AN 773: Drive-On-Chip Design Example for Intel® MAX® 10 Devices

Subscribe

Send Feedback

AN-773 | 2021.04.22

Latest document on the web: PDF | HTML

Contents

1. About the Drive-On-Chip Design Example for Intel® MAX® 10 Devices

The design demonstrates synchronous control of up to two three-phase permanent magnet synchronous motors (PMSMs) or brushless DC (BLDC) motors. Also the design controls a bidirectional DC-DC converter from the same FPGA. You can adapt the design to other motor types. The development kit can take power from a standard power supply or from a rechargeable battery pack, which shows the bidirectional power flow and battery state-of-charge estimation features.

Figure 1. Intel® Tandem Motion-Power 48 V Board with Intel® MAX® 10 FPGA Development Kit

Supported FPGA Development Kits

The design supports the Rev C (or later) Intel® MAX® 10 10M50 FPGA Development Kit. You can modify the design to run it on other Intel FPGA development kits that have an HSMC connector.

Supported Motor Control Boards

Table 1. Supported Motor Control Boards

Board	Vendor	Website	Power Stage	Sample Rate (kHz max)	Supported Feedback
Tandem Motion-Power 48 V Board	Terasic	www.terasic.com	MOSFET	125	Quadrature encoder, resolver, sensorless, trapezoidal

The design requires you to attach a power board to the FPGA development kit. The power board must, at a minimum, implement the motor drive electronics (e.g., IGBT or MOSFET switches), current and voltage feedback signal conditioning and DC link power bus to provide power to the motor via the inverter. The design requires position feedback for some control algorithms.

AC and Servo Drive Systems

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:

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

Intel MAX 10 Devices and DSP Builder for Intel FPGAs

Intel MAX 10 devices offer high-performance fixed- and floating-point DSP functionality, and Nios II soft processors. Intel MAX 10 FPGA devices offer a scalable and flexible platform for integration of single- and multiaxis drives on a single FPGA. The design comprises IP, software libraries, and a hardware platform. The design demonstrates DSP Builder for Intel FPGAs and Qsys for creating the the Avalon® Memory-Mapped interface between IP and the processor. The design includes all software and IP components. You can extend and customize the design to meet your own application needs. The design supports partitioning of algorithms between software running on an integrated processor and IP performing portions of the motor control algorithm in the FPGA, to accelerate performance. For example, depending on the performance requirements of your system or the number of axes you need to support, you may implement the field-oriented control (FOC) loop in hardware designed using DSP Builder for Intel FPGAs, or in software on the Nios II processor. The design allows you to connect to the motor and power stages through on chip or off-chip ADCs, feedback encoder devices and transistor gate drive circuitry. You can connect to higher-level automation controllers by adding off-the-shelf IP, for example for industrial Ethernet or CAN.

DSP Builder for Intel FPGAs provides a MATLAB Simulink* based work flow that allows you to create hardware-optimized fixed-latency implementations of algorithms without requiring HDL or hardware programming skills. The design provides fixed- and floating-point examples of the FOC algorithm. You can use the DSP Builder folding feature to reduce the resource usage of the logic compared to a direct parallel implementation.

Related Information

- Tandem Motion-Power 48 V Board Reference Manual
- Intel MAX 10 FPGA Development Kit
- Battery Management System Reference Design
  The Battery Management System (BMS) Reference Design demonstrates battery state of charge (SOC) estimation in an FPGA-based real-time control platform that you can extend to include other BMS functionality such as battery state-of-health monitoring and charge equalization (cell balancing).

2. Features of the Drive-on-Chip Design Example for Intel MAX 10 Devices

- Multiple FOC loop implementations:
  - Fixed- and floating-point implementation with Nios II processors targeting Intel MAX 10 FPGA devices
  - Fixed- and floating-point accelerator implementations designed using Simulink model-based design flow with DSP Builder for Intel FPGAs
  - Selectable 16 kHz or 32 kHz control loop update
- Integration in a single Intel MAX 10 FPGA of single and multiaxis motor control IP including:
  - High performance PWM IP at 300 MHz for two-level IGBT or MOSFET power stages
  - Sigma delta ADC interfaces for motor current feedback and DC link voltage measurement
  - Direct connection to MAX 10 integrated ADC
  - Multiple position feedback interfaces (default quadrature encoder)
- Bidirectional DC-DC converter for Tandem Motion-Power 48 V Board
  - 9 to 16 V input
  - 12 to 48 V output
  - System Console toolkit GUI for motor feedback information and control of motors
- Optional support for rechargeable battery power and BMS development with state-of-charge (SOC) estimation using an adaptive Dual Extended Kalman Filter (DEKF) algorithm

3. Getting Started with the Drive-On-Chip Design Example for Intel MAX 10 Devices

3.1. Software Requirements for the Drive-On-Chip Design Example for Intel MAX 10 Devices

- The Intel FPGA Complete Design Suite version 17.0.2, which includes:
- — Intel Quartus® Prime Standard Edition v17.0.2
  - — DSP Builder for Intel FPGAs v17.0.2
  - — Intel FPGA Nios® II Embedded design Suite (EDS) v17.0.2 (installed with Intel Quartus Prime)

3.2. Hardware Requirements for the Drive-On-Chip Design Example for Intel MAX 10 Devices

- Tandem Motion-Power 48 V Board
- Optionally, to estimate the SOC of the battery pack:
  - — Four-cell lithium polymer battery (for example Turnigy Accucell T100)
  - — Lithium polymer battery balancer/charger (for example Turnigy 2200mAh 4S 30C)
  - — Charging cable converter
  - — Discharging cable converter
  - — Custom lead to connect HXT 4 mm connector from battery to 6-pin connector on power board.

3.2.1. Preparing the Rechargeable Battery

You must charge the rechargeable battery to the level set by the specified charger before using it with the the Drive-on-Chip Design Example.

Intel tuned the state-of-charge estimator for the Turnigy Accucell T100 battery based on experimental results at room temperature.The state-of-charge estimator does not give accurate results with other battery types. Natural battery variability, temperature or changes in the specification of the cells used by the manufacturer may also affect accuracy. You must use the battery only within its recommended operating range. Intel recommend you keep the state-of-charge above 10%.

1. Make a converter using a HXT 4mm connector and the XT60 charger connector.

Figure 2. XT60 Connector

Figure 3. HXT 4 mm Connector

2. Connect the battery to the charger using both the charging connector (the red banana connector), and the monitor connector (white 5-pin connector) to the charger.

Figure 4. Connecting Battery Charger

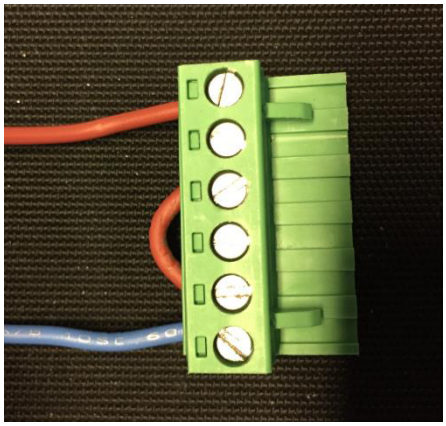
3. Make a battery power connector with a 6-pin connector

Table 2. Battery Power Connector (J1) Pin Assignments

To enable regeneration, link pins 3 and 5 of the battery power connector

Pin	Function
1	9 – 16 V
2	9 – 16 V
3	REGEN_EN
4	VDD_IO
5	0V
6	0V

Figure 5. 6-pin Battery Power Connector



3.3. Downloading and Installing the Design

The Drive-On-Chip Design Example for Intel MAX 10 Devices includes a precompiled `.sof` in the `master_image` directory.

1. Download the relevant design `.par` file for your development kit and power board from the Intel FPGA Design Store.
2. Install the relevant design `.par` file for your development kit and power board.

Archive file	Development Kit	Power Board
`DOC_TANDEM_MAX10.par`	MAX 10 10M50	Tandem Motion-Power

3. In the Quartus Prime 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 design and browse to the destination directory where you want to install it.
11. Click **OK** on the design template installation message.
12. Select **Drive on Chip Design Example**.
13. Click **Next**.
14. Click **Finish**.
    The Intel Quartus Prime software expands the archive and sets up the project, which may take some time.
15. Click **Tools ➤ Compile Design ➤ .**

Related Information

Drive-on-Chip motor control and autonomous DC-DC control (Tandem) at the Intel FPGA Design Store

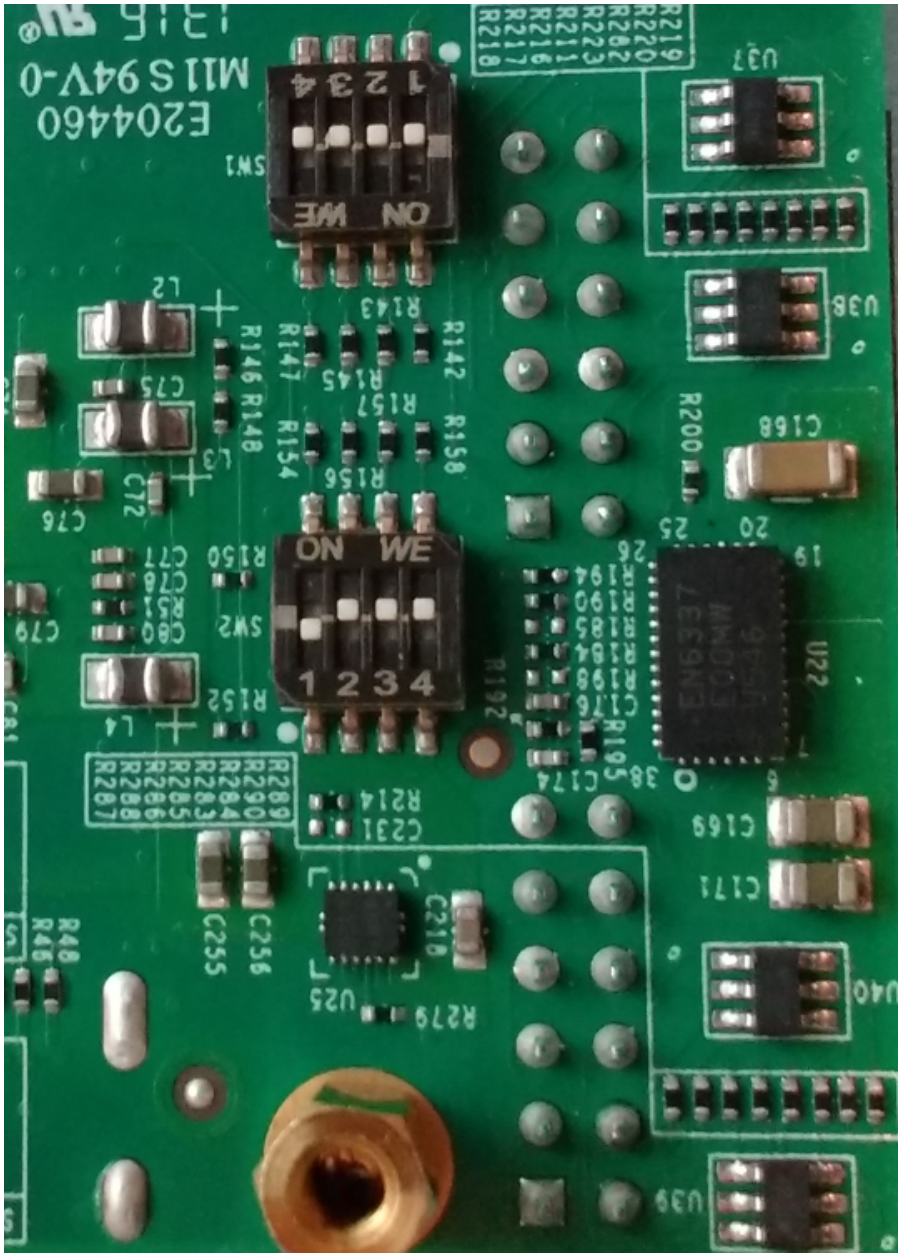
3.4. Setting Up the Motor Control Board with your Development Board for the Drive-On-Chip Design Example for Intel MAX 10 Devices

To prevent damage to the motor control board, ensure development board and power board are turned off and do not apply power until you have made all connections.

1. Ensure DIP SW2 is set to OFF-ON-ON-ON.

Figure 6. DIP SW2 Setting

DIP SW2 is on the lower side of the board.

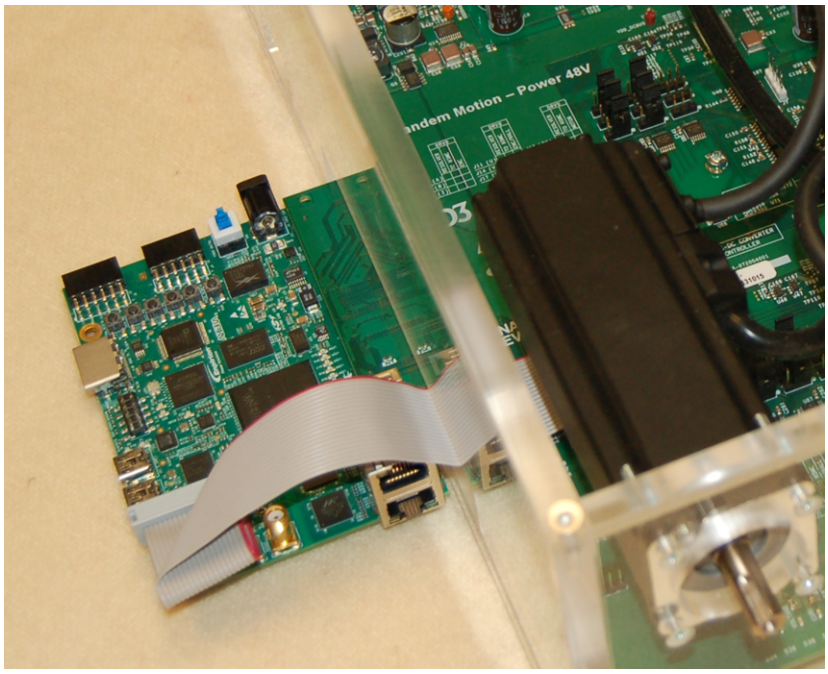


2. Connect the power board to the development board using the HSMC connector.

   For the optional estimation of SOC, connect the battery pack to connector J1 on the Tandem Motion-Power 48 V Board.

3. Connect the development board ADC connector to J20 on the power board.

Figure 7. Correct Cable Direction from Development Board to Tandem Motion-Power 48 V Board



4. Connect a USB cable from the USB connector J12 on the development board to your computer.
5. Apply power to the development board.

Related Information

- Applying Power to the Power Board on page 15
- MAX 10 FPGA Development Kit User Guide
- Tandem Motion-Power 48 V Board Reference Manual

3.5. Importing the Drive-On-Chip Design Example Software Project

Download and install the design

1. Start Nios II EDS. In the Intel Quartus Prime software click **Tools** > **Nios II Software Build Tools for Eclipse**.
2. Browse to the `\software` folder in the design project 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\` and click **OK**.

e. Click **Finish**.

5. Generate the BSP project: right-click <variant>_bsp project in the **Project Explorer** tab, point to Nios II, and click **Generate BSP**.
6. Build the application project: right-click <variant> project in the **Project Explorer** tab and click **Build Project**.

   On Windows, building the project for the first time might take up to one hour to build the newlib C libraries with support for the Nios II floating point custom instructions.

Related Information

Downloading and Installing the Design on page 10

3.6. Configuring the FPGA Hardware for the Drive-On-Chip Design Example for Intel MAX 10 Devices

Set up the motor control board with your development board.

Note: Always remove power from the motor control power board, before reprogramming the FPGA, or removing power from the development boards.

1. In the Quartus Prime software, click **Tools** > **Programmer**.
2. In the Programmer pane, select **USB-Blaster II** under **Hardware Setup** and **JTAG** under **Mode**.
3. Click **Auto Detect** to detect devices.
4. Select the **10M50DA** device.
5. Double-click on the **File** field for the **10M50** device from the pop-up list.
6. Select the `output_files/<project name>.sof` file from the `master_image` directory and click **Open**
7. Turn on **Program/Configure**.
8. Click **Start**.

Related Information

- Downloading and Installing the Design on page 10
- Setting Up the Motor Control Board with your Development Board for the Drive-On-Chip Design Example for Intel MAX 10 Devices on page 11

3.7. Programming the Nios II Software to the Device for the Drive-On-Chip Design Example for Intel MAX 10 Devices

Configure the FPGA with the design example hardware

1. In the Nios II EDS Project explorer, click the ***<project variant>*** to highlight the project.
2. 1. 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 ***<project variant>*** project in the **Project** name drop-down.

d. On the **Target Connection** tab, click **Refresh** Connections. The software finds the Intel FPGA Download Cable.

e. Click **Apply** to save changes, optionally specifying a name for the new configuration.

f. Click **Run** to start the software.

3. Check that the Nios II console shows the correct FPGA and power board combination. For example for the Tandem Motion-Power 48 V Board project variant:

```
[DECODE SYSID] Decoding hardware platform from QSYS SYSID data : 0x003043FE
[DECODE SYSID] Design Version : 3.0
[DECODE SYSID] FPGA Board : MAX 10M50 Dev Kit
[DECODE SYSID] Power Board : Intel Tandem Motion Power
```

4. Check the five LEDs (LED0 to LED4) on the Intel MAX 10 development board illumiunate. These five status LEDs indicate errors in the system. Any LED not illuminating indicates a fault and you should switch off the power board immediately.

LED	Description
LED0	Indicates either overcurrent or overvoltage issue on the first motor.
LED1	Indicates either overcurrent or overvoltage issue on the second motor.
LED2	Indicates undervoltage, overvotlage, or overcurrent on DC input voltage.
LED3	Indicates a fault on the DC-DC converter.
LED4	Indicates start of ISR.

Related Information

- Downloading and Installing the Design on page 10
- Configuring the FPGA Hardware for the Drive-On-Chip Design Example for Intel MAX 10 Devices on page 14
- Configuring the FPGA with the Drive-On-Chip Design Hardware

3.8. Applying Power to the Power Board

Note: Always stop the motors, then remove power from the motor control power board, before reprogramming the FPGA, or removing power from the development boards.

Caution: Never connect the battery and the power supply simultaneously.

1. Apply power to the power board when the design running on Nios II ptrocessor shows the message `DC Input Voltage error check power connections`. After a few seconds both motors begin turning. The Nios II console shows further diagnostic messages as the control loop starts.

Related Information

Preparing the Rechargeable Battery on page 8

3.9. Debugging and Monitoring the Drive-On-Chip Design Example for Intel MAX 10 Devices with System Console

1. In the Intel Quartus Prime software, click **Tools** > **System Debugging Tools** > **System Console**.
2. In Tcl console type `toolkit_register toolkits/doc_toolkit/DOC.toolkit` and press enter.
3. In the **Drive On A Chip Debug GUI** area, click **Launch**.
4. Check that the console display shows the correct FPGA and power board combination. For example for the Tandem Motion-Power 48 V Board project variant look for the following lines:

```
Version = 3.0 Device Family = 3 Powerboard Id = 4 Design Id = 254
FPGA Board : MAX10 10M50 Dev Kit
Power Board : Intel Tandem 48V
Version : 3.0
```

   You can right-click on the **Drive On A Chip Debug GUI** tab and select **Detach to display the GUI in its own window**. Close the window to reattach it to the System Console window.

A number of tabs are populated in the Drive-On-A-Chip Debug GUI, depending on the project variant. The tabs are grouped into two panes. Use the upper pane, starting with the **Data Source** tab to configure the design. Use the lower pane, starting with the **General** tab to start demonstrations and monitor the state of the design.

3.10. System Console GUI Upper Pane for the Drive-On-Chip Design Example for Intel MAX 10 Devices

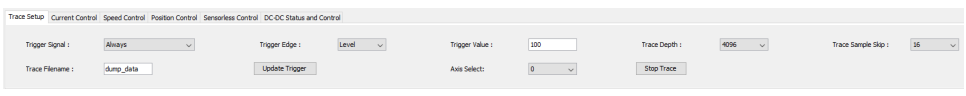
Trace Setup Tab

On the **Trace Setup** tab setup select:

- The waveform tracing by specifying a trigger
- Axis to trace
- Trace depth
- A filename to store the trace data.

Click **Update Trigger** after making any changes. Click **Start Trace** to start tracing. See the **Waveform** tab for trace display. When saving trace data to a file, be aware that the design overwrites the file with each trace; it does not append new traces to an existing file.

Figure 8. Trace Setup Tab



Current Control Tab

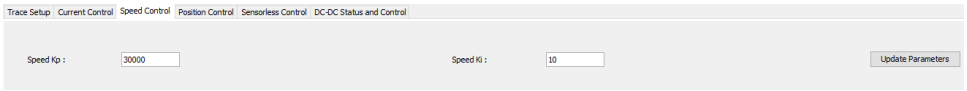
On the **Current Control** tab, enter the P (Kp) and I (Ki) coefficients for the current control loop. These quantities are preset to the correct values for the motor type configured in the application software. Click **Update Parameters** after making a change.

