Drive-On-Chip Reference Design

2016-12.-15

AN-669 Subscribe Send Feedback

About the Drive-On-Chip Reference Design

The Altera® Drive-On-Chip reference design demonstrates concurrent multiaxis control of up to four three-phase AC 400-V permanent magnet synchronous motors (PMSMs) or sinusoidally wound brushless DC (BLDC) motors.

AC and servo drive system designs comprise multiple distinct but interdependent functions to realize requirements to meet the performance and efficiency demands of modern motor control systems. The system's primary function is to efficiently control the torque and speed of the AC motor through appropriate control of power electronics. A typical drive system includes the following items:

- Flexible pulse-width modulation (PWM) circuitry to switch the power stage transistors appropriately
- Motor control loops for single- or multiaxis control
- Industrial networking interfaces
- Position encoder interfaces
- Current, voltage, and temperature measurement feedback elements.
- Monitoring functions, for example, for vibration suppression.

The system requires system software running on a processor for high-level system control, coordination, and management.

Altera Cyclone® and MAX® 10 devices offer high-performance fixed- and floating-point DSP functionality. Cyclone V SoC devices offer the integrated ARM-based hard processor subsystem (HPS); other Cyclone FPGA and MAX 10 FPGA devices offer support for Nios II soft processors. Cyclone and MAX 10 FPGA devices offer a uniquely scalable and flexible platform for integration of single- and multiaxis drives on a single FPGA. The Altera motor control development framework allows you to create these integrated systems easily. The framework provides a reference design that comprises IP cores, software libraries, and a hardware platform. The framework seamlessly integrates Altera system-level design tools such as DSP Builder and Qsys, and software and IP components that allow you to extend and customize the reference design to meet your own application needs. The framework also supports optimal partitioning decisions between software running on an integrated processor and IP performing portions of the motor control algorithm in the FPGA, to accelerate performance as required. For example, depending on the performance requirements of your system or the number of axes you need to support, you may implement all of the inner current control loop in hardware or entirely in software. The framework flexibly allows you to connect to the motor and power stages through off-chip ADCs, and feedback encoder devices and to connect to higher-level automation controllers through off-the-shelf digital encoder and industrial Ethernet IP cores, respectively.

The reference design offers vibration suppression, when you target the Cyclone V SoC devices. The design demonstrates how you can implement standard components (FFTs, FFT-post processing, IIR filter), to enable you to develop an automatic method for vibration suppression. You may use the FFTs and FFT postprocessing for condition monitoring, which detects vibrations that indicate degradation or wear and communicate the results to another system.

The reference design integrates Motor Control IP Suite components, a Nios® II soft processor subsystem, or ARM-based HPS, and software that uses an FOC algorithm. The reference design uses Altera's DSP Builder system-level design tool to implement the FOC algorithm.

DSP Builder provides a MATLAB and Simulink flow that allows you to create hardware optimized fixed latency representations of algorithms without requiring HDL/hardware skills. Altera provides DSP Builder fixed-point and floating point algorithms to demonstrate both options. The DSP Builder folding feature reduces the resource usage of the logic as an alternative to a fully parallel implementation. The reference design also includes an efficient Avalon® Memory-Mapped (Avalon-MM) interface that you can integrate in Qsys.

Related Information

Optimize Motor Control Designs with an Integrated FPGA Design Flow

Motor Control Boards

Table 1: FPGA Host Control Board Options

Board	Vendor	Website
Cyclone V SoC development board	Altera	**https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-cyclone-v-soc.html**
MAX 10 FPGA development board	Altera	**https://www.altera.com/products/boards_and_kits/dev-kits/altera/max-10-fpga-development-kit.html**
Terasic SoCKit	Terasic	**http://www.terasic.com**

Table 2: Motor Control Power Board Options

Board	Vendor	Website
Multiaxis Motor Control Board	Altera	**https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-multi-axis-motor-control.html**
FalconEye 2 HSMC Motor Control Board	devboards	**http://www.devboards.de**

An HSMC connector connects:

- The Cyclone V SoC development board to the Multiaxis Motor Control Board or the FalconEye 2 HSMC Motor Control Board
- The Cyclone V SoC and MAX 10 FPGA Development Boards and SoCKit to the FalconEye 2 HSMC Motor Control Board.

Note: The Multiaxis Motor Control Board supports multiple position feedback interfaces, however, the reference design only supports EnDat or BiSS.

For availability of the Multiaxis Motor Control Board, contact your Altera sales representative.

Related Information

- **Multiaxis Motor Control Board Reference Manual**
  For more information on the Multiaxis Motor Control Board
- **Motor Control IP Suite Components Data Sheet.**
  For more information on the Motor Control IP components

Features

The reference design offers the following features:

- Multiple field-oriented control (FOC) loop implementations:
  - Fixed- or floating-point implementation targeting the ARM Cortex A9 processor on SoC devices
  - Fixed- or floating-point implementation with Nios II processors targeting MAX 10 FPGA devices
  - Fixed and floating-point accelerator implementations designed using Simulink model-based design flow with DSP Builder
- Integration in a single FPGA of single and multiaxis motor control IP including:
  - PWM for two-level IGBT
  - Sigma delta ADC interfaces for motor current feedback and DC link voltage measurement
  - Position feedback with EnDat or BiSS
- Vibration suppression (SoC devices only):
  - DSP Builder-designed FFT accelerator
  - Peak detection
  - Suppression filter
- System Console GUI for motor feedback information, vibration suppression demonstration, and control of motors

Getting Started

Software Requirements on page 4

Downloading and Installing the Reference Design on page 4

Setting Up the Boards on page 6

Compiling the Design on page 7
Either compile your design or use the Altera-provided pre-compiled `.sof` from the **master_image** directory of your reference design variant.

Compiling the µC/OS-II HPS Software on page 7
If you are targeting the Cyclone V SoC development board or SocKit, compile the µC/OS-II version of the HPS software.

Compiling the Nios II Software on page 8

Programming the Hardware onto the Device on page 9

Downloading the DS-5 HPS Software to the Device on page 9

Downloading the Nios II Software to the Device on page 11
Prerequisites: import and compile the Nios II software in the Nios II Software Build Tools for Eclipse.

Creating and Booting an HPS SD Card Software Image on page 11
You create a bootable SD card that contains the software image, so you can boot the software automatically without using the DS-5 Debugger

Software Requirements

The reference design requires:

- The Altera Complete Design Suite version 15.0, which includes:
  - The Quartus II software v15.0
  - DSP Builder v15.0
  - The Altera Nios II Embedded design Suite (EDS) v15.0
- The Altera SoC EDS v15.0 (for designs that target the Cyclone V SoC), which includes ARM Development Studio 5 (DS-5) Altera Edition.
- Terminal emulator such as Tera Term for SoC based designs

Downloading and Installing the Reference Design

1. Download the relevant reference design **.par** file for your development kit and power board from the Altera Design Store.

   To obtain further support on the reference design, contact your local Altera sales representative.
2. Install the relevant reference design **.par** file for your development kit and power board.

Table 3: Design .par Files

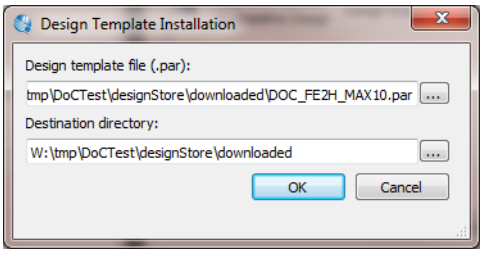
Variant	Development Kit	Power Board	Processor
DOC_4AXIS_CVSX	Cyclone V SoC	Altera 4 Axis	HPS
DOC_FE2H_CVSX	Cyclone V SoC	FalconEye 2 HSMC	HPS
DOC_FE2H_CVSX_XiP	Cyclone V SoC	FalconEye 2 HSMC	HPS
DOC_FE2H_MAX10	MAX 10 FPGA	FalconEye 2 HSMC	Nios II
DOC_FE2H_SoCKit	SoCKit	FalconEye 2 HSMC	HPS

Note: The XiP variant does not use external DDR3 memory and executes all code directly from flash memory.

3. In the Quartus II software, click **File** > **New Project Wizard**.
4. Click **Next**.
5. Enter the path for your project working directory and enter *variant name* from the table for the project name.
6. Click **Next.**

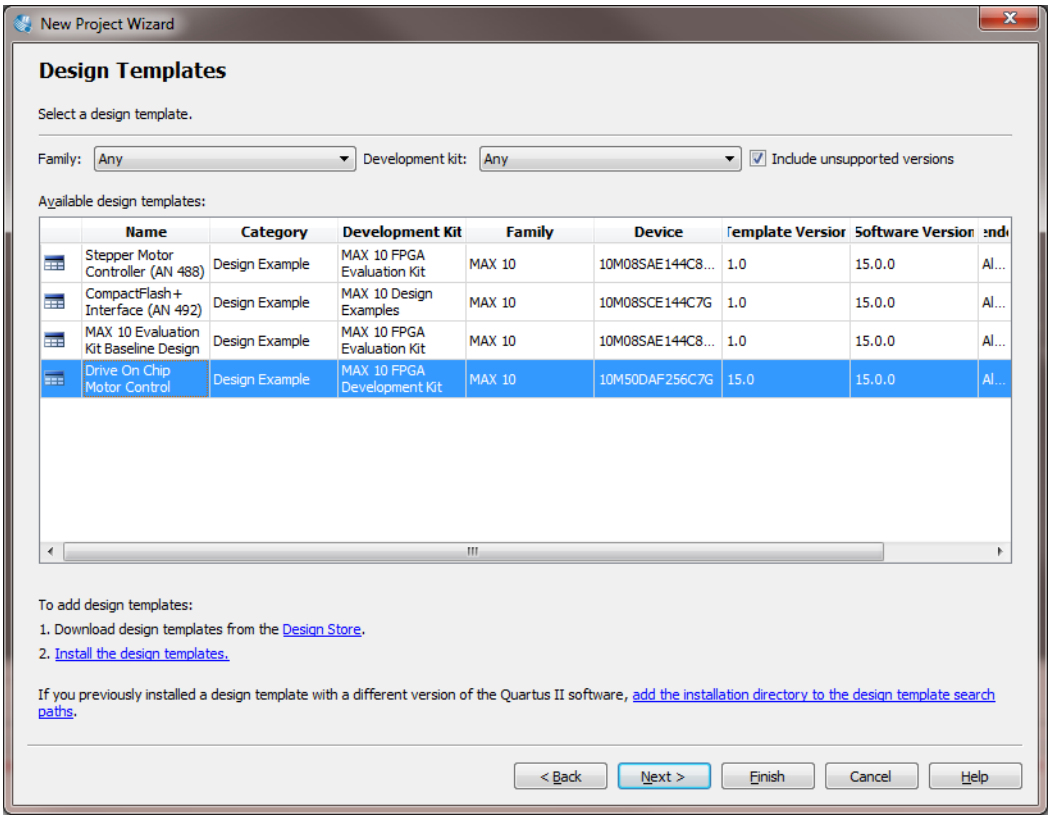
7. Select **Project Template**.
8. Click **Next**
9. Click **Install the design templates**.
10. Browse to select the **.par** file for the reference design and browse to the destination directory where you want to install it.

Figure 1: Design Template Installation



11. Click **OK** on the design template installation message.
12. Select the **Drive on Chip Reference design** design example.

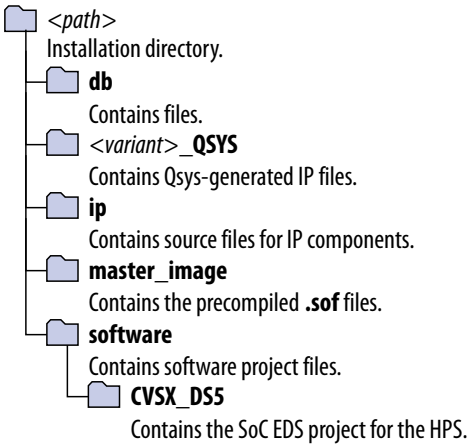
Figure 2: Design Template



13. Click **Next**.
14. Click **Finish**. The Quartus II software expands the archive and sets up the project, which may take some time.

Figure 3: Directory Structure

The reference design directory structure for the DOC_FE2H_CVSX variant



Related Information

Altera Design Store

Setting Up the Boards

Related Information

- **Cyclone V SoC Development Board Reference Manual**
  For information on setting up the Cyclone V SoC Development Board
- **MAX 10 FPGA Development Kit User Guide**
  For information on setting up the MAX 10 FPGA Development Board
- **Terasic**
  For information on setting up the SoCKit Development Board
- **Multiaxis Motor Control Board Reference Manual**
  For information on setting up the Multiaxis Motor Control Board
- **devboards**
  For information on setting up the FalconEye 2 HSMC Board

Setting Up the Motor Control Board with your Development Board

Caution: Before you begin, to prevent damage to the motor control board, ensure development board and power board are turned off.

