Application Note: Robotic Arm Sample Manipulator Example

Michael Klopfer, Crystal Lai, Zihan (Bronco) Chen, Jonathan Tang, & G.P. Li

UCI Microsemi Innovation Laboratory

California Institute for Telecommunications and Information Technology (Calit2)

University of California, Irvine (UC Irvine)

Tim McCarthy

Microsemi Programmable Solutions Division Products Division

Introduction

In the presented application example, a Microsemi SmartFusion 2 system-on-chip (SoC) which is comprised of a FPGA + ARM Cortex M3 microcontroller is used to provide precise communication and control for a basic automation and robotics application. The integrated design presented incorporates numerous common design elements and serves as a template for similar robotic and motion control designs with asynchronous serial control interfacing. The application demonstrated can provide a start for a PWM based motor control or actuator driver. SmartFusion2 system-on-chip is a field programmable gate array with an on-die, hardware implemented ARM Cortex M3 microcontroller. The presented application example was developed using the Future Electronics Creative Demo Board. This Future Electronics product has two versions, including the SmartFusion2 (M2S025) [Future Electronics P/N: FUTUREM2SF-EVB] and the IGLOO2 (M2GL025) [Future Electronics P/N: FUTUREM2GL-EVB]. Both are Microsemi 4th generation flash products. The Microsemi SmartFusion 2 includes a hardware design based on an integrated ARM Cortex M3 microcontroller and FPGA fabric. The IGLOO2 (M2GL025) features an FPGA without the microcontroller [Future Electronics P/N: FUTUREM2GL-EVB]. The Future Electronics Creative Demo Board product line is focused toward end-application R&D, educational, and hobby markets and form a low price point entry into developing applications with mid-range Microsemi FPGA solutions.

Design Overview

In this application example, a 6-axis robotic arm is constructed using hobby-grade PWM digital control servos. The completed device can function as an example laboratory sample manipulation tool. This device is driven by an FPGA application deployed on the low-cost Future Electronics Creative Board (SmartFusion 2). Control of motion could be implemented in full in the FPGA or with the use of soft processors such as RISC-V, but in this example, the authors intended to demonstrate an application heavily reliant on interfacing between the microcontroller subsystem (MSS) and the FPGA fabric to illustrate the flexibility and power of the SmartFusion2 design. The FPGA fabric design includes a PWM controller, a timer, a GPIO interface, and a UART controller. The PWM controller provides servo control, a MSS driven timer is used to schedule "heartbeat" messages to check for an active link, the GPIO interface provides program control via on-PCB buttons and enables three LED diagnostic indicators, and the UART controller provides asynchronous serial (UART) communications to control program execution. In this application, an Android application provides commands for triggering operation of built-in motion routines. The hardware timer schedules transmission of the "heartbeat" over UART to verify an active communication link. A UART to Bluetooth bridge (Sparkfun Bluetooth Mate Gold) and an example Android application are used to provide motion cues and provide wireless control of the motion of the laboratory sample manipulation tool.

- *Design Target Silicon: SmartFusion2 (M2S025-VF256)*
- *Design Target Board: Future Electronics Creative Board – SmartFusion2 (P/N: FUTUREM2SF-EVB)*
- *Design Required Software Development Toolchain:*
    - *Libero SoC v 11.8 SP2(minimum "Silver"(Free), "Gold"(Purchasable), or "Platinum"(Purchasable) license) NOTE:* The evaluation license does not support programming - it is intended as a starting point to allow designers to use the tools for free to go through the flow for the packages commonly on demo boards. Silver, Gold or Platinum are required to program more capable package versions.
    - *SoftConsole 4.0 or SoftConsole 5.1*
    - *Modelsim ME 10.5 (for waveform simulation included in Libero SOC)*

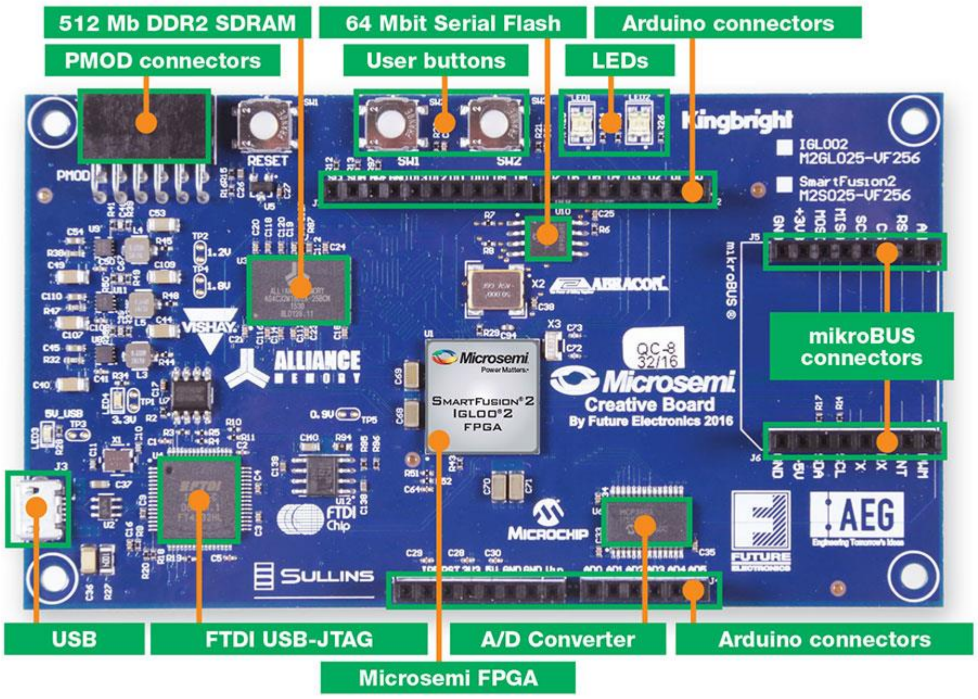


Figure 1: Future Electronics Creative board featuring the SmartFusion2 (M2S025)

Application Design Overview: Hardware

The hardware component of the project consists of the items described in Table assembled in the configuration shown in Figure . A high current power supply is used as a dedicated power supply for the servos. Minor adjustability permits fine control of output voltage to tune servo responsiveness and reduce servo oscillation due to overcorrection at the end of motion. The robotic arm itself is assembled from a kit by Aiboully (Figure 3).   An alternative product is the DOIT DoArm S6. Numerous kits for robotic arms (and clones thereof) exist that are very similar that will work equivalently provided they provide 6 DoF of motion (in a standard robotic/industrial arm configuration) and a 7th motion for a pincher or gripper attachment. Common low end, generic MG996R servos will not provide consistent operation for this application. In evaluation, these MG996R servos were prone to failure in this application. The MG996R servo lacks substantial heat dissipation for the control IC. This has led to multiple servo failures as witnessed first-hand by the authors.   If these servos are present in the purchased kit, please strongly consider replacement with Sinoning K2 Model KS-3518, Generic DS3115MG, or Generic DS3218MG digital servos with end stops. Each of these servos in end-stop and continuous rotation have been tested field tested by the authors in this application.