Figure 9. Current Control Tab

Speed Control Tab

On the **Speed Control** tab, enter the P (Speed Kp) and I (Speed Ki) coefficients for the speed control loop. These quantities are preset to the correct values for the motor type configured in the application software. Click **Update Parameters** after making a change.

Figure 10. Speed Control Tab



Position Control Tab

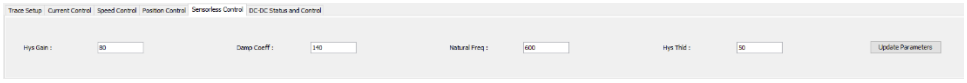
On the **Position Control** tab, enter the P (Position Kp) coefficient for the position control loop. This quantity is preset to the correct values for the motor type configured in the application software. Click **Update Parameters** after making a change.

Figure 11. Position Control Tab

Sensorless Control Tab

On the **Sensorless Control** tab, enter the Hys Gain (Hysteresis control gain), Damp Coeff (damping coefficient), natural freq (natural frequency) and Hys Thld (Hysteresis threshold) values for the sensorless control loop. These quantities are preset to the correct values for the motor type configured in the application software. Click **Update Parameters** after making a change. Modifying these four control parameters may cause the system to become unstable. For example, the motor speed may oscillate or accelerate to a much higher value than the command.

Figure 12. Sensorless Control Tab



DC-DC Status and Control Tab

The **DC-DC Status and Control** tab allows you to adjust the target voltage of the DC-DC converter on the Tandem Motion- Power 48 V Board. The converter runs with a DC Link Voltage of 48 V, if it is powered by a rechargeable battery and you turn on regeneration. If the motors suddenly stop, the energy regenerated from the motors' kinetic energy transfers to the battery. However, if a battery is not connected or a battery is connected but you turn off regeneration and the motors stop suddenly, the regenerated energy might cause the DC Link voltage to rise suddenly and damage components on the board. Therefore the default voltage of the DC Link is 32 V to leave some margin for regeneration.

Figure 13. DC-DC Status and Control Tab

Related Information

- Controlling the DC-DC Converter on page 21
- Tuning the PI Controller Gains on page 21
- Monitoring Performance on page 22
- Controlling the Speed and Position Demonstrations on page 22

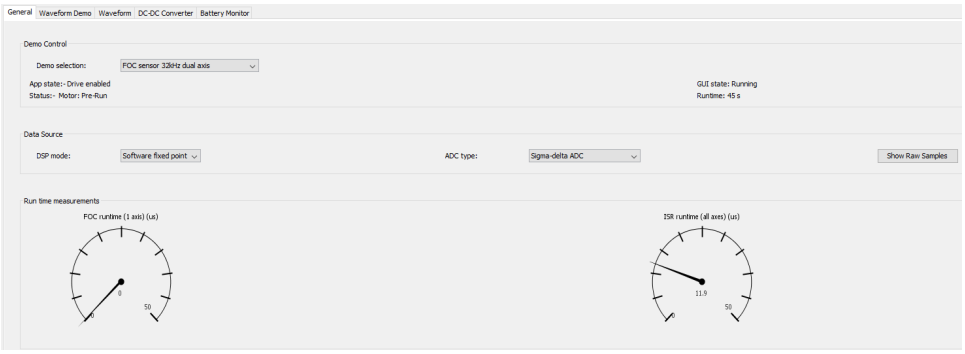
3.11. System Console GUI Lower Pane for the Drive-On-Chip Design Example for Intel MAX 10 Devices

General Tab

Under **Data Source**:

- In the **DSP mode** drop-down menu select **DSP calculation** mode to use (Software Fixed Point; DSP Builder for Intel FPGAs Fixed point; DSP Builder for Intel FPGAs Floating Point or Software Floating Point)
- Under the **ADC Type** drop-down menu, select the ADC to use for feedback samples (depending on the power board you use)
- Click **Show Raw Samples** to show raw or scaled samples.

Figure 14. General Tab



On the **Demo selection:** drop-down menu select the control algorithm, type of commutation, and update rate to be use in the demonstration. The available selections depend on which motor control hardware you use.

The **Status:** field reports the status of the demonstration. The **Runtime:** field updates from the application software.

The **Run time measurement** dials display the processing time of the FOC control loop and the overall Interrupt Service Routine (ISR) processing time, including handling debug trace data. in the currently selected DSP mode.

Waveform Demo Tab

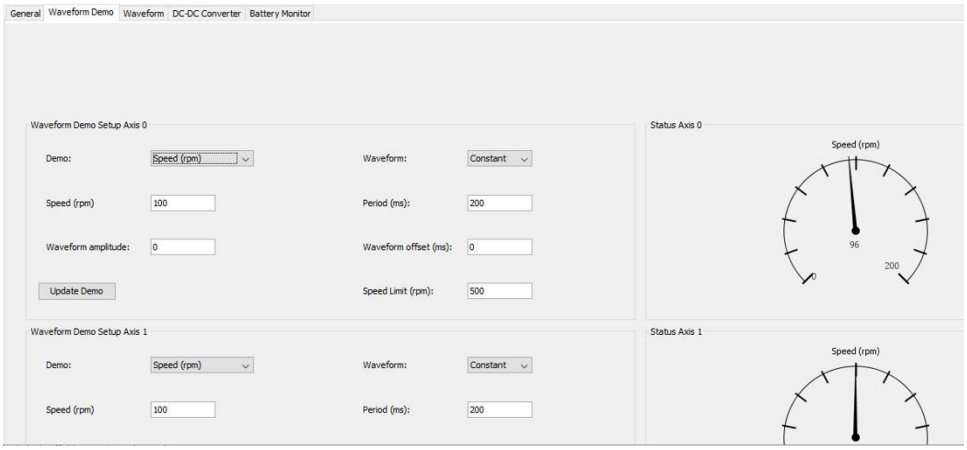
In the **Demo** drop-down menu select speed, position, or other demonstration.

In the **Waveform** drop down select the dynamic behavior of the speed or position demo (constant or varying with sine, square, triangle, sawtooth waveform).

Set the nominal speed or position, waveform period, amplitude and offset and click **Update Demo**.

Note: Large step changes in commands (e.g. using square or triangle wave speed demo) may result in unstable behavior, especially when using sensorless control.

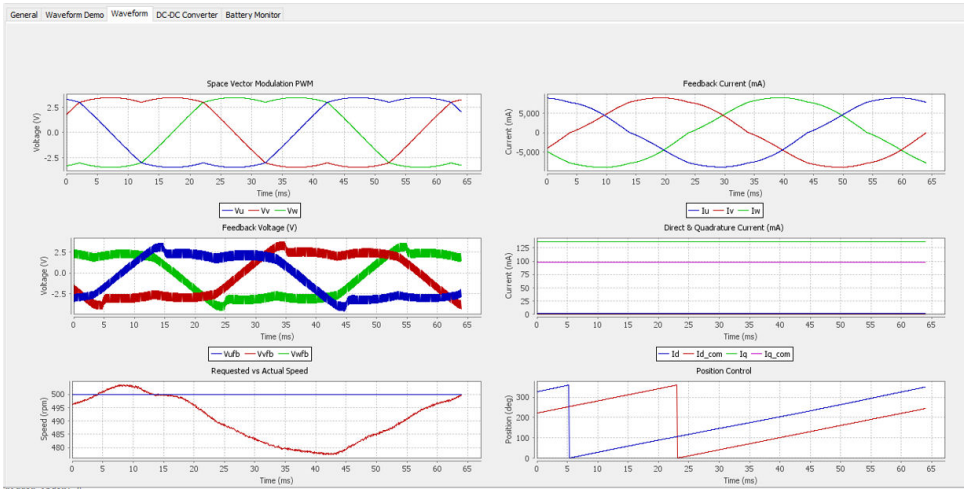
Figure 15. Waveform Demo Tab



Waveform Tab

The **Waveform** tab shows the motor control waveform captured as a result of the trigger settings in the **Trace Setup** tab..

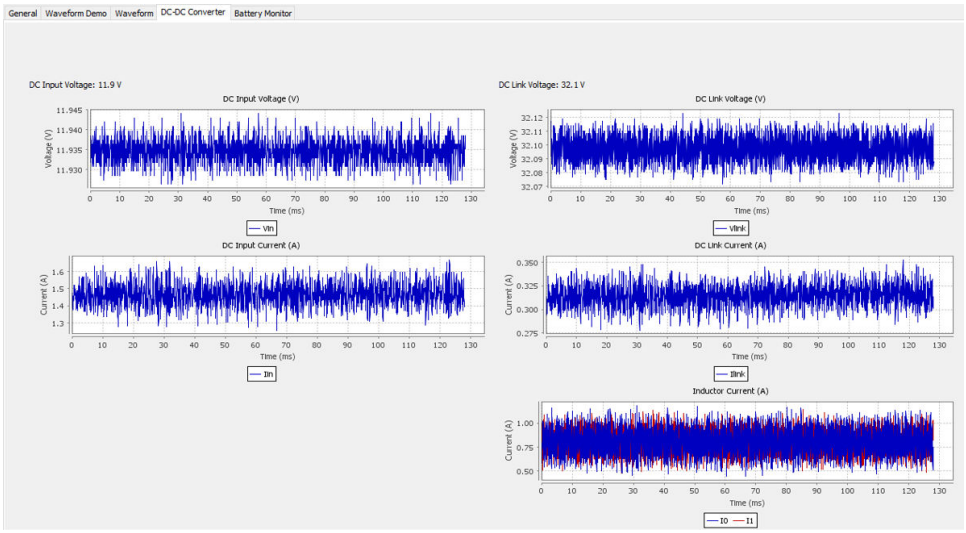
Figure 16. Waveform Tab



DC-DC Converter Tab

The **DC-DC Converter** tab shows the DC-DC converter waveforms captured as a result of the trigger settings in the **Trace Setup** tab.

Figure 17. DC-DC Converter Tab



Demonstration Selection

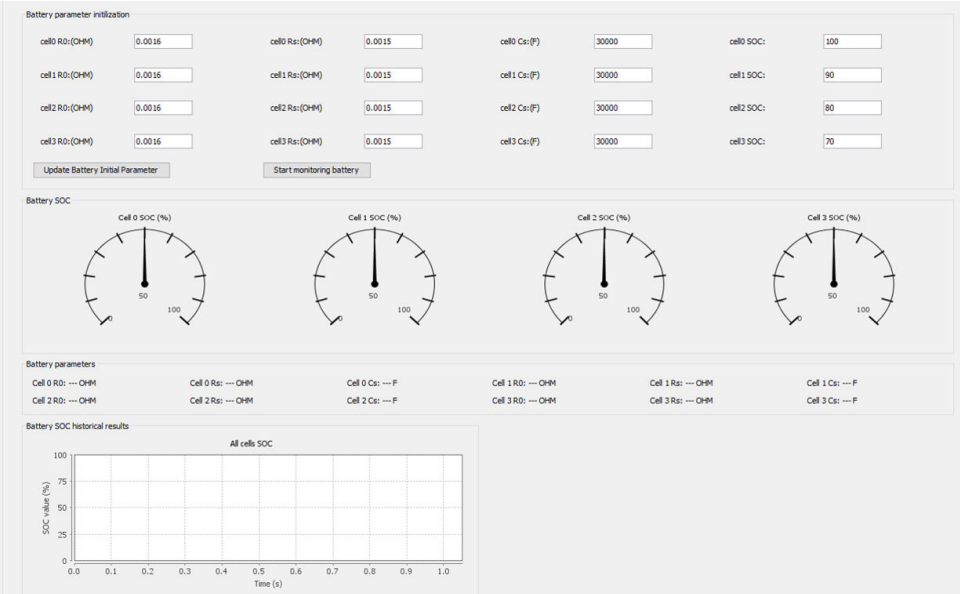
The **Demo selection:** drop-down on the **General** tab selects the demo to run:

- Reset
- Open loop sinusoidal 16 kHz Volts/Hz
- Open loop sinusoidal 32 kHz Volts/Hz
- FOC sensor 16 kHz single axis
- FOC sensor 32 kHz dual axis
- FOC sensorless 16 kHz dual axis
- FOC sensorless 32 kHz dual axis
- Trapeziodal Hall sensor 32 kHz dual axis

Battery Monitor

The **Battery Monitor** tab shows the battery initial parameters, battery monitor control, and status of battery, including SOC and parameter values.

Figure 18. Battery Monitor Tab



3.12. Controlling the DC-DC Converter

1. On the **DC-DC Status and control** tab enter the desired DC link voltage of the DC-DC converter.
2. Monitor the changes in the waveforms on the **DC-DC Converter** tab.

3.13. Tuning the PI Controller Gains

The Drive-On-Chip Design Example contains three PI control loops for current (inner most loop), speed and position. 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 **Waveform** tab.

1. On the **Current Control** tab, enter values for:
   - Kp (proportional gain).
   - Ki (integral gain).
2. Click **Update Parameters**.
3. On the **Speed Control** tab:
   - Enter values for Kp (proportional gain) and Ki (integral gain).
   - Click **Update Parameters**.
4. On the **Position Control** tab:
   - Enter a value for Position Kp.
   - Click **Update Parameters**.

3.14. Controlling the Speed and Position Demonstrations

The Drive-On-Chip Design Example speed and position demonstrations show constant or varying speed and position.

1. Selects the way the speed or position varies during the demonstraiton in the **Waveform** drop down.

   The speed or position varies according to the selected waveform.
2. Specify the **Speed (position)** to control the nominal speed or position for the respective demonstrations.

   If you select a non-constant waveform, the speed and position vary around this nominal value.
3. Specify the **Period (ms)** to control the period of the speed and position variation waveform.
4. Specify the **Waveform amplitude** to control the amplitude of the waveform. For example, a speed of 100 rpm with an amplitude of 50 rpm gives a speed varying between 50 and 150 rpm.
5. Specify the waveform offset (ms): to change the waveform phase (shift in time).
6. Specify the **Position Speed Limit (rpm)** to control the maximum speed in position demo mode.

   If you select **Speed** in **Demo** on the **Waveform Demo** tab, the design ignores this value.
7. Click **Update Demo** to apply changes to the design.

3.15. Monitoring Performance

The Drive-On-Chip Design Example offers many way to monitor the performance.

1. On the **Trace Setup** 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 a trigger type:

- **Level** (trigger signal must match this value)
- **Rising Edge** (trigger signal must transition from below to above this value)
- **Falling Edge** (trigger signal must transition from above to below this value)
- **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.
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. Select a lower number of samples to make 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** to stop the trace.

4. Rebuilding the Drive-On-Chip Design Example for Intel MAX 10 Devices

4.1. Changing the Intel MAX 10 ADC Thresholds or Conversion Sequence

Only change thethresholds or conversion sequence for the Drive-On-Chip Design Example for Intel MAX 10 devices by modifying hardware parameters.

The Intel MAX 10 ADC thresholds detect over or under voltage and current faults by comparing the sampled signals against preset limits. Errors cause the design to shut down the motor(s) and/or DC-DC converter and inform the software application of the error condition.

1. Open the design project in the Intel Quartus Prime software.
2. Click **Tools** > **Qsys** to open the Qsys editor.
3. Click **Close**.
4. Select the <*project variant*> _QSYS.qsys file and click **Open**.
5. Click **Close** if any warning dialog appears.
6. Double click on the **max10_adc component** in the **System Contents** tab.
7. In the **Channels** tab select the ADC and channel to edit the thresholds.
8. Enter the desired maximum and minimum thresholds. You must calculate the absolute voltage in the range 0..1.2 V from the scaling of feedback signals.
9. On the **Sequencer** tab set the desired **Conversion Sequence Length**.

   Intel recommends a **Conversion Sequence** length of 8 for the design.
10. In the **Sequencer** tab select the ADC and use the drop-down menus for each slot to set the desired conversion sequence.

    Intel recommends the sequence is each channel in numeric order CH 1...CH 8. You must ensure each channel is converted at least once in the sequence.

    Note: Failure to include all channels in the conversion sequence can cause damage to the Tandem Motion-Power 48 V Board. For example, by not allowing the application to detect overcurrent errors.
11. Close the **Parameters** tab.

Generate the system in Qsys.

4.2. Generating the Qsys System

After making any changes in the Qsys project for the Drive-On-Chip Design Example, generate the system.

1. In the Qsys software click **File** > **Save**.
2. Click **Generate HDL...**.
3. Click **Generate**.
4. Click **Close**.
5. If your changes result in new exported connections you can view the Qsys component template by clicking **Generate** > **Show Instantiation Template...**.

   Add new ports to the Qsys component instantiation in the top level RTL of the project `<project variant>.v`.
6. Close Qsys.

After making a change to the Qsys system you must:

- Regenerate the Nios II BSP and rebuild the software
- Compile the hardware

Related Information

- Compiling the Hardware in the Quartus Prime Software on page 25
- Generating and Building the Nios II BSP for the Drive-On-Chip Design Example on page 25

4.3. Compiling the Hardware in the Quartus Prime Software

1. In the Quartus Prime software select **Processing** > **Start Compilation**.

Related Information

Generating and Building the Nios II BSP for the Drive-On-Chip Design Example on page 25

4.4. Generating and Building the Nios II BSP for the Drive-On-Chip Design Example

1. Start Nios II EDS: in the Intel Quartus Prime software click **Tools** > **Nios II Software Build Tools for Eclipse**.
2. Browse to the **/software** workspace directory in the project folder.
3. Click **OK**.
4. Generate the BSP project: right-click ***<variant>*_bsp** project in the **Project Explorer** tab, point to Nios II, and click **Generate BSP**.

- Compile the software application.
- Optionally configure the software application.

Related Information

- Software Application Configuration Files on page 26
- Compiling the Software Application for the Drive-On-Chip Design Example on page 28

4.5. Software Application Configuration Files

You can modify the operation of the software application for the Drive-On-Chip Design Example by editing some C source code and header files.

Table 3. Software Application Configuration Files

File	Path	Function
demo_cfg.c	.	Declare motors[] Array
demo_cfg.h	.	Configuration macros and include file for demo_cfg.c
motor_types.c	Platform/motors	Declares motor types and encoders
motor_types.h	Platform/motors	Defines motor and encoder types and include file for motor_types.c

Table 4. Configuration Macros

This table lists the configuration macros that you can use to configure the design in demo_cfg.h.

Macro	Default State	Range	Function
FIRST_MULTI_AXIS	0	0 - 1	Index of first motor axis to be controlled.
LAST_MULTI_AXIS	1	0 - 1	Index of last motor axis to be controlled.
DEFAULT_ADC_TYPE	ADC_TYPE_SIGMA_DELTA	ADC_TYPE_SIGMA_DELTA	Use sigma delta ADC samples in control loop.
		ADC_TYPE_MAX10	Use MAX10 ADC samples in control loop.
SD_ADC_FILTER	ADC_D_10US	ADC_D_10US	Sinc3 filter delay 10us.
		ADC_D_20US	Sinc3 filter delay 20us.
DC_LINK_STARTUP_TARGET_VOLTS	32	12 - 48	Target voltage for DC-DC converter.
OPEN_LOOP_INIT	0	0	Start motors in closed loop mode.
		1	Start motors in open loop mode.
INTERACTIVE_START	0	0	Normal startup 1:
		1	User prompted via Nios II console at each stage of startup
ENCODER_SERVICE	Undefined	Undefined	Normal operation.
DBG_DEFAULT	DBG_INFO	DBG_NEVER	No console output.
		DBG_ALWAYS	Always output.
		DBG_FATAL	Debug level set to fatal errors .
		DBG_ERROR	Debug level set to non-fatal errors and above .
		DBG_WARN	Debug level set to warnings and above .

continued...

Macro	Default State	Range	Function
		DBG_INFO	Debug level set to information and above .
		DBG_PERF	Debug level set to performance data and above .
		DBG_DEBUG	Debug level set to debug messages and above .
		DBG_DEBUG_MORE	Debug level set to more debug messages and above .
		DBG_ALL	Debug level set to all messages.

4.5.1. Defining a New Motor or Encoder Type

1. To use a different motor type or position feedback encoder with the Drive-On-Chip Designs, declare a new motor type array of type `motor_t` in `motor_types.c`.

   The structure of `motor_t` is defined in `motor_types.h`. The array length must match the number of axes available (e.g. two for the Tandem Motion-Power 48 V Board).
2. Provide C source code for the three functions `encoder_init_fn`, `encoder_service_fn` and `encoder_read_position_fn` if none of the existing functions are suitable.
3. Use the functions provided with the design as templates to write your own functions.
4. Initially, use the gain constants from an existing motor type and then determine new values when you first run the motor by following a standard PI controller tuning process.