1. Remove the SD card from the SoC board if it is fitted.
2. Connect the power board to the development board using the HSMC connector.
3. Connect a USB cable from the USB-Blaster connector on the development board to your computer. The Cyclone V SoC development board and SoCKit require an additional USB cable connected to the UART connector.
4. Open the terminal emulator to connect to the virtual COM port (which has a large number, e.g. 12) at 115K baud.

5. Apply power to the development board.
6. Compile the hardware and software and program the hardware and software to the development board:

   The terminal must display the correct FPGA and power boards, otherwise you might damage either board.
7. Apply power to the motor control power board.
8. Before reprogramming the FPGA, or removing power from the development boards, always remove power from the motor control power board first.

Using BiSS with FalconEye 2 Board

1. Remove the EnDat cable from the header on the HSMC adapter board.
2. Connect a BiSS cable from the header on the HSMC adapter board to the BiSS encoder.

Compiling the Design

Either compile your design or use the Altera-provided pre-compiled `.sof` from the **master_image** directory of your reference design variant.

1. By default, Altera configures <*variant*>**.v** for EnDat encoders for FalconEye 2 HSMC Motor Control Board projects; BiSS for all Multiaxis Motor Control Board projects. In <*variant*>**.v**, to use BiSS, add the following line; to use EnDat, remove it (if present).

   ```
   "define BISS"
   ```
2. Add the paths to the license files for the EnDat and BiSS IP components to the Quartus II license path.
   a. Click **Tools** > **License Setup**.
   b. Copy any existing license paths into a text editor.
   c. Browse to the EnDat license file `ip\endat_OCP\dsgn\vhdl_ava_ALT_enc_ocp\ocp _license.dat` and click **Open**.
   d. Copy and paste this license path into your text editor. Add a semicolon to separate this text from the original text.
   e. Browse to the BiSS license file `ip\biss_OCP\license.dat` and click **Open**.
   f. Copy and paste this license path into your text editor. Add a semicolon to separate this text from the original text and teh EnDat license.
   g. Copy and paste the text from the text editor to the Quartus II **License File** dialog box.

      The final license path is `<Project directory>\ip\endat_OCP\dsgn\vhdl_ava_ALT_enc_ocp;<Project directory>\ip\biss_OCP\license.dat;<original license paths>`. Alternatively, you can add this path to the Windows environment variable `LM_LICENSE_FILE`.
3. Click **Processing** > **Start Compilation** .

   Note: You may edit the reference design project in Qsys.

Compiling the µC/OS-II HPS Software

If you are targeting the Cyclone V SoC development board or SocKit, compile the µC/OS-II version of the HPS software.

1. To ensure that the environment is set up correctly, run SoC EDS command shell, then start DS-5 by typing `eclipse&` at the command prompt.

Note: Do not use the Windows start menu entry to start DS-5.

2. Specify the **\CVSX_DS5** directory as the workspace by browsing to the <*path*>**\software\CVSX_DS5** directory.
3. Click **OK** to create the workspace.

Note: Close the welcome to DS-5 tab if it shows.

4. Import the makefile project **CVSX_DS5**:
   a. Select **File** > **Import** to open the **Import** window.
   b. Select **General** > **Existing Projects into Workspace** and click **Next**.
   c. Click **Select Root Directory** and browse to the project directory, **DOC_CVSX** in the workspace directory, **CVSX_DS5**.
   d. Click **OK.**
   e. Click **Finish**.

Table 4: Software Project Directories

The linked directories that appear in the Project Explorer view under DOC_CVSX

Directory	Description
app	Application main entry point.
components	Motor control component source and header files.
hwlibs	The build process copies selected files from the Altera Complete Design Suite hardware libraries to this directory.
mc	Motor control source and header files.
objects	The build process creates object files here.
perf	Performance monitor source and header files.
platform	Platform (development kit and RTOS) specific source and header files.
source	Source code. This directory contains other linked subdirectories.
vibration	Vibration suppression demo source and header files.

5. Single-axis power boards default to EnDat; multiaxis power boards default to BiSS. To override the default encoder, edit **mc\mc.h** to define the `OVERRIDE_ENCODER` macro:

```
#define OVERRIDE_ENCODER SYSID_ENCODER_ENDAT

#define OVERRIDE_ENCODER SYSID_ENCODER_BISS
```

6. Rebuild the **DOC_CVSX** project: right-click **DOC_CVSX** project and click **Build Project**.

Compiling the Nios II Software

1. Start Nios II EDS. Click **Start** > **Altera** > **Nios II EDS** > **Nios II Software Build Tools**
2. Specify the **\software** folder as the workspace by browsing to the reference design **\software** directory.
3. Click **OK** to create the workspace.
4. Import application and board support package (BSP) projects:

a. Click **File** > **Import.**
b. Expand **General** and click **Existing Projects into Workspace**.
c. Click **Next**.
d. Browse to **\software**<*variant*> and click **OK**.
e. Click **Finish**.
f. Repeat steps **a** to **e** for <*variant*>**_bsp**.

5. Single-axis power boards default to EnDat; multiaxis power boards default to BiSS. To override the default encoder, edit **mc.h** to define the `OVERRIDE_ENCODER` macro:

   #define OVERRIDE_ENCODER SYSID_ENCODER_ENDAT

   #define OVERRIDE_ENCODER SYSID_ENCODER_BISS
6. Rebuild the BSP project: right-click <*variant*>**_bsp** project, point to **Nios II**, and click **Generate BSP**.
7. Build the application project: right-click <*variant*> project and click **Build Project**.

   Note: On Windows, the first time you build the project might take up to one hour.

   Note: Repeat steps 6 and 7 if you make any changes to the Qsys project.

Programming the Hardware onto the Device

Before you begin

Set up the motor control board.

1. In the Quartus II software, click **Tools** > **Programmer**.
2. In the **Programmer** pane, select **USB-Blaster II** (or **CV SoCKit** for the Terasic SoCKit) under **Hardware Setup** and **JTAG** under **Mode**.
3. Click **Auto Detect** to detect devices.
4. If you are targeting a Cyclone V SoC, select any **5CSX** device from the pop-up list. For MAX 10 devices, select any **10M50** device.
5. Double-click on the **File** field for the **5CSX** or **10M50** device from the pop-up list.
6. Select the **output_files/<project name>_time_limited.sof** and click **Open**.
7. Click **OK** on the time-limited **.sof** message.
8. Turn on **Program/Configure**.
9. Click **Start**.

   Note: Do not close the OpenCore Plus message that appears.

Related Information

Setting Up the Motor Control Board with your Development Board on page 6

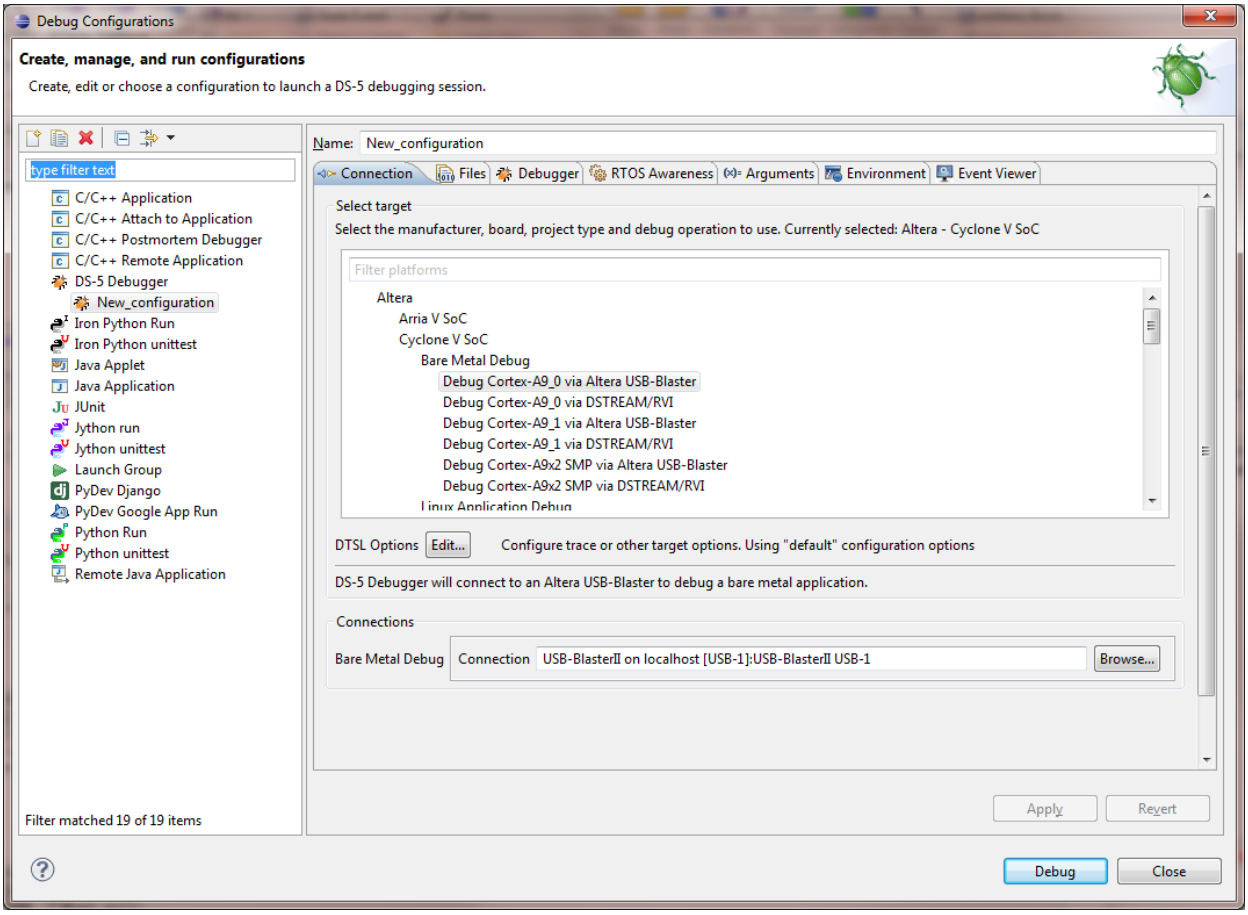
Downloading the DS-5 HPS Software to the Device

Before you begin

Import and compile the HPS software in DS-5. You must have a DS-5 debugger license to download HPS software.

1. Select **Run** > **Debug Configurations**.
2. Create new DS-5 debugger configuration:
   a. Right click on the **DS-5 Debugger launch type**
   b. Select **New**.
3. On the Connection tab, select **Debug Cortex-A9_0** under **Cyclone V SoC (Dual Core)|Bare Metal Debug**.
4. In the **Target Connection** drop-down select **USB-Blaster**.
5. In the **Bare Metal Debug Connection** field, click **Browse…** and select the **USB Blaster II** on localhost (or **CV SoCKit** on localhost for the SoCKit).

Figure 4: Connection Tab



6. Select the **Debugger** tab.
7. In **Run Control**, select **Connect Only**.
8. In **Host working directory** turn off **Use default** and change the path to `${workspace_loc:/DOC_CVSX}` by clicking **Workspace** and selecting **DOC_CVSX**
9. Click **Apply** to save the configuration, optionally specifying a name for the new configuration.
10. Click **Debug**.
11. Click **Yes** when the application asks you to switch to debug perspective.
12. In the **Command** text field enter the command `source debug.ds`.
13. Click **Submit** to load and run the preloader then load the application and run to `main()`.

14. Click **Continue** (green arrow) to run the application.
15. Check that the terminal console display shows the correct FPGA and power board combination. For example:

```
0: [DECODE SYSID] Decoding hardware platform from QSYS SYSID data : 0x00D112FE
0: [DECODE SYSID] Design Version : 15.0
0: [DECODE SYSID] FPGA Board : Cyclone V SX SoC Dev Kit
0: [DECODE SYSID] Power Board : FalconEye v2.0 HSMC Single-axis
0: [DECODE SYSID] Encoder Type : EnDat
0: [DECODE SYSID] 1 axes available
0: [DECODE SYSID] Axis 0 : Enabled
```

16. When the application is running, right-click on **Cortex-A9_0** and click **Disconnect from Target**.
17. Apply power to the power board. The motor starts to turn.

Downloading the Nios II Software to the Device

Prerequisites: import and compile the Nios II software in the Nios II Software Build Tools for Eclipse.

