AN 973: Three-phase Boost Bidirectional AC/DC Converter for Electric Vehicle (EV) Charging

Updated for Intel® Quartus® Prime Design Suite: **17.0**

Contents

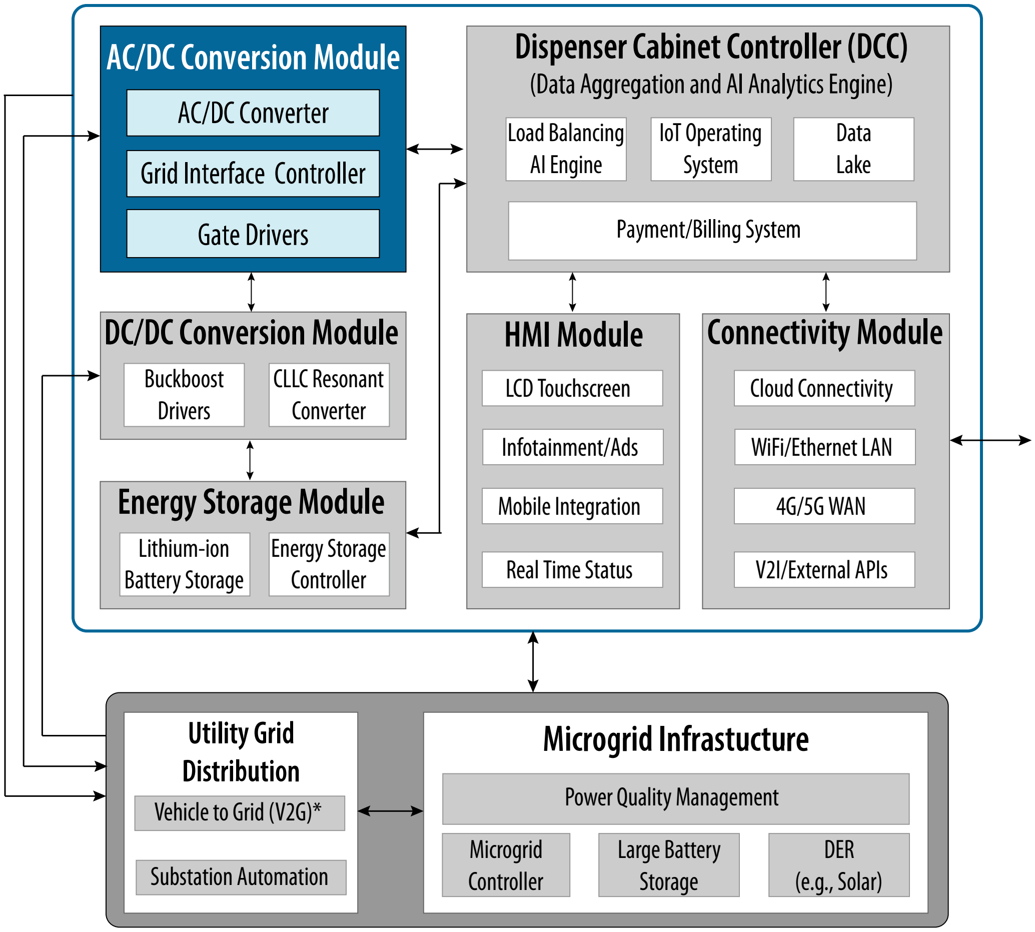
1. Three-phase Boost Bidirectional AC/DC Converter for Electric Vehicle (EV) Charging Design Example Overview

As transportation vehicles are electrified, attention switches from fuel consumption to electrical energy consumption and the efficiency and cost of power converters. Fixed-function chips (ASICs) or microcontrollers with limited control-update rates control several power converters.

FPGAs are unique in enabling custom digital control at very high frequencies. They are beneficial in reducing the size and cost of passive components required to stabilize voltage or change from traditional Pulse Width Modulation (PWM) switching waveforms to other methods to reduce switching losses. This may be an advantage in heavily used high-voltage electric vehicle charging stations.

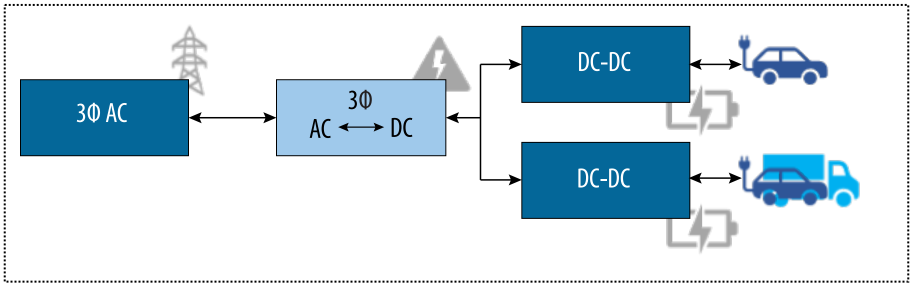
At a high level, the *Three-phase Boost Bidirectional AC/DC Converter for Electric Vehicle (EV) Charging* design example shows an EV charging station's small but indispensable building block, usually called the AC/DC conversion module. In general, an EV charging station is composed of different blocks, as follows:

Figure 1. AC/DC Conversion Module Within the EV Charger Infrastructure



As shown in Common EV Charger Power Architecture diagram, in EV charging power stations, an AC/DC power converter unit can feed multiple DC-DC power converter units that ultimately charge the electric cars. Many EV charging manufacturers use this topology as *modules*, which you can buy and connect to as you desire.

Figure 2. Common EV Charger Power Architecture



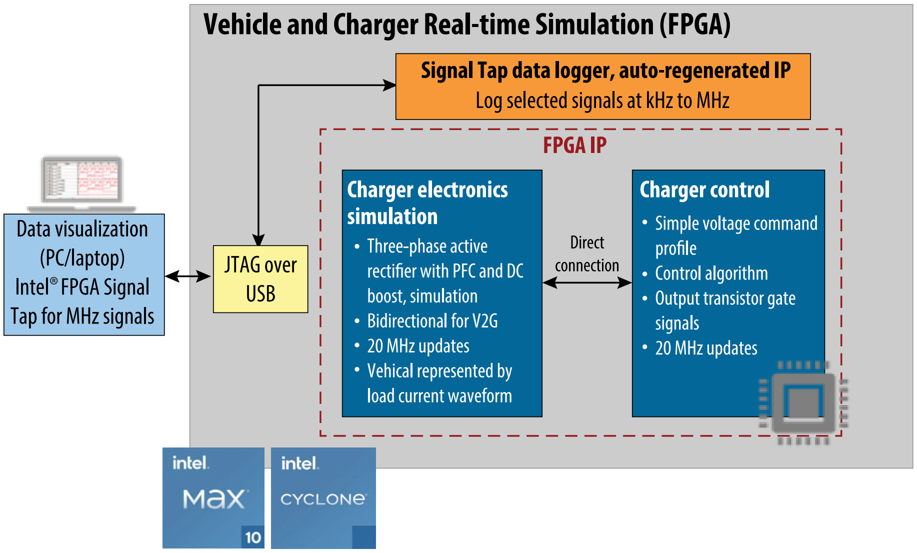
This design example demonstrates an implementation of a three-phase boost bidirectional AC/DC converter for EV charging. It uses the HDL Coder to generate synthesizable VHDL code for control and to emulate power electronics. Intel® MAX® 10 and Cyclone® V SoC FPGA Development Kits are the target devices for this reference design.

The *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example includes the following:

- Simulation of the three-phase boost bidirectional AC/DC converter for EV charging on hardware from MATLAB Simulink* and Simscape Electronics*.
- Examination of on-chip signals using the Signal Tap logic analyzer.
- Instructions for generating HDL code based on MATLAB HDL Coder and tools.
- Files to program and run Intel MAX 10 and Cyclone V SoC FPGA Development Kits.