   Refer to the declaration of `tamagawa_resolver` software source file as an example.
5. Edit the declaration of the `motors[]` array in `demo_cfg.c` to use your motor.

   The default `motors[]` definition for the Tandem Motion-Power 48 V Board is two Tamagawa motors with resolvers:

   ```
   motor_t * motors[] = {&tamagawa_resolver[1], &tamagawa_resolver[1], NULL,
   NULL};
   ```

   The resolver interface on the Tandem Motion-Power 48 V board converts the resolver output into quadrature equivalent or Hall equivalent encoder signals. The design supports a maximum of two axes so the third and fourth elements of the `motors[]` array are set to NULL for clarity.

Related Information

Tandem Motion-Power 48 V Board Reference Manual

4.6. Compiling the Software Application for the Drive-On-Chip Design Example

1. Start Nios II EDS. In the the Quartus Prime software click **Tools** > **Nios II Software Build Tools for Eclipse**.
2. Build the application project: right-click <*variant*> project in the **Project Explorer** tab and click **Build Project**.

4.7. Programming the Design into Flash Memory

For the Drive-On-Chip Design Example for Intel MAX 10 devices, you can store the FPGA configuration file in the Intel MAX 10 on-chip flash memory; you can store the software executable in external QSPI flash memory.

1. Rebuild the design with the Nios II reset vector pointing to the QSPI memory

   The quartus.ini file with PGMIO_SWAP_HEX_BYTE_DATA=ON content is required in the project directory.
2. Compile the software and generate the software programmer object file.
   a. In the Nios II SBT, open the BSP editor.
   b. Unselect all advanced.hal.linker option.
   c. Modify the linker script to point the reset section to the qspi memory.
   d. Build the BSP project and the main project.
   e. Generate the `.hex` file by right-clicking **DOC_TANDEM_MAX10 ➤ Make Targets ➤ Build ➤ mem_init_generate**.
   f. In the Quartus Prime software click **File ➤ Convert Programming Files** and enter these settings:.
      - **Configuration device: CFI_512Mb**.
      - **Mode: 1-bit Passive Serial**.
   g. Change the file name to the desired path and name. For example `SW.pof`.
   h. In **Input files to convert**, remove **SOF Page_0**.
   i. Click **ADD HEX Data**,
   j. Choose the `generic_quad_spi_controller_0.hex` file generated previously in step 2e.

      This file is in the `mem_init` subdirectory of the software project.
   k. Select **Absolute Addressing** and click **OK**.
   l. Click **Generate** to create the `.pof` file.
3. Program the software into QSPI flash.
   a. Ensure DIP SW2 is set to OFF-ON-ON-ON.
   b. Download the parallel Flash Loader from rocket boards https://rocketboards.org/foswiki/pub/Documentation/AlteraMAX1010M50RevCDevelopmentKitLinuxSetup/max10_qpfl.sof.
   c. Program the parallel flash loader (`max10_qpfl.sof`) into the MAX 10 device to program the QSPI flash, using Quartus Programmer.
   d. Right click on the MAX 10 FPGA and select **Edit ➤ Change File**.

e. Choose the `max_qpfl.sof` file.
f. Turn on MAX 10 device under **Program/Configure**.
g. Click **Start** to start programming.
h. Click on **Auto Detect** after `max10_qpfl.sof` was successful.
   A new QSPI flash device is shown, attached to the MAX10.
i. Program the software image into QSPI flash.
j. Right click on the SQPI device and select **Edit ➤ Change File**
k. Choose the generated `.pof` file (`SW.pof`).
l. Check the `.hex` file under **Program/Configure**.
m. Click **Start** to start programming.

4. Program hardware `.sof` file into the MAX 10 FPGA.
   a. Right click on the MAX 10 FPGA and select **Edit ➤ Change File**.
   b. Choose the `.sof` file generated from Quartus Prime project compilation.
   c. Click **Start** to start programming.

Related Information

AN730: Nios II processor booting methods in MAX 10 devices

5. About the Scaling of Feedback Signals

Voltage, current, and position feedback signals from the hardware require scaling into the appropriate physical units in software before you can use the data in the control loop

The Drive-On-Chip Design Example for Intel MAX 10 devices requires scaling to convert the feedback samples from alternative ADCs (sigma-delta ADCs versus Intel MAX 10 ADCs) into the same units for use in the FOC algorithm. Also, the design requires scaling to convert current and voltage feedback values to the units expected by DC-DC module. The design treats some feedback as "dimensionless" data and scales it into a convenient range (e.g. signed 16-bit integer) for use in the control loop. The design presents data for diagnostic purposes in a GUI provided as a System Console Toolkit. The `.tcl` toolkit script `DOC_debug_gui.tcl`, which creates this GUI, performs further scaling into physical units for waveform displays.

5.1. Signal Sensing in Sigma-Delta and MAX 10 Integrated ADCs

The Drive-On-Chip Design Example for Intel MAX 10 devices configures the Intel MAX 10 ADCs as a dual ADC with sequencer and sample store using the internal 2.5 V reference. It uses 16 channels, channels 1 to 8 on each of the ADC submodules.

Each Intel MAX 10 ADC submodule converts the 8 input channels in sequence. The Intel MAX 10 ADC Qsys component configures the sequence. Intel chooses the order in which the design connects signals to the ADC inputs. Also, Intel chooses the sequence in the Qsys component to minimize the time difference between the most recent feedback current samples for motor control.

Sigma-delta modulators on the power board convert analog signals to a one-wire digital bitstream. The design demodulates or filters the bitstream in the FPGA. The FPGA uses two types of sigma-delta filter IP in the FPGA, ADC modules and DC link modules, each with different scaling and offset.

The design downloads and filters all sigma delta inputs in parallel so no skew exists between the samples that it feeds to the software application.

Each ADC type has a different input and output ranges with the corresponding 'C' data type..

Table 5. ADC Output Data

ADC Type	Input Range	Count Range	C Data type
Sigma-delta ADC	-320…+320mV	-32768…+32767	Signed 16-bit
Sigma-delta DC link	0…+320mV	0…+32767	Unsigned 16-bit
MAX 10	0…2.5V	0…4097	Unsigned 16-bit

The input current and DC bus current are only available via sigma-delta ADCs.

Position feedback samples are scaled to a 23 bit unsigned integer, for consistency across all encoder types supported by this and previous Drive-On-Chip reference designs.

Table 6. ADC Scaling

This table shows the ADC scaling for all signals, ADC type and board revision. The scaling depends on the way the power board processes the signals (e.g., value of current shunts, scaling, and offset in sense amplifiers).

Feedback Quantity	Sigma Delta Interface IP	Sigma Delta Scaling for Tandem Motion Power Board	Intel MAX 10 Scaling for Tandem Motion Power Board
Motor Phase Voltages	ADC interface	545 counts/V	67.7 counts/V
DC Bus Voltage	ADC interface	40 counts/V	67.7 counts/V
Input Voltage	DC Link	895 counts/V	223 counts/V
Input Current	DC Link	256 counts/A	N/A
DC-DC Inductor Current	ADC interface	717 counts/A	57.3 counts/A
DC Bus Current	DC Link	1638 counts/A	N/A
Motor Phase Currents	ADC interface	1024 counts/A	81.9 counts/A

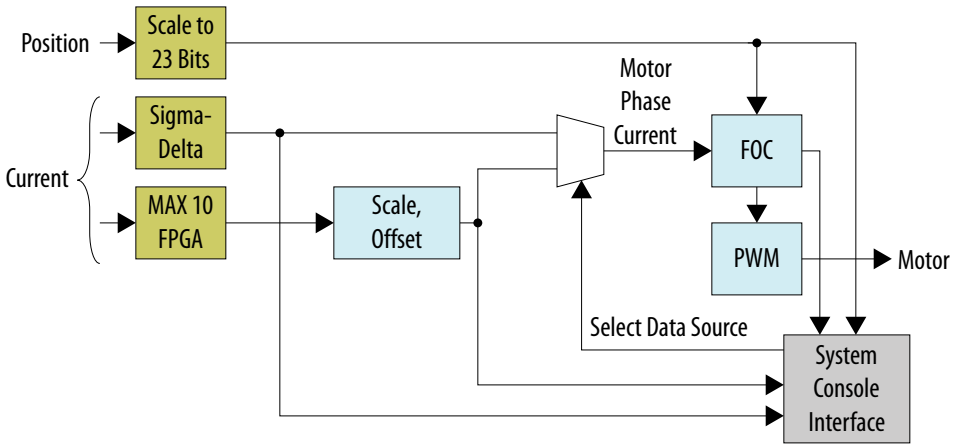
5.2. Signal Scaling in the Software of the Drive-On-Chip Design Example for Intel MAX 10 Devices

The software performs scaling to:

- Normalize sigma-delta and MAX 10 ADC samples for use in the FOC algorithm
- Apply zero offsets
- Position feedback scaling

Figure 19. Signal Scaling Architecture

This figure shows a simplified block diagram of the scaling in the software application supporting the Tandem Motion-Power 48 V Board.



Scaling of Motor Phase Current Samples

The design treats motor phase current samples as dimensionless numbers in the FOC algorithm, rather than real current measurements.

To compensate for the differences in signal conditioning between the different ADCs, the design scales Intel MAX 10 ADC samples as it reads them from the ADC to normalize them to represent the same physical quantity as the sigma-delta ADC samples.

Table 7. Scaling of Intel MAX 10 Motor Phase Current Samples

This table shows the ADC responses for the motor phase currents and the scaling applied to the Intel MAX 10 ADC samples to normalize them to the Sigma-Delta samples. The scaling is also shown with a power-of-2 divisor to simplify integer arithmetic.

Item	Sigma-Delta	Intel MAX 10
Motor Phase Currents	1024 counts/A	81.9 counts/A
Scaling	1	1024/81.9 or 12803/1024

Calculation of Zero Offsets

Offset errors arise in the ADC conversion process from a number of factors, including

- Component tolerance in sense circuits
- Offsets in sense amplifiers
- Errors in Vdd supply to sense amplifiers and ADCs
- Offsets in the ADC converters

Offsets are most noticeable when converting low level signals where they lead to a larger error in percentage terms. For the most crucial feedback, the design attempts to calculate and correct for the offsets.

Motor Phase Current Zero Offset

The design calculates the zero offset for the motor phase current during startup. the design samples a number of conversions while no motor current is flowing. The design averages the samples to calculate the offset and applies them as a correction to the offset register in the sigma delta ADC module, or stores them in the `drive_params` structure for use in software for the Intel MAX 10 ADCs.

Inductor Current Zero Offset on Tandem Motion Power Board

You cannot shut off the current flow through the DC-DC inductors. The design calculates approximate offsets from the average of the offsets previously calculated for the motor phase currents. The design applies power to all the converters from the same Vdd supply and in the same ambient surroundings.

5.3. Scale Factors for the Drive-On-Chip Design Example in the System Console Toolkit

The design applies scale factors to signals in the system console toolkit for diagnostic display in human readable, physical units (e.g. volts, amps).

Table 8. Scale Factors in System Console

This table shows the scale factors that the GUI uses, based on the scaling of the motor phase currents as in *Scaling of Motor Phase Current Samples*.

Item	Sigma Delta Scaling	MAX 10 Scaling
Motor Phase Voltages	545 counts/A	67.7 counts/V
DC Bus Voltage	545 counts/V	67.7 counts/V
Input Voltage	895 counts/V	223 counts/V
Input Current	252 counts/A	N/A
Inductor Current	717 counts/A	57.3 counts/A
DC Bus Current	1638 counts/A	N/A
Motor Phase Currents	1.024 counts/mA	1.024 counts/mA

Table 9. Scale Factors for Id and Iq in System Console

The table shows that scaling of Id (requested and actual) and Iq (requested and actual) in the GUI is the same as the motor phase current scaling

Item	Sigma Delta Scaling (counts/mA)	MAX 10 Scaling (counts/mA)
Id Direct Current	1.024	1.024
Iq Quadrature Current	1.024	1.024

SVM Voltage

The design calculates the maximum count of the PWM from the the PWM frequency, and passes it to the software from the `system.h` header file generated with the Nios II board support package (BSP). The maximum count varies with the PWM frequency and sample rate and is (PWM frequency in Hz)/((Sample rate) *1000). For example, with a PWM frequency of 300 MHz and a sample rate of 16 kHz the maximum count is 18,750.

Voltage demand signals for the PWM IP have a full-scale value equal to the maximum count, so setting the voltage demand to the maximum count value achieves 100% duty cycle and 100% of DC link voltage. Setting the voltage demand to 0 achieves 0% duty cycle and 0% of the DC link voltage. By convention, voltages for display purposes are centred around 0. For example, if the DC link voltage is 48 V voltage demand signals between 0 and maximum count map to 0 to +48 V outputs, but these signals are offset and show in System Console as -24 V to +24 V.

Using the above example of 300 MHz PWM and 16 kHz sample rate for the Tandem Motion-Power 48 V Board, in System Console:

Offset 18,750/2 = 9,375

Scaling 9,375/24 = 391

Related Information

Signal Scaling in the Software of the Drive-On-Chip Design Example for Intel MAX 10 Devices on page 31

6. Motor Control Software

The Drive-on-Chip Design Example motor control software is in C, runs under the Micrium µC/OS-II real-time Operating System on the Nios II processor, and is in two parts.

The BSP is generated from the Qsys system via the `.sopcinfo` file, which contains a description of the system interconnectivity and module base addresses. The design includes drivers for Nios II peripherals that the Nios II Hardware Abstraction Layer (HAL) supports.

The application program comprises a number of threads handling initialization, status reporting, and communication functions and an Interrupt Service Routine (ISR), triggered by the PWM timebase, which covers the real-time aspects of running the motor control FOC algorithm. The design includes header files and basic drivers for motor control peripherals that the Nios II HAL does not directly support.

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

For Instructions on how to install doxygen software and generate the HTML help files, refer to "Viewing Motor Control Software Help Files".

Figure 20. Main Program

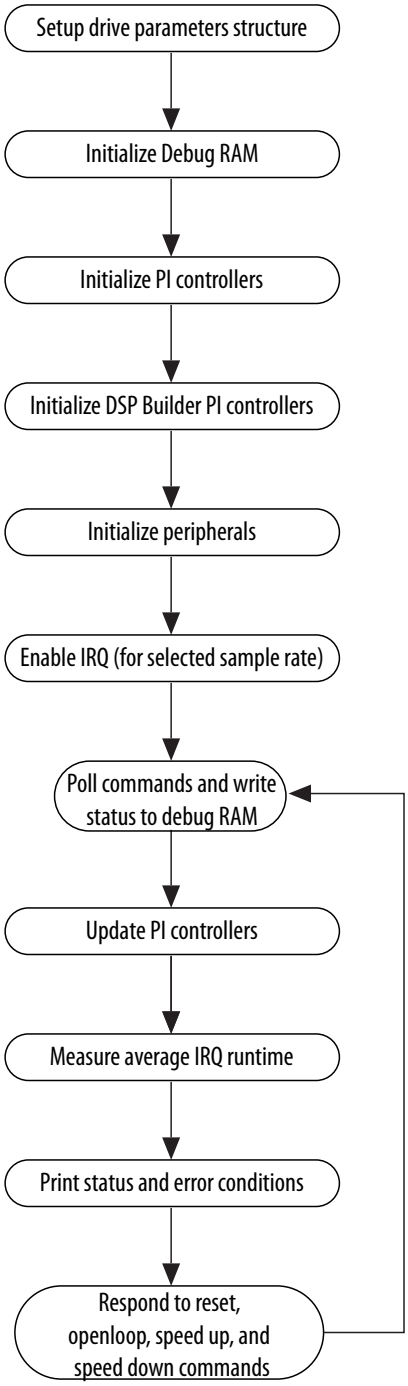
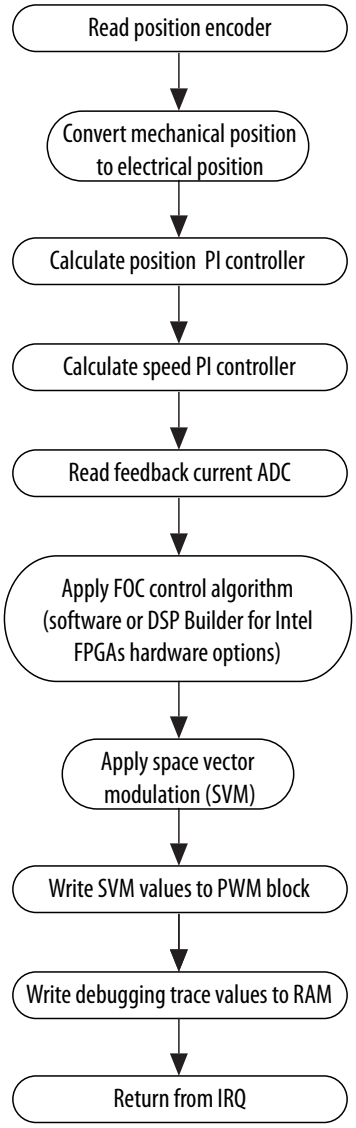


Figure 21. IRQ Routine



Related Information

Rebuilding the Drive-On-Chip Design
Instructions to rebuild the BSP after making hardware changes and rebuilding the application software.

6.1. Viewing Motor Control Software Help Files

Doxygen generated HTML help files are in the `software\doxygen` directory.

1. Install Doxygen and Graphviz (to generate diagrams).
2. Check that the environment variables are set to point to the correct path:
   a. Click **Advanced system settings ➤ Environment variables**.

b. Click on **Path** in the **System Variables** pane and select **Edit**

c. Select **Browse...** and add <*path*>\doxygen\bin and <*path*>\Graphviz2.44\bin directories.

3. Open a Nios II command shell and change directory to `software/doxygen` in your project directory.
4. Write the command "`make doxygen`" to generate HTML help files in the same directory.
5. Open the `index.html` file in a browser to view the help files.

Related Information

- Doxygen
- Graphviz

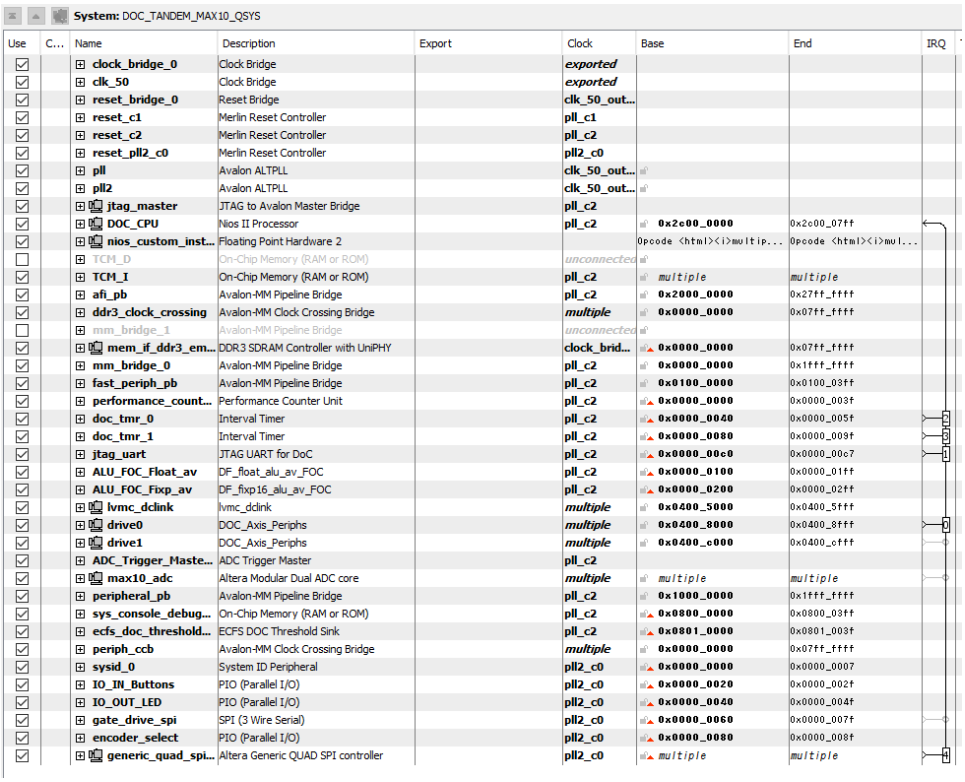
7. Functional Description of the Drive-On-Chip Design Example for Intel MAX 10 Devices

The design consists of two main elements: Qsys, DSP Builder for Intel FPGAs, IP, and RTL sources compiled into an FPGA programming file; and C source code compiled to run on a Nios II processor in the FPGA.