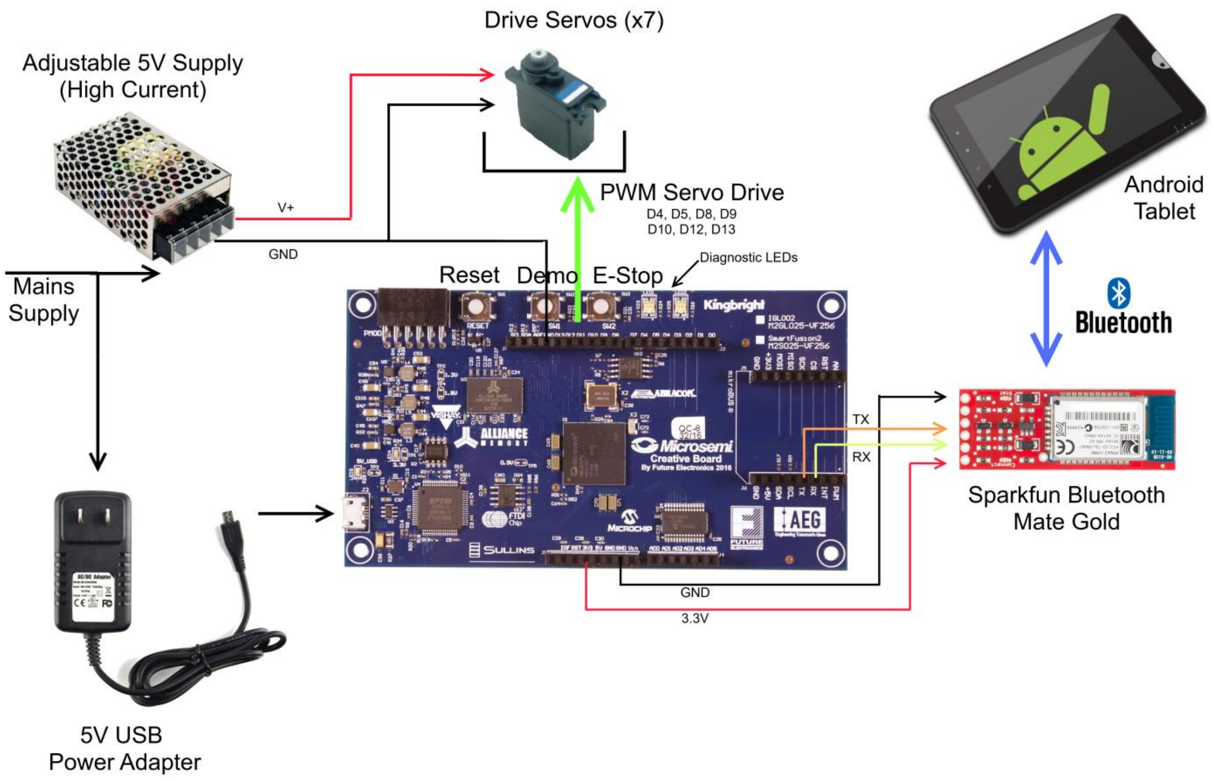


Figure 2: A connection diagram for the servo driven robotic arm sample manipulator design. Observe proper RX and TX orientations for interfacing UART devices.

Table 1: Major hardware components required for construction.

Quantity Req.	**Manufacturer**	**Model**
1	Future Electronics	Creative Board: SmartFusion 2 (FUTUREM2SF-EVB)
1	Aiboully	iSmaring 6DOF Robotic Arm
7 (total)	Sinoning K2 [Varies] [Varies]	Model KS-3518 (or) DS3115MG (or) DS3218MG (digital servos with end stops)

1	Sparkfun	Bluetooth Mate Gold
1	[Varies]	Android Tablet
1	[Varies]	5V USB Power Supply with micro-USB male interface
1	[Varies]	5V, 5A Adjustable high current power supply (4.5-5.5V)

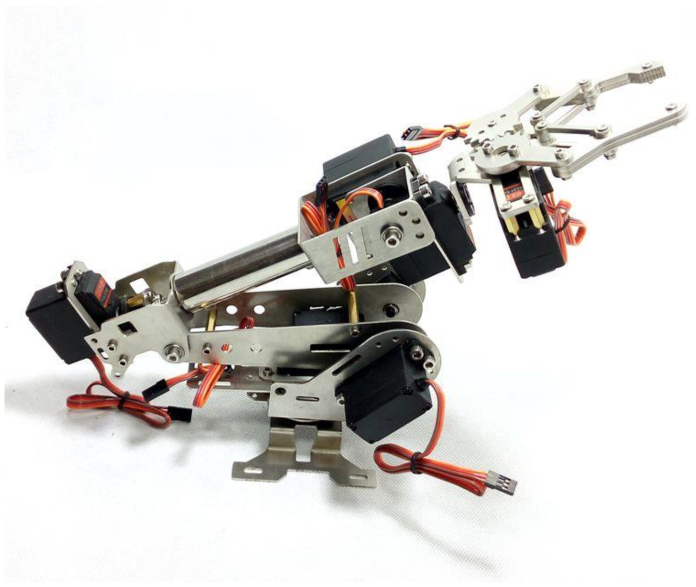
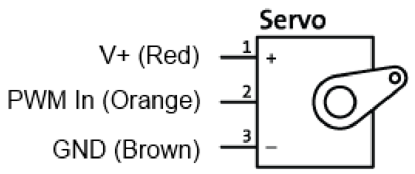


Figure 3: Robotic arm kit with 6 degree of freedom (DoF) motion (Aiboully iSmaring 6DOF). This kit is similar to the DoArm S6 which would work as a substitute with added manipulator jaw.



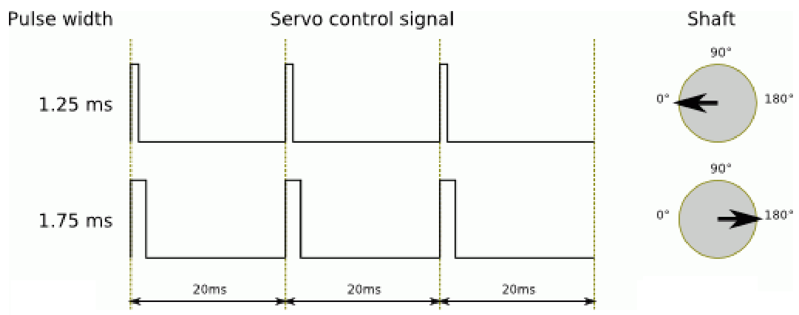


Figure 4: Required PWM pulse and cycle duration for driving a typical hobby servo between mechanical limits

Application Design Overview: Firmware

The project was developed in Libero SoC version 11.8 and SoftConsole version 4.0. Per conventional Microsemi SmartFusion2 designs, Libero SoC is used to develop the FPGA fabric design while SoftConsole is used to develop (in the C programming language) the program executed on the microcontroller. If the program for the microcontroller is intended to be executed on boot without debugging by a connected PC (as illustrated as a component of Figure), Libero is also used to flash an auto executing SoftConsole C program into the SmartFusion2 eNVM memory. While RISC-V is compatible with the Future Electronics Creative board, it is not implemented in this design. The proposed design potentially could be implemented on an IGLOO2 FPGA by substituting the microcontroller functionality with that provided by the RISC-V soft processor (Note: Requires SoftConsole 5.1 or 5.2). This conversion would require the use of alternative hardware abstraction layer schemes other than those used for interfacing fabric soft cores to the ARM Cortex M3 microcontroller.

FPGA Fabric:

The PWM are implemented in the FPGA fabric logic. While the GPIO and UART are implemented later in the MSS. The ARM Cortex M3 microcontroller subsystem (MSS) provides capabilities to perform similar or identical functionality and interfacing as designs implemented in the fabric. For example, asynchronous serial interfacing via UART can be accomplished using a software core UART (CoreUARTapb) in the fabric or on the microcontroller subsystem (MSS). In the current version of the application, the timer is implemented in the MSS versus the fabric. In many cases using MSS resources wisely can save fabric space versus redundant designs.

The UART and GPIO interfacing was implemented in the MSS of the FPGA. It should be known that while this provides a usage example, this is not necessarily the best approach as it uses redundant resources. The UART, GPIO, and (one of two) timer blocks is implemented in the MSS for simplicity. In more complicated designs this adding more into the fabric can impinge on requirements for total logic elements but may be more efficient for completed designs. Often designers wish to know how much of the FPGA fabric was used to implement a solution: the numbers are in the reports that are available after the layout completes. By adding soft peripherals in the fabric that could have been replaced by MSS peripherals there are fewer resources available for other logic functions and accordingly, designers might think they would need a larger device to implement a design they are considering. Designers should bear this in mind a strategy for scalability when building from this example design.