The following diagram shows the main components of the design example and how an FPGA is incorporated to emulate power electronics and implement an actual PWM switching controller:

Figure 3. Main Components of the *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example


Vehicle and Charger Real-time Simulation (FPGA)
Signal Tap data logger, auto-regenerated IP
Log selected signals at kHz to MHz
FPGA IP
Data visualization (PC/laptop) Intel® FPGA Signal Tap for MHz signals
JTAG over USB
Charger electronics simulation
• Three-phase active rectifier with PFC and DC boost, simulation
• Bidirectional for V2G
• 20 MHz updates
• Vehical represented by load current waveform
Direct connection
Charger control
• Simple voltage command profile
• Control algorithm
• Output transistor gate signals
• 20 MHz updates
intel MAX 10
intel CYCLONE

The three-phase boost bidirectional AC/DC converter within this design example is composed of three main parts:

Table 1. Main Parts of the Three-phase Boost Bidirectional AC/DC Converter

Block	Description
Power Electronics	Composed of a six-switch power converter and high-power electronics, such as capacitors or inductors. In an actual EV charging station, this is the section of the system that is controlled by an FPGA using PWM signals.

continued...

Block	Description
	In this reference design, the power electronics subsystem is modeled with MATLAB* and synthesized to emulate the behavior of power transistors, capacitors, inductors, and resistors in the FPGA programmable logic.
Source Voltage Generator	A MATLAB Simulink* model synthesized to mimic the UK 230 V_{RMS} national power grid. With this model, HDL code is generated to emulate the behavior of the three-phase grid within the FPGA fabric.
Controller	In a real-product scenario, this is the only block in this example design that is implemented on FPGA. The controller generates the necessary PWM signals to drive the power transistors in the six-switch power converter and senses different currents and voltages so that it generates the desired DC output voltage.

The six-switch boost converter is a three-phase AC/DC converter capable of bidirectional power flow. It has the following specifications:

- Emulated 650 $V_{peak-peak}$ input voltage at 50 Hz to replicate UK 230 V_{RMS} phase voltage in the national power grid.
- Output voltage of 800 V
- Output power of 100 kW
- High efficiency (>95%)
- Minimal overshoot while maintaining quick-settling time.
- Very high frequency operation

This document describes the reference design of a new high-performance power converter. It looks for:

- Maximizing the FPGA clock frequency used to generate PWM waveforms.
- Reducing the FPGA resource use by optimizing the fixed-point signal formats.
- Simulating the interface to external sensors and transistors in a detailed manner.
- Simulating the external sensors and ADCs.
- Considering different control algorithms to the existing PI and PWM control to improve efficiency while limiting worst-case voltage error.
- Extending the design from DC input to a single or three-phase AC input to simulate electric vehicle charging.
- Implementation on FPGA using Signal Tap to acquire signals in real-time from the FPGA.

Related Information

- Refer to MAX 10 - DC-DC Converter Design Example (AN959) for DC-DC controller reference design for a single-phase converter.
- Refer to MAX 10 - Drive On Chip motor control and power conversion (Tandem) (AN773) for two-phase converter design used in the motor kit control.
- Visit the Intel FPGA Design Store for prototype designs that you must implement on FPGA or optimize.

2. Downloading and Installing the Design Example

Before downloading and installing the design, ensure that your system complies with the hardware and software requirements listed in the following sections. Then, download the reference design from the Intel FPGA Design Store and install it.

2.1. Software Requirements

To install and run the *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example, your system must meet the following minimum software requirements:

- The Intel FPGA complete design suite version 17.0.2, which includes:
  - Intel Quartus® Prime Standard Edition software version 17.0.2 (Windows | Linux) (Hint: See **Updates** tab)
  - DSP Builder for Intel FPGAs software version 17.0.2 (Windows | Linux) (Hint: See **Updates** tab)
- MATLAB* and Simulink* R2020b that includes the following:
  - Simulink, Electrical, or Specialized Power Systems libraries
  - Fixed-point Designer*

2.2. Hardware Requirements

Before installing the *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example, ensure your hardware conforms to the following hardware requirements:

Intel Quartus Prime Requirements

- A Windows PC or Linux workstation.
- A minimum CPU of 64-bit Intel Nethalem (2008) or an AMD Bulldozer (2011) microarchitecture processor with SSE4.2 instruction set or later.
- A monitor capable of at least 1024 x 768 display resolution.
- At least 36 GB of free disk space to contain copies of uncompressed version installation files. For disk space requirements for individual software components, follow these instructions:
  1. Visit the FPGA Software Download Center.
  2. Select the desired software product from the collection.
  3. Within the selected software product landing page, see **System Requirements** section. For additional memory recommendations, refer to the release notes under **Documentation Links** section.

Important:

- For the most up-to-date FPGA software operating system requirements, refer to the Operating System Support page.
- Intel recommends that your system be configured to provide virtual memory equal in size or larger than the recommended physical RAM size that is required to process your design.
- The disk space may be significantly more based on the device families included in the install.
- Prior to installation, the disk space should be enough to hold both zipped tar files and uncompressed installation files.
- After successful installation, delete the downloaded zipped files to release the disk space.

MATLAB Requirements

Refer to System Requirements - Release 2020b - Windows for MATLAB hardware requirements.

Three-phase Boost Bidirectional AC/DC Converter for EV Charging Design Example Requirements

Obtain one of the following development kits:

- Intel MAX 10 FPGA Development Kit
- Cyclone V SoC FPGA Development Kit

2.3. Downloading and Installing the Design

Perform the following steps to download and install the *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example:

Note: The *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example includes a precompiled `.sof` in the `master_image` directory.

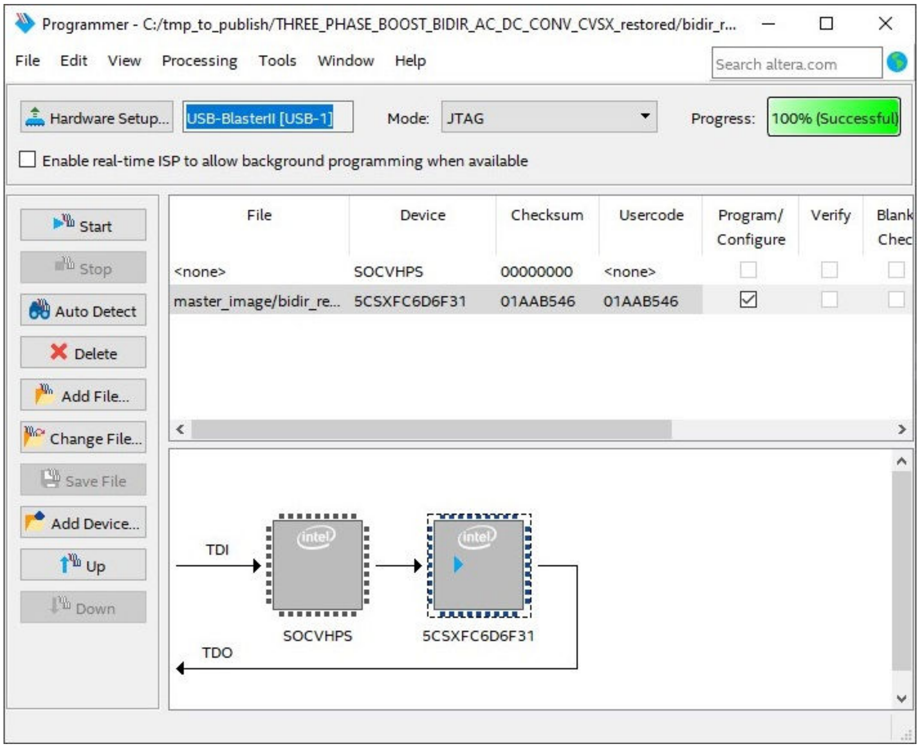
1. Download the relevant design `.par` file for your development kit and power board from the Intel FPGA Design Store:
   - Intel MAX 10-Three-phase boost bidirectional AC/DC converter FPGA design example for EV charging
   - Cyclone V-Three-phase boost bidirectional AC/DC converter FPGA design example for EV charging

   The file you downloaded is in the form of a `<project>.par` file, which contains a compressed version of the design example (similar to a `.qar` file) and metadata describing the project. The combination of this information is what constitutes a `<project>.par` file.
2. In the Intel Quartus Prime software, click **File ➤ New Project Wizard**. The **New Project Wizard** dialog box launches with the **Introduction** screen.
3. Click **Next**. The **Directory, Name, Top-Level Entity** screen displays.
4. Enter the path for your project working directory.
5. Specify a project name.
6. Click **Next**. The **Project Type** screen displays.