To program the Nios II software to the device:

1. In the Nios II EDS, on the Run menu, click **Run configurations...**.
   a. Double click **Nios II Hardware** to generate a new run configuration.
   b. Click **New_configuration**.
   c. On the **Project** tab select the **DOC_FE2H_MAX10** project in the **Project name** drop-down.
   d. Turn on **Enable browse for file system ELF file**. Browse to the **software\DOC_DE2115** and select **DOC_DE2115.elf**.
   e. On the **Target Connection** tab, click **Refresh Connections**. The software finds the USB-Blaster cable.
   f. Click **Apply** to save changes, optionally specifying a name for the new configuration.
   g. Click **Run** to start the software.
2. Check that the terminal console display shows the correct FPGA and power board combination. For example:

```
0: [DECODE SYSID] Decoding hardware platform from QSYS SYSID data : 0x00D111FE
0: [DECODE SYSID] Design Version : 15.0
0: [DECODE SYSID] FPGA Board : Cyclone V E Dev Kit
0: [DECODE SYSID] Power Board : FalconEye v2.0 HSMC Single-axis
0: [DECODE SYSID] Encoder Type : EnDat
0: [DECODE SYSID] 1 axes available
0: [DECODE SYSID] Axis 0 : Enabled
```

3. Apply power to the power board. The motor starts to turn.

Creating and Booting an HPS SD Card Software Image

You create a bootable SD card that contains the software image, so you can boot the software automatically without using the DS-5 Debugger

1. Create a bootable SD card image using the prebuilt image that the SoC EDS includes.
   a. On Windows, download and install win32diskimager.
   b. Find the bootable image for your development board.

      For the Cyclone SoC development board find `sd_card_linux_boot_image.img` in the Altera SoC EDS default installation directory: `C:\altera\15.0\embedded\embeddedsw\socfpga\`

`prebuilt_images` For the Terasic SoCKit, find `sd_image_sockit_20140902.img` (see related information).

c. Use win32diskimager or Linux to write the bootable image onto your SD card.

2. Copy **DOC_CVSX_uImage** to the FAT partition on the SD card.
3. Enter U-Boot:
   a. Insert the SD card in the board
   b. Apply power to the board.
   c. Open your Terminal Emulator software and connect to the board.Press the 'Cold Reset' (Altera SoC dev board) or 'HPS_RST' (SoCKit), while monitoring the terminal.
   d. Press **HPS_RST** (SoCKit) or **Cold Reset** (Altera development boards) and monitor the terminal.
   e. Press a key in the terminal window when prompted, before the countdown finishes and the operating system boots.
4. At the U-Boot prompt, edit the U-Boot environment by typing:

   setenv mmcload "mmc rescan;fatload mmc 0:1 0x01000000 doc_cvsx_uImage 0 0x40; go 0x01000000"
5. Save the new environment setting by typing:

   env save
6. Configure the FPGA.
7. Boot the software: press **HPS_RST** (SoCKit) or **Cold Reset** (Altera development boards) again.

Related Information

- **Programming the Hardware onto the Device** on page 9
- **Windows Disk Imager**
- **Terasic SoCKit bootable image**

Debugging and Monitoring Hardware with System Console

1. In Qsys, click **Tools** > **System Console** or in the Quartus II software, click **Tools .** > **System Debugging Tools** > **System Console** The Quartus II software sets the working directory to the project directory.
2. In System Console, in the Tcl console type:

   ```
   source DOC_debug_gui.tcl
   ```

General Tab

Connecting Demonstration

1. On the **General** tab, click **Connect JTAG** to connect to hardware. The **Messages** window displays:

   ```
   INFO: Connecting to JTAG Master /devices/5CSEBA6(.|ES)|5CSEMA6|..@2#USB-1/(link)/
   JTAG/(110:132 v1 #0)/phy_0/master
   INFO: Checking for System ID at 0x1000_0000 : Value = 0x00f012fe
   INFO: Version = 15.0 Device Family = 2 Powerboard Id = 1 Design Id = 254
   INFO: FPGA Board : Cyclone V SX SoC Dev Kit
   ```

```
INFO: Power Board : FalconEye v2.0 HSMC Single-axis
INFO: Design Version : 15.0
```

2. Under **Axis Select**, select **0**, **1**, **2**, or **3**. for multiaxis power board only.
3. Click **Reset Motor** to reset the motor control state machine and restart the motor in constant speed mode (100 rpm).
4. To test the motor without any current or position sensor feedback, click **Enable Open Loop Mode**.
5. To re-enable closed-loop control and continue to other demonstrations, press **Disable Open Loop Test Mode**.

Changing DSP Mode

1. On the **General** tab, click **Change DSP Mode** to change between the following DSP modes:
   - Software fixed-point
   - Software floating-point
   - DSP Builder fixed-point
   - DSP Builder floating-point

Using Speed Control

1. Under **Speed Demo Setup** enter **Speed Value 1**, **Speed Value 2**, and **Speed Interval**, to set the speed and time interval between speed value switches. You can select speeds from –3,000 to +3,000 rpm.
2. Click **Run Speed Demo** to start a sequence where System Console switches the motor speed command alternately between the two speeds at the specified time interval.
3. Alternatively, select a new **Speed Request** value in the drop-down list.

Using Position Control

1. Type in two position values (from –36,000 to + 36,000 degree), a speed limit and a position interval.
2. Click **Run Position Demo**.

System Console Dials

On the **General** tab, System Console displays the following dials:

- DSP Latency:
  - Measures the runtime of the field-oriented control (FOC) algorithm with software timers in the interrupt service routine (ISR)
  - Shows the time taken to execute the algorithm in the different DSP modes
- Speed. The instantaneous speed (rpm) of the motor
- Position. The position of the motor shaft in degrees. System Console sets the scale automatically based on the parameters you configure in the position demonstration.

Note: If you set **Position Interval** to less than several seconds, you may not see the changes on the dial because of the update rate of the GUI.

Control and Diagnostics Tab

On the **Control and Diagnostics** tab you can configure an integrated storage buffer that behaves similarly to a storage oscilloscope. System Console captures a range of variables within the FOC DSP loop at a 16-kHz sample rate and then displays them interactively on graphs. You can configure the trace trigger settings and storage depth. System Console displays the results and automatically writes them to a `.csv` file for later analysis.

Monitoring Performance

1. On the **Control and Diagnostics** tab, under **Trigger Signal**, select the signal you want to trigger the trace data capture. If you select **Always**, the trigger is always active.
2. Under **Trigger Edge**, select one of the following trigger types:
   a. **Level** (trigger signal must match this value)
   b. **Rising Edge** (trigger signal must transition from below to above this value)
   c. **Falling Edge** (trigger signal must transition from above to below this value)
   d. **Either Edge** (triggers on both falling and rising edge conditions).
3. Under **Trigger Value**, select the value that **Trigger Edge** uses to compare the signal value against.

   Note: These values are raw integer values, not the scaled values that System Console displays in the graphs. Convert the raw integer values into scaled values.
4. Click **Update Trigger**, if you update the **Trigger Value**.
5. Under **Trace Depth**, select the number of samples to capture and display. System Console can store up to 4,096 samples on SoC or 2,048 samples on MAX 10 devices. Select a lower number of samples to make the System Console update rate faster, and zoom in on the graph as the graph scale autosizes to the number of samples.
6. Specify a **Trace Filename**. System Console saves the trace data saved to a **.csv** file.
7. Click **Start Trace** to start the trace; click **Disable Trace** again to stop the trace.

Related Information

Signals and Raw Versus Physical Values on page 14

Signals and Raw Versus Physical Values

Table 5: Signals and Physical Versus Raw Integer Values

Signal Name	Description	Raw Integer Values	Equivalent Physical Value
Vu Vv Vw	Space vector modulation voltages applied to motor terminals u, v and w.	0 to 3,125	-200 to +200 V
Iu Iv Iw	Measured feedback currents. Iv is derived from Iu and Iw.	-5,120 to +5,120	-1 to 1 A

Signal Name	Description	Raw Integer Values	Equivalent Physical Value
Id IdCommand Iq IqCommand	FOC direct and quadrature currents and their commanded values. IdCommand=0 for a PMSM.	-5, 120 to +5,120	-1 to 1 A
Speed Speed Command	Speed derived from encoder readings and its commanded value.	-32,768 to +32,768	-2048 to +2048 rpm
Position Position-Command	Position derived from encoder reading, and its commanded value.	0 to 65,536	0 to 1rev
Vd Vq	FOC direct and quadrature voltages. The **Voltage Output Limit** also uses this scaling, but only takes positive values.	-1,563 to +1,563	-200 V to +200 V
Vd_unfilt Vq_unfilt	Vd and Vq before the suppression filter (if your design includes vibration suppression)	-1,563 to +1,563	-200 V to +200 V

System Console Graphs

On the **Control and Diagnostics** tab, System Console displays the following graphs

- Space vector modulation (U,V,W) voltage.
- Feedback current (U,W,V). System console measures U and W, but derives V.
- Requested direct and quadrature currents versus actual currents.
- Requested versus actual speed.
- Position from encoder versus position command. Only available when you use position demonstration.

Tuning the PI Controller Gains

The reference design contains three PI control loops. You can tune the gain of each PI control loop. When tuning these gains, only change the values a little at a time while monitoring the performance on the adjacent graphs.

1. To tune PI controller gains, on the **PI Tuning** tab, under **Current Control Loop**, enter values for:

   - **Kp** (proportional gain).
   - **Ki** (integral gain).
   - **Current Command Limit**
   - **Output Voltage Limit**

   The design applies the current command limit in three places to limit the current:

   - The speed PI loop integrator.
   - The speed PI loop output (current command).
   - The current PI loop error. .

   See `I_sat_limit` in function `update_axis` in `motor_task.c`

   The design applies the output voltage limit in two places to limit the applied voltage:

- Current PI loop integrator.
- Current PI loop output (Voltage command)

See `V_sat_limit` in function `update_axis` in `motor_task.c`.

Note: For the **Current Command Limit** and **Output Voltage Limit**, the values you enter are based on raw values. The scaling is the same as for the trigger function values.

2. Click **Update Parameters.**
3. Under **Speed Control Loop**:
4. Enter values for **Kp** (proportional gain) and **Ki** (integral gain).
5. Under **Position Control Loop**:
6. Enter values for **Position Kp** and **Position Ki.** Click **Update Parameters**.

Related Information

Signals and Raw Versus Physical Values on page 14

Vibration Suppression Tab

About the Demonstration

The demonstration sets the control parameters and runs waveforms to demonstrate vibration suppression. Alternatively, you may change the settings and run waveforms manually. Stop the demonstration before making manual changes.

Note: Do not run the demonstration and change the settings simultaneously.

Note: The reference design does not support Vibration suppression on MAX 10 devices or XiP variants.

The demonstration shows how to improve speed control. Speed is never perfectly constant because motors include discrete parts such as magnets, electrical coils and steel cages to hold these components. As the motor rotates, the torque produced for a given current magnitude varies. These variations (cogging torques) produce accelerations leading to speed variations. The feedback controller adjusts the current to counteract the speed variations depending on the strength of the control gains. Stronger gains reduce speed variation, but may lead to control instability or amplify mechanical resonances.

The demonstration shows the level of speed variation with default speed control gains. It then shows the speed control step response. The step response should be fast and with little overshoot or oscillation, so the speed output should look similar to the square-wave input command. The speed control step response with standard gains is slow and may stop briefly at zero speed because of friction when the motor changes direction. Increasing the speed control loop proportional gain makes the response look much faster and without any visible stop at zero speed. However, the motor produces high-frequency noise that is audible and visible in the FFTs as a broad peak around 1kHz. In a real system with a mechanism connected to the motor, this noise can easily excite mechanical vibrations. To counteract this undesirable change, apply the suppression filter using a broad notch characteristic centred on 1kHz. The resulting waveform still has a fast, square shape, but the filter suppresses the noise. The speed variation with the combination of high gain and filter is much less than the original graph, showing the benefit of the faster control.

Starting the Demonstration

1. Click **Start demo**.
2. Click **Advance demo** to start the demonstration step 1. The demonstration highlights the text that describes this step.
3. Click **Advance Demo** again to start each consecutive step.
4. The demonstration finishes when you click **Advance Demo** and all text shows again with no highlighting.
5. To repeat the demonstration, click **Advance Demo** again.

FFT0 and FFT1 Graphs

On the **Vibration Suppression** tab**,** System Console displays the following graphs:

- Input signal to FFT0 against time.
- The magnitude of FFT0 against frequency.
- Input signal to FFT1 against time.
- The magnitude of FFT1 against frequency.