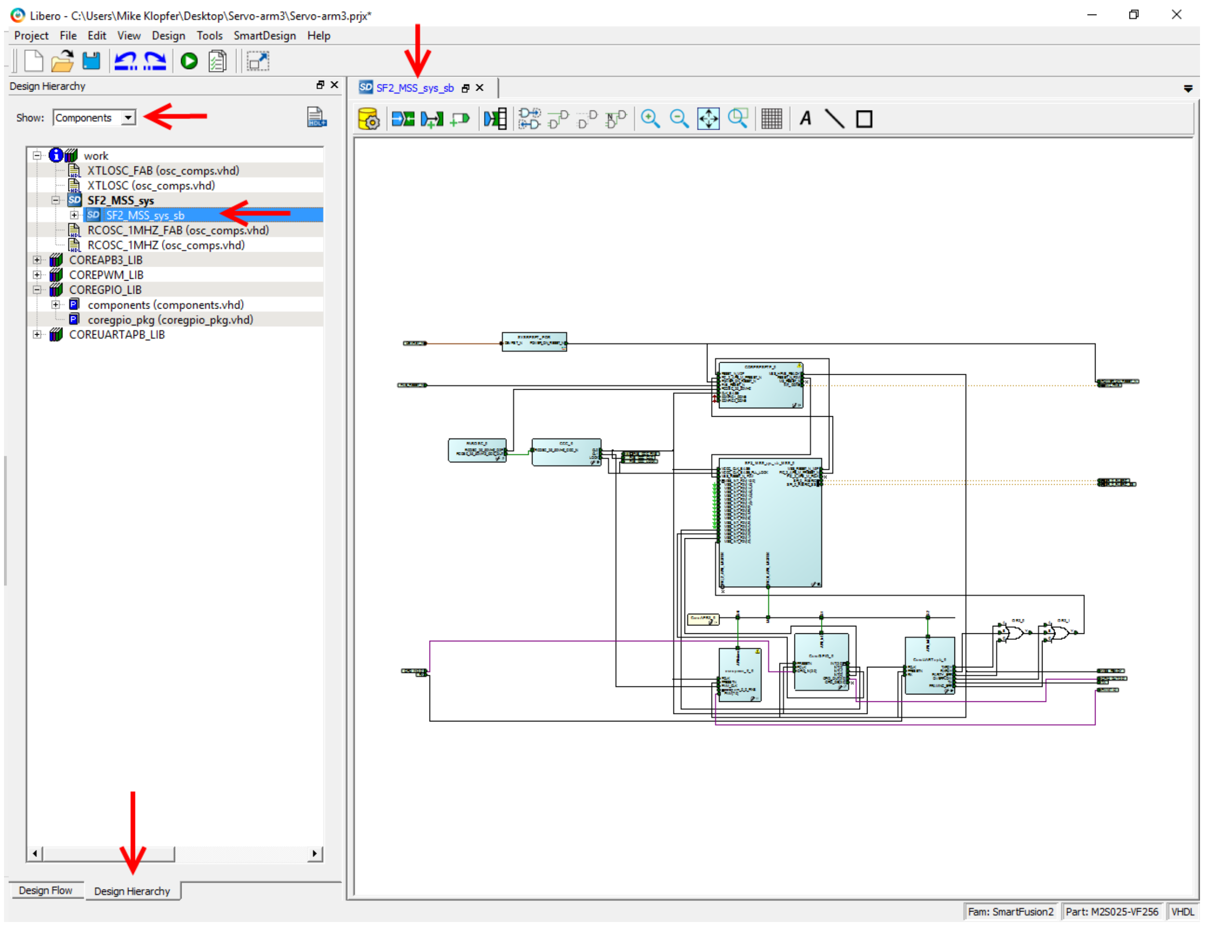
It is often good practice to reserve the fabric for resources for functions that are not in the MSS (such as a PWM controller) or to increase the number of MSS resources (i.e. more timers, UARTs, GPIO, etc.). Implementing the current design with MSS components properly utilized versus adding additional fabric components (UART and GPIO) has not been evaluated, but example figures are available in the hierarchical compile report.

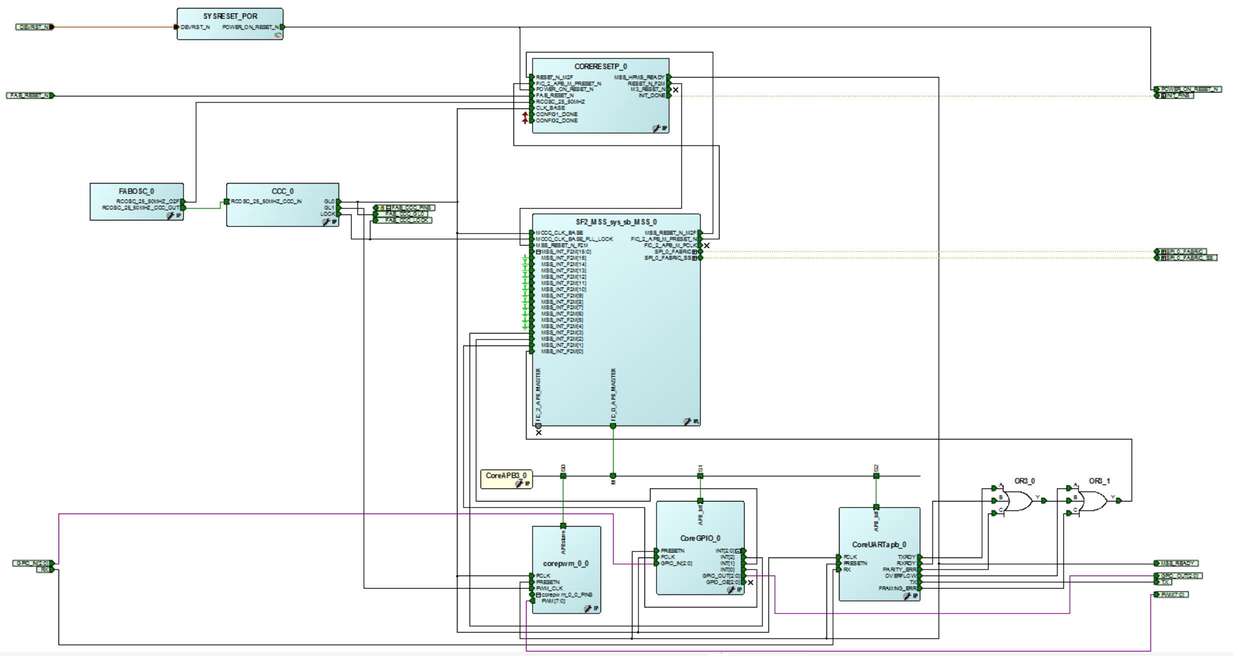
While not shown in this example, there are a lot of other things that can be done in the fabric as well. The implementation of a soft processor in the fabric that runs while the Cortex-M3 is running can reduce operational load and serve as an offload engine for the Cortex M3.

Microsemi provides a proprietary processor (CoreABC) in the IP catalog and have introduced a soft Cortex-M1 (PolarFire) and the RISC-V processors which would provide expanded design capabilities.

The microcontroller UART interfaces (MMUART_0 and MMUART_1) are disabled as a result, there is a single fabric UART (CoreUARTapb) in the design. A total of 8 PWM outputs are enabled in the fabric logic in this design with a resolution of 16 bits. Other resolutions can be used: there is a trade-off between the PWM resolution and the number of logic elements required to implement CorePWM. For this design, a resolution of 16 bits was chosen because it provided acceptable resolution and reduced the size of CorePWM. Seven of the PWM drives provide control for the seven servos on the robotic arm (6-DoF and the pincher), and one of the PWM outputs is routed to a diagnostic LED for testing purposes. A block diagram of the design is shown above in Figure 6.

This project design consists of the SmartFusion2 Cortex-M3, CorePWM, CoreGPIO, CoreUARTapb, CORERESETP and SYSRESET_POR. Clocking is provided by the 25/50 MHz RCO (FABOSC) which is connected to one of the fabric PLLs to generate a 70 MHz clock which is the reference clock for the SmartFusion2 MSS (GL0) and a 1 MHz clock which is the PWM clock (GL1). There are 4 clocks in the design: the internal 25/50 MHz oscillator is the reference clock for the fabric phase lock loop (PLL). The fabric PLL has two outputs - 70 MHz which is used as the reference clock for the MSS PLL and the APB clock for CorePWM, CoreGPIO and CoreUARTapb; and a 1 MHz clock which is used as the PWM clock. The MSS PLL generates a 140 MHz clock which is used as the clock for the Cortex-M3SYSRESET_POR is the SmartFusion2 dedicated reset and CORERESETP generates various resets to keep the device and peripherals reset during the boot process, etc. The design includes a 16k byte eNVM data storage client which could be used to hold the Cortex-M3 application. In the current design, three fabric based interrupts are used. For these three, one is for CoreUARTapb to trigger on UART input and two are used with on board push buttons to modify program execution. One button executes a demo mode application in the Cortex-M3, to move all axes in a prescribed manner. This permanently executes until the Future Creative Demo Board is reset. The second button provides an emergency stop functionality. Pressing this button halts the Cortex-M3's program execution (places it in an infinite loop: aka hold and catch fire) and disables the CorePWM cores. Details of the FPGA design and the MSS design is shown in Figure 5 and Figure 6.