7. Select **Project template**.
8. Click **Next**. The **Design Templates** screen displays.
9. Click the **Install the design templates** link. The **Design Template Installation** dialog box appears.
10. In the **Design template file (.par)** field, browse to select the `.par` file of the design example and browse to the destination directory (optional) where you want to install it.
11. Click **OK**. The design template installation success message appears.
12. Click **OK**.
13. Select the **Three-phase Boost Bidirectional AC/DC Converter for EV Charging** design example.
14. Click **Next**. The **Summary** screen appears.
15. Click **Finish**. The Intel Quartus Prime software expands the archive and sets up the project, which may take some time.
16. Click **Processing ➤ Start Compilation**.
17. Connect the relevant USB cable from the USB connector on the development board to your computer.
    - J12 for Intel MAX 10 FPGA Development Kit
    - J37 for Cyclone V SoC Development Kit
18. Supply power to the development board.
19. After compilation ends, click **Tools ➤ Programmer** to program your Intel MAX 10 Development Kit or Cyclone V SoC Development Kit. Use the *Auto Detect* feature if necessary.
20. Select the `master_image/bidir_rectifier_fixed_quartus.sof` file or your newly generated `output_files/bidir_rectifier_fixed_quartus.sof` file.

    Note: The Programming the Board figure shows the programmer for Cyclone V SoC Development Kit FPGA (5CSXFC6D6F31).
21. Select the **Program/Configure** check box in the table.
22. Click **Start**. The **Progress** bar displays in green a *100% (Successful)* legend.

Figure 4. Programming the Board



Programmer - C:/tmp_to_publish/THREE_PHASE_BOOST_BIDIR_AC_DC_CONV_CVSX_restored/bidir_r...

USB-BlasterII [USB-1] Mode: JTAG Progress: 100% (Successful)

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank Chec
<none>	SOCVHPS	00000000	<none>	☐	☐	☐
master_image/bidir_re...	5CSXFC6D6F31	01AAB546	01AAB546	☑	☐	☐

23. Return to the Intel Quartus Prime software and click **Tools ➤ Signal Tap logic analyzer**.

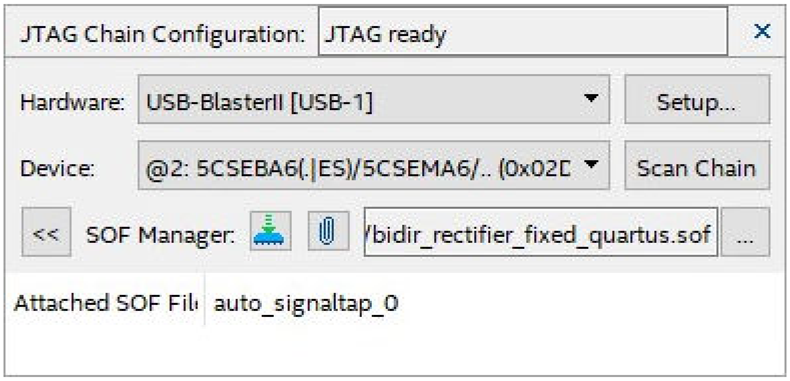
24. In the **Signal Tap logic analyzer** window, in the top right corner, click **Setup** to select USB-BlasterII as shown in Hardware Setup.

25. Select the appropriate FPGA device.

    — **Intel MAX 10 Development Kit**: Select @1: `10M50DA(.ES)/10M50DC (0x031050DD)`

    — **Cyclone V SoC Development Kit**: Select @2: `5CSEBA6(.ES)/5CSEMA6/.. (0x02D020DD`.

Figure 5. Hardware Setup



JTAG Chain Configuration: JTAG ready

Hardware: USB-BlasterII [USB-1]

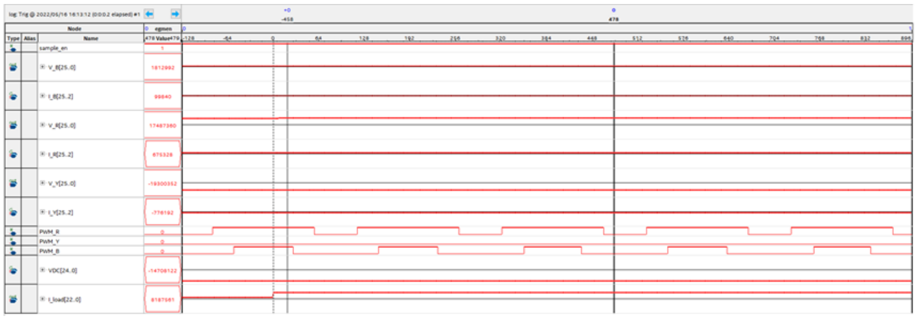
Device: @2: 5CSEBA6(.|ES)/5CSEMA6/.. (0x02D

SOF Manager: /bidir_rectifier_fixed_quartus.sof

Attached SOF Fil auto_signaltap_0

26. In the **Signal Tap logic analyzer** window, select the **Data** tab and click . The following image shows the waveform:

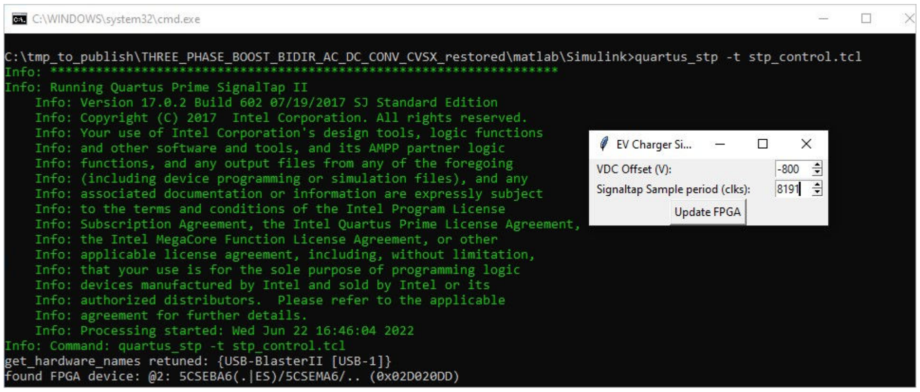
Figure 6. Signal Tap Logic Analyzer Waveform



27. Navigate to the `matlab/Simulink` directory and double-click the `run_stp_control.bat` file to adjust the sampling resolution.

28. In the **EV Charger** dialog box, set the values of **VDC Offset (V)** to -800 and **Signaltap Sample period (clks)** to 8191 as shown in the following image:

Figure 7. Changing VDC Offset and Signaltap Sample Period



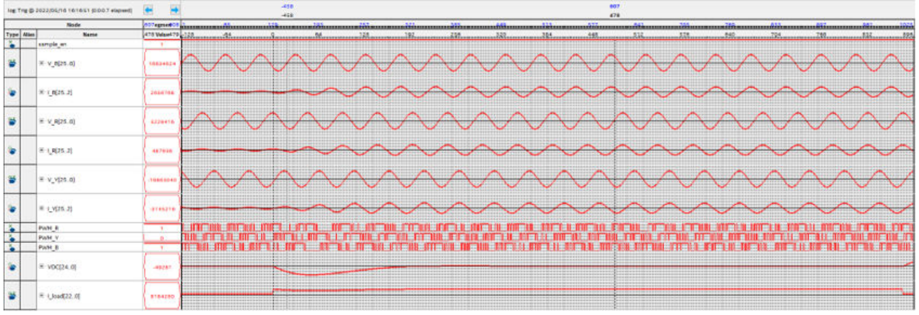
```
C:\tmp_to_publish\THREE_PHASE_BOOST_BIDIR_AC_DC_CONV_CVSX_restored\matlab\Simulink>quartus_stp -t stp_control.tcl
Info: *******************************************************************
Info: Running Quartus Prime SignalTap II
    Info: Version 17.0.2 Build 602 07/19/2017 SJ Standard Edition
    Info: Copyright (C) 2017  Intel Corporation. All rights reserved.
    Info: Your use of Intel Corporation's design tools, logic functions
    Info: and other software and tools, and its AMPP partner logic
    Info: functions, and any output files from any of the foregoing
    Info: (including device programming or simulation files), and any
    Info: associated documentation or information are expressly subject
    Info: to the terms and conditions of the Intel Program License
    Info: Subscription Agreement, the Intel Quartus Prime License Agreement,
    Info: the Intel MegaCore Function License Agreement, or other
    Info: applicable license agreement, including, without limitation,
    Info: that your use is for the sole purpose of programming logic
    Info: devices manufactured by Intel and sold by Intel or its
    Info: authorized distributors.  Please refer to the applicable
    Info: agreement for further details.
    Info: Processing started: Wed Jun 22 16:46:04 2022
Info: Command: quartus_stp -t stp_control.tcl
get_hardware_names retuned: {USB-BlasterII [USB-1]}
found FPGA device: @2: 5CSEBA6(.|ES)/5CSEMA6/.. (0x02D020DD)
```

VDC Offset (V): -800

Signaltap Sample period (clks): 8191

29. Return to the **Signal Tap logic analyzer** window and click . The view updates as shown in the following images:

Figure 8. Scaled Output Waveform in the Signal Tap Logic Analyzer



The following are waveform components' details:

- In the model, the output DC Voltage (VDC) is regulated to 800V. Setting the **VDC Offset** to -800 V allows the maximum range to be easily visible in the Signal Tap logic analyzer.
- The **SignalTap Sample Period** sets the period of a counter, which enables waveforms sampling. A value of *n* results in samples every *n*-1 clock cycles. So, a value of 0 samples every clock cycle results in no aliasing with a very short visible period. The value of 8191 is shown in step 28 samples every 8192 cycles, so the Pulse Width Modulation (PWM) signals are highly aliased, but a long time frame becomes visible.

You can run the GUI script from the Intel Quartus Prime software TCL console with the following command:

```
exec quartus_stp -t ../../stp_control.tcl &
```

Note: The `../../` path depends on the current directory of the Intel Quartus Prime software.

3. Model Description

This section describes various models provided with the *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example.

3.1. Simscape* Model

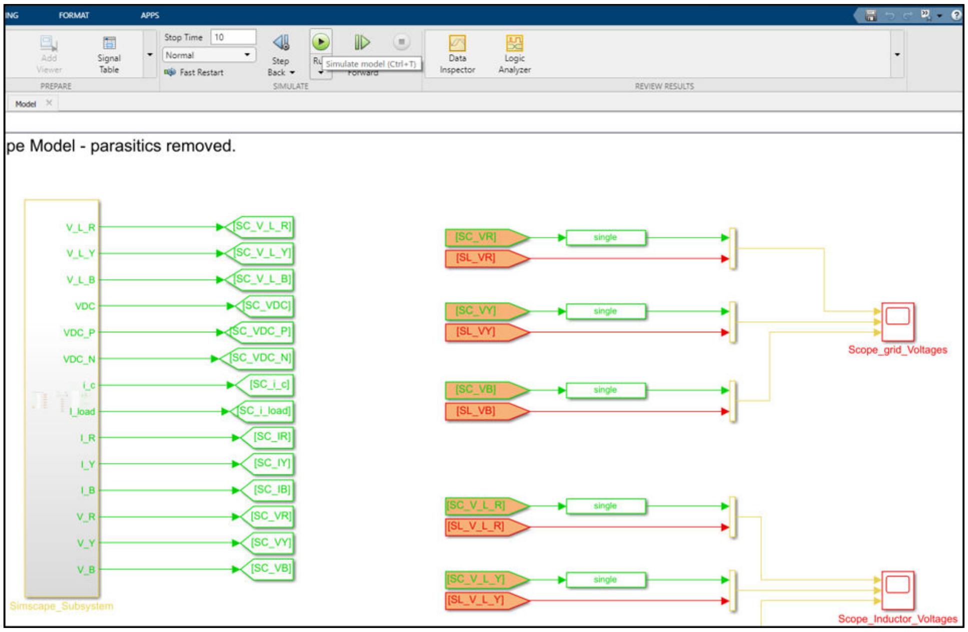
You can find the Simscape* model in the `matlab/Simscape` directory under the `bidir_rectifier_2models.slx` file. This model implements Simscape Electronics* blocks, and it is the primary approach to implementing the three-phase boost bidirectional AC/DC converter for EV Charging.

The `bidir_rectifier_2models.slx` file compares the Simscape model with an implementation of a floating-point Simulink* model. This is intended to check the behavior of the power converter across multiple implementations.

You cannot use the Simscape model in this context to generate HDL code for FPGA. However, the model provides the first insight on how to implement power and control models, which can be converted to *synthesizable* blocks for programming an FPGA fabric (see Simulink* Model on page 15).

The Simscape model has all components and subsystems as the Simulink model described in the Simulink* Model on page 15. To run this simulation, click **Run** on the Simulink GUI menu.

Figure 9. Launching Simscape Model Simulation

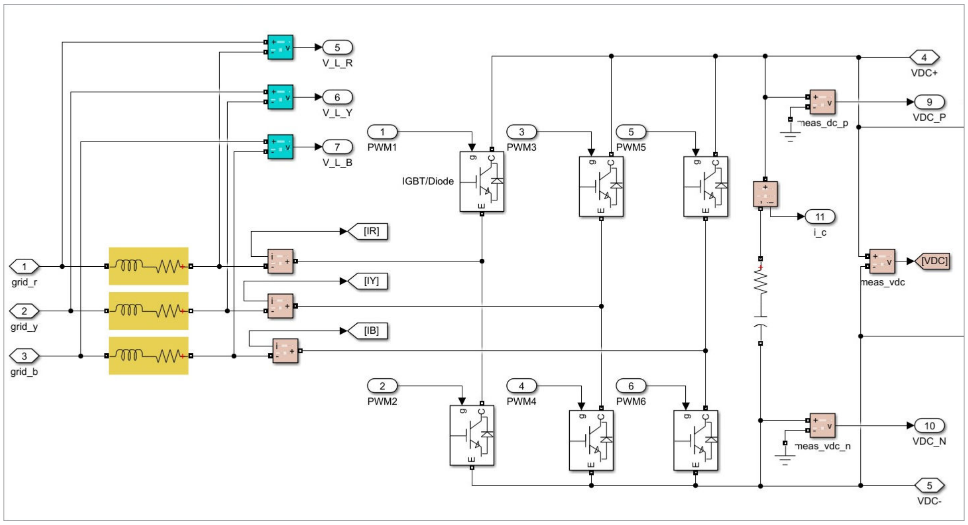


Simscape Power Electronics Block

The Simscape power electronics block is equivalent to the block described in Power Electronics Block Model on page 21, and it is highlighted here to show the difference between implementing power circuitry using Simscape electronics and Simulink blocks. This implementation allows understanding the array of transistors, capacitors, and inductors to implement the six-switch power converter and the power electronics block to be built using Simulink blocks that emulate these passive and active components.