The Qsys system consists of:

- Nios II processor subsystem
- DC link monitors
- Intel MAX 10 modular dual ADC
- DC-DC converter
- FOC subsystem
- One or two motor drive axes comprising the following motor control peripheral components:
  - 6-channel PWM
  - Drive system monitor
  - Quadrature encoder interface
  - Resolver SPI interface
  - ADC interface

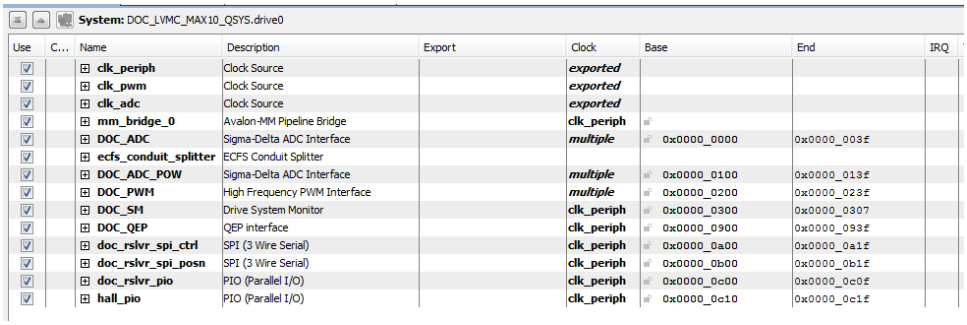
Figure 22. Qsys System Top-Level Design



System: DOC_TANDEM_MAX10_QSYS

Use	C...	Name	Description	Export	Clock	Base	End	IRQ
☑		clock_bridge_0	Clock Bridge		exported			
☑		clk_50	Clock Bridge		exported			
☑		reset_bridge_0	Reset Bridge		clk_50_out...			
☑		reset_c1	Merlin Reset Controller		pll_c1			
☑		reset_c2	Merlin Reset Controller		pll_c2			
☑		reset_pll2_c0	Merlin Reset Controller		pll2_c0			
☑		pll	Avalon ALTPLL		clk_50_out...			
☑		pll2	Avalon ALTPLL		clk_50_out...			
☑		jtag_master	JTAG to Avalon Master Bridge		pll_c2			
☑		DOC_CPU	Nios II Processor		pll_c2	0x2c00_0000	0x2c00_07ff	
☑		nios_custom_inst...	Floating Point Hardware 2			Opcode <html><i>multip...	Opcode <html><i>mul...	
☐		TCM_D	On-Chip Memory (RAM or ROM)		unconnected			
☑		TCM_I	On-Chip Memory (RAM or ROM)		pll_c2	multiple	multiple	
☑		afi_pb	Avalon-MM Pipeline Bridge		pll_c2	0x2000_0000	0x27ff_ffff	
☑		ddr3_clock_crossing	Avalon-MM Clock Crossing Bridge		multiple	0x0000_0000	0x07ff_ffff	
☐		mm_bridge_1	Avalon-MM Pipeline Bridge		unconnected			
☑		mem_if_ddr3_em...	DDR3 SDRAM Controller with UniPHY		clock_brid...	0x0000_0000	0x07ff_ffff	
☑		mm_bridge_0	Avalon-MM Pipeline Bridge		pll_c2	0x0000_0000	0x1fff_ffff	
☑		fast_periph_pb	Avalon-MM Pipeline Bridge		pll_c2	0x0100_0000	0x0100_03ff	
☑		performance_count...	Performance Counter Unit		pll_c2	0x0000_0000	0x0000_003f	
☑		doc_tmr_0	Interval Timer		pll_c2	0x0000_0040	0x0000_005f	
☑		doc_tmr_1	Interval Timer		pll_c2	0x0000_0080	0x0000_009f	
☑		jtag_uart	JTAG UART for DoC		pll_c2	0x0000_00c0	0x0000_00c7	
☑		ALU_FOC_Float_av	DF_float_alu_av_FOC		pll_c2	0x0000_0100	0x0000_01ff	
☑		ALU_FOC_Fixp_av	DF_fixp16_alu_av_FOC		pll_c2	0x0000_0200	0x0000_02ff	
☑		lvmc_dclink	lvmc_dclink		multiple	0x0400_5000	0x0400_5fff	
☑		drive0	DOC_Axis_Periphs		multiple	0x0400_8000	0x0400_8fff	
☑		drive1	DOC_Axis_Periphs		multiple	0x0400_c000	0x0400_cfff	
☑		ADC_Trigger_Maste...	ADC Trigger Master		pll_c2			
☑		max10_adc	Altera Modular Dual ADC core		multiple	multiple	multiple	
☑		peripheral_pb	Avalon-MM Pipeline Bridge		pll_c2	0x1000_0000	0x1fff_ffff	
☑		sys_console_debug...	On-Chip Memory (RAM or ROM)		pll_c2	0x0800_0000	0x0800_03ff	
☑		ecfs_doc_threshold...	ECFS DOC Threshold Sink		pll_c2	0x0801_0000	0x0801_003f	
☑		periph_ccb	Avalon-MM Clock Crossing Bridge		multiple	0x0000_0000	0x07ff_ffff	
☑		sysid_0	System ID Peripheral		pll2_c0	0x0000_0000	0x0000_0007	
☑		IO_IN_Buttons	PIO (Parallel I/O)		pll2_c0	0x0000_0020	0x0000_002f	
☑		IO_OUT_LED	PIO (Parallel I/O)		pll2_c0	0x0000_0040	0x0000_004f	
☑		gate_drive_spi	SPI (3 Wire Serial)		pll2_c0	0x0000_0060	0x0000_007f	
☑		encoder_select	PIO (Parallel I/O)		pll2_c0	0x0000_0080	0x0000_008f	
☑		generic_quad_spi...	Altera Generic QUAD SPI controller		pll2_c0	multiple	multiple	

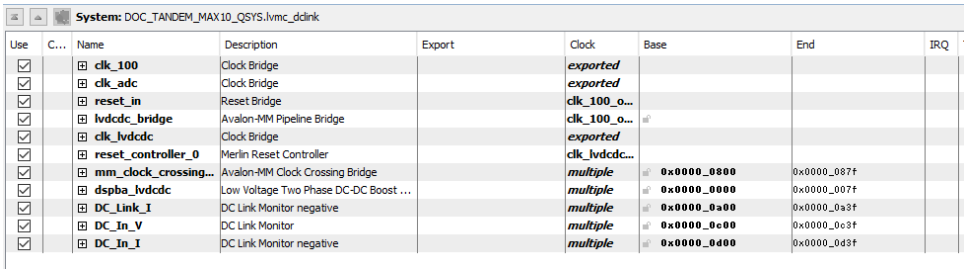
Figure 23. Qsys System for a Drive Axis



System: DOC_LVMC_MAX10_QSYS.drive0

Use	C...	Name	Description	Export	Clock	Base	End	IRQ
☑		clk_periph	Clock Source		exported			
☑		clk_pwm	Clock Source		exported			
☑		clk_adc	Clock Source		exported			
☑		mm_bridge_0	Avalon-MM Pipeline Bridge		clk_periph			
☑		DOC_ADC	Sigma-Delta ADC Interface		multiple	0x0000_0000	0x0000_003f	
☑		ecfs_conduit_splitter	ECFS Conduit Splitter					
☑		DOC_ADC_POW	Sigma-Delta ADC Interface		multiple	0x0000_0100	0x0000_013f	
☑		DOC_PWM	High Frequency PWM Interface		multiple	0x0000_0200	0x0000_023f	
☑		DOC_SM	Drive System Monitor		clk_periph	0x0000_0300	0x0000_0307	
☑		DOC_QEP	QEP interface		clk_periph	0x0000_0900	0x0000_093f	
☑		doc_rslvr_spi_ctrl	SPI (3 Wire Serial)		clk_periph	0x0000_0a00	0x0000_0a1f	
☑		doc_rslvr_spi_posn	SPI (3 Wire Serial)		clk_periph	0x0000_0b00	0x0000_0b1f	
☑		doc_rslvr_pio	PIO (Parallel I/O)		clk_periph	0x0000_0c00	0x0000_0c0f	
☑		hall_pio	PIO (Parallel I/O)		clk_periph	0x0000_0c10	0x0000_0c1f	

Figure 24. Qsys System for DC-DC Converter



System: DOC_TANDEM_MAX10_QSYS.lvmc_dclink

Use	C...	Name	Description	Export	Clock	Base	End	IRQ
☑		clk_100	Clock Bridge		exported			
☑		clk_adc	Clock Bridge		exported			
☑		reset_in	Reset Bridge		clk_100_o...			
☑		lvdcdc_bridge	Avalon-MM Pipeline Bridge		clk_100_o...			
☑		clk_lvdcdc	Clock Bridge		exported			
☑		reset_controller_0	Merlin Reset Controller		clk_lvdcdc...			
☑		mm_clock_crossing...	Avalon-MM Clock Crossing Bridge		multiple	0x0000_0800	0x0000_087f	
☑		dspba_lvdcdc	Low Voltage Two Phase DC-DC Boost ...		multiple	0x0000_0000	0x0000_007f	
☑		DC_Link_I	DC Link Monitor negative		multiple	0x0000_0a00	0x0000_0a3f	
☑		DC_In_V	DC Link Monitor		multiple	0x0000_0c00	0x0000_0c3f	
☑		DC_In_I	DC Link Monitor negative		multiple	0x0000_0d00	0x0000_0d3f	

7.1. Nios II Processor Subsystem

The Drive-On-Chip Design Example Nios II processor subsystem offers a fully functional processor system with debugging capabilities:

The Nios II processor subsystem comprises the following Qsys components:

- Nios II fast processor
- Floating-point hardware custom instructions (optional)
- Tightly-coupled instruction and data memory
- JTAG master
- Performance counters
- DDR controller
- MOSFET gate driver SPI
- 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 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.

7.2. Six-channel PWM Interface

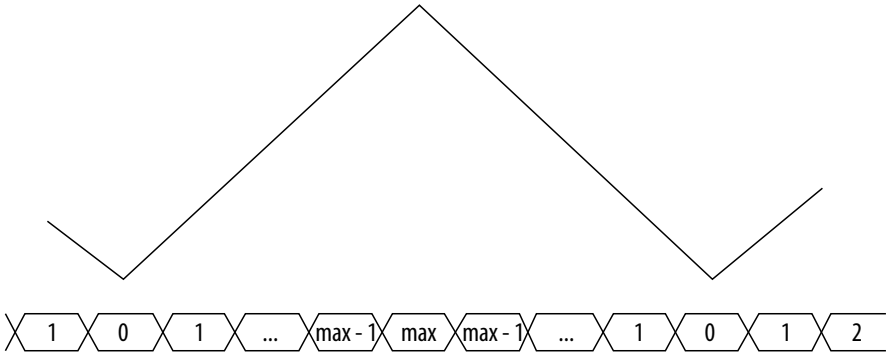
The Drive-On-Chip Design Example six-channel PWM interface operates as three pairs of outputs, with each pair operating differentially to drive the upper and lower power transistors (e.g., IGBT or MOSFET driven via external drivers) in a half-bridge power stage.

The PWM interface operates with a PWM carrier clock of 300 MHz for high resolution control of the MOSFET switching times.

The PWM interface ensures a dead time between switching to ensure both outputs are not high at the same time; the dead time prevents short circuit "shoot-through" in the power transistors. The input clock and a PWM counter set the PWM frequency. The counter alternately ramps up from zero to a maximum value and ramps down from the maximum value to zero. The sequence is as follows:

```
0, 1, 2, … max - 1, max, max - 1, … 2, 1, 0, …
```

Figure 25. PWM Counter Value



The maximum value of the counter ramp, max, is software configurable. The PWM frequency is fPWM = fCLK/(2 x max)

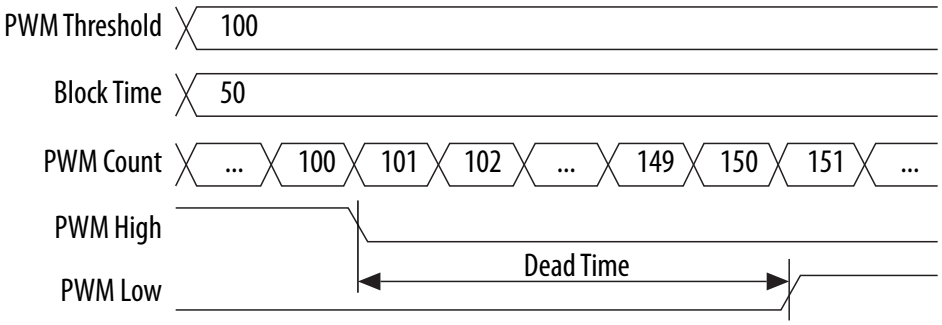
The 16-bit counter resolution is sufficient to generate an 8-kHz PWM output. The design generates high- and low-side drive signals for the insulated gate bipolar transistor (IGBT) module by comparing the ramp counter value with the values you set in the PWM threshold configuration registers. The design inserts a dead period between the switching of the upper and lower drive signals according to the value set in the PWM blocking time configuration register.

The design sets carrier_latch output signal high for one clock cycle when the PWM counter is at 0 or max. This signal triggers a position encoder to take a position reading.

The start output signal is a trigger for the ADC IP to start conversion. The trigger_up configuration register sets the PWM count value and the start signal is high for one clock cycle while the PWM is counting up. The trigger_down configuration register sets the PWM count value and the start signal is high for one clock cycle while the PWM is counting down. Set the trigger_up and trigger_down registers symmetrically to ensure a regular ADC sample position offset before the reversal point of the counter. In other words, trigger_up = MAX - offset, and trigger_down = offset.

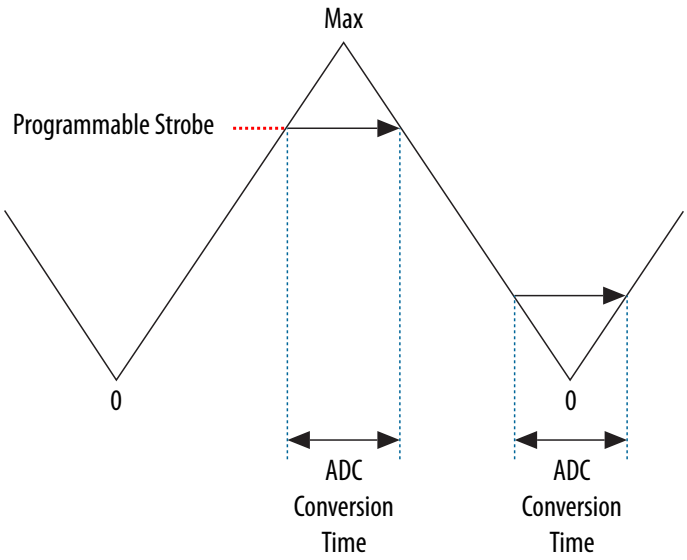
The design calculates the PWM blocking time configuration register as pwm_block = dead time x fCLK. Dead time refers to the time when the design turns off both upper and lower transistors, to prevent short circuits. You must obtain specific dead time values for the specific IGBT or MOSFET module you are using. For example, with a dead time requirement of 2µs and a PWM module clock of 300 MHz, the pwm_block value is 600 (=2µs x 300 MHz. Figure 5 shows PWM output generation (including dead time).

Figure 26. PWM Output Generation (Including Dead Time)



Based on the PWM counter value, the PWM component generates configurable timing output strobes for triggering ADC conversion for feedback-current readings. Configure the ADC start pulse to perform the conversion during the quietest period of the PWM cycle away from PWM switching events (around the min and max values of the PWM counter).

Figure 27. Configurable Timing Output Strobes



7.3. DC Link Monitor

The Drive-On-Chip Design Example DC-link monitor uses an instance of the sinc3 filter module, similar to the instance that the sigma-delta interface for drive axes uses. Three DC link monitor modules monitor DC link input voltage, DC link input current, and DC link output current. All three modules reside in `lvmc_dclink.qsys` subsystem.

The design compares the software configurable reference values with the filtered DC-link voltage and current values to determine if the DC-link voltage and current are within the expected range. Status outputs indicate overvoltage, undervoltage, and overcurrent conditions to external protection circuitry.

The design feeds these status output signals to the DC-DC converter module and shuts down the DC link output when any abnormal state occurs.

ADC Interface Result

The DC link voltage restricts the demodulated result of the DC-link monitor to a positive value because the DC-link voltage cannot be negative. The design clips any negative result after applying the offset correction to zero. The DC link input current and output current clips the demodulated results to fit within 16 bits signed integer range.

Offset Adjustment for DC-Link Monitor

The design adds offset values to demodulator results. The design specifies offset values in the Offset register. During normal operation, the offset value is 16,384.

7.4. Drive System Monitor

The Drive-On-Chip Design Example drive system monitor is a state machine that responds to state requests from the software and fault signals from hardware.

Application software writes to the drive system monitor to request a change of state. The hardware may accept or decline the change of state request, depending on the system status (for example, overvoltage status, undervoltage status, and current measurements alter the system status). A subsequent read from the Status register verifies if the design accepts the change of state.

The drive system monitor latches status signals from the system so the signals are available as status register bits and direct outputs. For example, the direct outputs can drive status LEDs.

7.4.1. Drive System Monitor States for the Drive-On-Chip Design Example

Table 10. Drive System Monitor States

State	Name	System State
0	Idle	Reset state, moves immediately to preinit
1	Precharge	PWM counter running, low side outputs enabled, voltage errors monitored
2	Prerun	PWM counter running, low side outputs enabled, voltage and current errors monitored
3	Run	PWM counter running, low and high side outputs enabled, voltage and current errors monitored
4	Error	Error state, PWM counter running, outputs disabled
5	init	PWM counter running, outputs disabled, voltage errors monitored
6	preinit	PWM counter running, outputs disabled

7.5. Quadrature Encoder Interface

The Drive-On-Chip Design Example quadrature encoder interface monitors and decodes the A, B and I signals from a quadrature encoder. The resulting output is a count value representing the position of the motor shaft.

The quadrature encoder interface allows you to:

- Program maximum count value to match a wide range of encoders.
- Increment or decrement the counter on each A or B input edge.
- Capture the latest count value on an index pulse.
- Reset the count value on an index pulse.
- Reverse the direction of the count, equivalent to swapping the A and B inputs.
- Capture the latest count by an external strobe to synchronise with the PWM module and ADC sampling.

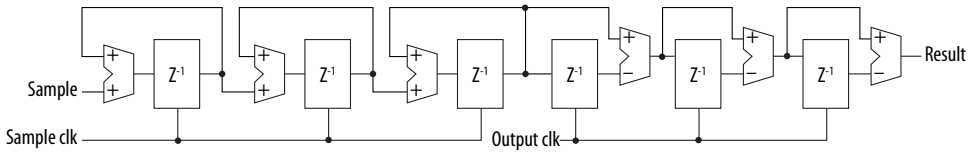
7.6. Sigma-Delta ADC Interface for Drive Axes

The Drive-On-Chip Design Example sigma-delta ADC interface samples the 20-MHz 1-bit ADC serial input for 3 phase current inputs for each drive axis. A decimating sinc^3 filter in the FPGA then low-pass filters the serial input. The sinc^3 filter does not require hardware multipliers.

Sinc^3 Filter

Figure 28. Sinc^3 Filter Topology

The input samples pass through three integrator stages before a factor M decimates them. The design reserves every Mth sample and discards M-1 samples. The design passes the reserved samples through three differentiators to produce a final output value.



The pulse-width modulation (PWM) block triggers ADC conversion with a reset signal that resets the filters and control logic. The design calculates:

- The direct-current gain of the sinc3 filter as GainDC = MK (where K = 3 for sinc3).
- The internal bus width of the filters as Internal bus width = 1 + Klog2M, to account for word growth in the filter stages
- The output data rate for an input sample rate fS and decimation factor M as Data rate = fS/M.

When the settling time satisfies and the ADC conversion completes, the design sends an interrupt to the processor. The design calculates the performance of N-bit ADC as SNR = 6.02N + 1.76dB, where SNR is the signal to noise ratio. Additional noise in the system affects the performance value. The design calculates the effective number of bits (ENOB) as ENOB = (SINAD - 1.76dB)/6.02, where SINAD is the signal to noise and distortion. The design determines SNR, SINAD, and ENOB by decimation ratio.

The sinc^3 filter requires a time period 3× longer than the time period of the output data rate to settle. The standard settings of M=128 keeps the settling time short and a deliver a suitable ENOB of 16bits. By choosing to synchronize sampling to the quiet periods of the PWM waveform, signal quality is acceptable when sampled at 16 kHz despite the theoretical output data rate of 156.2 kHz.

Table 11. Sinc^3 Filter: Fs = 20 MHz

Decimation (M)	GainDC	Word Size	Bus Width	Data rate (kHz)	Settling Time (µs)	ENOB
8	512	9	10	2500	1.2	6.4
16	4096	12	13	1250	2.4	8.9
64	262,144	18	19	312.5	9.6	13.9
128	2,097,152	21	22	156.2	19.2	16.4

Two Filter Paths

The design has two separate filter paths: a control loop filter path and an overcurrent detection filter path.