Libero - C:\Users\Mike Klopfer\Desktop\Servo-arm3\Servo-arm3.prjx*
Project File Edit View Design Tools SmartDesign Help
Design Hierarchy
Show: Components
work
XTLOSC_FAB (osc_comps.vhd)
XTLOSC (osc_comps.vhd)
SF2_MSS_sys
SF2_MSS_sys_sb
RCOSC_1MHZ_FAB (osc_comps.vhd)
RCOSC_1MHZ (osc_comps.vhd)
COREAPB3_LIB
COREPWM_LIB
COREGPIO_LIB
components (components.vhd)
coregpio_pkg (coregpio_pkg.vhd)
COREUARTAPB_LIB
Design Flow
Design Hierarchy
SF2_MSS_sys_sb
Fam: SmartFusion2
Part: M2S025-VF256
VHDL


SYSRESET_POR
DEVRST_N
POWER_ON_RESET_N
FAB_RESET_N
CORERESETP_0
FABOSC_0
CCC_0
SF2_MSS_sys_sb_MSS_0
CoreAPB3_0
corepwm_0_0
CoreGPIO_0
CoreUARTapb_0
OR3_0
OR3_1

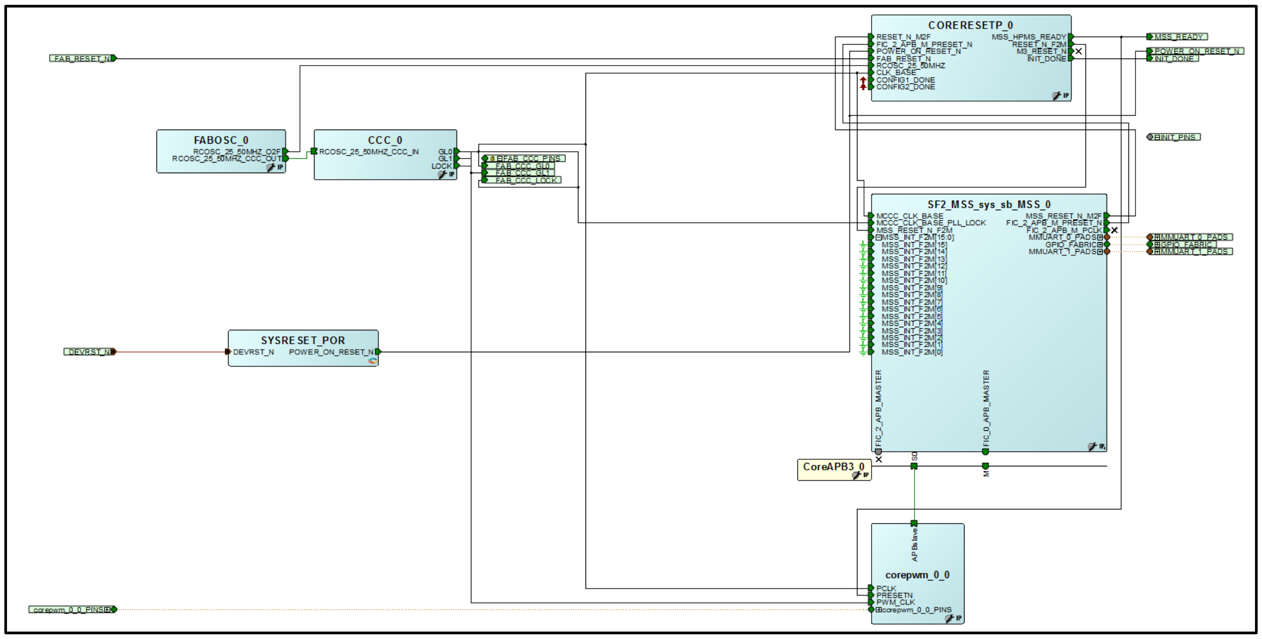
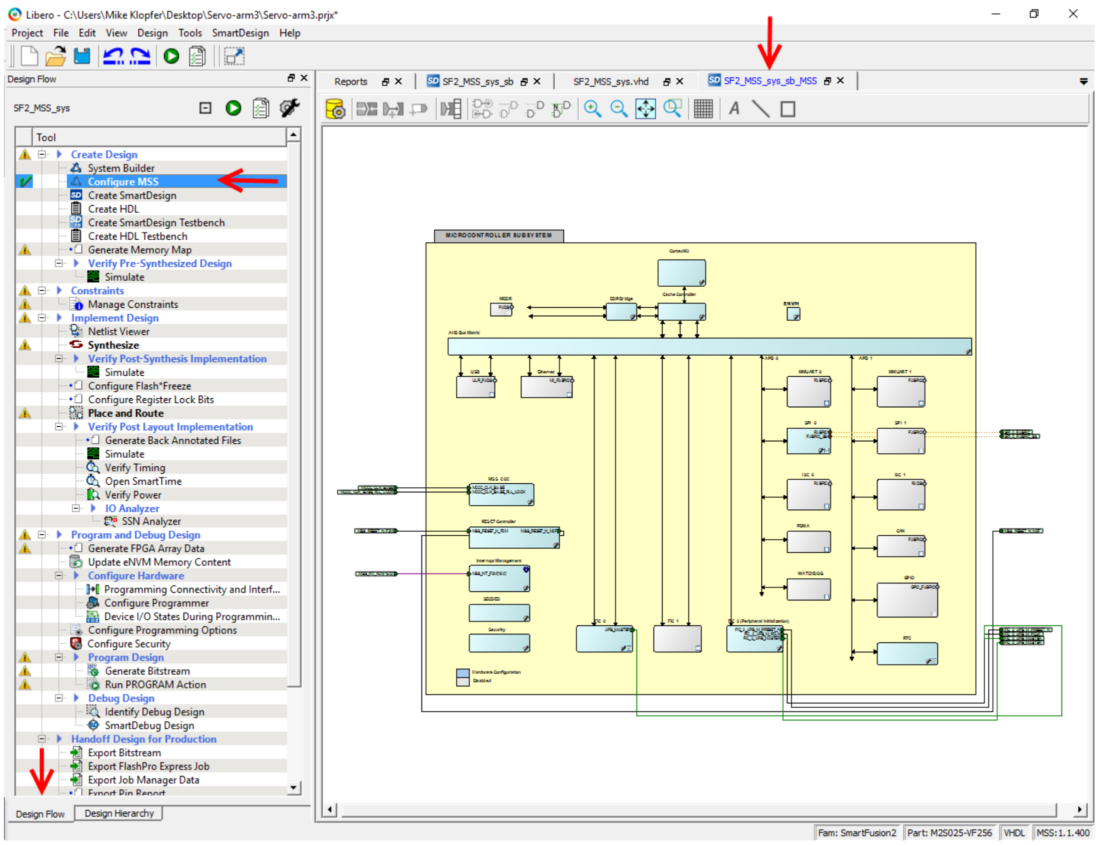


Figure 5a,b,c: (a-top) Location in Libero SoC,(b – middle) detail of project Fabric Design, and (c – bottom) a preliminary design implementing only CorePWM with no CoreUART or CoreGPIO implemented.