The following image shows the power electronics block within the Simscape model in the `bidir_rectifier_2models.xsl`:

Figure 10. Six-switch Power Converter Implemented with Simscape Blocks



3.2. Simulink* Model

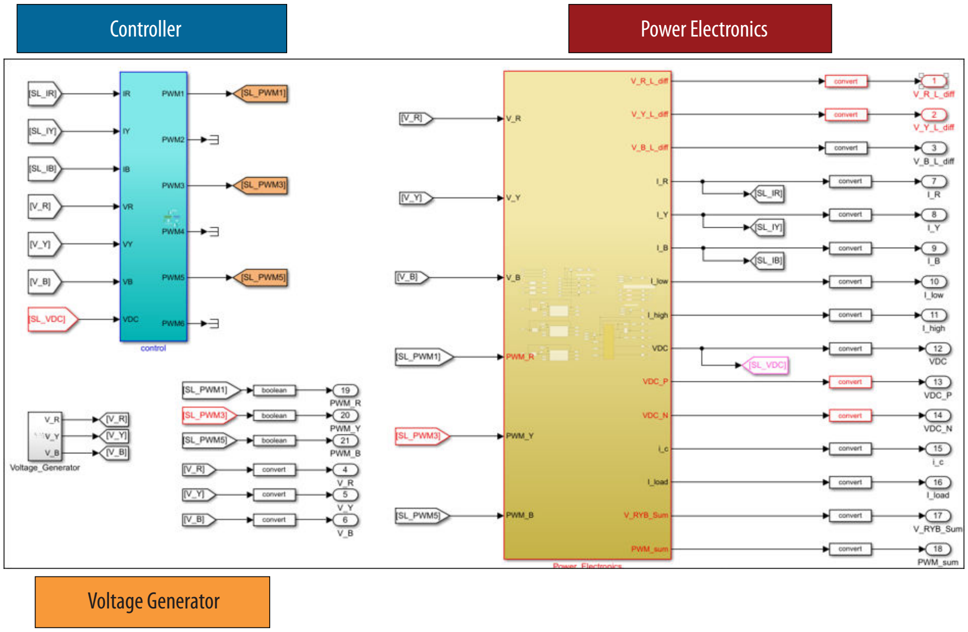
The Simulink* model `bidir_rectifier.slx` within the `matlab/Simulink` directory produces VHDL code that implements the three-phase boost bidirectional AC/DC converter for EV charging simulation. The complete system can run in real-time on FPGA.

Top-level Overview

The load current is simulated as a square waveform with positive and negative values, generating current and voltage waveforms inside the design. When the load current switches, you can examine the transient current and voltage signals using the `rectifier.stp` file in Signal Tap logic analyzer within the Intel Quartus Prime software on a computer.

The following image shows the expanded three-phase boost bidirectional AC/DC converter block:

Figure 11. Synthesizable Controller and Power Electronics Blocks Inside the Top Level

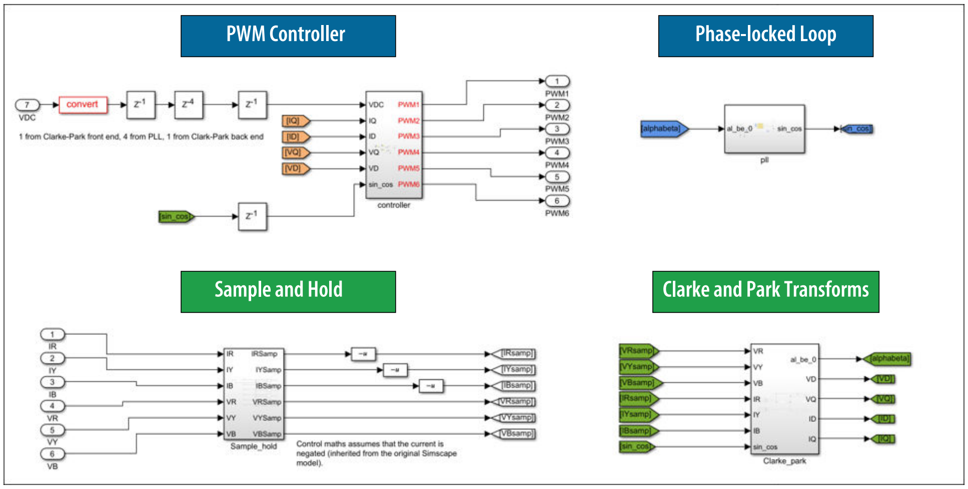


The bidirectional AC/DC converter block has the portion of the simulation for which VHDL code is generated. The outputs are defined in the top-level model file using the blocks' in-port or out-port port interfaces for the VHDL code. The MATLAB Simulink in-port and out-port signals define the VHDL signal names, and the VHDL data formats are the signal formats you typically set with the convert block.

3.2.1. Controller Block

The controller block subsystem is composed of the PWM controller (PWM signal generator for power transistor switching), Phase-locked Loop (PLL), sample and hold block, and Clarke and Park transforms.

Figure 12. Controller-block Subsystem Implemented in Simulink

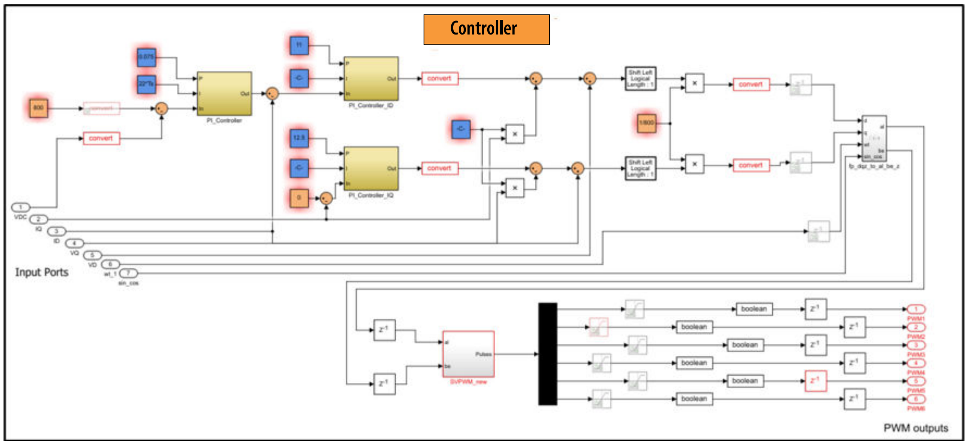


You can find more information about each block in the following sections:

3.2.1.1. Pulse Width Modulation (PWM) Controller

The controller for the power switches that generate the PWM (power transistor switching controller) outputs is composed of an outer Proportional Integral (PI) voltage control loop surrounding two internal current PI controllers, as shown in the following image:

Figure 13. Controller Implemented in Simulink with HDL Coder Blocks



An external reference voltage (VDC), the capacitor voltage, or an output voltage (up to 800 V) provides the output reference voltage. The output voltages are initially subtracted from the reference voltage, and the current reference controls it. Direct Quadrature (DQ) synchronous reference frame voltages are achieved at the control output. Normalize these two voltage values by a factor of 1 (shift-left logical) and 1/800 to get a waveform that you can use in the Space Vector PWM (SVPWM) block. After normalization, the values are converted from the DQ frame to the alpha-beta frame (orthogonal stationary) and used in the SVPWM block to control the switching of the power transistors in the six-switch converter module.

To find the values of the PI constants, refer to the *Analysis and Control of Three-Phase PWM Rectifier for Power Factor Improvement of IM Drive*[1] paper for the following formulas:

$$K_{p-current} = \frac{L}{\lambda_i} = 9.42$$

$$K_{I-current} = \frac{R_L}{\lambda_i} = 484$$

where:

- $$\frac{1}{\lambda_i} = \frac{2\pi f_s}{10} = 62832$$
- L is the input inductance.
- R_L is the associated resistance.

Similarly, you can calculate voltage loop constants as follows:

$$K_{p-voltage} = \frac{C}{3V\lambda_i} = 0.051$$

$$K_{i-voltage} = \frac{C}{3V\lambda_i^2} = 320.4$$

where:

- C is the DC link capacitance.
- V is the input RMS voltage.

Even though the values calculated using the equations provide a good control response, manual tuning is necessary to achieve the desired overshoot and settling time. After manual tuning, you can find values for the constants in the Simulink model blocks as follows:

- $K_{\texttt{p-voltage}}$ = 0.075
- $K_{\texttt{i-voltage}}$ = 22 T_s
- $K_{\texttt{p-current1}}$ = 11

[1] P. Scholar, 'Analysis and Control of Three Phase PWM Rectifier for Power Factor Improvement of IM Drive', International Journal of Innovations in Engineering and Technology, vol. 10, no. 2, p. 7, 2018

- $K_{i\text{-current1}} = 500\ T_s$
- $K_{p\text{-current2}} = 12.5$
- $K_{i\text{-current2}} = 500\ T_s$

Note: Consider T_s = 50 ns and the circuit is running at 20 MHz in the discrete domain.