The graphs update at the same time as the graphs on the **Control and Diagnostics** tab whenever the trigger function in that tab activates. Within the SoC, the 4,096pt FFT data recalculates after every 64 new data points. The data sample rate is 16 kHz, thus the FFTs recalculate at 250 Hz.

System Console saves both the time- and frequency-based data to the `.csv` file that you specify in the **Control and Diagnostics** tab. You can then verify offline the correctness of the FFT calculation.

The FFT magnitude data is in dB (i.e. 20*log10 absolute value in physical units) similar to a control system Bode plot.

A basic peak detection algorithm enables you to measure particular peaks and develop automated vibration suppression. For each of the two FFTs, the algorithm finds the maximum magnitude value between two specified frequency limits. Thus, you can search for peaks within a physically relevant range without seeing large FFT magnitudes that often occur at low frequencies. System Console shows the peaks with a cross-hair superimposed on the FFT magnitude graph. It shows the frequency and magnitude values below the graph.

Figure 5: Using the Graphs

You can zoom into an area of the graph by clicking and dragging the cursor over that area. To return to the original scaling, right-click on the graph and select **Auto Range, Both Axes**.'

Setting Up FFT

1. On the **Vibration Suppression** tab, select the FFT signal.
2. Select the frequency scale: **linear** or **log**.
3. Select the peak detection low and high limit frequencies.
4. Click **Update Limits**.

Related Information

Signals and Raw Versus Physical Values on page 14

Setting Up Command Waveform

You can set up the command waveform to generate repeating waveforms that allow you to examine the quality of control.

1. Enter values for the parameters.

Table 6: Command Waveform Parameters

Parameter	Value	Description
Waveform type	Sine, Triangle, Square, Sawtooth	The shape of the repeating waveform.
Period (samples)	2 to 32768	Period of the waveform, in 16 kHz samples. For example, 2000 is equivalent to 8 Hz.
Position amplitude	-180 to +180	+/- amplitude of the waveform, if running in position control. Negative values create a 180 degree phase shift.
Speed amplitude (rpm)	-2048 to +2048	+/- amplitude of the waveform, if running in speed control. Negative values create a 180 degree phase shift.
On/Off	On or Off	Switches the waveform on or off. The waveform is superimposed on the speed or position set points from the **General** tab.

2. Click **Update waveform**.

Setting Up Second Order IIR Filter

1. Enter values for the IIR filter.

Table 7: IIR Filter Parameters

Parameter	Values	Description
Fn (Hz)	2 to 8000	The 'natural frequency' of the filter numerator, $\Omega_n/2\pi$.
Fd (Hz)	2 to 8000	The 'natural frequency' of the filter denominator, $\Omega_d/2\pi$.
Zn (nondimensional)	0 to 1000	The damping factor of the filter numerator, ζ_n. Values less than 0.707 produce a trough around Fn before the numerator gain increases; larger values give a smoother (more damped) characteristic.
Zd (nondimensional)	0 to 1000	The damping factor of the filter denominator, ζ_d. Values less than 0.707 produce a peak around Fd before the denominator gain decreases; larger values give a smoother (more damped) characteristic.
On/Off	On or off.	Switches the filter on or off.

Parameter	Values	Description
Input gain	0 to 1	This gain is in series with the filter and multiplies the complete control loop. Normally leave at 1, but setting it to smaller values reduces the overall control loop gain if required to improve control loop stability.

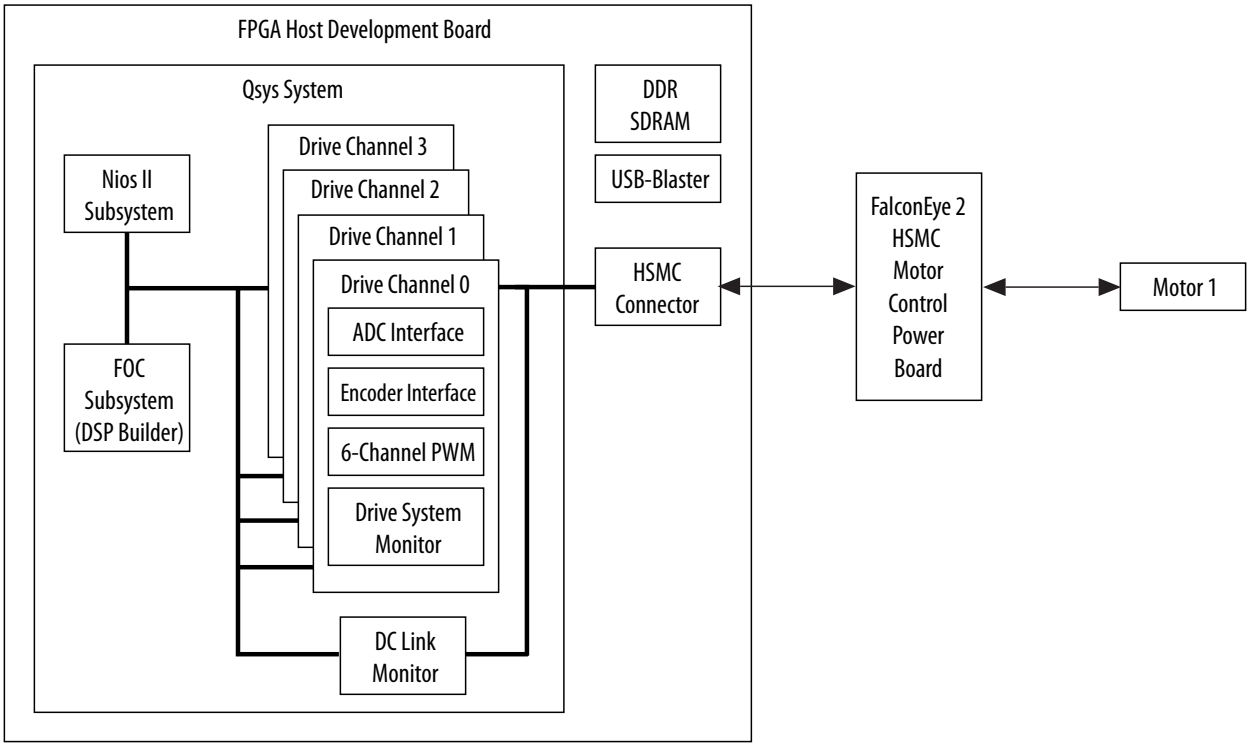
2. Click **Update Filter**.

Related Information

IIR Filter Tuning Parameters on page 29

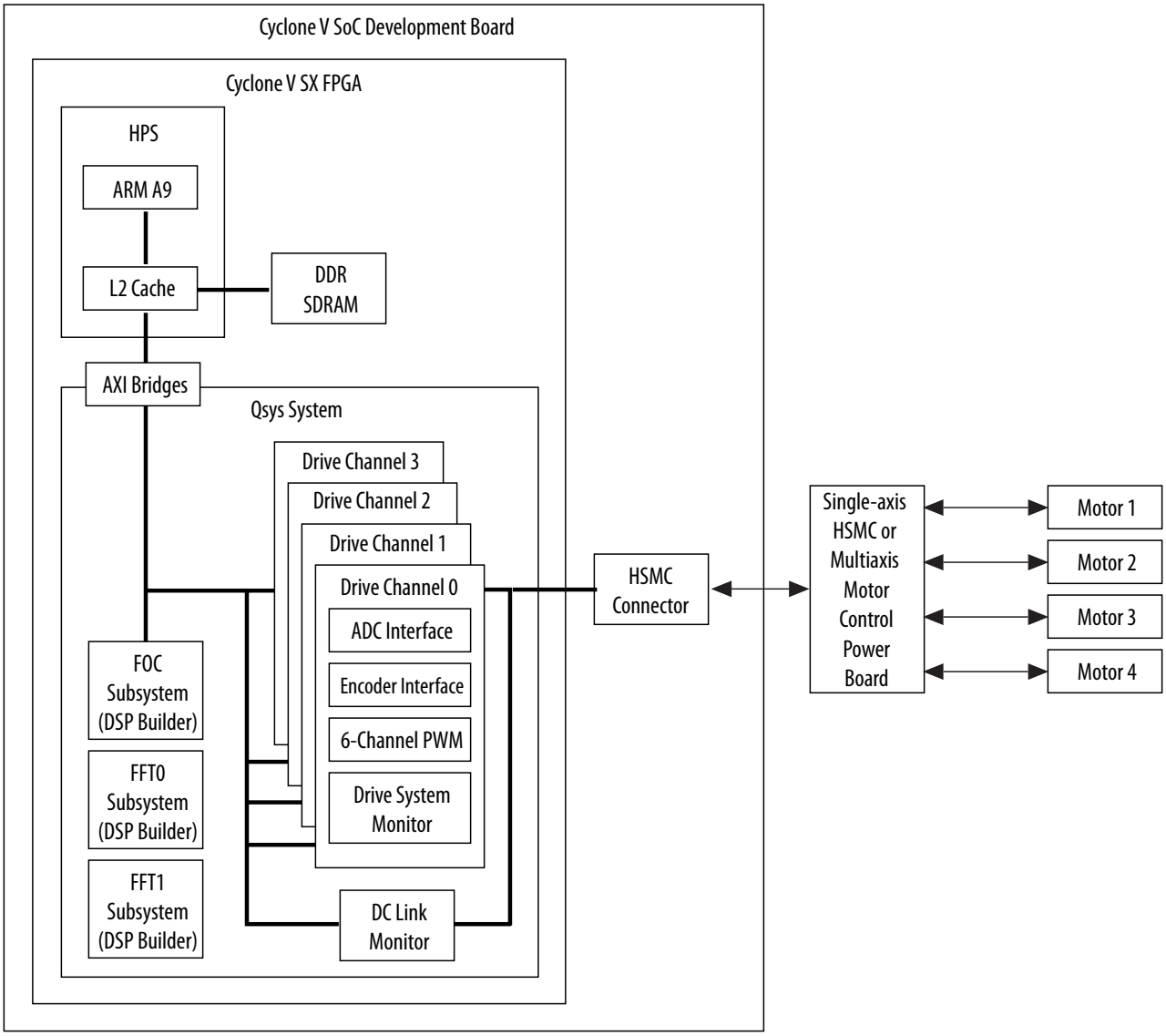
Functional Description

Figure 6: Block Diagram for MAX 10 FPGA Development Boards



Note: Only the multiaxis power board can drive four motors.

Figure 7: Block Diagram for Cyclone V SoC Development Board and SoCKit



The Qsys system consists of:

- DC link monitor
- FOC subsystem
- FFTs (SoC only)
- Processor subsystem
- Four drive channels comprising the following motor control peripheral components:
  - 6-channel PWM
  - ADC interface
  - Drive system monitor
  - Encoder interface (BiSS or EnDat)

DC Link Monitor

The DC link monitor comprises:

- Parallel input and output (PIO) for PFC monitor
- DC link voltage monitor
- DC link current monitor

FOC Subsystem

The Drive-On-Chip Reference Designs use DSP Builder to generate the HDL code for floating-point and fixed-point implementations of the field-oriented control (FOC) algorithm. The Nios II processor uses this DSP Builder-generated FOC IP as a coprocessor and moves the data between the FOC IP and the peripherals .

Note: Alternatively, the reference design includes software implementations of the FOC algorithm with the same FOC functionality. You can select which implementation to run using the Debug GUI. In all FOC implementations, the reference design performs the reverse Clarke transform as part of the SVM function in software.

FOC controls a motor's sinusoidal 3-phase currents in real time to create a smoothly rotating magnetic flux pattern, where the frequency of rotation corresponds to the frequency of the sine waves. FOC controls the current vector to keep:

- The torque-producing quadrature current, Iq, at 90 degrees to the rotor magnet flux axis
- The direct current component, Id, (commanded to be zero) inline with the rotor magnet flux.

The FOC algorithm:

1. Converts the 3-phase feedback current inputs and the rotor position from the encoder into quadrature and direct current components using Clarke and Park transforms.
2. Uses these current components as the inputs to two proportional and integral (PI) controllers running in parallel to limit the direct current to zero and the quadrature current to the desired torque.
3. Converts the direct and quadrature voltage outputs from the PI controllers back to 3-phase voltages with inverse Clarke and Park transforms.

The FOC algorithm includes:

- Forward and reverse Clarke and Park transforms
- Direct and quadrature current
- Proportional integral (PI) control loops
- Sine and cosine
- Saturate functions

About DSP Builder

DSP Builder advanced blockset supports bit-accurate simulation and VHDL generation of the full range of fixed-point and floating-point data types available in Simulink. Floating-point data types give a high dynamic range, avoid arithmetic overflows, and avoid the manual floating- to fixed-point conversion and scaling steps necessary in algorithm development. You can optimize the data types to adjust hardware usage and calculation latency, and run Simulink simulations to confirm adequate performance.