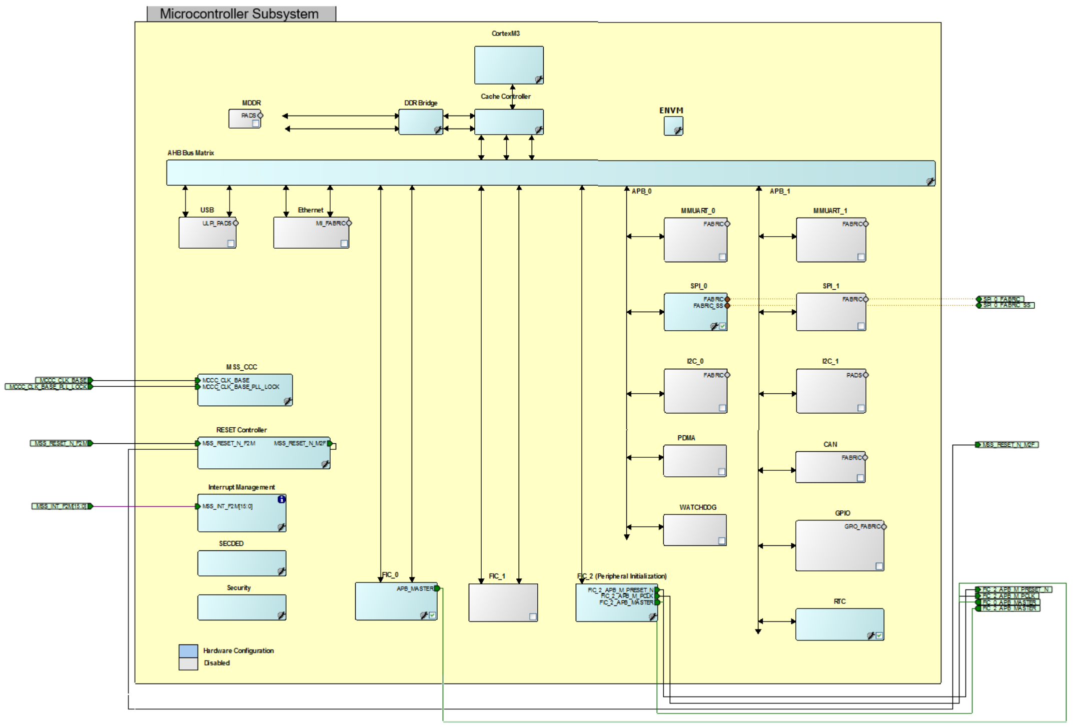
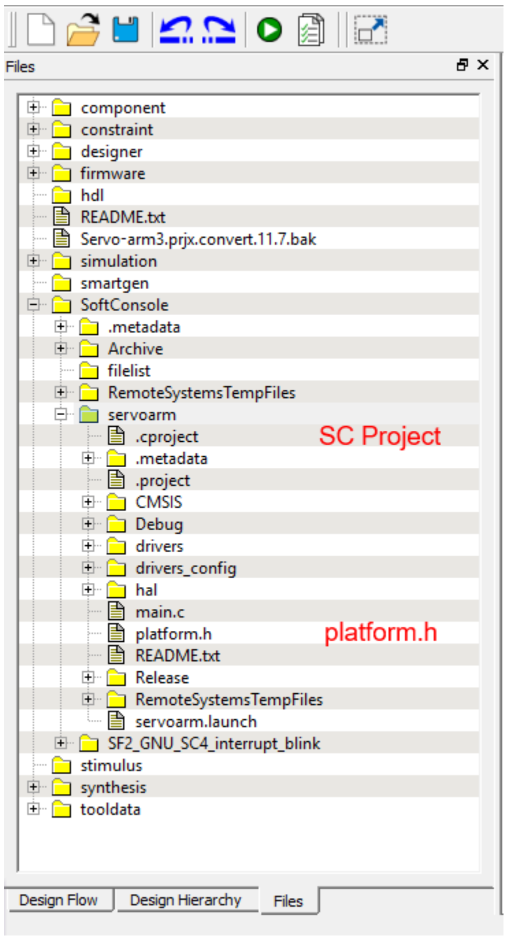


Figure 6a,b: (a – top) Location in Libero SoC and (b – bottom) detail of project Microcontroller Subsystem design. Notice grayed-out elements in the MSS are disabled. SPI peripherals enabled for convenience to start other SPI projects, however, are not used this project.

Cortex M3 Microcontroller Interfacing:

Configuration of the Cortex-M3 is split between the settings for the MSS in Libero SoC and the project for the designed application in SoftConsole. The hardware abstraction layer provides a means of communication between the microcontroller program and the FPGA fabric components. Communication is made via an abstracted hardware bus, the Advanced Peripheral Bus (APB). On the microcontroller side to communicate with APB interfaced cores, driver files and a project file "platform.h" which includes the fabric memory addresses are used for interfacing.

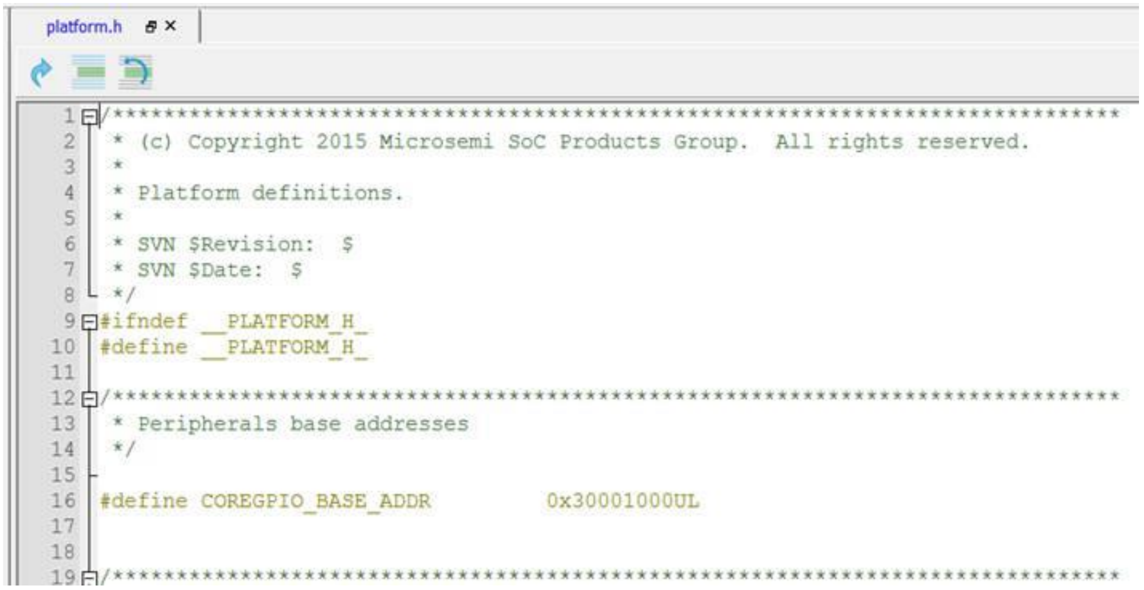
In the current design example the communication between the FPGA fabric implemented software cores and the microcontroller is functional. A short tutorial for establishing the driver setup is shown below to permit the addition of additional software cores on the APB interface:



```
component
constraint
designer
firmware
hdl
README.txt
Servo-arm3.prjx.convert.11.7.bak
simulation
smartgen
SoftConsole
    .metadata
    Archive
    filelist
    RemoteSystemsTempFiles
    servoarm
        .cproject
        .metadata
        .project
        CMSIS
        Debug
        drivers
        drivers_config
        hal
        main.c
        platform.h
        README.txt
        Release
        RemoteSystemsTempFiles
        servoarm.launch
    SF2_GNU_SC4_interrupt_blink
stimulus
synthesis
tooldata
```

Figure 7: Project files highlighting the location of the platform.h file.

If one were to create a sample firmware project for one of the fabric peripherals (such as CorePWM or CoreGPIO), the project contains a file named platform.h which has the address for the fabric peripheral. An example of the platform.h file is shown below in Figure 8:



```
/*******************************************************************************
 * (c) Copyright 2015 Microsemi SoC Products Group.  All rights reserved.
 *
 * Platform definitions.
 *
 * SVN $Revision:  $
 * SVN $Date:  $
 */
#ifndef __PLATFORM_H_
#define __PLATFORM_H_

/*******************************************************************************
 * Peripherals base addresses
 */

#define COREGPIO_BASE_ADDR          0x30001000UL


/*******************************************************************************
```

Figure 8: Example platform.h file generated from a template project.