3.2.1.2. Phase-locked Loop (PLL)

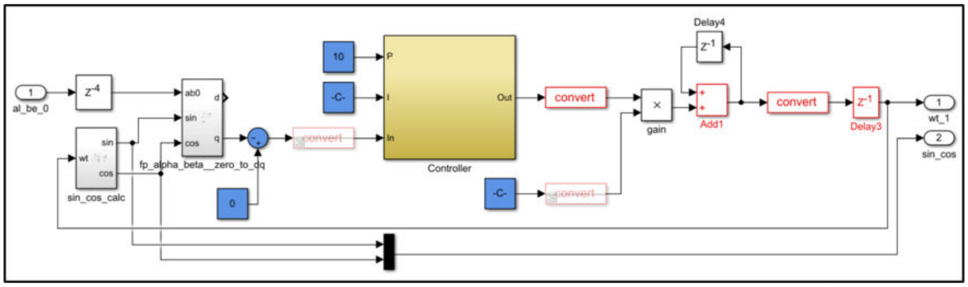
The Phase-locked Loop (PLL) block is required to assert that the phases of the generated current references are synchronous with the power grid. The rotating angle `wt`, an output of the PLL, is the same DQ frame rotation angle. The resulting rotation angle is used as feedback to calculate the angle `wt` for another DQ transform.

Note: The quadrature component of the voltage Vq is controlled to a value of `0`. When `Vq = 0`, the PLL *locks*, which means that the references are fully synchronized with the grid.

To achieve a faster *lock*, the PI controller within the PLL uses high values of K_p and K_i. For this implementation, $K_p = 10$ and $K_i = 100000\ T_s$ (T_s = 50 ns).

The following figure shows the final implementation of the PLL using Simulink blocks in the discrete-time domain for FPGA HDL coder generation:

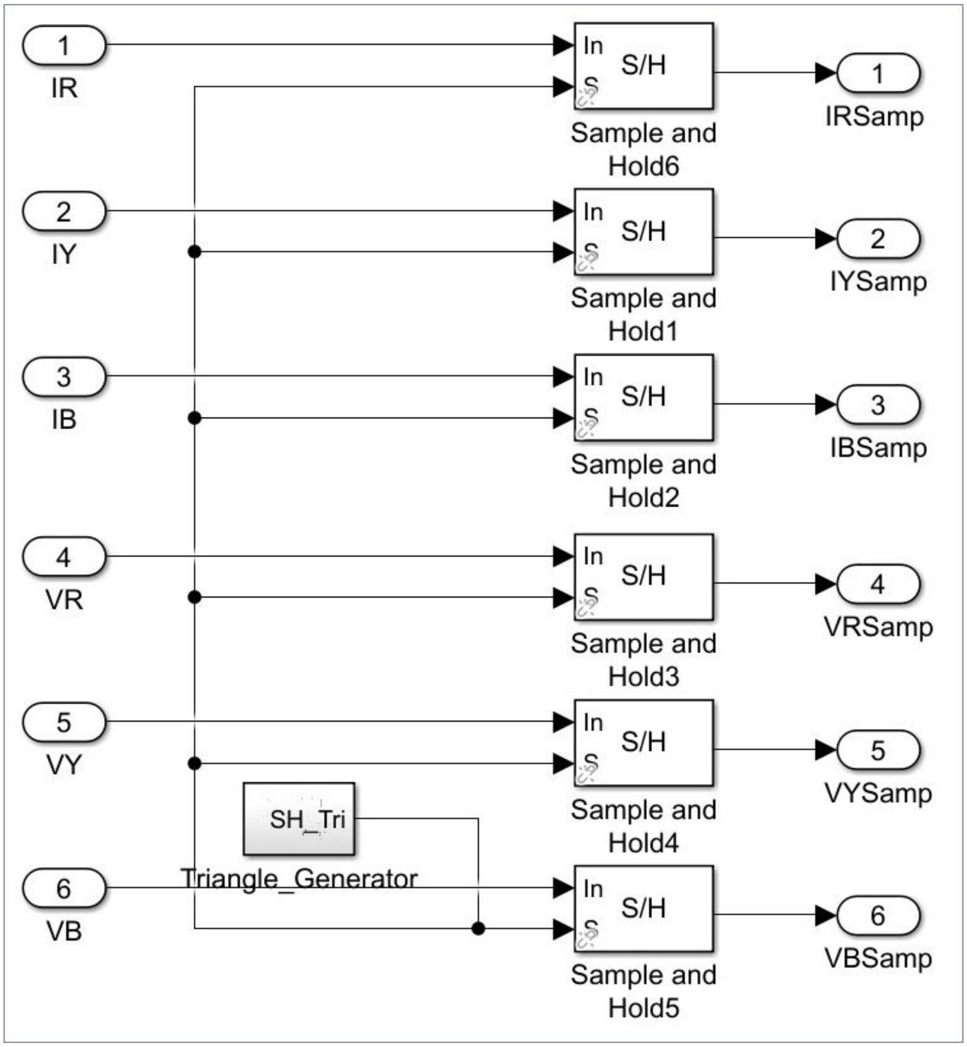
Figure 14. Phase-locked Loop Block for Grid Synchronization



3.2.1.3. Sampling Circuit

The sample and hold block is required to safely guarantee the control logic digital operation while the main circuit keeps operating. The PWM signals to switch on and off the transistors, which cannot be allowed to transition between allotted switching events (100 KHz).

Figure 15. Sampling Circuit Model



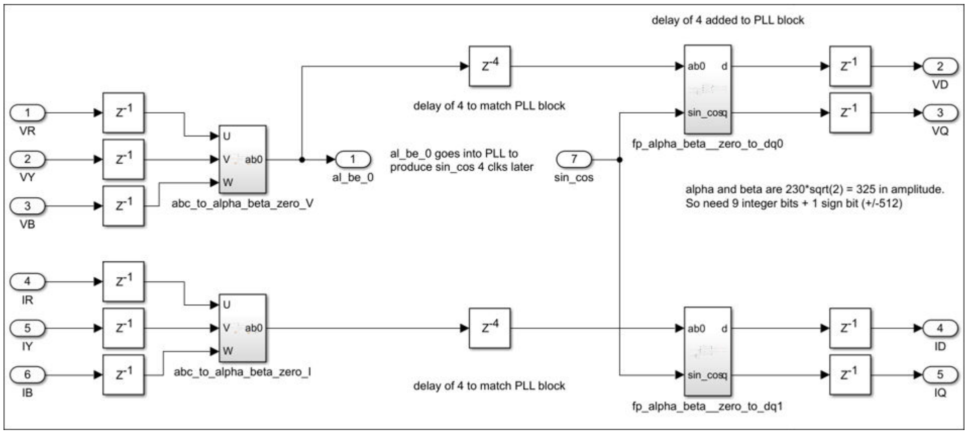
3.2.1.4. Clarke and Park Transforms

The *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example uses Clarke and Park transforms to convert current and voltage three-phase values to DC signals. The transforms establish three phases as vectors on a two-dimensional rotating coordinate system. The coordinate system rotates at the same rate as the voltage and current vectors.

For power conversion, controlling voltages and DQ frame current allows controlling the level of reactive and active power, giving the ability to control the power factor independently.