After you develop the algorithm in Simulink, DSP Builder can automatically generate pipelined HDL that it targets and optimizes to the chosen FPGA device. You can use this VHDL in a HDL simulator such as

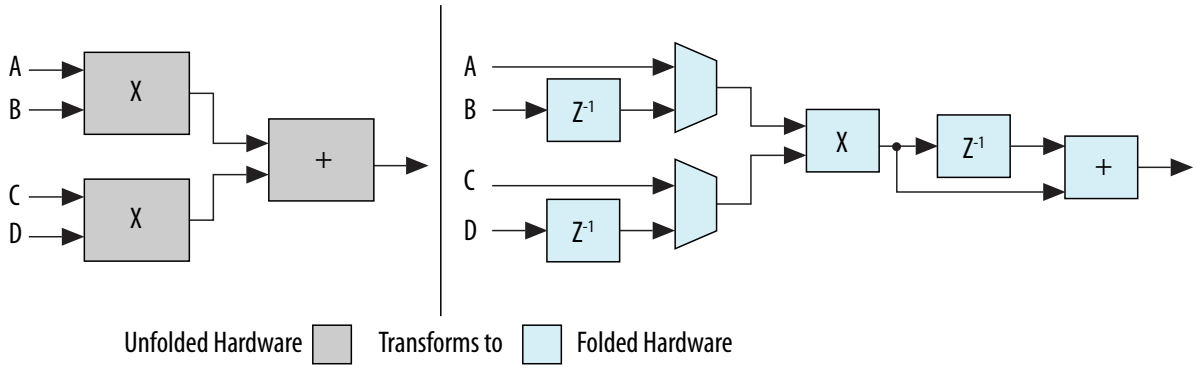
ModelSim to verify the generated logic versus Simulink and in the Quartus Prime software to compile the hardware. DSP Builder gives instant feedback of the VHDL's logic utilization and algorithm latency in automatically generated Simulink reports.

DSP Builder Folding

DSP Builder generates flat parallel models that can receive and process new input data every sample time. However, designs which have a much lower sample rate than the FPGA clock rate, such as this FOC design (16 kHz versus 100 MHz), can use the DSP Builder folding feature to trade off an increase in algorithm latency for a decrease in the used FPGA resources. This feature allows the design to use as much hardware parallelism as necessary to reach the target latency with the most cost effective use of FPGA resources without making any changes to the algorithm.

The DSP Builder folding feature reuses physical resources such as multipliers and adders for different calculations with the VHDL generation automatically handling the complexity of building the time division multiplexed (TDM) hardware for the particular sample to clock rate ratio.

Figure 8: Unfolded and Folded Hardware Examples



DSP Builder Model Resource Usage

Altera compared the FOC algorithm as a single precision floating-point model and a model that uses the folding feature. When you use folding, the model uses fewer logic elements (LEs) and multipliers but has an increase in latency. In addition, a fixed-point model uses significantly fewer LEs and multipliers and has lower latency than the floating-point model.

Altera compared floating- and fixed-point versions of the FOC algorithm with and without folding. In addition, Altera compared using a 26-bit (17-bit mantissa) instead of standard single-precision 32-bit (23-bit mantissa) floating point implementation. 26-bit is a standard type within DSP Builder that takes advantage of the FPGA architecture to save FPGA resources if this precision is sufficient.

Cyclone V devices use ALMs instead of LEs (one ALM is approximately two LEs plus two registers) and DSP blocks instead of multipliers (one DSP block can implement two 18-bit multipliers or other functions).

Table 8: Resource Usage Comparison for Cyclone V Devices

Algorithm	Precision (Bits)	Folding	Logic Usage (ALMs)	DSP Usage	Algorithm Latency (µs)
FOC, floating point including filter, DFf_float_alu_av.slx	32	No	11.5k	31	0.71
FOC, floating point including filter, DFf_float_alu_av.slx	32	Yes	3.9k	4	2.30
FOC, floating point including filter, DFf_float_alu_av.slx	26	No	11k	31	0.70
FOC, floating point including filter, DFf_float_alu_av.slx	26	Yes	3.6k	4	2.34
FOC, fixed point, including filter, DFf_fixp16_alu_av.slx	16	No	1.6k	36	0.22
FOC, fixed point, including filter, DFf_fixp16_alu_av.slx	16	Yes	2.3k	2	3.31

Table 9: Resource Usage Comparison for MAX 10 Devices

Algorithm	Precision (Bits)	Folding	Logic Usage (LEs)	Multiplier Usage	Algorithm Latency (µs)
FOC, floating point without filter, DF_float_alu_av.slx	32	No	30k	53	0.52
FOC, floating point without filter, DF_float_alu_av.slx	32	Yes	6.5k	10	1.75
FOC, floating point without filter, DF_float_alu_av.slx	26	No	23k	23	0.47
FOC, floating point without filter, DF_float_alu_av.slx	26	Yes	5.4k	6	1.61
FOC, fixed point, without filter, DF_fixp16_alu_av.slx	16	No	2.2k	12	0.14
FOC, fixed point, without filter, DF_fixp16_alu_av.slx	16	Yes	2.7k	2	2.08

The results show:

- The model with folding uses fewer processing resources but has an increase in latency.
- A floating-point model with folding also uses significantly fewer logic resources (LEs/ALMs) than a model without folding.
- The 26-bit floating-point format saves significant resources compared to 32-bit in MAX 10 devices but proportionally less in Cyclone V SoCs.
- The fixed-point algorithms without folding use the fewest logic resources and give the lowest latency.

The reference design implements these FOC configurations:

- Floating-point 26-bit model on MAX 10 devices ; 32-bit on Cyclone V, both with folding.
- Fixed-point 16-bit model without folding.

Related Information

Generating VHDL for the DSP Builder Models for the Drive-On-Chip Reference Designs on page 26

DSP Builder Design Guidelines

Use these design guidelines to reduce FPGA resource usage with folding.

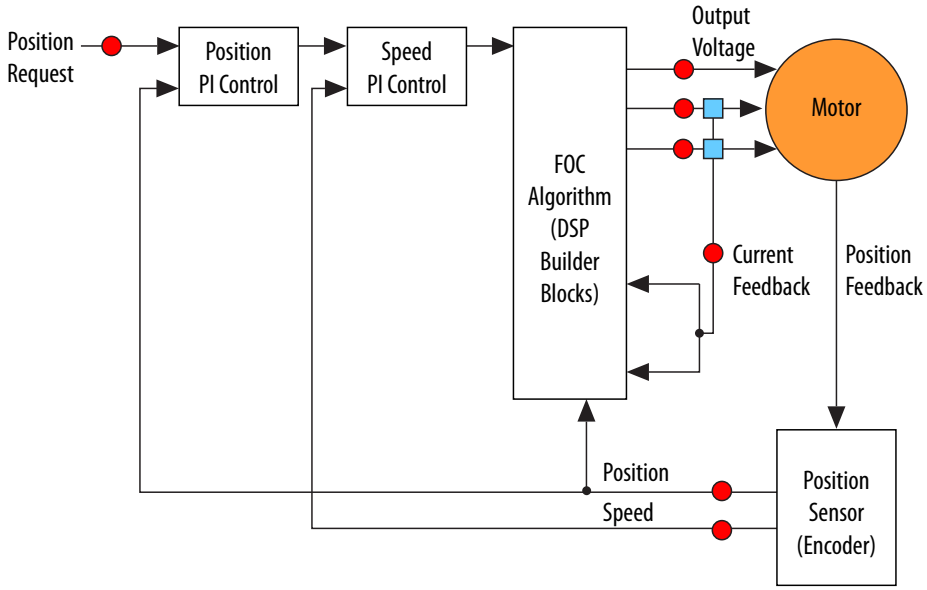
In your design:

- For fixed-point designs use the variable precision support in DSP Builder. Instead of using classical 32-bit datapath, investigate the algorithm and reduce the datapath to a dimension closer to the DSP block size.
- For fixed-point datapaths, disable bit growth for adders and subtracters. For example, use 27-bit datapaths on Cyclone V devices. The bit width should provide sufficient dynamic range for handling the values in the algorithm.
- Reduce the output of fixed-point multipliers to the same size as the inputs to better integrate in the datapath.
- Use smaller components when available. For example, pure sin and cos blocks require a range reduction stage. Use the smaller sin(pi*x) and cos(pi*x).
- Restructure a sin(pi*x) and a cos(pi*x) into a sin(pi*x) and sin(pi*(0.5-x)) to allow folding to reduce resource usage.
- Ensure that the select line of a multiplexer does not use more bits than necessary. For example, for a 2:1 multiplexer, the select line should be 1 bit.

DSP Builder Model for the Drive-On-Chip Reference Designs

The top-level model is a simple dummy testbench with constant inputs of the correct arithmetic types to control hardware generation, which includes the FOC algorithm model.

Figure 9: DSP Builder Model



The FOC algorithm comprises the FOC algorithm block and a latch block for implementing the integrators necessary for the PI controllers in the FOC algorithm. DSP Builder implements the latches outside because of limitations of the folding synthesis.

The reference design includes fixed-point and floating-point models that implement the FOC algorithm.

Each model calls a corresponding `.m` setup script during initialization to set up the arithmetic precision, folding factor, and target clock speed. The folding factor is set to a large value to minimize resource usage.

Table 10: Default settings in Setup Script

Model	Folding Factor	Clock Speed (MHz)	Input Precision	Output Precision
Fixed point	500	100	sfix16En10	sfix32En10
Floating point	500	100	sfix32En10	sfix32En10

The following models generate the FOC block including the Avalon-MM interface:

- `DF_float_alu_av.slx` for floating-point designs
- `DF_fixp16_alu_av.slx` for fixed-point designs

Verification models stimulate the FOC algorithm using dynamically changing inputs:

- `verify_DF_float_alu.slx`
- `verify_DF_fixp16_alu.slx`

Closed-loop simulation models validate that the FOC correctly controls a motor in simulation:

- `sim_DF_float_alu.slx`
- `sim_DF_fixp16_alu.slx`

A Simulink library model contains the main FOC algorithm code, which the models reference:

- `foc_blocks.slx`

Generating VHDL for the DSP Builder Models for the Drive-On-Chip Reference Designs

1. Start DSP Builder.
2. Change the directory to the **ip\dspba**.
3. If you want a different numeric precision, edit the **setup_**<*Simulink Model*>**.m** file corresponding to the model before opening it.
4. Load the model. Check the status of the orange DSP Builder folding block. If the model includes it, folding is enabled. If it is removed or commented out, the model does not use folding.
5. On the Simulation menu, click **Start**.

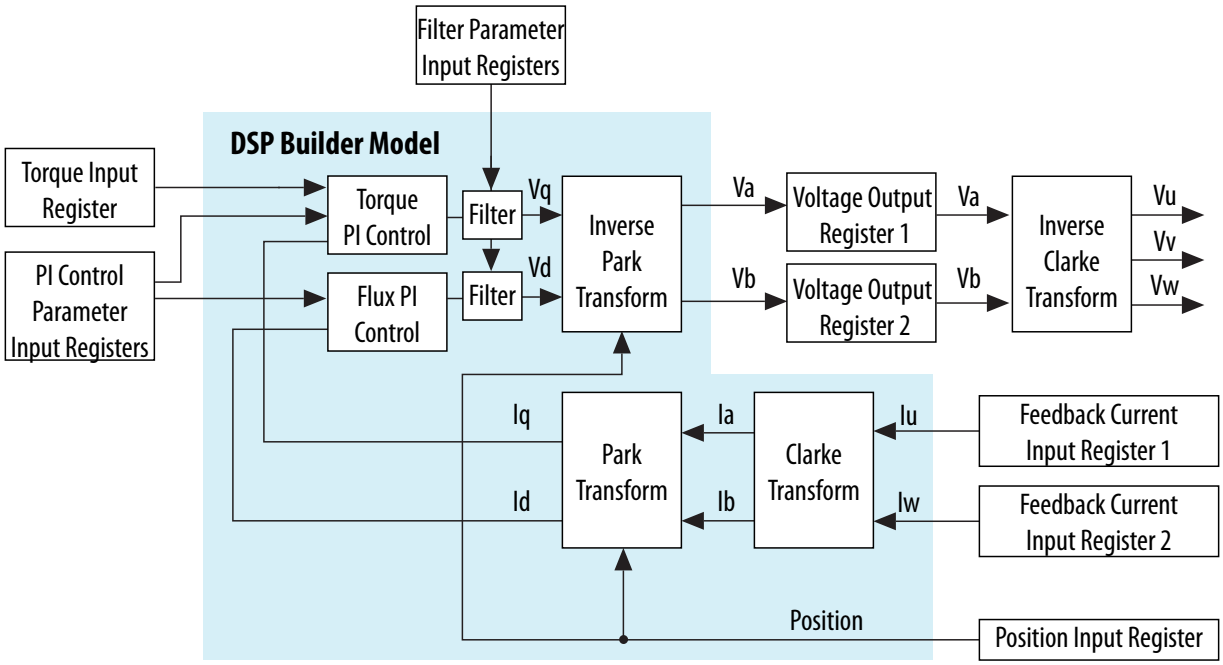
   DSP Builder generates the VHDL files in **ip\dspba\rtl** (for Cyclone V devices) or **ip\dspba\rtlmax10** (for MAX 10 devices).