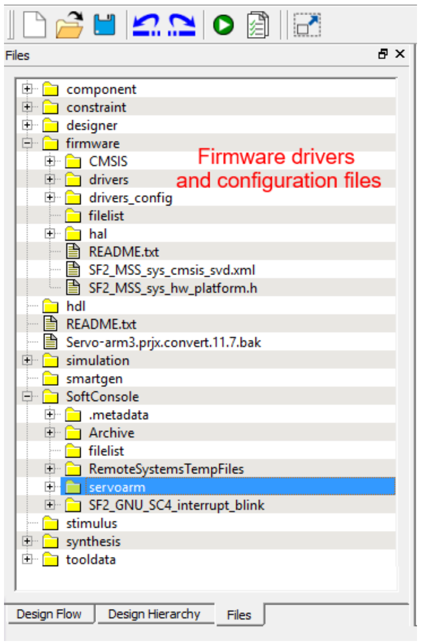
Once a new sample project is created from a template project found in the library of Libero, the main.c in the sample project will reference the name in the platform.h file (in this case shown above only CoreGPIO was used and correspondingly COREGPIO_BASE_ADDR is shown). The address for the peripheral in the platform.h file is always the same – it is not generated from knowledge of the design. As a result, the address in the file might be incorrect and must be verified and updated. In addition, the sample projects contain other firmware drivers and configuration files, but these are also generated without knowledge of the hardware design, so they must be replaced with firmware drivers and configuration files that are exported from Libero and match the hardware design. By default these files are saved in a folder named firmware (see Figure) in the Libero project. They must be imported into the sample project and will replace the existing drivers and driver configuration files in the sample project. This step must be repeated if the hardware design is changed to ensure that the firmware project matches the hardware. Details of this operation are referenced in the videos jointly produced between Microsemi and UC Irvine (see Table 2).

Table 2: List of videos co-produced between Microsemi and UCI to date.

Video URL	**Video Title**	**Video Description**
https://www.youtube.com/watch?v=khzBknH-Zh8	Getting Started with Microsemi SmartFusion2 SoC (Part 1) - Product Architecture and Capabilities	Review of the SF2 and Igloo 2 Product Line, Architecture, and Capabilities
https://www.youtube.com/watch?v=gwJxhTQYtIg&t=598s	Getting Started with Microsemi SmartFusion2 System on Chip (Part 2) – Libero v11.x Setup	Review of Libero and toolchain licensing and setup (video partially obsoleted by Libero 11.8’s new licensing structure)
https://www.youtube.com/watch?v=ibIARgh7N1M&t=33s	Getting Started with Microsemi SmartFusion2	Setup of microcontroller subsystem and general use

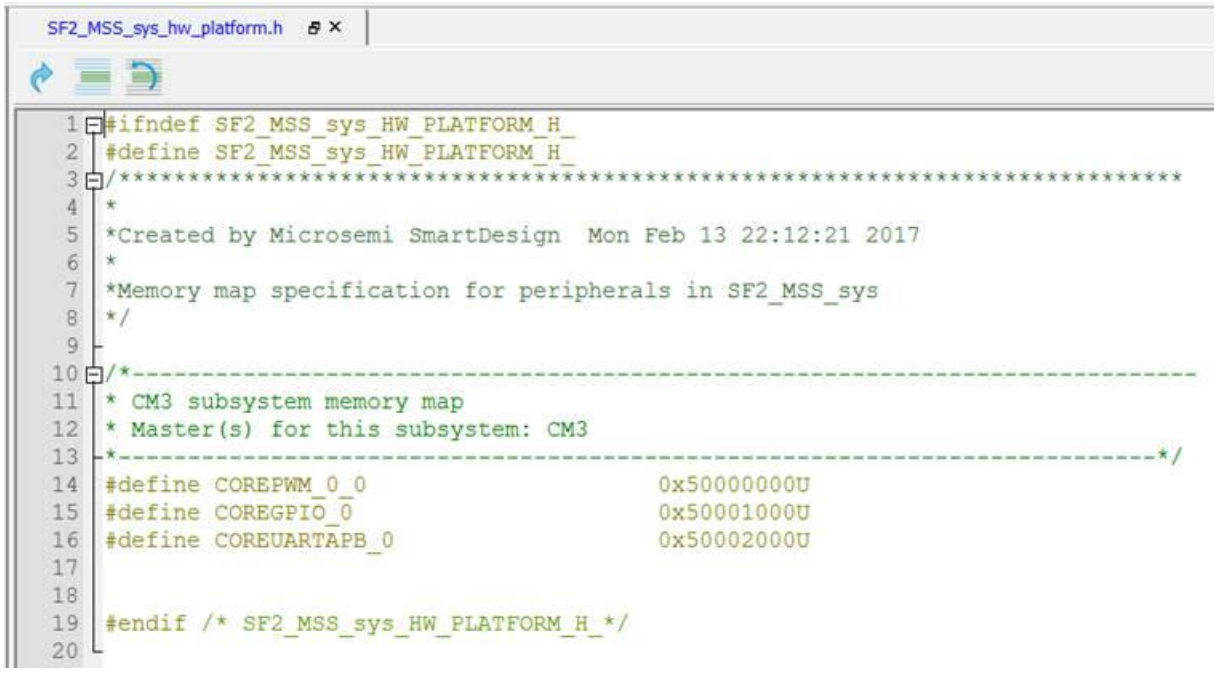
	System on Chip (Part 3) – ARM Microcontroller Subsystem	of Libero in design
https://www.youtube.com/watch?v=AE90gaJudD0&t=17s	Getting Started with Microsemi SmartFusion2 System on Chip (Part 4) – FPGA Fabric Demonstration	Interfacing FPGA fabric peripherals with microcontroller designs and an example of simulation testing.
https://www.youtube.com/watch?v=mknriOro6ts	Getting Started with Microsemi SmartFusion2 System on Chip (Part 5) – FPGA Fabric Peripherals	Interfacing FPGA fabric peripherals with microcontroller designs
https://www.youtube.com/watch?v=Dr3AYogENdc	Getting Started with Microsemi SmartFusion2 System on Chip (Part 6) – AVNET Kickstart Example	Building an example project with the AVNET Kickstart board.
https://www.youtube.com/watch?v=PLGkq17wgqA&t=1s	Getting Started with Microsemi SmartFusion2 System on Chip (Part 7) – UART Example	Specific UART example implemented in the FPGA fabric.

In the figure above you will see there is a second platform.h file named SF2_MSS_sys_hw_platform.h (see Figure). This file has the correct addresses for the fabric peripherals, but the names do not match the names in the project's main.c file.



```
component
constraint
designer
firmware
    CMSIS
    drivers
    drivers_config
    filelist
    hal
    README.txt
    SF2_MSS_sys_cmsis_svd.xml
    SF2_MSS_sys_hw_platform.h
hdl
README.txt
Servo-arm3.prjx.convert.11.7.bak
simulation
smartgen
SoftConsole
    .metadata
    Archive
    filelist
    RemoteSystemsTempFiles
    servoarm
    SF2_GNU_SC4_interrupt_blink
stimulus
synthesis
tooldata
```

Figure 9: Location of project firmware files.



```
#ifndef SF2_MSS_sys_HW_PLATFORM_H_
#define SF2_MSS_sys_HW_PLATFORM_H_
/***************************************************************************
*
*Created by Microsemi SmartDesign  Mon Feb 13 22:12:21 2017
*
*Memory map specification for peripherals in SF2_MSS_sys
*/

/*-------------------------------------------------------------------------
* CM3 subsystem memory map
* Master(s) for this subsystem: CM3
*------------------------------------------------------------------------*/
#define COREPWM_0_0                     0x50000000U
#define COREGPIO_0                      0x50001000U
#define COREUARTAPB_0                   0x50002000U


#endif /* SF2_MSS_sys_HW_PLATFORM_H_*/
```

Figure 10: Example SF2_MSS_sys_hw_platform.h file.

Note that in this example SF2_MSS_sys_hw_platform.h uses the name COREGPIO_0 instead of COREGPIO_BASE_ADDR which is used in platform.h and main.c. This pattern is common to other situations. To properly coordinate a design, there are two options:

1. Edit platform.h so that the addresses for the fabric peripherals match those in SF2_MSS_sys_hw_platform.h.
2. Edit main.c to use SF2_MSS_sys_hw_platform.h instead of platform.h.

Of these two presented approaches, the first approach is typically easier as less file editing is involved.

In the current implemented design, the following memory locations (

Table 3) are used to provide interface communication between the microcontroller interface Hardware Abstraction Layer (HAL) drivers (implemented in C) and the corresponding fabric control elements. Please refer to Part 5 in the video series for more details.