The control loop filters are slower but more accurate than the overcurrent detection filters with a software selectable decimation factor of M=128 or M=64. The control loop filters have an offset correction feature for zero-offset correction. The filter output is a signed 16 bit (2's complement) format.

The overcurrent detection filters are faster but less accurate than the control loop filters with a software selectable decimation factor of M=16 or M=8. A software configurable overcurrent output provides a direct output to disable the motor when under hardware control.

The control loop and overcurrent detection filters use the same control bit for decimation selection. The possible selections are:

- control loop M=128, overcurrent M=16
- control loop M=64, overcurrent M=8.

Clocks

The design performs synchronization between the ADC clock and the FPGA system clock at the output stage before the design delivers output data in the Avalon memory-mapped interface agent registers.

The external ADC components require a clock source from the FPGA and return samples synchronous to the FPGA-sourced clock. The same clock within the FPGA drives the ADC filters.

You must apply appropriate timing constraints in the Intel Quartus Prime software project to guarantee correct sampling of the ADC interface data. Base the sampling on the clock to output specification of the ADC.

7.6.1. Offset Adjustment for Sigma-Delta ADC Interface

Use the offset adjustment to calculate the output voltages in the Drive-On-Chip Design Example.

Table 12. Sigma-Delta ADC Characteristics

The table describes typical characteristics of a sigma-delta ADC and the demodulated output of the $sinc^3$ filter. The output code is a positive value.

Analog Input	Voltage Input (mV)	Density of 1s	Demodulated ADC Code (16-bit)
Full-scale range	640	-	-
+ Full scale	+ 320	100%	65,535
+ Recommended input range	+ 200	31.25%	53,248
Zero	0	50%	32,768
- Recommended input range	- 200	18.75%	12,288
- Full scale	- 320	0%	0

The design adds offset values to demodulator results to represent the bipolar input signal and to allow for zero-offset adjustment. The offset values are in the offset_u or offset_w registers.

During normal operation, the offset value is 32,768, or 50% of the full-scale range, to bring the demodulated result into the range of -32,768 to +32,767. The design adjusts the offset value to correct for zero-offset errors during calibration.

7.7. Intel MAX 10 ADCs

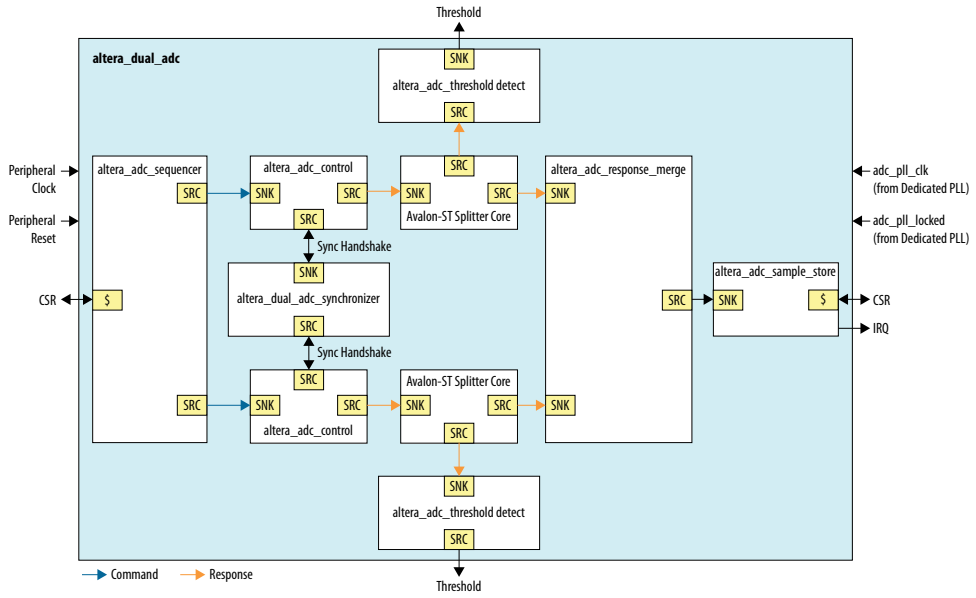
The Drive-On-Chip Design Example for Intel MAX 10 devices has dual Intel MAX 10 ADCs with Avalon memory-mapped sample storage and threshold violation detection.

Software reads converted samples from an Avalon memory-mapped agent interface.

Threshold violation errors are output on two Avalon streaming sources, one for each of the ADC modules that make up the dual ADC.

To change the thresholds: edit the component settings in Qsys, regenerate the Qsys project, and recompile in the Intel Quartus Prime software.

Figure 29. Intel MAX 10 Dual ADC with Avalon Memory-Mapped Sample Storage and Threshold Violation Detection



Related Information

- Changing the Intel MAX 10 ADC Thresholds or Conversion Sequence on page 24
- Intel MAX 10 Analog to Digital Converter User Guide

7.8. ADC Threshold Sink

The Intel MAX 10 ADC threshold sink module provides an interface between the Avalon streaming threshold sources of the Intel MAX 10 dual ADC and the drive system monitor modules.

The Avalon streaming sink interfaces capture threshold violation errors from the Intel MAX 10 ADC. Each Avalon streaming interface can indicate eight under- or over-threshold violations corresponding to the eight channels of each of the two ADC modules that make up the dual ADC.

The software selectively captures and latches errors for later checking and clearing.

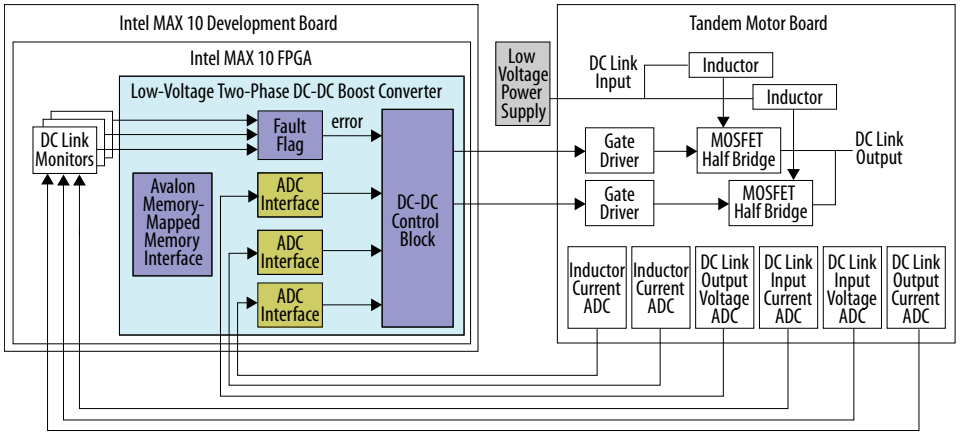
The design selectively enables latched errors for output to one or more drive system monitor modules via the under and over conduits. The drive system monitors use the error signals to safely shut down the DC-DC converter and one or more drive axes in the event of an error condition such as overcurrent or overvoltage.

You can selectively set the error latches, to simulate error conditions, for test purposes.

7.9. DC-DC Converter

The Drive-On-Chip Design Example for Intel MAX 10 devices DC-DC converter comprises two-phase DC-DC converter power electronics on the Tandem Motion-Power 48 V board and IP on Intel MAX 10 FPGA.

Figure 30. DC-DC Converter Block Diagram



The power electronics hardware includes:

- Two inductors (phases)
- Two MOSFET half-bridges
- MOSFET gate driver chip per MOSFET half-bridge
- Current sensing of each inductor current: the total DC link input current and the total DC link output current
- Voltage sensing of the DC Link input voltage and DC Link output voltage

The low voltage two-phase DC-DC boost converter includes:

- Three sigma-delta ADC interfaces
- DC-DC control block
- Avalon memory-mapped agent interface for control and status
- Fault flag logic

Intel developed the converter using DSP Builder for Intel FPGAs.

The DC-DC converter consists of 2 phases that provide bidirectional power flow from a low voltage power source or battery (typically 12 V DC) to a DC link output bus (maximum 48 V DC) that feeds one or more motor drive inverters. The DC-DC converter provides the boost function to increase the voltage. It also provides a buck function during periods of regenerative braking to deliver power from the DC link output bus back to the low voltage source (i.e., battery in this case).

The gate driving signals for the two phases are 180 degrees out of phase so that they alternate in supplying current during buck-boost function, which gives smoother output current and voltage.

The control consists of two independent inner current loops and an outer voltage loop that regulates the DC link output bus voltage to a specified value (e.g., 32 V DC). An external host (e.g. the Nios processor) gives the target value via the Avalon memory mapped agent interface.

For feedback control, the DC-DC control block takes the output of the two inductor current and DC link output voltage ADCs outputs. The design connects the outputs of the DC link input voltage, the DC link input current, and the DC link output current ADCs with DC link monitors to detect fault conditions for external host monitoring.

Low-Voltage Two-phase DC-DC Boost Converter

The top-level module of the DC-DC converter is a Qsys component, which is a manually-created HDL module that:

- Instantiates the VHDL entity generated by DSP Builder for Intel FPGAs (the DC-DC control block)
- Instantiates sigma-delta ADC interface module
- Adds an Avalon memory-mapped register agent and conduit signals.

The design instantiates the the DC-DC Converter in the `lvmc_dclink.qsys` subsystem.

You can instantiate the Qsys component in a Qsys system and connect it to the Nios II processor and other modules. The register agent allows software access to the DC-DC converter parameters, control, and status. The conduits connect to various system-wide control and status signals that are outside the software domain.

The top-level module implements safety features in the fault flag logic that you may use with external logic, to protect the system in the case of a malfunction. The fault flag logic combines (logial ORs) three fault bit signals provided by the DC link monitors. They monitor the total DC link input current, the DC link input voltage, and the total DC link output current, and gate the following two independent enable sources that enable the DC-DC converter:

- The `enable` bit in the control register
- The `enable` input for the DSP Builder block.

The ADC interface in the DC-DC converter has three demodulators for one-bit sigma-delta signals, which sense the following DC link feedback signals:

- The output voltage
- The two output inductor currents

The ADC interface delivers 16-bit ADC values at 156 kHz, which is 1/128 of 20 MHz. The design feeds the ADC output values to the DC-DC control block to carry out the feedback control loops. The voltage feedback control and two inductor current feedback control loops run autonomously in hardware, without software interaction in the loop execution.

Related Information

Avalon Interface Specification

7.9.1. DC-DC Control Simulink Models

The Drive-On-Chip Design Example includes three MATLAB Simulink models, which lead step by step from offline simulation to HDL code generation while maintaining the numerical simulation results.

The models are:

- `lvdcdc_simpower.slx` where the power electronics simulate in SimScape and the PI control loops in Simulink standard blocks
- `lvdcdc_2phase_hwsim.slx` is similar to `lvdcdc_simpower.slx` but the power electronics simulate in standard Simulink blocks, not SimScape
- `lvdcdc_adsp_vhdl.slx` is like `lvdcdc_2phase_hwsim.slx` but the algorithm is implemented using DSP Builder for Intel FPGAs so that you can generate HDL code.

Figure 31. DC-DC Converter Linear MATLAB Model

The figure shows the linear MATLAB model (`lvdcdc_simpower.slx`). The linear model cannot generate VHDL, but you create it to provide a rapid simulation to develop control dynamics and determine controller gains.

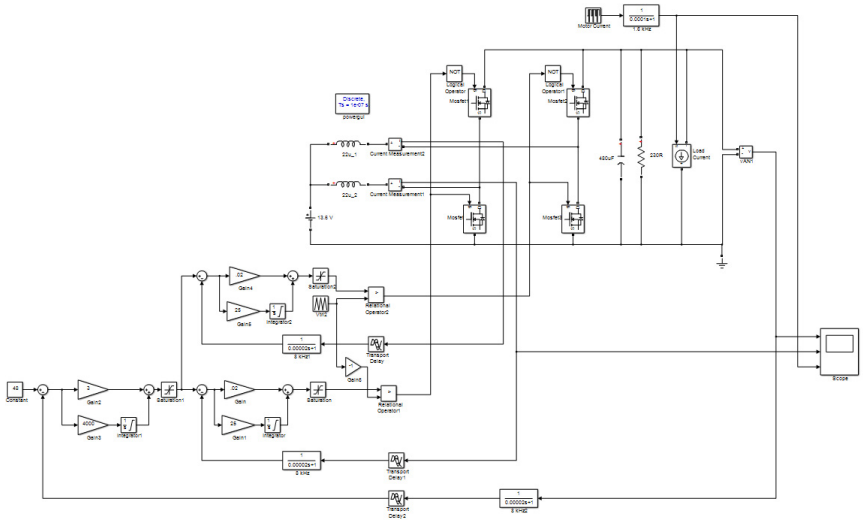


Figure 32. DC-DC Converter: DC bus Voltage, Inductor Currents, Motor Load Current (stimulus)

The figure shows the linear MATLAB model (`lvdcdc_simpower.slx`) and simulation.

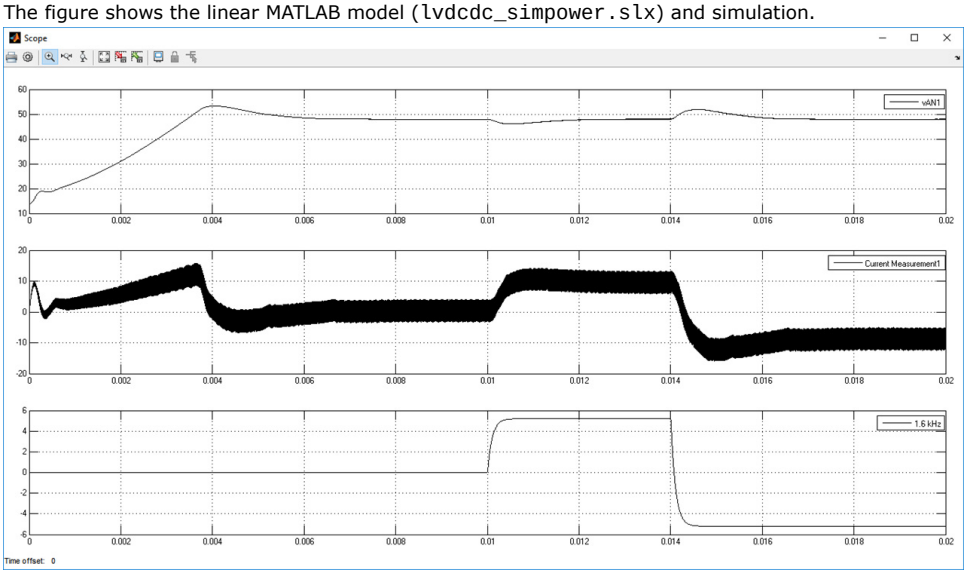
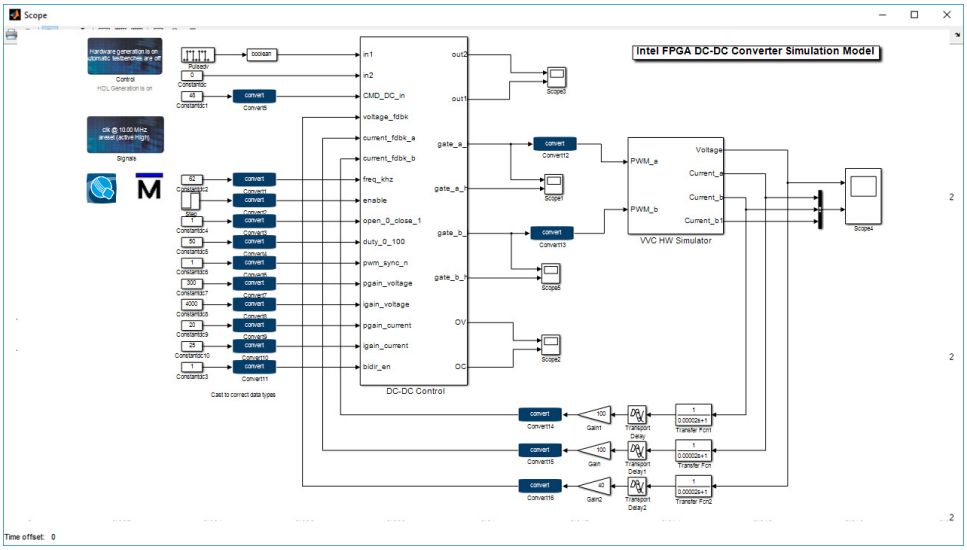


Figure 33. DSP Builder for Intel FPGAs Top-level Model

The DSP Builder for Intel FPGAs model (lvdcdc_adsp_vhdl.slx) performs the same simulation as above, but includes DSP Builder for Intel FPGAs blocks that allow simulation of VHDL and auto-generation of VHDL code.



7.9.1.1. DC-DC Control Block

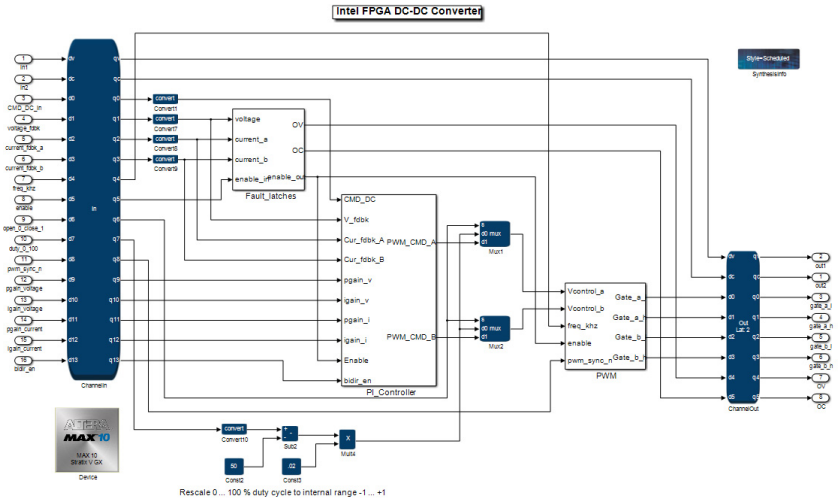
The Drive-On-Chip Design Example for Intel MAX 10 devices DC-DC Converter block contains a DC-DC Control block, which is implemented in a DSP Builder for Intel FPGAs model (lvdcdc_adsp_vhdl.slx)

The DC-DC control block consists of

- Fault latches block
- PI controller block
  - Two independent current control loops
  - Voltage control loop
- PWM block

Figure 34. DSP Builder for Intel FPGAs DC-DC Control Block

The figure shows the expanded DC-DC Control block



The DC-DC control block has the portion of the simulation for which you generate VHDL code. The **ChannelIn** and **ChannelOut** blocks are the port interface for the VHDL code. The MATLAB Simulink inport and outport signals define the VHDL signal names, and the VHDL data formats are the signal formats that you typically set with the **Convert** block.

The **Convert** DSP block sets the data format. This model uses signed-fractional data format for the feedback signals and the control math inside the **DC-DC Control** block.

For instance, the voltage feedback signal `voltage_fdbk` comes into the **DC-DC Control** block with data format sfix13 and scaling "2^0" ("13bits . 0bits", where 13bits includes sign bit), which matches the 12 bit ADC twos-complement format. DSP Builder for Intel FPGAs also uses twos-complement maths to perform any calculations.

After the signal `voltage_fdbk` is inside the DC-DC control block the resolution is increased with another convert block to "sfix(27)" with output scaling "2^-12" ("15bits . 12bits").

In the PWM block, the design generates a triangular wave bounded within [-1.1] using a SR latch and counter counting at the frequency of the system clock of 10 MHz. After every 5000/`freq_khz` steps, the counter changes the direction of up-down counting. The design compares the triangular signal with thresholds representing the PWM duty cycle commands bounded within [-0.9, 0.9] to produce pulses for driving gates for each phase. The design inserts dead-time of five samples time duration (for clk=10MHz) at every transition of gate driving signals. You can extend the dead time by increasing the number of sample delays.

If the design asserts the fault input to the DC-DC converter, the enable bit in the DC-DC converter's control register is cleared and the DC-DC converter turns off. The enable bit remains cleared, and writing to the control register cannot set it again until the system negates the fault input.