Avalon-MM Interface

The Drive-On-Chip Reference Design DSP Builder-generated VHDL has a signal interface that matches the connections in Simulink. In the DSP Builder models, feedback currents, position feedback, torque command, and gain parameters are all parallel inputs into the system and voltage commands are parallel outputs.

To allow direct connectivity in Qsys, the top-level DSP Builder design adds blocks to terminate the parallel inputs and outputs and handshaking logic with an Avalon-MM register map.

Figure 10: FOC Model integrated in Simulink with Avalon-MM Register Map



DSP Builder generates a `.h` file that contains address map information for interfacing with the DSP Builder model.

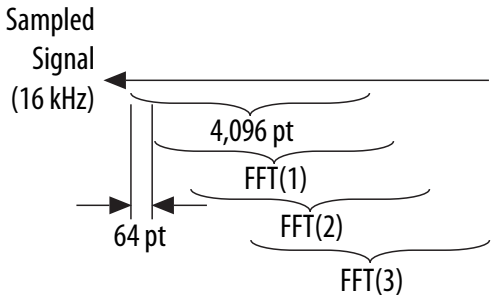
To run the DSP Builder model as part of the drive algorithm, a C function passes the data values between the processor and DSP Builder. The handshaking logic ensures synchronization between the software and hardware. The software sets up any changes to hardware parameters such as PI gains, writes new feedback currents, position feedback and torque command input data before starting the DSP Builder calculation. The software then waits for the DSP Builder calculation to finish before reading out the new voltage command data.

The ISR that runs the FOC algorithm calls the C function with an option to switch between software and DSP Builder implementations at runtime.

FFTs

FFT0 and FFT1 correspond to the two 4,096pt FFT blocks that the design implements on the SoC for vibration suppression. The design provides two FFTs to enable immediate comparison between the FFTs of different internal control signals. Each calculates an FFT on the last 4,096 samples input to it. The number of samples between each new calculation is configurable. The design uses 64 samples between each new calculation, so each new FFT is based on the same data set as the previous one except for 64 new points. Given the sample rate of 16kHz, the design produces new FFT data sets at 250Hz. Thus vibration detection can react quickly, before equipment is damaged.

Figure 11: FFT Overlapping Data Sets



Servo FFTs

The design uses two parallel instantiations of the DSP Builder Advanced Blockset servo FFT. The servo FFT is a folded FFT design that uses minimal FPGA resources but sufficient performance for typical industrial drive applications.

Table 11: FFT Resource Usage

Device	Logic	Memory	Other
Cyclone V	550 ALMs	29 M9K	2 DSP blocks

The processing time with a 100-MHz clock is around 0.6ms including the time to clock data out of the FFT block again. The processing time for two parallel FFTs is the same. You may choose other FFT implementations for the FPGA for faster processing times if required, at the expense of FPGA resources.

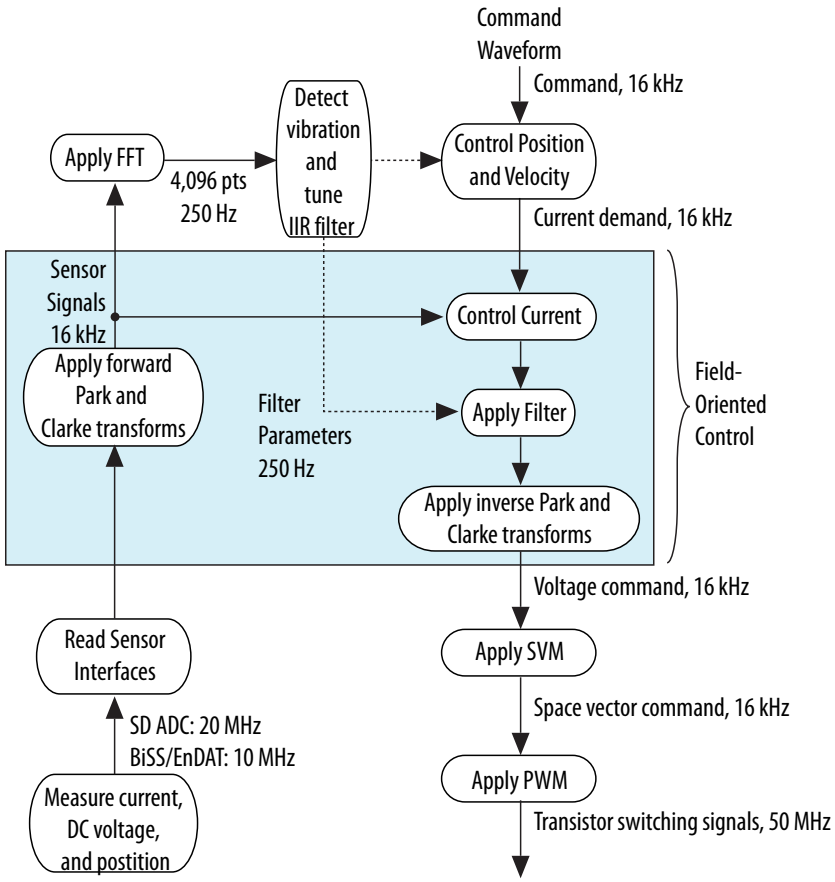
Running test_servofft.mdl

1. When you run the `test_servofft.mdl` Simulink model in the project **dspba** directory, **setup_test_servofft.m** runs the set up configuration parameters for the FFT and imports sampled motor current data from the input data for the simulation file (**test_servofft_sampledata.csv**).
2. DSP Builder generates the HDL code.
3. Simulink runs the simulation, using the sampled current data from **test_servofft_sampledata.csv** as FFT input.
4. **analyze_test_servofft.m** collects the simulation output and creates some MATLAB plots to verify correct calculation. It compares the simulated servo FFT output with that from the MATLAB FFT and with output data collected from servo FFT running in hardware.

Vibration Suppression

The standard FOC loop has a configurable digital filter to enable you to adjust the loop frequency response, which avoids exciting identified vibrations. An outer control loop analyzes motor feedback signals using FFTs to identify any vibrations that may be present. If the design detects vibrations, it automatically adjusts the control gains in the standard control loop and the filter settings to minimize the vibration. The reference design does not include automatic detection and suppression. For detection use System Console; for suppression manually configure the IIR filter.

Figure 12: Control Loop Flow Diagram for Vibration Suppression



Methods of automatically tuning filter and control gains depend on:

- Knowledge of the mechanical and electrical properties of the mechanisms that the motor drives
- The effects of different filter settings on the overall control response

Command Waveform

The command waveform generator provides repeating waveforms. The design uses these waveforms for developing the overall control system including any vibration suppression features. The waveforms generate repeatable behavior for given control system settings.

You can choose the waveform shape, period (expressed as a number of 16 kHz samples), and amplitude.

IIR Filter

The design applies an IIR filter to the FOC voltages Vd and Vq before it applies them to the motor.

The IIR filter is a configurable second-order digital filter, based on the second-order continuous time transfer function (in Laplace notation):

$$G(s) = \frac{1 + 2\zeta_n\left(\frac{s}{\Omega_n}\right) + \left(\frac{s}{\Omega_n}\right)^2}{1 + 2\zeta_d\left(\frac{s}{\Omega_d}\right) + \left(\frac{s}{\Omega_d}\right)^2}$$

The design converts the continuous time formulation to a discrete-time formulation using a combination of the Tustin (bilinear) transformation and frequency prewarping, which ensures that the discrete time and continuous time filter characteristics match at a specified frequency Ω:

$$s \rightarrow \frac{\Omega}{\tan\left(\frac{\Omega T}{2}\right)}\frac{z-1}{z+1}$$

The design applies prewarping to the denominator terms with Ω = Ωd and to the numerator terms with Ω = Ωn, which ensures that the filter preserves the corner frequencies of Ωd and Ωn after the transformation.

The design applies the IIR filter to the Vd and Vq voltage signals. These signals are the final outputs of the linear control system before the trigonometric transformation to stator-fixed voltages (V_α and V_β) and the space vector modulation (SVM) conversion to transistor gate signals.

The IIR filter is integrated into the FOC algorithm. Different DSP modes use different implementations of the FOC algorithm. Each implementation of the FOC algorithm includes a matching implementation of the IIR filter.

IIR Filter Tuning Parameters

Table 12: IIR Filter Tuning Parameters

Parameter	Description
Ω_d	Corner frequency (rad/s) above which the denominator response magnitude begins to decrease.

Parameter	Description
ζ_d	Damping factor that determines the sharpness of the corner at Ω_d; ζ_d < sqrt(0.5) results in a filter response peak (filter resonance) around Ω_d, whereas ζ_d > sqrt(0.5) gives a more gradual (more strongly damped) transition with no peak.
Ω_n	Corner frequency (rad/s) above which the numerator response magnitude begins to increase.
ζ_n	Damping factor that determines the sharpness of the corner at Ω_n; ζ_n < sqrt(0.5) results in a filter response trough around Ω_n, whereas ζ_n > sqrt(0.5) gives a more gradual transition with no trough.

Use these four parameters to create different filter response shapes. For vibration suppression, use a notch characteristic to decrease the system gain only around the frequency of interest. Set $\Omega_n = \Omega_d$ and $\zeta_n < \zeta_d$. The gain at $\Omega_n = \Omega_d$ is then equal to
ζ_n / ζ_d

IIR Filter Examples

Table 13: Exanple IIR Filters

Description	Ωd (Hz)	Ωn (Hz)	ζd (nondimen-sional)	ζn (nondimen-sional)
Sharp notch with 10x gain reduction at 2 kHz	2,000	2,000	0.3	0.03
Wide notch with 10x gain reduction at 1 kHz	1,000	1,000	100	10