The following image shows the implementation of Clarke and Park transforms using Simulink blocks in discrete time for FPGA implementation using HDL coder:

Figure 16. Clarke and Park Transforms Implemented in Simulink with HDL Coder Blocks


delay of 4 added to PLL block
z-4
delay of 4 to match PLL block
ab0 d
sin_cosq
fp_alpha_beta__zero_to_dq0
z-1
2 VD
3 VQ
1 VR
2 VY
3 VB
U
V ab0
W
abc_to_alpha_beta_zero_V
1 al_be_0
al_be_0 goes into PLL to produce sin_cos 4 clks later
7 sin_cos
alpha and beta are 230*sqrt(2) = 325 in amplitude. So need 9 integer bits + 1 sign bit (+/-512)
4 IR
5 IY
6 IB
abc_to_alpha_beta_zero_I
z-4
delay of 4 to match PLL block
ab0 d
sin_cosq
fp_alpha_beta__zero_to_dq1
4 ID
5 IQ

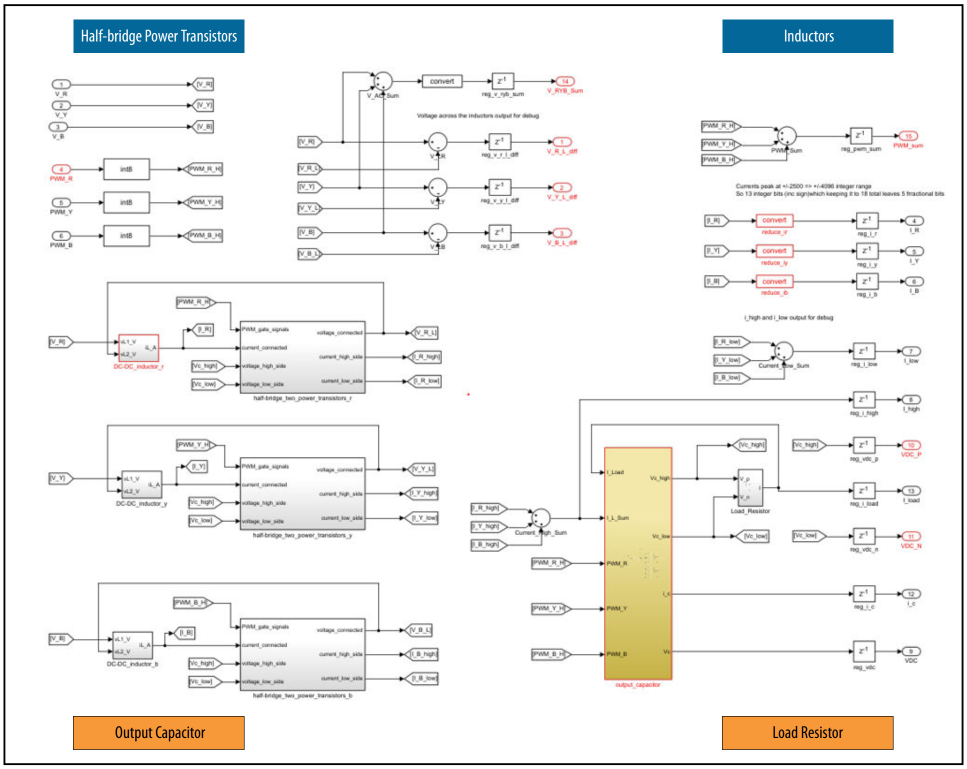
3.2.2. Power Electronics Block Model

The power electronics block is a synthesizable model of the power electronics section of an EV charger. The blocks described in this section are models of an actual power stage you can find in any EV charger three-phase boost bidirectional AD/DC converter. It includes the following components:

- Insulated Gate Bipolar Transistors (IGBT)
- Diodes
- High-voltage capacitors
- Modeling of parasitic resistance and capacitance

The significance of modeling the power electronics in HDL and FPGA is to enable you to emulate your power stage in FPGA and fine-tune the controller parameters according to your system characteristics. In an actual design, the power electronics block is removed, and actual implementation of the power electronic circuit is placed in the EV charger station.

Figure 17. Power Electronics Block Modeling



3.2.2.1. Six-switch Power Converter

The six-switch converter comprises six Insulated Gate Bipolar Transistors (IGBTs) or any other suitable power transistor with antiparallel diodes. The circuit consists of three half-bridges, and each half-bridge is connected to a voltage phase.

In this Simulink* model, to obtain synthesizable HDL code, each half-bridge is modeled as one voltage switch and two current switches, and the outputs of such blocks are a voltage, a `current_high`, and a `current_low`.

3.2.2.2. Inductors

Inductors comprise of three subblocks, and each emulating one of the three inductors. The inductor model has two inputs (the voltage at either end of the inductor) and one output (the current through the inductor).

One end of each inductor is connected to the respective phase of the power grid voltage. The other end of the inductor is connected to the voltage switch. The inductor model integrates the difference between these inputs, multiplied by the inductance, to produce the resulting current through the inductor.

The voltage switch's high and low sides are the output capacitor's respective pins.

3.2.2.3. Output Capacitor

In this model, there is a subblock to emulate the output capacitor or DC-link capacitor, for which you must consider an acceptable output voltage ripple. In the continuous-time domain, you can use the following standard electronics formula to obtain the capacitor value:

$$C = \frac{I\,t}{\Delta V} = \frac{125A * 1x10^{-5}s}{0.5V} = 2.5mF$$

Where:

- ΔV = 0.5 V, which is the acceptable output voltage ripple.
- 1 x 10^{-5}s is the inverse of the switching frequency.
- 125 A is the maximum output current.

The voltage across the output capacitor (V_{DC}) is the integral of the current flowing into it as follows:

$$V_{DC} = C \int I_C \, dt$$

The inductor and switch models calculate the high side current, which is a split between the output capacitor (I_C) and load resistor (I_{LOAD}). Although V_C is controlled to a DC voltage, the voltage on the capacitor terminals is relative to the ground (V_{DC_P} and V_{DC_N}) switches up and down with the PWM of the switches. The calculation of V_{DC_P} and V_{DC_N} depends on the state of the PWM signals (0 or 1).

$$V_{DC\_N} = \frac{V_R + V_Y + V_B - V_{DC}(PWM_R + PWM_Y + PWM_B)}{3}$$

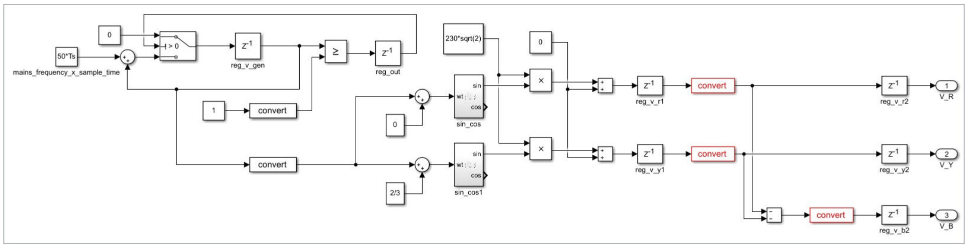
Then, calculate V_{DC_P} using:

$$V_{DC\_P} = V_{DC\_N} + V_{DC}$$

3.2.3. Source Voltage Generator Block

The source voltage generator block helps to emulate the three-phase power grid. The circuit output voltage is a 430 V phase to phase at 50 Hz (based on UK national grid specification). The voltages produced in this block are measured and used in the control circuit after sampling.

Figure 18. Source Voltage Generator Modeling



3.3. HDL Inputs, Outputs, and Parameters

The following tables list the inputs, outputs, and PI controller block parameters on the HDL coder block:

Table 2. Inputs for HDL Coder Block

Input Signal Name	Type	Description
clk	Clock	Input clock for the HDL coder block. Generated by PLL 20 MHz.
reset	Reset	Active low reset
clk_enable	Enable	Active high reset

Table 3. Outputs for HDL Coder Block

Output Signal Name	Data Type	Description
V_R_L_diff V_Y_L_diff V_B_L_diff	Fix32_En16	The voltage across inductors for debugging when compared to the input voltage.
V_R, V_Y, V_B	Fix32_En16	Three-phase input voltage from the voltage generator block.
I_R, I_Y, I_B	Fix32_En16	Measured current in each inductor model block.
I_LOW, I_HIGH	Fix32_En16	Sum of the low and high-side switch currents, respectively
VDC_P, VDC_N	Fix32_En16	Voltage relative to ground of the DC output +/- ports.
I_C	Fix32_En16	Current measured in the output capacitor.
I_LOAD	Fix32_En16	Current measured in the output load.
V_RYB_SUM	Fix32_En16	Sum of the input AC voltages.
PWM_SUM	Fix32_En16	Sum of the three PWM-generated signals by the controllers.
PWM_R, PWM_Y, PWM_B	Fix32_En16	Controller-generated PWM signals for power transistor switching.

