Serial GPIO specification.

Going Forward

The PSoC Creator library project provided along with this application note can be shared freely with other designers and reused in your own PSoC Creator design projects. To make use of this feature, copy the **SGPIOLibrary** folder that was provided with the example projects to a convenient location on your hard drive or shared network.

When you begin your next SGPIO design project in PSoC Creator, the **Solutions** Component Catalog will not be automatically available to you.

To gain access to the SGPIO_Initiator component, select **Project > Dependencies** from the pull-down menus. Under **User Dependencies** click on the folder icon and navigate to the location of the *SGPIOLibrary.cyprj* file.

Select the **Components** and **Code** check boxes and click **OK**. The Solutions component catalog will now be available in your design project and the SGPIO_Initiator and SGPIO_Target components will be available to use in your new project.

Summary

The SGPIO_Initiator component enables designers to quickly and easily design HBAs for system storage applications providing support for many hard drives. PSoC’s custom logic capability enables designers to implement custom protocol interfaces like SGPIO without the issue of firmware implementations such as bit-banging with complex and difficult timing requirements. PSoC’s unique ability to combine custom digital logic, analog signal chain processing and an MCU in a single device enables system designers to integrate many external fixed-function ASSPs. This powerful integration capability not only reduces BOM cost, but also results in PCB board layouts that are less congested and more reliable.

Document History

Document Title: PSoC® 3 and PSoC 5 – SFF-8485 Serial GPIO (SGPIO) Initiator Interface - AN66019

Document Number: 001-66019

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3122236	JK	12/28/2010	New Spec – Initial Release
*A	3265814	JK	07/03/2011	Added support for PSoC 5. Changed application note title. Updated the example projects to use SGPIO_Target Component v1.20 and SGPIO_Initiator Component v1.10
*B	3323585	JK	07/25/2011	Updated AN# number in the header throughout the document.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/