Table 3: Memory Location for Core Interfacing. NOTE: There is an aliasing for the fabric peripherals connected to the FIC so either address range could be used. As presented, "&" can be replaced with "|" as either address range could be used. This is described on page 793 of the SmartFusion2 Microcontroller Subsystem User's Guide (https://www.microsemi.com/document-portal/doc_download/130918-ug0331-smartfusion2-microcontroller-subsystem-user-guide)

Fabric Software Core	**Memory Location**
CoreUART (apb)	0x50002000 – 0x50002FFF | 0x30002000 – 0x30002FFF
CoreGPIO	0x50001000 – 0x50001FFF | 0x30001000 – 0x30001FFF
CorePWM	0x5000000 – 0x50000FFF | 0x3000000 – 0x30000FFF

Design constraints for the FPGA fabric design are used to set external interface pins. Documentation provided with the Future Creative board provides a table showing corresponding die pins and the location of interface headers or peripherals on the board. For this design, Table 4 summarizes the corresponding constraints, physical interfaces, and connected peripherals. Timing constraints are generated automatically to ensure to meet the 70MHz clock design requirements

Table 4: List of interfaced peripherals

Interface	**Peripheral**	**Implemented Interface**
Arduino Header – D4	Servo – Pincher Jaw Rotation	Fabric (Core PWM)
Arduino Header – D5	Servo – Pincher Jaw Close	Fabric (Core PWM)
Arduino Header – D8	Servo – Arm Extension	Fabric (Core PWM)
Arduino Header – D9	Servo – Pincher Jaw Pivot	Fabric (Core PWM)
Arduino Header – D10	Servo – Rotation of Arm Base	Fabric (Core PWM)
Arduino Header – D12	Servo – Rotation of Pincher Jaw Arm	Fabric (Core PWM)
Arduino Header – D13	Servo – Second Extension of Pincher Jaw Arm	Fabric (Core PWM)
MikroBUS – RX	UART RX (to BT Mate TX)	Fabric (Core UART)
MikroBUS – TX	UART TX (to BT Mate RX)	Fabric (Core UART)
LED 1 (Green)	Auxiliary PWM Channel Diagnostic LED (DIAGLED 1)	Fabric (Core PWM)
LED 1 (Red)	Mode Diagnostic LED 0	Fabric (Core GPIO)
LED 2 (Green)	Mode Diagnostic LED 1	Fabric (Core GPIO)
LED 2 (Red)	Mode Diagnostic LED 2	Fabric (Core GPIO)
RESET Button	Board Reset	Hardware
Button 1 (SW1)	Demo Mode Begin	Fabric (Core GPIO - Interrupt)
Button 2 (SW2)	Emergency Stop (E-STOP)	Fabric (Core GPIO - Interrupt)

PWM Design Configuration:

In this design, CorePWM is configured as PWM Only Mode with 8 PWM channels and 16-bit resolution. Channel [8:1] configurations are set by writing the PWMx_POSEDGE and PWMx_NEGEDGE registers. Note that this could be changed in the CorePWM configurator to set the positive edge to a fixed value if desired. The number of PWM channels can also be additionally modified if desired. The port numbering for CorePWM matches the PWM output number so the lowest index is 1 instead of 0.

This design uses CorePWM v4.3.101 because is supports a separate PWM clock. Having a separate PWM clock allows the PWM clock to be slower (supporting longer PWM periods) while allowing a faster PCLK frequency for writes to the CorePWM registers by the Cortex-M3.

Note that the System Builder tool does not show CorePWM v4.3.101. This design was implemented using System Builder, then opening the component in SmartDesign and replacing the CorePWM instance with v4.3.101 and manually connecting the CorePWM PWM clock input to the GL1 output of the fabric CCC. See previous discussion on this topic within this document.

Theory of Operation

The CorePWM register block is accessed by the Cortex-M3 via the FIC_0 MSS master interface. In this design the interface is configured as APB3.

The Cortex-M3 writes to the CorePWM PRESCALE (offset 0x00) and PERIOD (0x04) registers. The PRESCALE register setting determines the PWM period granularity (PWM_PG) given by

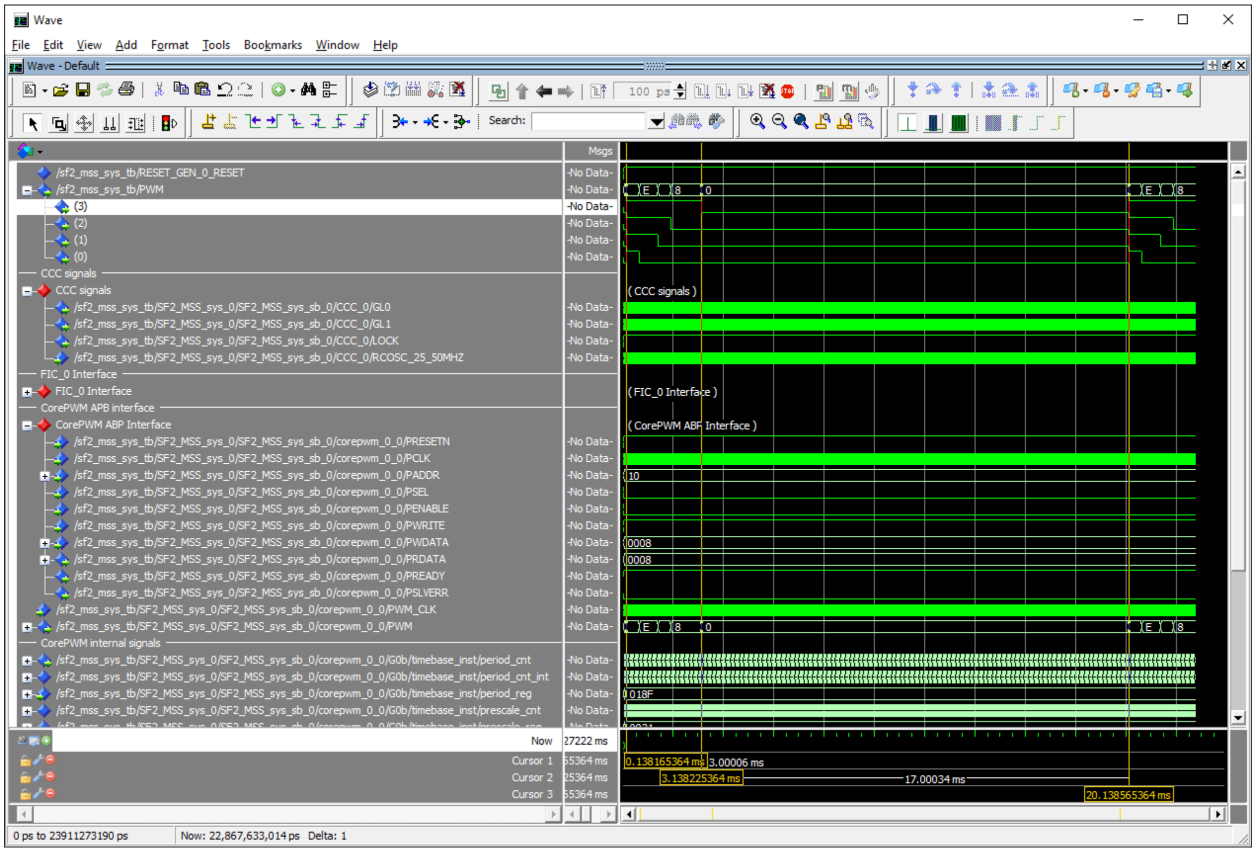
PWM_PG = clock period * (PRESCALE +1). This design uses the separate CorePWM PWM_CLK of 1 MHz which is generated by the fabric CCC GL1 output. Setting PRESCALE to 49 (0x31) sets the PWM_PG to 50 us as seen in Table 5. Please see simulation table for details of PWM_PRESCALE and PWM_Period required for a desired PWM frequency and duty cycle.

The PERIOD register setting determines the PWM period given by PWM period = PWM_PG * (PERIOD + 1). Setting PERIOD to 399 (0x018F) gives a period of 20 ms (note that this requires 16-bit resolution). The PWM output waveforms are configured by writing the the PWMx_POSEDGE and PWMx_NEGEDGE registers.