Figure 13: Bode Plot—Sharp Notch with 10x Gain Reduction at 2 kHz

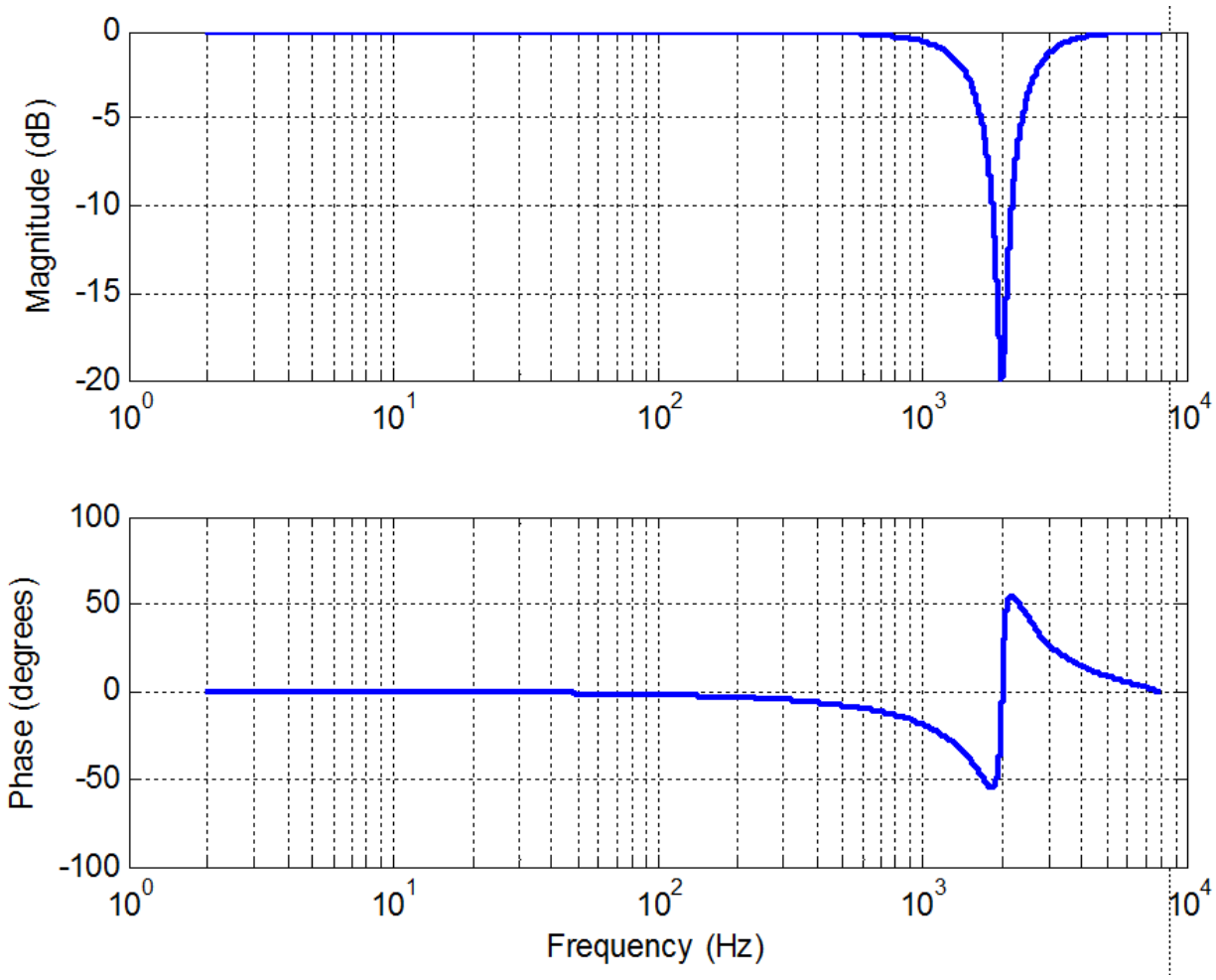
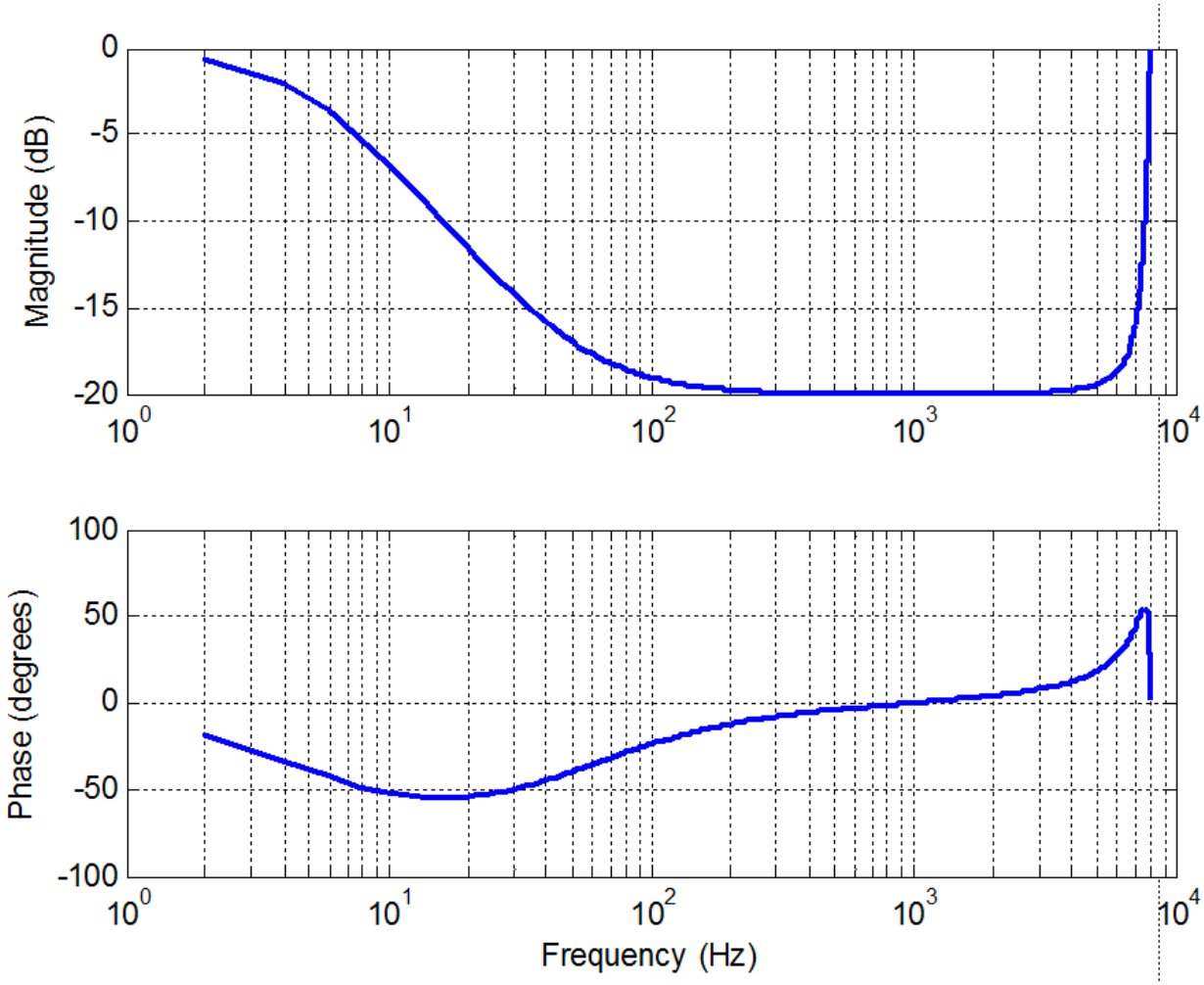


Figure 14: Bode Plot, Wide Notch with 10x Gain Reduction at 1 kHz



Vibration Detection

Raw FFT output is a complex-valued array of 4,096 points. The DMA peripheral in the SoC transfers this data from the FPGA part of the SoC to processor RAM. Software converts the real and imaginary parts to FFT magnitude (squared) and then applies a basic peak detection algorithm. A peak is the largest FFT magnitude value between two specified frequency bounds. Peak detection allows you to identify and read peak values from the FFT magnitude graphs in System Console.

Nios II Subsystem for MAX 10 FPGA Development Kits

The Drive-On-Chip reference design Nios II subsystem comprises the following Qsys components for a fully functional processor system with debugging capabilities:

- Nios II fast processor
- Floating-point hardware custom instructions 2 (optional)
- Tightly coupled instruction and data memory
- JTAG master
- Performance counters

- Clocking and bridge
- SDRAM controller
- JTAG UART
- System console debugging RAM
- Debugging dump memory

The ISR uses the tightly coupled memory blocks for code and data to ensure fast predictable execution time for the motor control algorithm.

The Nios II subsystem uses the JTAG master and debug memories to allow real-time interactions between System Console and the processor. The reference design uses the System Console debugging RAM to send commands and receive status information. The debugging dump memory stores trace data that you can display as time graphs in System Console.

Related Information

Qsys Interconnect and System Design Components
For more information about these components

Motor Control Peripheral Components

The motor control peripheral components comprise:

- 6-channel pulse width modulator (PWM)
- ADC interface
- BiSS encoder interface
- EnDat encoder interface
- Drive system monitor

The reference design puts the motor control peripheral components in a separate Qsys subsystem (`DOC_AXIS_Periphs.qsys`), which allows you to disable, enable, or add, extra axes.

This section describes the motor control peripheral components and describes specific connectivity issues when instancing the motor control suite peripherals in Qsys.

Related Information

Motor Control IP Suite Components for Drive-on-Chip Reference Designs.
For more information about the drive system monitor

6-Channel PWM

The 6-channel PWM provides an 8-kHz switching period, and a start pulse every 16 kHz shortly before the reversal point of the PWM counter. This pulse goes to the ADC to start capturing data and when the ADC finishes it sends an interrupt to the processor. The interrupt output from the first axis and drive0 (`doc_adc_irq`) is connected to the processor. You may leave the interrupt outputs from the other axes unconnected.

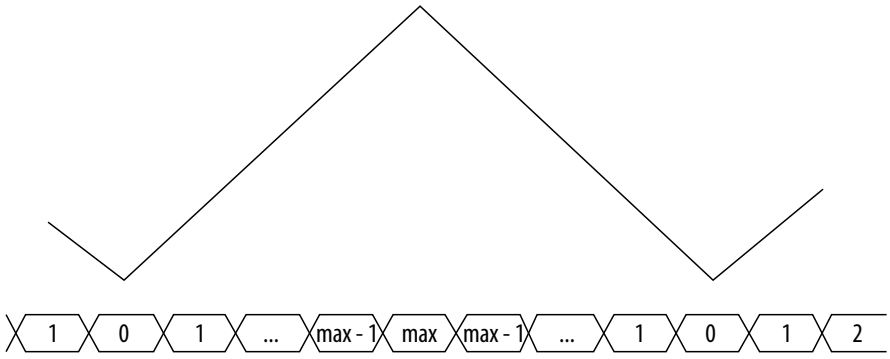
The PWM provides synchronization between multiple PWM instances. To implement this feature the reference design programs the PWMs identically: in Qsys one instance is the master PWM and connects the `sync_out` port to the other PWM `sync_in` port. For additional instances, continue chaining the `sync_out` port from the last instance to the next `sync_in` port. For example:

`sync_out` (PWM0) to `sync_in` (PWM1) and `sync_out`(PWM1) to `sync_in`(PWM2)

An internal counter defines the PWM period from 0 to PWM_{MAX} and then back to 0. The design configures it for an 8-kHz rate, and the internal clock rate of the PWM is 50 MHz. To achieve an 8-kHz period with an internal clock rate of 50 MHz, set the `carrier` register value to 3,125. Therefore:

`carrier` = (50 MHz / 8 kHz) / 2 = 3,125

Figure 15: PWM Counter Value



The `carrier_latch` output indicates to the position encoder to take a position reading. It is high for one clock cycle when the counter is at zero and at PWM_{MAX}.

The `start` output indicates to the phase current ADC interfaces to start sampling. It is offset from the `carrier_latch` position, so it occurs at a fixed position before it. The `pwm_trigger_up` sets the trigger value when the PWM counter is counting up; `pwm_trigger_down` sets the trigger value when counting down (**Table 14**).

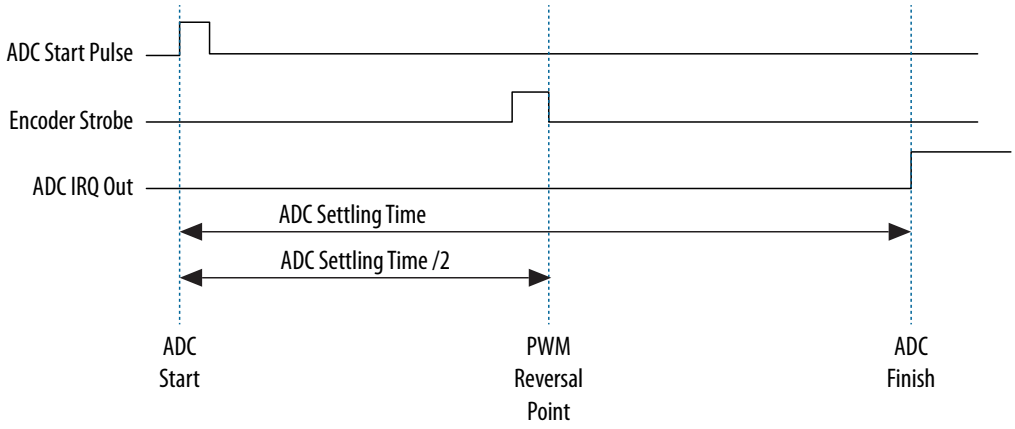
ADC Interface

The reference design uses a start pulse, which the PWM generates, on the ADC input to start the ADC interface. If you need to start multiple ADC interfaces from the same start pulse, use a conduit splitter component.

To offset the effect of current ripple, the reference design centers the ADC reading at the PWM reversal point. The reference design configures the start pulse to be at:

(settling time)/2

Figure 16: ADC Start Pulse and Settling Time



The reference design configures the phase current ADC interfaces to a decimation rate of M = 128, which requires a settling time of 19.2 s. To centre the ADC sampling, apply an offset to the PWM trigger of 9.6 s (480 clock periods @ 50 MHz).

Note: If you change to decimation rate M = 64, you must also adjust the PWM trigger parameters in the PWM to correctly centre the ADC sampling.

For an ADC clock rate of 20 MHz,

Table 14: ADC Interface Offsets for M= 64 and 128

Decimation Rate	PWM Trigger Down	PWM Trigger Up	Offset (s)
64	240	PWM_{MAX} –240	4.8
128	480	PWM_{MAX} – 480	9.6

An additional ADC measures the combined current flowing from the DC link for power consumption measurements.