The port map for the DSP Builder for Intel FPGAs-generated VHDL entity is:

```
entity lvdcdc_adsp_vhdl_DC_DC_Control is
    port (
        in1 : in std_logic_vector(0 downto 0);  -- ufix1
        in2 : in std_logic_vector(7 downto 0);  -- ufix8
        CMD_DC_in : in std_logic_vector(13 downto 0);  -- ufix14
        voltage_fdbk : in std_logic_vector(12 downto 0);  -- sfix13
        current_fdbk_a : in std_logic_vector(12 downto 0);  -- sfix13
        current_fdbk_b : in std_logic_vector(12 downto 0);  -- sfix13
        freq_khz : in std_logic_vector(13 downto 0);  -- ufix14
        enable : in std_logic_vector(0 downto 0);  -- ufix1
        open_0_close_1 : in std_logic_vector(0 downto 0);  -- ufix1
        duty_0_100 : in std_logic_vector(13 downto 0);  -- ufix14
        pwm_sync_n : in std_logic_vector(0 downto 0);  -- ufix1
        pgain_voltage : in std_logic_vector(13 downto 0);  -- ufix14
        igain_voltage : in std_logic_vector(13 downto 0);  -- ufix14
        pgain_current : in std_logic_vector(13 downto 0);  -- ufix14
        igain_current : in std_logic_vector(13 downto 0);  -- ufix14
        bidir_en : in std_logic_vector(0 downto 0);  -- ufix1
        out1 : out std_logic_vector(0 downto 0);  -- ufix1
        out2 : out std_logic_vector(7 downto 0);  -- ufix8
        gate_a_l : out std_logic_vector(0 downto 0);  -- ufix1
        gate_a_h : out std_logic_vector(0 downto 0);  -- ufix1
        gate_b_l : out std_logic_vector(0 downto 0);  -- ufix1
        gate_b_h : out std_logic_vector(0 downto 0);  -- ufix1
        OV : out std_logic_vector(0 downto 0);  -- ufix1
        OC : out std_logic_vector(0 downto 0);  -- ufix1
        clk : in std_logic;
        areset : in std_logic
    );
end lvdcdc_adsp_vhdl_DC_DC_Control;
```

7.9.1.1.1. DC-DC Model and VHDL Entity Signal Names and Data Format

Signals for the Drive-on-Chip Design Example.

Table 13. DC-DC Model and VHDL Entity Signal Names and Data Format

Signal Name	Data Format	Scaling	Default/Notes
Inputs			
In1	ufix1		0
In2	ufix8		0
CMD_DC_In	ufix14	1 V = 1	48
voltage_fdbk	sfix13	0.025 V = 1 or 1 V = 40	
current_fdbk_a	sfix13	0.01 A = 1 or 1 A = 100	
current_fdbk_b	sfix13	0.01 A = 1 or 1 A = 100	
freq_khz	ufix14		62
enable	ufix1		1
open_0_close_1	ufix1		1
duty_0_100	ufix14		

continued...

Signal Name	Data Format	Scaling	Default/Notes
pwm_sync_n	ufix1		1 (low to reset PWM counter)
pgain_voltage	ufix14	1/100	300 (* 1/100 = 3)
igain_voltage	ufix14	1e-7 (1/fclk)	4000
pgain_current	ufix14	1/1000	20 (* 1/1000 = 0/02)
igain_current	ufix14	1e-7 (1/fclk)	25
clk	std_logic		10 MHz
bidir_en	ufix1		0 for PS, 1 for battery
areset	std_logic		0
Outputs			
out1	ufix1		
out2	ufix8		
gate_a_h	ufix1		MOSFET gate signal
gate_a_l	ufix1		MOSFET gate signal
gate_b_h	ufix1		MOSFET gate signal
gate_b_l	ufix1		MOSFET gate signal
ov	ufix1		High = overvoltage
oc	ufix1		High = overcurrent

7.9.1.2. Generating VHDL for the DSP Builder for Intel FPGAs Models for the DC-DC Converter

1. Start DSP Builder for Intel FPGAs.
2. Change the directory to the `ip\dspba\two_phase_dc_dc`.
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.
5. Check the status of the orange DSP Builder for Intel FPGAs folding block. If the model includes it, folding is enabled. If it is removed or commented out, the model does not use folding.
6. Click **Simulation ➤ Start** .

DSP Builder for Intel FPGAs generates the VHDL files in `ip\dspba\two_phase_dc_dc\rtl`.

7.9.2. Sigma-Delta ADC Interface for DC-DC Converter

This interface is like the sigma-delta ADC interface for the motor drive axes.

The interface samples the 20-MHz 1-bit ADC signal inputs and applies a low pass sinc3 filter. Unlike the ADC interface for drive axes, the ADC Interface for the DC-DC Converter demodulation is always running and it generates filtered 16-bit results at 156.25 kHz, which is 1/128 of 20 MHz.

Related Information

Sigma-Delta ADC Interface for Drive Axes on page 44

7.10. Motor Control Modes

The Drive-On-Chip Design Example supports various control algorithms and commutation modes.

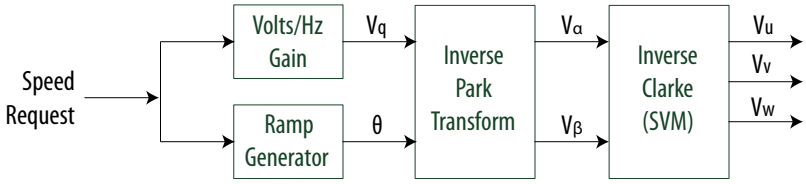
The design supports:

- Open-loop Volts/Hz speed control with sinusoidal commutation
- Speed and position control with field-oriented current control (FOC), sinusoidal commutation with quadrature encoder or resolver feedback
- Sensorless speed control with field-oriented current control using a sliding-mode speed and position observer using current feedback
- Speed control with trapezoidal commutation using Hall sensor feedback

Open Loop

The design supports open loop control using sinusoidal commutation and trapezoidal commutation. The design uses Volts per Hz control in which the voltage the design applies to the motor increases with increasing frequency (motor speed). After each interrupt the interrupt service routine (ISR) updates a ramp generator to represent the motor electrical angle based on the previous angle, desired speed, and sample rate. The ISR calculates the voltage to apply using a Volts per Hz control gain based on the frequency and motor parameters. In open loop sinusoidal commutation, the ISR applies the inverse Park Transform and SVM function from FOC to generate sinusoidal commutation.

Figure 35. Open Loop Sinusoidal Commutation

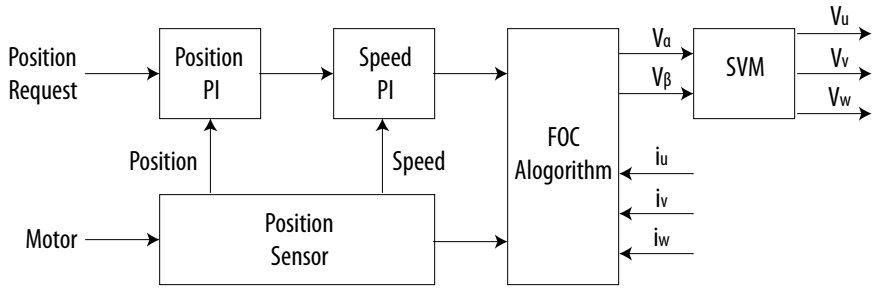


FOC with Position Sensor Feedback

The design supports FOC sensor control where the motor position feeds back to form a closed loop with position and speed PI control. The design senses the motor position by incremental (quadrature) encoders.

The design samples and uses the motor phase currents as feedback to the FOC algorithm.

Figure 36. FOC with Position Sensor Feedback



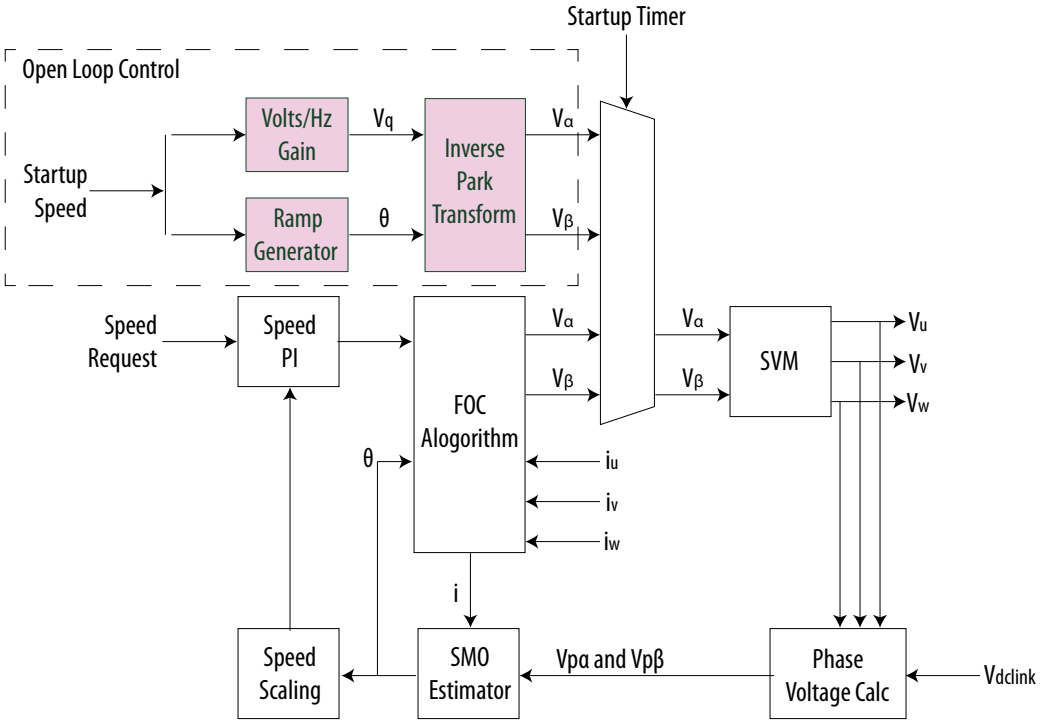
FOC Sensorless

The design supports FOC sensorless control in which the design samples and uses both the motor phase voltages and currents as the feedback to the control loop.

The phase voltage calc block derives signals V_α and V_β, scaled and normalized with respect to the DC bus voltage (V_{dclink}). The DC bus voltage may drop during quick acceleration or rise during regeneration. If you do not expect the bus voltage to change much (e.g. large bus capacitance), you may use V_α and V_β generated from the inverse Clarke transform. The software function `Phase_Volt_Calc_f()` implements the phase voltage calculations using floating-point arithmetic.

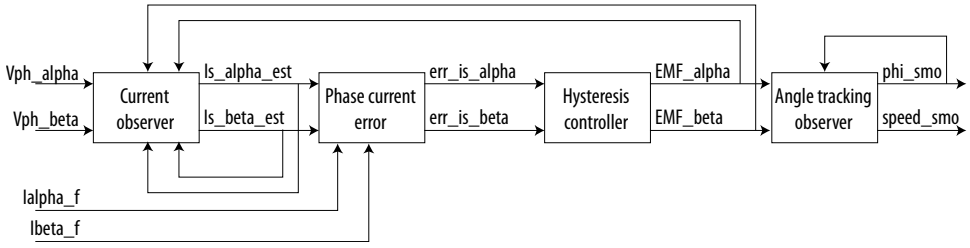
The design integrates the speed estimator with the sliding mode observer (SMO) to allow a second order observer to calculate both estimated angle and estimated speed together. In FOC sensorless mode, the motor starts initially in open loop with a requested speed and switches to sensorless mode after a preset time to allow the SMO to settle. The software function `SMO_Calc_f()` implements the SMO calculations using floating-point arithmetic.

Figure 37. FOC Sensorless Control



Sliding Mode Observer Theory

Figure 38. SMO_Calc_f() software function



The observer, based on the electrical model, estimates the rotating back-EMF vector. The rotor position determines the direction of the back-EMF, which enables estimation of position. At constant speed, back-EMF components are sinusoidal in quadrature, as are error signals. Considering voltage *V* and current *I* in one motor phase with resistance *Rs*, inductance *Ls*. Net applied voltage *V* is at the motor terminal, and the centre of the motor is at 0 V.

$$v = iR_s + \frac{di}{dt}L_s$$

Solvinge the differential equation for *i*, assuming a constant *v* applied over sample time *T*, initial current *i*, and current at the end of *T* *i*+1, gives:

$$i_{k+1} = i_k e^{-\left(\frac{R_s}{L_s}\right)T} + v_k \frac{1}{R_s}\left(1 - e^{-\left(\frac{R_s}{L_s}\right)T}\right)$$

Normalizing by some V_{max} and I_{max} to create variables that are non-dimensional and always <1, gives:

$$\left(\frac{i}{I_{max}}\right)k+1 = \left(\frac{i}{I_{max}}\right)k e^{-\left(\frac{R_s}{L_s}\right)T} + \left(\frac{v}{V_{max}}\right)\frac{V_{max}}{I_{max}R_s}\left(1 - e^{-\left(\frac{R_s}{L_s}\right)T}\right)$$

$$\left(\frac{i}{I_{max}}\right)k+1 = \left(\frac{i}{I_{max}}\right)Aparm + \left(\frac{v}{V_{max}}\right)Bparm$$

$$Aparm = e^{-\left(\frac{R_s}{L_s}\right)T}$$

$$Bparm = \left(1 - Aparm\right)\frac{V_{max}}{I_{max}R_s}$$

Aparm is non-dimensional, represents the electrical system dynamics. Ls/Rs is the electrical time constant, Ts is the discrete sample time. *Aparm* should be close to 1 if the sample time is much faster than the motor time constant.

Bparm uses $V_{max}/(I_{max}*R_s)$ so is also nondimensional. (1-*Aparm*) is small.

Aparm and *Bparm* are constant and are calculated once during initialization of the software.

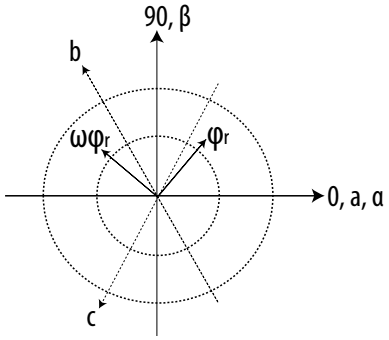
Angle Tracking Observer Theory

The angle tracking observer (part of `SMO_Calc_f()`) takes the estimated back-EMF vector as input and outputs an estimated rotor position (`phi_SMO`). The back-EMF estimate may be noisy so the observer filters it and converts it to position. The design provides two methods:

1. `#if (TRACKER_ENABLE == 0)`, arctan method: A first order filter with gain *Lpf_Gain* is applied to both alpha and beta components of the back-EMF before using arctan to convert them to `phi_SMO`. The advantage is simplicity, but velocity must be estimated separately. The design does not use this method as the speed estimation may introduce lag.
2. Angle observer method: The design combines the back-EMF estimates with the last `phi_SMO` to calculate *AObsError*, which is proportional to *speed**sin(*actual-estimate*). Assuming small errors and constant speed, *AobsError*=*K**(θ_{Elec} - `phi_SMO`). `Phi_SMO` has units of fraction of 1rev (0..1 for 0..2π rad).

Figure 39. Angle Tracking Observer

Induced voltage v_e = d/dt(φ) = Rotor back-EMF in q-direction in FOC. Rotor electrical angle θ_{elec} = direction of magnetic flux vector φ = d-direction in FOC.



The back-EMF operates in the *q*-direction,, which is at right-angles to the d-direction (θelec_rad). The components of the back-EMF in the fixed alpha and beta directions are:

- *vBemfAlpha_V* = -dθmech_dt_rad_s*Ke_Vs_rad*sin(θelec_rad)
- *vBemfBeta_V* = dθmech_dt_rad_s*Ke_Vs_rad*cos(θelec_rad)
- *AobsError* is the back-EMF in the estimated d direction (phi_SMO), which is zero if phi_SMO is equal to θelec_rad
- *AObsError* = -vBemfAlpha_V*cos(phi_SMO)-vBemfBeta_V*sin(phi_SMO)

Substituting the *Bemf* equations into the *AObsError* equation:

AObsError = dθmech_dt_rad_s*Ke_Vs_rad*(sin(θelec_rad)*cos(phi_SMO)-cos(θelec_rad)*sin(phi_SMO))

= dθmech_dt_rad_s*Ke_Vs_rad*sin(θelec_rad - phi_SMO)

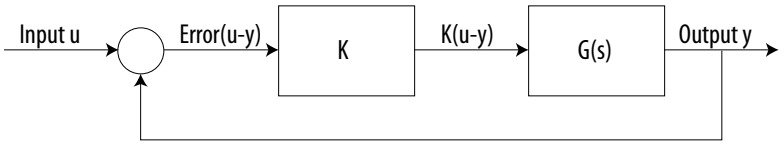
Hence, if the *BemfAlpha* and *BemfBeta* values are correct, *AObsError* measures the angle estimation error (θelec_rad - phi_SMO).

For small error angles and at constant speed, we can assume *AObsError* = K*(θelec - phi_SMO)

AObsError = $K(\theta_k - \varphi_k)$

The software creates a second-order filter from θ_{elec} to phi_SMO. Filter natural frequency Ω and damping coefficient ζ are design parameters tuned for the Tamagawa motors in the Tandem Motion-Power 48 V motor kit. The normalized scaling makes them suitable for other motors too with possible retuning. Carefully tune this non-linear observer because incorrect position estimates can make the FoC algorithm oscillate or become unstable by causing positive feedback.

Figure 40. Feedback System



SMO Parameters

The design derives various SMO parameters from the motor parameters for each motor type, such as resistance and inductance. Other SMO parameters, and default values, are:

Table 14. SMO Parameters

Parameter	Description
Lpf_Gain = 0.10	Arctan method of angle calculation only. The final two stages of the SMO are a low-pass filter on each component of the estimated BEMF followed by an inverse tangent (arctan observer). The output of the inverse tangent is the estimated angle. The parameter Lpf_Gain sets the cutoff frequency of the low-pass filter. Lpf_Gain = 2*pi*fc*Ts

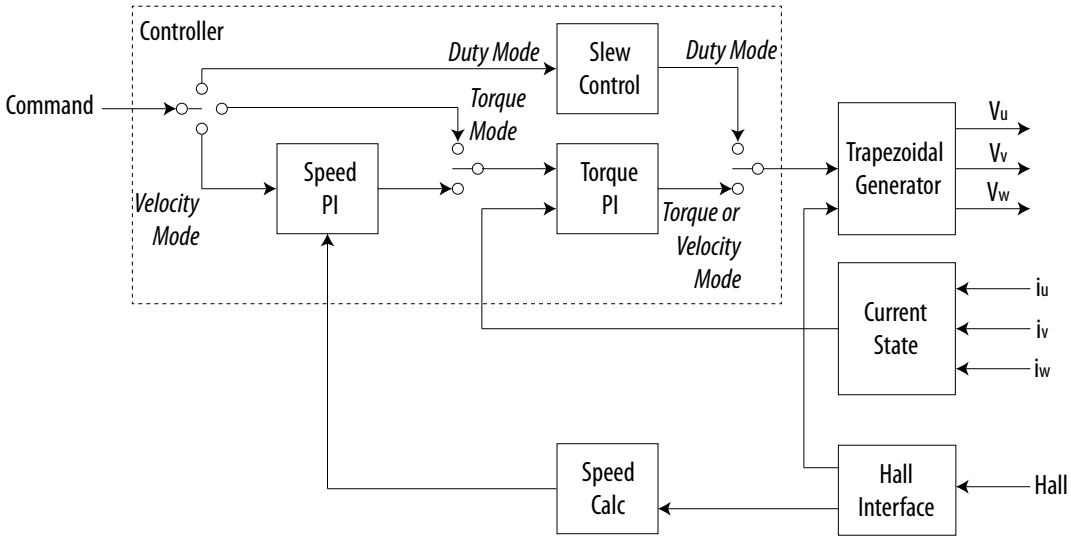
continued...

Parameter	Description
	where: Ts is the sample period and fc is the desired cutoff frequency.
damping_coefficient = 1.4	These are both parameters of the angle tracking observer, which takes in both unfiltered components of the estimated BEMF, extracts the angle and filters in one module. The angle tracking observer has no speed dependent phase lag, unlike the arctan observer. Setting natural_frequency too low can result in instability in the speed estimation.
natural_frequency = 600	
Hys_Gain = 0.8	This parameter sets the sliding mode gain on the current observer. This observer is responsible for estimating the BEMF signals that it ultimately feeds into the angle tracking observer. To adjust this parameter, run the motor and view the estimated angle waveform. If it looks like an undistorted triangle no adjustment should be necessary. If the triangle looks distorted, while running at constant speed, adjust this parameter to clean it up.

Trapezoidal

The design supports trapezoidal control of BLDC motors using Hall sensor feedback on the Tandem Motion-Power 48 V Board. The software supports **Duty Mode** and **Torque Mode**, but the demonstration GUI only uses **Velocity Mode**. The software reconstructs the motor current from the individual phase current readings using the Hall encoder state to determine which phase current is relevant.

Figure 41. Trapezoidal Commutation Using Hall Sensor Feedback



7.11. FOC Subsystem

The Drive-On-Chip Design uses DSP Builder for Intel FPGAs 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 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 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 adjust 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.