Simulation

The design was simulated using the Cortex-M3 Bus Functional model. A BFM script was used to configure the PWM granularity as 50 us, the PWM period as 20 ms and four output channels to produce output waveforms of 0.5 ms, 1.25 ms, 1.75 ms and 3 ms. A picture of the Modelsim wave window is shown below. A wave.do file is included in the project along with the BFM script file (CorePWM_config.bfm). Both files are visible on the Libero SoC Files tab under simulation.

Please refer to the video “Getting Started with Microsemi SmartFusion2 System on Chip (Part 4) – FPGA Fabric Demonstration” for details on simulation execution.


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/sf2_mss_sys_tb/RESET_GEN_0_RESET
/sf2_mss_sys_tb/PWM
(3)
(2)
(1)
(0)
CCC signals
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/GL0
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/GL1
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/LOCK
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/RCOSC_25_50MHZ
FIC_0 Interface
CorePWM APB interface
CorePWM ABP Interface
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PRESETN
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PCLK
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PADDR
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PSEL
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PENABLE
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PWRITE
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PWDATA
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PRDATA
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PREADY
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PSLVERR
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PWM_CLK
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PWM
CorePWM internal signals
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/G0b/timebase_inst/period_cnt
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/G0b/timebase_inst/period_cnt_int
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/G0b/timebase_inst/period_reg
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/G0b/timebase_inst/prescale_cnt
-No Data-
(CCC signals)
(FIC_0 Interface)
(CorePWM ABP Interface)
10
0008
0018F
Now 27222 ms
Cursor 1 55364 ms
Cursor 2 25364 ms
Cursor 3 55364 ms
0.138165364 ms
3.00006 ms
3.138225364 ms
17.00034 ms
20.138565364 ms
0 ps to 23911273190 ps
Now: 22,867,633,014 ps Delta: 1

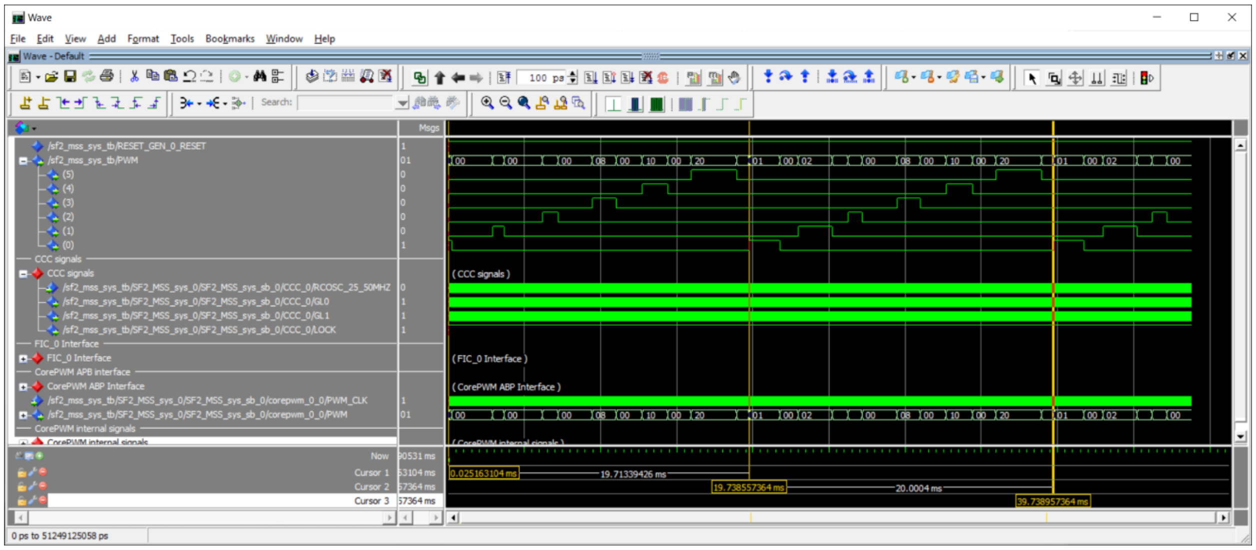

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/sf2_mss_sys_tb/RESET_GEN_0_RESET
/sf2_mss_sys_tb/PWM
(5)
(4)
(3)
(2)
(1)
(0)
CCC signals
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/RCOSC_25_50MHZ
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/GL0
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/GL1
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/CCC_0/LOCK
FIC_0 Interface
CorePWM APB interface
CorePWM ABP Interface
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PWM_CLK
/sf2_mss_sys_tb/SF2_MSS_sys_0/SF2_MSS_sys_sb_0/corepwm_0_0/PWM
CorePWM internal signals
(CCC signals)
(FIC_0 Interface)
(CorePWM ABP Interface)
Now 90531 ms
Cursor 1 63104 ms
Cursor 2 57364 ms
Cursor 3 57364 ms
0.025163104 ms
19.71339426 ms
19.738557364 ms
20.0004 ms
39.738957364 ms
0 ps to 51249125058 ps

Figure 11: Simulation example showing PWM timing.

Table 4: PWM Granularity for servo control

PWM pulse width	PWM count		
	1 us Granularity	**5 us Granularity**	**50 us Granularity**
0.25 ms	250	50	5
0.5 ms	500	100	10
0.75 ms	750	150	15
1.0 ms	1000	200	20
1.25 ms	1250	250	25
1.5 ms	1500	300	30
1.75 ms	1750	350	35
2.0 ms	2000	400	40
2.25 ms	2250	450	45
2.5 ms	2500	500	50
2.75 ms	2750	550	55

Table 5: The PWM_PRESCALE and PWM_PERIOD values for the granularity shown above are presented in this table

Granularity	PWM_PRESCALE	PWM_PERIOD
50 us	49	399
5 us	4	3999
1 us	0	19999

Appendix

Example ARM Cortex M3 code to drive an individual servo between limits along with a fabric design capable of providing control functionality.

```
#include "platform.h"

#include "core_pwm.h"

/******************************************************************************
 * Delay count used to time the delay between duty cycle updates.
 *****************************************************************************/
#define DELAY_COUNT     10000  //period for the delay counts

/******************************************************************************
 * PWM prescale and period configuration values.
 *****************************************************************************/
#define PWM_PRESCALE    49  //Prescale value 4
#define PWM_PERIOD      399 //full period
#define MAX_DEFLECT     55 //max deflection of servo
#define MIN_DEFLECT     5 //min deflection of servo
#define LOOP_SPEED      10 //speed of execution

/******************************************************************************
 * CorePWM instance data.
 *****************************************************************************/
pwm_instance_t the_pwm;

/******************************************************************************
 * Local function prototype.
 *******************************************************************************/
void delay( int mult );

/*******************************************************************************
 * main function.
 *******************************************************************************/
int main( void )
{
    uint32_t duty_cycle = 1;
    int direction = 1;

    /***************************************************************************
     * Initialize the CorePWM instance setting the prescale and period values.
     ***************************************************************************/
    PWM_init( &the_pwm, COREPWM_BASE_ADDR, PWM_PRESCALE, PWM_PERIOD ) ;

    /***************************************************************************
     * Set the initial duty cycle for CorePWM output 1.
     ***************************************************************************/
    PWM_set_duty_cycle( &the_pwm, PWM_1, duty_cycle );
    PWM_set_duty_cycle( &the_pwm, PWM_6, duty_cycle );

    while ( 1 )
    {
        /***********************************************************************
         * Update the duty cycle for CorePWM output 1.
         ***********************************************************************/
        PWM_set_duty_cycle( &the_pwm, PWM_1, duty_cycle );
        PWM_set_duty_cycle( &the_pwm, PWM_6, duty_cycle );
        /***********************************************************************
         * Wait for a short delay.
         ***********************************************************************/
        delay(LOOP_SPEED); //delay with "10" multiplier period

        /***********************************************************************
         * Calculate the next PWM duty cycle.
         ***********************************************************************/
        duty_cycle += direction;
        if ( duty_cycle >= MAX_DEFLECT )
        {
            direction = -1;
        }
        else if ( duty_cycle == MIN_DEFLECT )
        {
            direction = 1;
        }
    }

}
/*******************************************************************************
 * Delay function.
 *******************************************************************************/
void delay( int mult )
{
    volatile int counter = 0;

    while ( counter < (DELAY_COUNT*mult) )
    {
        counter++;
    }
}
```