Related Information

Motor Control IP Suite Components for Drive-on-Chip Reference Designs.
For more information on settling times and decimation rates

EnDat Encoder Interface

The reference design uses an evaluation version of the EnDat IP core version 2.2 from Mazet to read from the EnDat absolute position encoder attached to the motor. The reference design configures the EnDat IP core to respond to the trigger output that the PWM generates and reads a new position value.

The EnDat IP core requires a strobe to capture a position reading at a time synchronized with the ADC interface. The reference design generates the EnDat strobe at the exact reversal point of the PWM without offset.

Note: The reference design connects the strobe signal between the EnDat and PWM in the top level Verilog HDL design, not in the Qsys system.

Related Information

Mazet

For more information about and to purchase a license for the EnDat IP core

BiSS Encoder Interface

The reference design uses an evaluation of the BiSS Master IP core version 119 from iC-Haus to read from the BiSS absolute position encoder attached to the motor. The reference design configures the BiSS IP core to respond to the trigger output that the PWM generates and reads a new position value. The BiSS IP core can communicate with any device complying with the BiSS standard. However, Altera configures the reference design to work with the Hengstler AD36 series of BiSS B encoders.

The **components\biss_OCP** directory includes the datasheet for the BiSS Master IP core.

The BiSS IP core requires a strobe to capture a position reading at a time synchronized with the ADC interface. The reference design generates the BiSS strobe at the exact reversal point of the PWM without offset.

Note: The reference design connects the strobe signal between the BiSS and PWM in the top level Verilog HDL design, not in the Qsys system.

Related Information

iC-Haus BiSS interface

For more information about and to license the BiSS Master IP core

Motor Control Software

The motor control software is in C, runs on an ARM Cortex -A9 in the HPS or Nios II processor, and is in two sections. The main program covers the initialization, status reporting, and communication functions. The ISR covers the real-time aspects of running the motor control FOC algorithm.

Doxygen generated HTML help files are in the `software\CVSX_DS5\DOC_CVSX\source\doxygen` or `software\source\doxygen` directory. Open the `index.html` file in a browser to view the help files.

Figure 17: Main Program

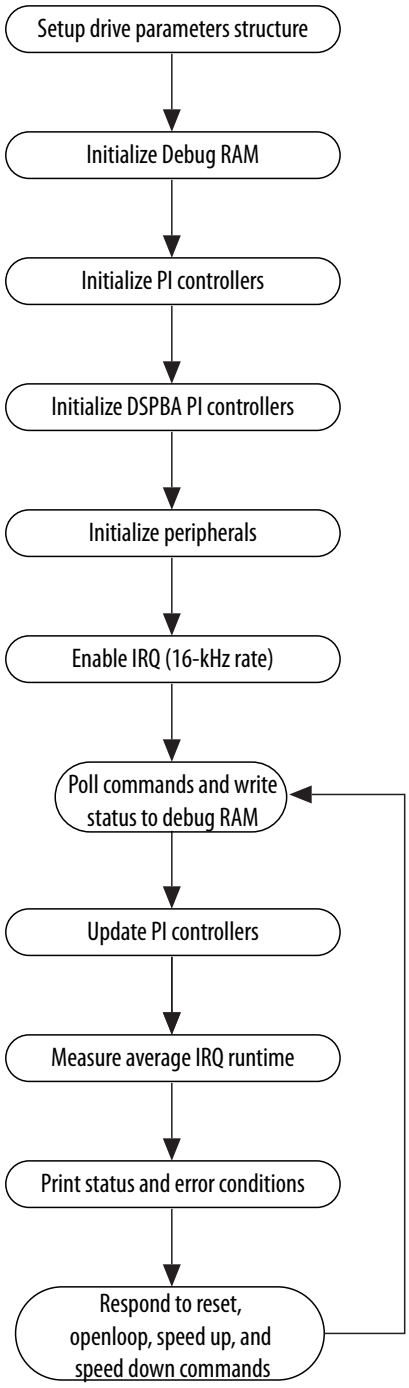


Figure 18: IRQ Routine

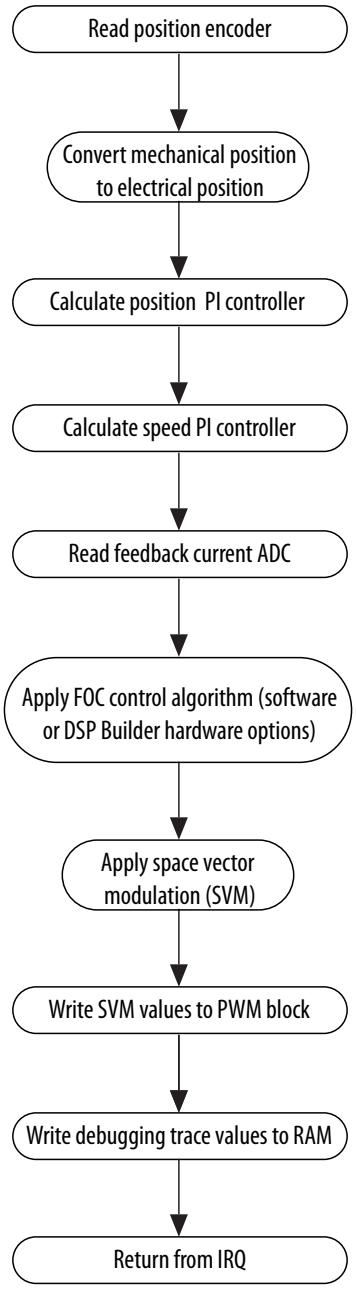
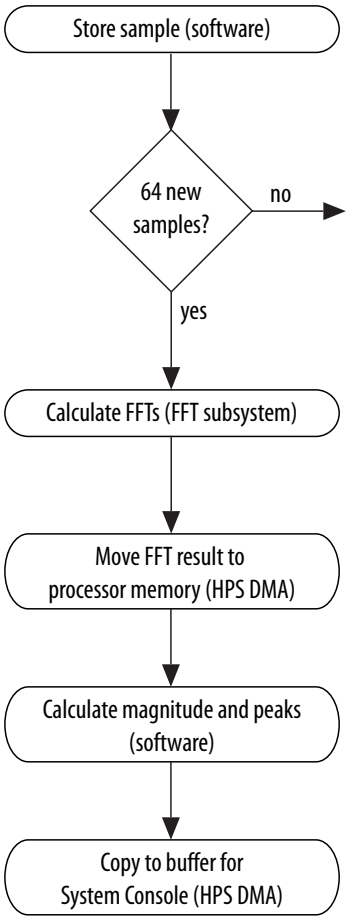


Figure 19: FFT Processing Flow (SoC only)



Nios II Motor Control Software Project

The Nios II motor control software project is a Nios II HAL-based project for the Nios II Software Build Tools for Eclipse. It has two components: the application and the BSP.

Setting Up the Nios II BSP

The Nios II board support package (BSP) setup requires two tightly-coupled memory blocks: **TCM_Data_Memory** and **TCM_Instruction_Memory**.

If you make any changes to the default BSP:

- Map **.exceptions** to **TCM_Instruction_Memory** (configured in Nios II parameters in System Console)
- Map **exceptions_stack_memory_region_name** to **TCM_Data_Memory** - size = 1,024
- Map **interrupt_stack_memory_region_name** to **TCM_Data_Memory** - size = 1,024
- Add section **.fast_math** and map to **TCM_Instruction_Memory**
- Add section **.table_data** and map to **TCM_Data_Memory**
- Add section **.drive_data** and map to **TCM_Data_Memory**

SoC HPS Motor Control Software Project

The SoC HPS motor control software project is a executable makefile project for the Altera SoC Embedded Design Suite. The single project contains both the application source and the µC/OS-II source. You can load this standalone application from DS-5 or using U-boot from an SD card.

Note: You may use the µC/OS-II source only for this reference design. If you wish to use the design that includes the C/OS-II in your production design, you must license µC/OS-II from Altera's partner Micrium who develop and support the BSP.

Altera provides DS-5 debug script that first loads and runs the preloader, then loads and runs the application to a breakpoint at `main()`. Execution can continue from this point or you can set further breakpoints or configure memory watch windows.

Building the HPS Preloader

Altera provides prebuilt preloader image in the **software\spl_bsp\uboot-socfpga\spl** directory. Only build the preloader if you change the HPS hardware settings.

1. Delete the existing **\software\spl-bsp** directory to remove the old preloader.
2. Run the bsp-editor in the **\software** directory to ensure that the preloader generate in the correct directory, so the software Makefile can find it when building the HPS software. The watchdog timer must be disabled in the bsp-editor (turn off **WATCHDOG_ENABLE**).

Related Information

Preloader User Guide in the SoC EDS User Guide

SoC HPS Execute-in-Place (XiP) Variant

The **DOC_FE2H_CVSX_XiP** variant is an example of a minimal motor control project for Cyclone V SoCs requiring no external memory. Execute-in-Place (XiP) refers to executing the application directly from flash memory (quad serial peripheral interface (QSPI) in this case). If the code footprint is small enough to fit entirely in the L2 cache, performance is similar to that achieved with external DDR memory and L2 cache. The variant uses on-chip RAM (OCRAM) for data storage. You may implement additional memory in the FPGA but do not use it for frequently accessed data. Accesses to this memory incur the latency through the HPS2FPGA bridges. The XiP variant uses FPGA memory for debug data to be shared with the System Console debug GUI.

Building the Preloader for XiP

Altera provides prebuilt preloader image in the **software\spl_bsp\uboot-socfpga\spl** directory. Only rebuild the preloader if you change the XiP hardware project.

1. Configure the preloader for XiP in the bsp-editor:

a. Turn off **BOOT_FROM_SDMMC**
b. Turn on **BOOT_FROM_QSPI**
c. Turn on **SKIP_SDRAM**
d. Turn off **WATCHDOG_ENABLE** (unless the application is modified to pet the watchdog)
e. Turn off **SDRAM_SCRUBBING**
f. Turn off **SDRAM_SCRUB_REMAIN_REGION**
g. Turn off **HARDWARE_DIAGNOSTIC**

2. Generate the preloader.
3. Compile the preloader (make all).
4. In **include/configs/socfpga_common.h**, search for the following macros and `#define` or `#undef`:

   ```
   #define CONFIG_SPL_SPI_XIP

   #define CONFIG_SPL_SPI_XIP_ADDR 0xFFA40040

   #undef CONFIG_USE_IRQ
   ```

5. Compile the preloader again (make).

Related Information

Preloader User Guide in the SoC EDS User Guide

Building the Application for XiP

The XiP application is an executable makefile project that you can build in DS-5. The steps required to import the project into DS-5 and build the application are similar to those for the DOC_CVSX variant.

Programming the QSPI FLASH With the XiP Preloader and Application

You must set the BOOTSEL jumpers on the Cyclone V SoC development board to boot from QSPI as `BOOTSEL[2:0]` left, left, left..

1. Turn off then turn on the power to the board.
2. Open the **quartus_hps** utility in a SoC EDS Command Shell.
3. Program the preloader:

   ```
   quartus_hps -c 1 -o PV --boot=0 preloader-mkpimage.bin
   ```

4. Program the application:

   ```
   quartus_hps -c 1 -o PV -a 0x40000 --boot=0 doc_xip-mkimage.bin
   ```

   If the command fails the first time because the HPS boots from the existing QSPI contents, try again.

XiP Application Debugging Script

Altera provide a debug script that allows you to debug the XiP application while running from QSPI flash. The script runs the preloader and then the application to a breakpoint at `main()`. Execution can continue from this point or you can set further breakpoints or configure memory watch windows.If the you boot the design from QSPI, `BOOTSEL[2:0]` is set to left, left, left. If the you run the design from DS5, `BOOTSEL[2:0]` is set to left, right, right.

Document Revision History

Table 15: Document Revision History

Date	Version	Changes
2016.12.15	3.1	Corrected iC Haus BISS interface link
June 2015	3.0	• Added support for MAX 10 FPGA devices. • Updated for Altera Complete Design Suite v15.0 • Removed support for Cyclone IV and Cyclone V (non SoC) devices. • Removed support for Terasic DE2115 development board • Updated DSP Builder model.
June 2014	2.1	• Improved *Licence Setup* instructions. • Improved *Programming the Nios II Device* instructions.
February 2014	2.0	• Added support for vibration suppression • Updated for the Quartus II software v13.1
May 2013	1.3	• Added Cyclone V SoC Development Kit • Updated for the Quartus II software v13.0
February 2013	1.2	• Added Cyclone V E FPGA Development Kit • Updated for the Quartus II software v12.1 SP1 • Added BiSS encoder
November 2012	1.1	Added FalconEye.
August 2012	1.0	Initial release.