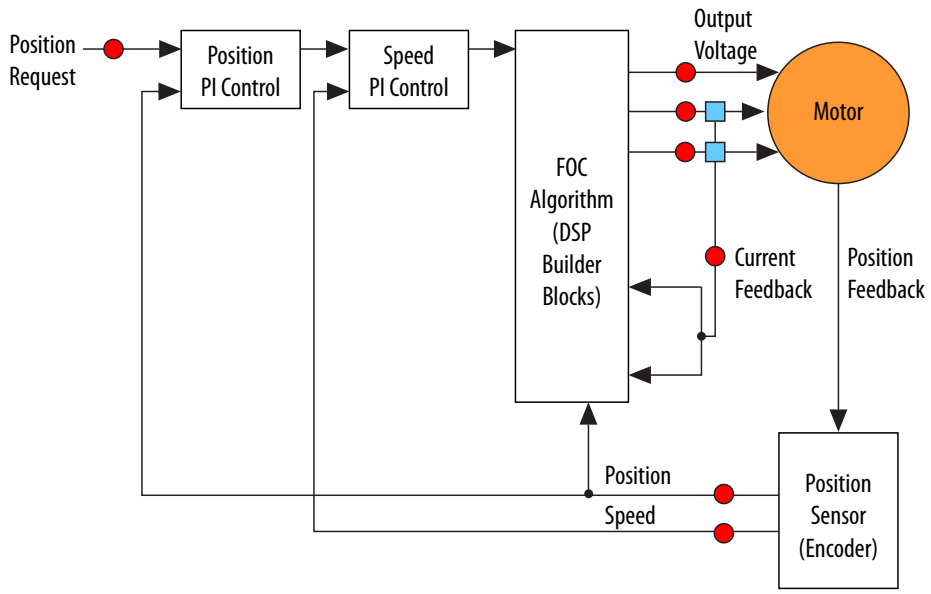
The FOC algorithm includes:

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

7.11.1. DSP Builder for Intel FPGAs Model for the Drive-On-Chip 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 42. DSP Builder for Intel FPGAs 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 for Intel FPGAs implements the latches outside because of limitations of the folding synthesis.

The 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 15. 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 memory-mapped 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 other models refer to:

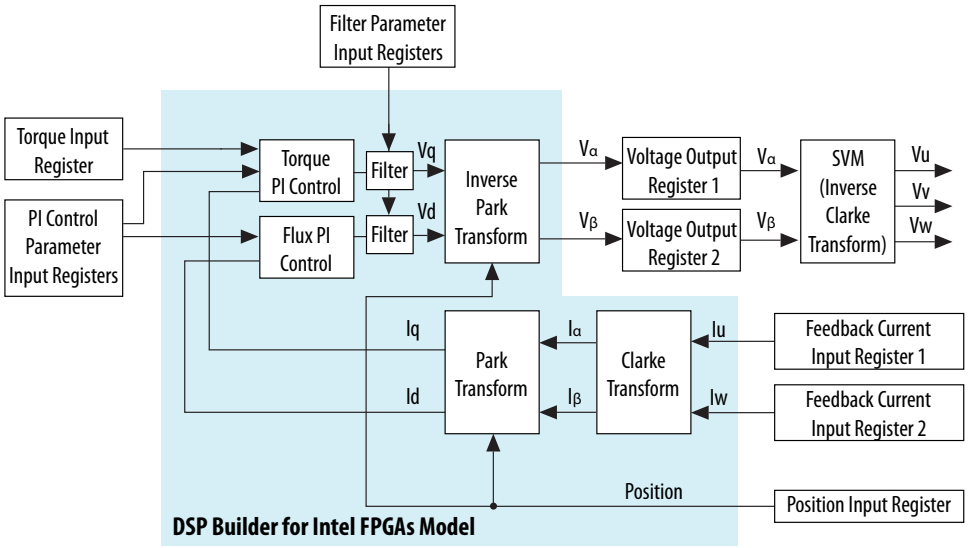
- `foc_blocks.slx`

7.11.2. Avalon Memory-Mapped Interface

The Drive-On-Chip Design DSP Builder for Intel FPGAs-generated VHDL has a signal interface that matches the connections in Simulink. 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 for Intel FPGAs design adds blocks to terminate the parallel inputs and outputs and handshaking logic with an Avalon memory-mapped register map.

Figure 43. FOC Model integrated in Simulink with Avalon-MM Register Map



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

To run the DSP Builder for Intel FPGAs model as part of the drive algorithm, a C function passes the data values between the processor and DSP Builder for Intel FPGAs. 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 for Intel FPGAs calculation. The software then waits for the DSP Builder for Intel FPGAs 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 for Intel FPGAs implementations at runtime.

Related Information

Avalon Interface Specification

7.11.3. About DSP Builder for Intel FPGAs

DSP Builder for Intel FPGAs 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.

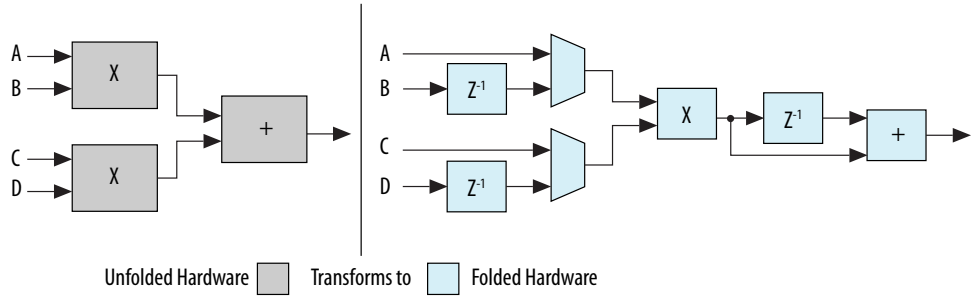
After you develop the algorithm in Simulink, DSP Builder can automatically generate pipelined HDL that it targets and optimizes to the chosen FPGA device. You can use this VHDL in a HDL simulator such as ModelSim* to verify the generated logic versus Simulink and in the Quartus Prime software to compile the hardware. DSP Builder for Intel FPGAs gives instant feedback of the VHDL's logic utilization and algorithm latency in automatically generated Simulink reports.

7.11.4. DSP Builder for Intel FPGAs Folding

DSP Builder for Intel FPGAs generates flat parallel models that can receive and process new input data on every clock pulse. However, designs that 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 for Intel FPGAs folding feature to trade off an increase in algorithm latency for a decrease in the 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 for Intel FPGAs 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.

Figure 44. Unfolded and Folded Hardware Examples



7.11.5. DSP Builder for Intel FPGAs Model Resource Usage

For the Drive-On-Chip Design, Intel compared the FOC algorithm as a single precision floating-point model to 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.

Intel compared floating- and fixed-point versions of the FOC algorithm with and without folding. In addition, Intel 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 for Intel FPGAs 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 16. Resource Usage Comparison for Cyclone V Devices

Design	Folding	Precision	ALMs	DSPs	Latency (us)	M10K
Floating-point	No	32	9968	31	0.99	19
Floating-point	Yes	32	3840	4	1.77	1
Floating-point	No	26	8995	31	0.99	15
Floating-point	Yes	26	3634	4	1.75	3
Fixed-point	No	16	1979	24	0.22	2
Fixed-point	Yes	16	2510	1	1.99	2

Table 17. Resource Usage Comparison for MAX 10 Devices

Design	Folding	Precision	LEs	Multipliers	Latency (us)	M9K
Floating-point	No	32	20010	53	0.74	24
Floating-point	Yes	32	6092	10	1.32	4
Floating-point	No	26	15450	23	0.67	17
Floating-point	Yes	26	4982	6	1.25	1
Fixed-point	No	16	2567	12	0.13	2
Fixed-point	Yes	16	2624	2	1.19	2

The results show:

- 26-bit floating-point precision uses fewer resources because datapaths are narrower and simpler with reduced precision.
- Fixed-point designs use significantly fewer resources than floating-point designs. Typically, implement fixed-point designs if you do not require the high dynamic range that floating-point offers. However, floating-point designs avoid arithmetic overflow during algorithm development and tuning.
- Fixed-point designs can achieve a processing latency down to 0.1 µs, which is ideal for designs that require very high update frequencies.
- Folded designs use significantly fewer resources than designs without folding. Folding increases latency to around 1 µs, which is still acceptable for the control loop.

7.11.6. DSP Builder for Intel FPGAs Design Guidelines

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

In your design:

- Use the variable precision support in DSP Builder for Intel FPGAs. Instead of using a 32-bit datapath, investigate the algorithm and reduce the numeric format as much as possible. Also keep sufficient accuracy to represent the range of allowed data values.
- Use components that use fewer FPGA resources. For example, sin(x) and cos(x) blocks require a range reduction stage. Use the smaller sin(pi*x) and cos(pi*x).
- Use fewer different components to enable resource reuse. For example, restructure a sin(pi*x) and a cos(pi*x) into a sin(pi*x) and sin(pi*(0.5-x)).
- 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.

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

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

7.12. DEKF Technique

In the DEKF technique, the design simultaneously executes two cooperating Kalman filters for nonlinear systems: one to estimate the states and the other to estimate the parameters.

The dual Kalman filter approach reduces the computation requirements compared to considering parameter changes as additional states in a single larger Kalman filter.

Equation 1. Parameter Evolution

This equation describes the parameter evolution that with the measurement equation builds the first EKF.

$$p(k+1) = p(k) + \chi(k)$$

Equation 2. State of Evolution

This equation represents the state evolution that combines with the measurement equation to form the second EKF.

$$x(k+1) = F(x(k), i_L(k), p(k)) + \xi(k)$$

Equation 3. Measurement Equation

$$v_T(k) = G(x(k), i_L(k), p(k)) + \psi(k)$$

The measurement equation is the same for both filters. In the above equations:

- *vT(k)* is the vector of measurements at time k
- *k* is the discrete time
- *iL(k)* is the load current at time k
- *p* is parameters vector
- x = [SOC; V_{RC1}] is the battery state vector
- χ, ξ and ψ are the parameters, the state and measurement noise, with zero mean and covariance matrix Σχ, Σξ and Σψ, respectively.

Equation 4. Circuit Equations

The circuit equation describes the actual circuit. OCV(SOC) is the open circuit voltage - the voltage that is measured externally on the battery. It is related to the internal state of charge by the empirical polynomial equation with fixed parameters P1 to P8.

$$x_k = f\left(SOC_k, V_{RC_k}, i_{L_k}\right) = \begin{bmatrix} SOC_k \\ V_{RC_k} \end{bmatrix} = \begin{bmatrix} SOC_{k-1} - \frac{T}{Q_r} i_{L_k} \\ v_{RC_{k-1}} e^{-T/\tau} + R\left(1 - e^{-T/\tau}\right) i_{L_k} \end{bmatrix}$$

$$v_{T_k} = g\left(SOC_k, V_{RC_k}, i_{L_k}\right) = OCV(SOC_k) - R_0 i_{L_k} - v_{RC_k}$$

$$OCV(SOC) = P_1 SOC^7 + P_2 SOC^6 + P_3 SOC^5 + P_4 SOC^4 + P_5 SOC^3 + P_6 SOC^2 + P_7 SOC^1 + P_8$$

Equation 5. DEKF Matrix Equations

The matrix equations are derived from the circuit equations.

$$x_k = \begin{bmatrix} SOC_k \\ V_{RC_k} \end{bmatrix}$$

$$q_k = \begin{bmatrix} SOC_k \\ 1/\tau \\ R_k \end{bmatrix}$$

$$A_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{-T/\tau} \end{bmatrix}$$

$$C_{x_k} = \left[\frac{dOCV(SOC)}{dSOC} - 1 \right]$$

$$C_{q_k} = \left[i_{L_k} \; 0 \; 0 \right] + C_{x_k} \frac{dx_k^-}{dq}$$

$$\frac{dx_k^-}{dq} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-T/\tau} \end{bmatrix} \frac{dx_{k-1}^+}{dq} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & Te^{-\frac{T}{\tau}\left(Ri_{L_k} - V_{RC_{k-1}}\right)} & \left(1 - e^{-T/\tau} i_{L_k}\right) \end{bmatrix}$$

$$\frac{dx_{k-1}^+}{dq} = \frac{dx_k^-}{dq} - L_{x_{k-1}} C_{q_{k-1}}$$

The following equations are standard Kalman filter equations, which you can calculate after you define the standard Kalman filter variables in terms of the specific variables of the battery model.

Equation 6. Initialization Equation

$$x_0, P_0, q_0, P_{q0}$$

Equation 7. Prediction Equations

$$q_k^- = q_{k-1}^+$$

$$P_{q_k}^- = P_{q_{k-1}}^+ + Q_q$$

$$x_k^- = f\left(x_{k-1}^+, u_{k-1}, q_{k-1}^+\right)$$

$$P_k^- = A_k P_{k-1}^+ A_k^T + Q$$

Equation 8. Correction Equation

$$L_{x_k} = P_k^- C_{x_k}^T \left(C_{x_k} P_k^- C_{x_k}^T + R\right)^{-1}$$

$$x_k^+ = x_k^- + L_{x_k}\left(y_k - g\left(x_k^-, u_k, q_{k-1}^+\right)\right)$$

$$P_k^- = \left(I - L_{x_k} C_{x_k}\right) P_k^-$$

$$L_{q_k} = P_{q_k}^- C_{q_k}^T \left(C_{q_k} P_{q_k}^- C_{q_k}^T + R\right)^{-1}$$

$$q_k^+ = q_k^+ + L_{q_k}\left(y_k - g\left(x_k^-, u_k, q_{k-1}^+\right)\right)$$

$$P_{q_k}^- = \left(I - L_{q_k} C_{q_k}\right) P_{q_k}^-$$

Related Information

R. Morello et al., "Comparison of state and parameter estimators for electric vehicle batteries," Industrial Electronics Society, IECON 2015 - 41st Annual Conference of the IEEE, Yokohama, 2015, pp. 005433-005438

7.13. Signals

The signals connect various blocks in the Drive-On-Chip Design Example for Intel MAX 10 devices.

Table 18. Six-Channel PWM Interface Signals

Signal Name	Direction	Description
Avalon-MM Interface Signals		
clk	Input	PWM and system clock input
reset_n	Input	System reset signal, active low
avs_read_n	Input	Avalon-MM read strobe, active low
avs_write_n	Input	Avalon-MM write strobe, active low
avs_address[3:0]	Input	Avalon-MM address bus
avs_writedata[31:0]	Input	Avalon-MM write data bus
avs_readdata[31:0]	Output	Avalon-MM read data bus
Conduit Signals		
pwm_enable	Input	PWM enable from drive system monitor
en_upper	Input	Upper switch enable from drive system monitor
en_lower	Input	Lower switch enable from drive system monitor
u_h	Output	Motor phase phase U upper gate drive
u_l	Output	Motor phase phase U lower gate drive
v_h	Output	Motor phase phase V upper gate drive
v_l	Output	Motor phase phase V lower gate drive

continued...

Signal Name	Direction	Description
w_h	Output	Motor phase phase W upper gate drive
w_l	Output	Motor phase phase W lower gate drive
sync_in	Input	Synchronization signal for multiple PWM modules
sync_out	Output	Synchronization signal for multiple PWM modules
start_adc	Output	ADC start conversion signal

Table 19. DC Link Monitor Signals

Signal Name	Direction	Description
Avalon-MM Interface Signals		
clk	Input	FPGA system clock input
clk_adc	Input	ADC clock input
reset_n	Input	System reset signal, active low
avs_read_n	Input	Avalon-MM read strobe, active low
avs_write_n	Input	Avalon-MM write strobe, active low
avs_address[3:0]	Input	Avalon-MM address bus
avs_writedata[31:0]	Input	Avalon-MM write data bus
avs_readdata[31:0]	Output	Avalon-MM read data bus
avs_irq	Output	Avalon interrupt
Conduit Signals		
sync_dat	Input	Sigma-delta ADC bit stream
dc_link_enable	Input	Enable
overvoltage	Input	Overvoltage status
undervoltage	Output	Undervoltage status
chopper	Output	Chopper circuit gate drive

Table 20. Drive System Monitor Interface Signals

Signal Name	Direction	Description
Avalon-MM Interface Signals		
clk	Input	FPGA system clock input
reset_n	Input	System reset signal, active low
avs_read_n	Input	Avalon-MM read strobe, active low
avs_write_n	Input	Avalon-MM write strobe, active low
avs_address[3:0]	Input	Avalon-MM address bus
avs_writedata[31:0]	Input	Avalon-MM write data bus
avs_readdata[31:0]	Output	Avalon-MM read data bus
Conduit Signals		
overcurrent	Input	Overcurrent status

continued...

Signal Name	Direction	Description
overvoltage	Input	Overvoltage status
undervoltage	Input	Undervoltage status
chopper	Input	Chopper status
dc_link_clk_err	Input	Clock monitor status
igbt_err	Input	IGBT error status
error_out	Output	Error output
overcurrent_latch	Output	Latched overcurrent status
overvoltage_latch	Output	Latched overvoltage status
undervoltage_latch	Output	Latched undervoltage status
dc_link_clk_err_latch	Output	Latched clock monitor status
igbt_err_latch	Output	Latched IGBT error status
chopper_latch	Output	Latched chopper status
pwm_control[2:0]	Output	PWM control

Table 21. Quadrature Encoder Interface Signals

Signal Name	Direction	Description
Avalon-MM Interface Signals		
clk	Input	FPGA system clock input
reset_n	Input	System reset signal, active low
avs_read_n	Input	Avalon-MM read strobe, active low
avs_write_n	Input	Avalon-MM write strobe, active low
avs_address[3:0]	Input	Avalon-MM address bus
avs_writedata[31:0]	Input	Avalon-MM write data bus
avs_readdata[31:0]	Output	Avalon-MM read data bus
Conduit Signals		
strobe	Input	Capture strobe
QEP_A	Input	Quadrature phase A
QEP_B	Input	Quadrature phase B
QEP_I	Input	Quadrature index

Table 22. Sigma-Delta ADC Interface Signals

Signal Name	Direction	Description
Avalon-MM Interface Signals		
clk	Input	FPGA system clock input
clk_adc	Input	ADC clock input
reset_n	Input	System reset signal, active low
avs_read_n	Input	Avalon-MM read strobe, active low

continued...

Signal Name	Direction	Description
avs_write_n	Input	Avalon-MM write strobe, active low
avs_address[3:0]	Input	Avalon-MM address bus
avs_writedata[31:0]	Input	Avalon-MM write data bus
avs_readdata[31:0]	Output	Avalon-MM read data bus
avs_irq	Output	Interrupt request
Conduit Signals		
start	Input	Start conversion signal
sync_dat_u	Input	Phase U sigma-delta bitstream
sync_dat_v	Input	Phase V sigma-delta bitstream
sync_dat_w	Input	Phase W sigma-delta bitstream
overcurrent	Output	Overcurrent status

Table 23. MAX 10 ADC Threshold Sink Interface Signals

Signal Name	Direction	Description
Avalon-MM Interface Signals		
clk	Input	FPGA system clock input
reset_n	Input	System reset signal, active low
avs_read_n	Input	Avalon-MM read strobe, active low
avs_write_n	Input	Avalon-MM write strobe, active low
avs_address[3:0]	Input	Avalon-MM address bus
avs_writedata[31:0]	Input	Avalon-MM write data bus
avs_readdata[31:0]	Output	Avalon-MM read data bus
Avalon-ST Sink Interface Signals		
st_1_valid	Input	ADC 1 threshold valid
st_1_channel[4:0]	Input	ADC 1 threshold channel index
st_1_data	Input	ADC 1 threshold data
st_2_valid	Input	ADC 2 threshold valid
st_2_channel[4:0]	Input	ADC 2 threshold channel index
st_2_data	Input	ADC 2 threshold data
Conduit Signals		
under[15:0]	Output	Under threshold errors
over[15:0]	Output	Over threshold errors

Table 24. DC-DC Converter Interface Signals

Signal Name	Direction	Description
Avalon memory-mapped interface signals		
avs_clk	Input	10MHz clock input

continued...

Signal Name	Direction	Description
reset_n	Input	System reset signal, active low
avs_read_n	Input	Avalon-MM read strobe, active low
avs_write_n	Input	Avalon-MM write strobe, active low
avs_address[4:0]	Input	Avalon-MM address bus
avs_writedata[31:0]	Input	Avalon-MM read data bus
avs_readdata[31:0]	Output	Avalon-MM write data bus
Conduit Signals		
enable_in	Input	Enable input
bidir_en_n	Input	Bidirectional conversion enable
fault	Input	Fault input. If the design asserts the fault input, it clears the enable bit of the control register, and turns off the DC-DC converter. The design keeps the enable bit clear, and does not set again, until the fault input is negated.
pwm_sync_n	Input	Synchronization signal
gate_a_h	Output	Phase 0 upper transistor gate drive
gate_a_l	Output	Phase 0 lower transistor gate drive
gate_b_h	Output	Phase 1 upper transistor gate drive
gate_b_l	Output	Phase 1 lower transistor gate drive
dc_dc_on	Output	DC-DC status
overvoltage	Output	Overvoltage error
overcurrent	Output	Overcurrent error
timeout_latch	Output	Sample timeout
sync_dat_i_pase_a	Input	Phase 0 current feedback sigma-delta bitstream
sync_dat_i_pase_b	Input	Phase 1 current feedback sigma-delta bitstream
sync_dat_v_out	Input	Voltage feedback sigma-delta bitstream
clk_adc	Input	ADC clock input

7.14. Registers

The Drive-on-Chip Design Example for Intel MAX 10 devices contains many registers that you can set.

Table 25. Six-Channel PWM Interface Control and Status Registers

Write reserved bits as zero and read as zero.

Address	Name	Bits	Description	Reset Value	Access
0x00	-	-	Reserved	-	-
0x04	pwm_u	[31:15]	Reserved	-	-
		[14:0]	phase U PWM switching threshold in PWM clocks	0x0	RW
0x08	pwm_v	[31:15]	Reserved	-	-

continued...

Address	Name	Bits	Description	Reset Value	Access
		[14:0]	phase V PWM switching threshold threshold in PWM clocks	0x0	RW
0x0C	pwm_w	[31:15]	Reserved	-	-
		[14:0]	phase W PWM switching threshold threshold in PWM clocks	0x0	RW
0x10	max	[31:15]	Reserved	-	-
		[14:0]	PWM maximum count threshold in PWM clocks	0x0	RW
0x14	block	[31:8]	Reserved	-	-
		[7:0]	PWM blocking (dead time) register threshold in PWM clocks	0x0	RW
0x18	trigger_up	[31:15]	Reserved	-	-
		[14:0]	PWM up count trigger for ADC threshold in PWM clocks	0x0	RW
0x1C	trigger_down	[31:15]	Reserved	-	-
		[14:0]	PWM down count trigger for ADC threshold in PWM clocks	0x0	RW
0x20	gate	[31:6]	Reserved	-	-
		[5]	Phase U lower transistor gate signal	0x0	R
		[4]	Phase U upper transistor gate signal	0x0	R
		[3]	Phase V lower transistor gate signal	0x0	R
		[2]	Phase V upper transistor gate signal	0x0	R
		[1]	Phase W lower transistor gate signal	0x0	R
		[0]	Phase W upper transistor gate signal	0x0	R
0x24	carrier	[31:16]	Reserved	-	-
		[15:0]	PWM count value threshold in PWM clocks	0x0	R
0x28	multi_cycle	[31:4]	Reserved	-	-
		[3:0]	Cycles to skip for ADC sample strobes	0x0	RW

Table 26. DC Link Monitor Interface Control and Status Registers

Write reserved bits as zero and read as zero.

Address	Name	Bits	Description	Reset Value	Access
0x00	-	-	Reserved	-	-
0x04	offset	[31:16]	Reserved	-	-
		[15:0]	Offset. A value of 16384 corresponds to a zero offset.	0x0	RW
0x08	k_64	[31:1]	Reserved	-	-
		[0]	sinc3 filter decimation rate. When set to 0, the sinc3 decimation rate is M=64; when set to 1, the sinc3 decimation rate is M=128.	0x0	RW
0x0C	ref_disable	[31:16]	Reserved	-	-
		[15:0]	DC-link voltage disable level. This register provides the maximum allowable voltage for link voltage. If the maximum value is exceeded the overvoltage output is driven, to shut down the system.	0x0	RW

continued...

Address	Name	Bits	Description	Reset Value	Access
0x10	link_ref	[31:16]	Reserved	-	-
		[15:0]	DC-link chopper voltage level. The chopper IGBT transistor is turned on when the DC-link voltage exceeds this value.	0x0	RW
0x14	bottom_ref	[31:16]	Reserved	-	-
		[15:0]	DC-link undervoltage reference level. If the link voltage falls below the reference level the undervoltage output is driven.	0x0	RW
0x18	brake_t	[31:11]	Reserved	-	-
		[10:0]	This register is not used.	0x0	RW
0x1C	brake_max_level	[31:16]	Reserved	-	-
		[15:0]	This register is not used.	0x0	RW
0x20	dc_link	[31:16]	Reserved	-	-
		[15:0]	Current link voltage reading	0x0	R
0x24	brake_level	[31:16]	Reserved	-	-
		[15:0]	This register is not used.	0x0	R
0x28	status	[31:3]	Reserved	-	-
		[2]	DC link overvoltage status	0x0	R
		[1]	DC link undervoltage status	0x0	R
		[0]	Chopper gate signal status	0x0	R

Table 27. Drive System Monitor Control and Status Registers

Write reserved bits as zero and read as zero. R/W1C bits are read, write a 1 to clear the bit

Address	Name	Bits	Description	Reset Value	Access
0x00	control	[31:3]	Reserved	-	-
		[2:0]	Control. Write to this register to request a change of state in the drive system monitor.	0x0	RW
0x04	status	[31:12]	Reserved	-	-
		[11:9]	Current DSM state.	0x0	R
		[8]	PWM control, upper PWM enable	-	-
		[7]	PWM control, lower PWM enable	0x0	R
		[6]	PWM control, PWM enable	-	-
		[4]	IGBT error	0x0	R/W1C
		[3]	ADC clock status	-	R/W1C
		[2]	Undervoltage status	0x0	R/W1C
		[1]	Overvoltage status	-	R/W1C
		[0]	Overcurrent status	0x0	R/W1C

Table 28. Quadrature Encoder Interface Control and Status Registers

Write reserved bits as zero and read as zero.

Address	Name	Bits	Description	Reset Value	Access
0x00	control	[31:3]	Reserved.	-	-
		[2]	direction bit. Reverses the count direction when set.	0x0	RW
		[1]	index_reset_en bit. Count will reset on index pulse if this bit is set.	0x0	RW
		[0]	index_capture_en bit. Count will be captured in index capture reg, when index pulse occurs, if this bit is set.	0x0	RW
0x04	count capture	[31:0]	Captures current count on each strobe.	0x0	R
0x08	maximum count	[31:0]	Maximum count. Count will reset to zero when it reaches this value.	0x3FFF	RW
0x0C	count	[31:0]	Current count value.	0x0	RW
0x10	index capture	[31:0]	Captures current count when index pulse occurs if index_capture_en bit is set.	0x0	R

Table 29. Sigma-Delta ADC Interface Control and Status Registers

Write reserved bits as zero and read as zero.

Address	Name	Bits	Description	Reset Value	Access
0x0	-	-	Reserved	-	-
0x04	offset_u	[31:16]	Reserved.	-	-
		[15:0]	Offset for phase U. A value of 32,768 corresponds to 0 offset.	0x0	RW
0x08	offset_w	[31:16]	Reserved.	-	-
		[15:0]	Offset for phase W. A value of 32,768 corresponds to 0 offset.	0x0	RW
0x0C	i_peak	[31:10]	Reserved.	-	-
		[9:0]	Overcurrent detection threshold.	0x0	RW
0x10	d	[31:3]	Reserved.	-	-
		[2]	sinc3 filter decimation rate. When set to 0, the sinc3 decimation rate is M=128 for the control loop and M=16 for overcurrent detection; when set to 1, the sinc3 decimation rate is M=64 for the control loop and M=8 for the overcurrent detection.	0x0	RW
		[1]	Overcurrent enable	0x0	RW
		[0]	Overvoltage enable	0x0	RW
0x14	irq_ack	[31:1]	Reserved.	-	-
		[0]		0x0	W1C
0x18	status	[31:5]	Reserved.	-	-
		[4]		0x0	R
		[3]		0x0	R
		[2]	Overcurrent for phase U	0x0	R
		[1]	Overcurrent for phase W	0x0	R

continued...

Address	Name	Bits	Description	Reset Value	Access
		[0]	Overcurrent for any phase	0x0	R
0x1C	i_u	[31:10]	Reserved.	-	-
		[9:0]	Current in phase U.	0x0	R
0x20	i_w	[31:10]	Reserved.	-	-
		[9:0]	Current in phase W.	0x0	R
0x24	i_peak	[31:10]	Reserved.	-	-
		[9:0]	Overcurrent detection threshold.	0x0	RW
0x28	i_v	[31:10]	Reserved.	-	-
		[9:0]	Current in phase V.	0x0	R
0x2C	offset_v	[31:16]	Reserved.	-	-
		[15:0]	Offset for phase V. A value of 32,768 corresponds to 0 offset.	0x0	RW
0x2C	Overcurrent_u	[31:10]	Reserved.	-	-
		[9:0]	Overcurrent value for phase U	0x0	R
0x2C	Overcurrent_v	[31:10]	Reserved.	-	-
		[9:0]	Overcurrent value for phase V	0x0	R
0x2C	Overcurrent_w	[31:10]	Reserved.	-	-
		[9:0]	Overcurrent value for phase W	0x0	R

Table 30. MAX10 ADC Threshold Sink Control and Status Registers

Write reserved bits as zero and read as zero

Address	Name	Bits	Description	Reset Value	Access
0x00	capture under enable	[31:16]	Reserved.	-	-
		[15:0]	Enable latching of under threshold errors. One bit per ADC channel.	0	RW
0x04	capture over enable	[31:16]	Reserved.	-	-
		[15:0]	Enable latching of over threshold errors. One bit per ADC channel.	0	RW
0x08	output under enable	[31:16]	Reserved.	-	-
		[15:0]	Enable output of under threshold errors. One bit per ADC channel.	0	RW
0x0C	output over enable	[31:16]	Reserved.	-	-
		[15:0]	Enable output of over threshold errors. One bit per ADC channel.	0	RW
0x10	latch under	[31:16]	Reserved.	-	-
		[15:0]	Latched under threshold errors. One bit per ADC channel.	0	R
0x14	latch over	[31:16]	Reserved.	-	-
		[15:0]	Latched over threshold errors. One bit per ADC channel.	0	R
0x18	output under	[31:16]	Reserved.	-	-

continued...

Address	Name	Bits	Description	Reset Value	Access
		[15:0]	Under threshold output status. One bit per ADC channel.	0	R
0x1C	output over	[31:16]	Reserved.	-	-
		[15:0]	Over threshold output status. One bit per ADC channel.	0	R
0x20	set under error	[31:16]	Reserved.	-	-
		[15:0]	Set under threshold errors. One bit per ADC channel. Write 1s to set error bits.	0	W1S
0x24	set over error	[31:16]	Reserved.	-	-
		[15:0]	Set over threshold errors. One bit per ADC channel. Write 1s to set an error bits.	0	W1S
0x28	clear under error	[31:16]	Reserved.	-	-
		[15:0]	Clear under threshold errors. One bit per ADC channel. Write 1s to clear error bits.	0	W1C
0x2C	clear over error	[31:16]	Reserved.	-	-
		[15:0]	Clear over threshold errors. One bit per ADC channel. Write 1s to clear error bits.	0	W1C

Table 31. DC-DC Converter Control and Status Registers

Write reserved bits as zero and read as zero

Address	Name	Bits	Description	Reset Value	Access
0x00	control	[31:5]	Reserved	-	-
		[4]	State of REGEN_EN signal from the power board	-	R
		[3]	Reserved	-	-
		[2]	Enable regeneration	0	RW
		[1]	Enable closed loop mode	0	RW
		[0]	Enable Dc-DC gated with enable_in input	0	RW
0x04	cmd_dc	[31:14]	Reserved	-	-
		[13:0]	Commanded DC-DC output level in 1V increments	0	RW
0x08	fault_reg	[31:6]	Reserved	-	-
		[5]	Input overvoltage detected	0	RW
		[4]	Input undervoltage detected	0	RW
		[3]	Output overvoltage detected	0	RW
		[2]	Output undervoltage detected	0	RW
		[1]	Input overcurrent detected	0	RW
		[0]	Output overcurrent detected	0	RW
0x0C	-	-	Reserved	0	-
0x10	duty	[31:14]	Reserved	-	-
		[13:0]	Duty cycle for open loop mode, 0 – 100	50	RW
0x14	freq	[31:14]	Reserved	-	-

continued...

Address	Name	Bits	Description	Reset Value	Access
		[13:0]	Frequency of operation, kHz	64	RW
0x18	-	-	Reserved	-	-
0x1C	-	-	Reserved	-	-
0x20	fb_current_a	[31:13]	Reserved	-	-
		[12:0]	Phase 0 current feedback sample, 100 = 1A	0	RW
0x24	fb_current_b	[31:13]	Reserved	-	-
		[12:0]	Phase 1current feedback sample, 100 = 1A	0	RW
0x28	fb_voltage	[31:13]	Reserved	-	-
		[12:0]	DC-DC output voltage feedback sample, 40 = 1V	0	RW
0x2C	-	-	Reserved	0	-
0x30	offset_fb_current_a	[31:16]	Reserved	-	-
		[15:0]	Phase 0 current feedback ADC offset	0x8000	RW
0x34	offset_fb_current_b	[31:16]	Reserved	-	-
		[15:0]	Phase 1 current feedback ADC offset	0x8000	RW
0x38	offset_fb_voltage	[31:16]	Reserved	-	-
		[15:0]	DC-DC output voltage feedback ADC offset	0x8000	RW
0x3C	-	-	Reserved	0	-
0x40	pgain_voltage	[31:14]	Reserved	-	-
		[13:0]	P gain coefficient for voltage control loop * 100 [AC TODO] resolution? Scale?	300	RW
0x44	igain_voltage	[31:14]	Reserved	-	-
		[13:0]	I gain coefficient for voltage control loop * 1e-7 (1/ avs_clk)	4000	RW
0x48	pgain_current	[31:14]	Reserved	-	-
		[13:0]	P gain coefficient for current control loop * 1000	20	RW
0x4C	igain_current	[31:14]	Reserved	-	-
		[13:0]	I gain coefficient for current control loop * V	25	RW

8. Achieving Timing Closure on a Motor Control Design

Intel Quartus Prime may not achieve full timing closure when it first compiles and fits this designs. The design requires around 90% of the Intel MAX 10 device resources and uses some IP blocks with high clock frequencies, especially the PWM block at 300 MHz and the DDR3 SDRAM IP.

Intel Quartus Prime assume worst-case timing parameters over a wide range of temperature, which is good practice for a commercial design. If you run this design at room temperature, it is unlikely that real timing violations occur. To achieve full timing closure with Intel Quartus Prime, you may include additional pipeline registers. However, the place-and-route process is sensitive to an initial seed value and the result of different seeds is not easy to predict. Before optimizing the design, try the seed sweep function with many different seed values in case the design immediately fits.

1. In Intel Quartus Prime, select **Tools ➤ Launch Design Space Explorer II**.

   Design space Explorer opens in a seaparate window.
2. For a basic seed sweep use the following settings:
   a. In **Setup**, select **Local**.
   b. In **Exploration**, select **Design exploration, exploration mode: Seed Sweep Only, create 10 seeds**.
   c. Click **Start** to run.

Figure 45. Exploration window

Use **Results** to view when all are finished

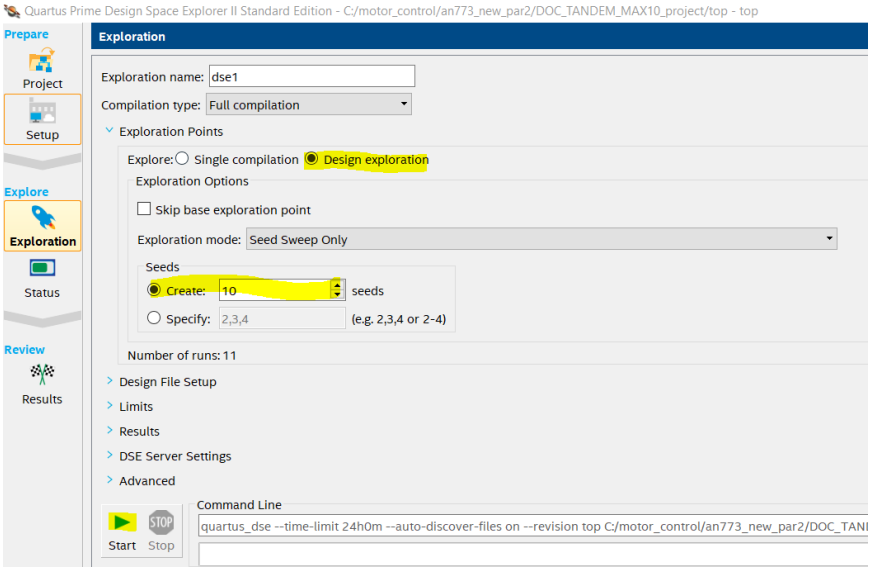
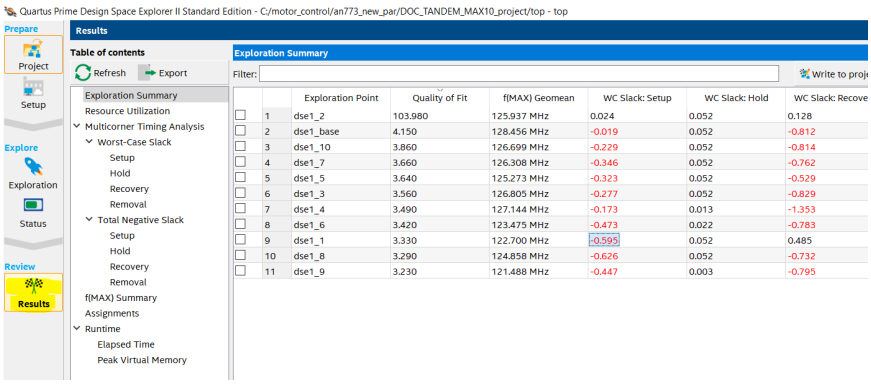


Figure 46. Result window



d. Calculate the seed number from the exploration point name. It is the *_number* plus 1. In this example, the best seed will be 2(dse1_2) + 1 = 3.

3. Select **Assignments ➤ Settings**.
4. In complier settings, click **Advanced settings (Fitter)...**.
5. Update seed value **Fitter Initial Placement Seed 3**.

8.1. Optimizing Motor Control Designs

1. In Intel Quartus Prime, **Assignments ➤ Settings**.
2. In complier settings, change the default **Optimization mode** from **Balanced (Normal flow)**, to either **Performance (High effort – increases run time)** or **Performance (High effort – increases run time)**. If the design density is already very high, then use Performance (High effort – increases run time) because Performance (High effort – increases run time) may cause congestion.
3. In complier settings, click **Advanced settings (Fitter)...**and change these settings to improve timing.
   - Change **Router Effort Multiplier** from 2.0 to 4.0
   - Turn on **Perform Clocking Topology Analysis During Routing**
   - Turn on **Eco Optimize Timing**
   - Turn on **Final Placement Optimization Always Physical Synthesis Register Duplication**
   - Turn on **Physical Synthesis Asynchronous Signal Pipelining**

9. Design Security Recommendations

Follow these recommendations if you use the Drive-On-Chip Design Example for Intel MAX 10 Devices as the basis for a commercial product:

- Add more checks on current, voltage, and gain values in the design to ensure all the parameters are in the correct range.
- Analyze the security risk of incorrect use of the JTAG interface while implementing the design. Incorrect use of the JTAG interface can cause system malfunction and damage the power electronics, motors, or machinery that the design drives. It might even create a hazard to people working with the design.
- Ensure you follow guidelines for secure use of JTAG to protect the system from any unauthorized write or read accesses.

10. Reference Documents for the Drive-on-Chip Design Example for Intel MAX 10 Devices

Related Information

- White Paper: Motor Control Designs with an Integrated FPGA Design Flow
- DC-DC Converter Reference Design
- Tandem Motion-Power 48 V Board Reference Manual
- Battery Management System Reference Design
  The Battery Management System (BMS) Reference Design demonstrates battery state of charge (SOC) estimation in an FPGA-based real-time control platform that you can extend to include other BMS functionality such as battery state-of-health monitoring and charge equalization (cell balancing).
- AN730: Nios II processor booting methods in MAX 10 devices
- ALU Folding

11. Document Revision History for AN 773: Drive-on-Chip Design Example for Intel MAX 10 Devices

Version	Changes
2021.04.22	• Changed ADC scaling value. • Changed signal scaling architecture figure. • Deleted *Scaling for DC-DC Converter Feedback Samples* tables • Changed *DC Link Monitor* • Updated *DC-DC Converter* • Added *DC-DC Control Simulink Models* • Added *Sigma-Delta ADC Interface for DC-DC Converter* • Updated *DC-DC Converter Control and Status Registers*
2021.02.15	Changed PWM frequency from 333 MHz to 300 MHz
2021.01.30	• Removed FalconEye 2 HSMC board. • Renamed to design example. • Added *Achieving Timing Closure on Motor Control Designs* • removed *EnDat Encoder Interface* and *BiSS Encoder Interface*
2017.11.24	• Added support for estimating battery state of charge (SOC). • Changed name from *Drive-on-Chip Reference Design v16.0* to *Drive-on-Chip Reference Design for MAX 10 Devices*. • Rebranded as Intel.
2017.01.10	Initial release.