Table 4. PI Controller Block Parameters

PI Controller Block Parameters	Value
`Voltage Loop Pgain`	0.075
`Voltage Loop Igain`	22 * Ts
`Dcurrent Loop Pgain`	11
`Dcurrent Loop Igain`	500 * Ts
`Qcurrent Loop Pgain`	12.5
`Qcurrent Loop Pgain`	500 * Ts

4. FPGA Resource Use Comparison

The following tables show FPGA resource use for the fixed-point Simulink* implementation of the *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example based on Cyclone V SoC FPGA Development Kit and Intel MAX 10 FPGA Development Kit after Intel Quartus Prime software compilation. Tables show the main entities' resource utilization and the total utilization of the FPGA.

Table 5. FPGA Resource Use for Intel MAX 10 FPGA Development Kit

Entity	LEs	Regs	Mem. Bits	M9Ks	DSPs	DSP 9x9	DSP 18x18
Bidir_rectifier_fixed (from MATLAB)	6983	3530	35328	6	137	5	66
Power electronics (model)	1923	622	0	0	27	3	12
Controller	4542	2559	11776	2	98	2	48
Voltage Generator	530	349	23552	4	12	0	6
Signal Tap Block	4214	3808	209920	23	0	0	0
Others	648	456	0	0	0	0	0
Total	**11845 (24%)**	**7794 (15%)**	**245248 (15%)**	**29 (16%)**	**137 (48%)**	**5 (2%)**	**66 (46%)**

Table 6. FPGA Resource Use for Cyclone V SoC FPGA Development Kit

Entity	ALMs	Registers	Mem. Bits	M10Ks	DSP Blocks
Bidir_rectifier_fixed (from MATLAB)	1912	3651	35328	6	56
Power electronics (model)	590	674	0	0	13
Controller	1168	2618	11776	2	377
Voltage Generator	156	359	23552	4	6
Signal Tap	1124	3848	209920	21	0
Others	561	471	0	0	0
Total	**3597 (9%)**	**7970 (10%)**	**245248 (4%)**	**27 (5%)**	**56 (50%)**

5. Locating Top-level VHDL Wrapper

You can find the top-level VHDL wrapper `top.vhd` in the `M10` or `C5` project directory. It mainly includes the following three sections:

- Auto-generated three-phase boost bidirectional AC/DC converter for EV charging from the HDL coder.
- Auto-generated PLL.
- Control sample period and VDC offset components.

6. Generating HDL Code with MATLAB and HDL Coder

The *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example provides means to generate HDL code that you can import into an Intel Quartus Prime software project and program the Intel FPGA boards with it.

The following sections provide instructions to generate HDL code (based on MATLAB*, Simulink* toolbox, and HDL Coder toolbox) and Intel Quartus Prime software projects for Intel MAX 10 FPGA Development Kit and Cyclone V SoC Development Kit:

6.1. Generating HDL and QPF Using MATLAB Scripts

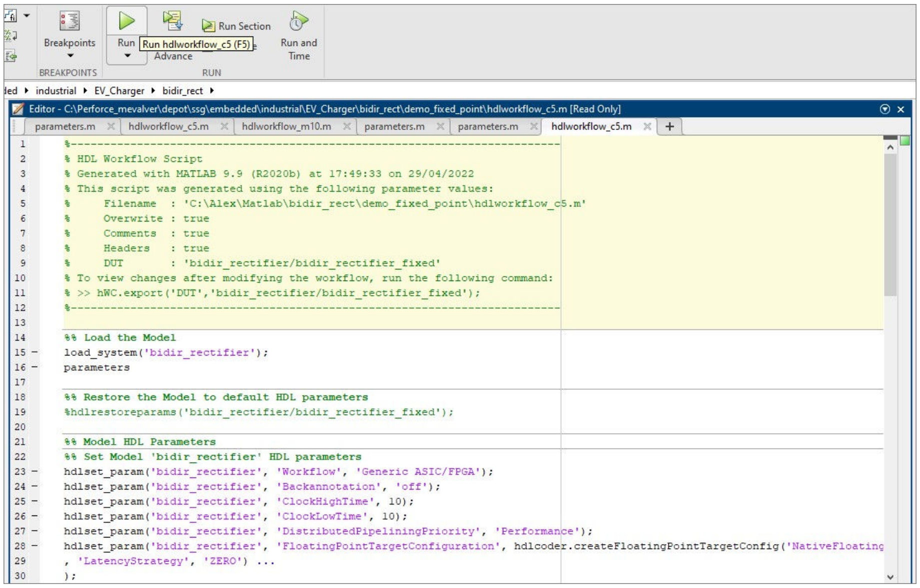
The *Three-phase Boost Bidirectional AC/DC Converter for EV Charging* design example provides MATLAB* files and scripts necessary to generate an Intel Quartus Prime software project and HDL code for the three-phase bidirectional AC/DC converter for an electric vehicle charging station. Under the `matlab/Simulink` directory, `bidir_rectifier.slx` and `bidir_rectifier_2models.slx` MATLAB files contain MATLAB models. The `bidir_rectifier.slx` file compares fixed-point and floating-point Simulink* models. The `bidir_rectifier_2models.slx` file compares the Simscape Electronics and Simulink floating-point models.

The model to generate HDL code for FPGA used in this design example is the Simulink fixed-point model contained in the `bidir_rectifier.slx`, named `bidir_rectifier_fixed`.

Follow these instructions to generate an Intel Quartus Prime software project and HDL using HDL coder for Intel MAX 10 FPGA Development Kit or Cyclone V SoC Development Kit:

1. Navigate to the `matlab/Simulink` directory and open one of the following files:
   - **Cyclone V SoC Development Kit**: `hdlworkflow_c5.m` to generate an Intel Quartus Prime software project and VHDL code for the three-phase boost bidirectional AC/DC converter for EV charging.
   - **Intel MAX 10 Development Kit**: `hdlworkflow_m10.m` to generate an Intel Quartus Prime software project and VHDL code for the three-phase boost bidirectional AC/DC converter for EV charging.
2. Click **Run** to execute the script in the MATLAB interface.

Figure 19. Run MATLAB Script to Generate a Project


Breakpoints
BREAKPOINTS
Run
Run hdlworkflow_c5 (F5)
Run Section
Advance
Run and
Time
RUN
ied ▸ industrial ▸ EV_Charger ▸ bidir_rect ▸
Editor - C:\Perforce_mevalver\depot\ssg\embedded\industrial\EV_Charger\bidir_rect\demo_fixed_point\hdlworkflow_c5.m [Read Only]
parameters.m
hdlworkflow_c5.m
hdlworkflow_m10.m
parameters.m
parameters.m
hdlworkflow_c5.m

```
%------------------------------------------------------------------------
% HDL Workflow Script
% Generated with MATLAB 9.9 (R2020b) at 17:49:33 on 29/04/2022
% This script was generated using the following parameter values:
%     Filename  : 'C:\Alex\Matlab\bidir_rect\demo_fixed_point\hdlworkflow_c5.m'
%     Overwrite : true
%     Comments  : true
%     Headers   : true
%     DUT       : 'bidir_rectifier/bidir_rectifier_fixed'
% To view changes after modifying the workflow, run the following command:
% >> hWC.export('DUT','bidir_rectifier/bidir_rectifier_fixed');
%------------------------------------------------------------------------

%% Load the Model
load_system('bidir_rectifier');
parameters

%% Restore the Model to default HDL parameters
%hdlrestoreparams('bidir_rectifier/bidir_rectifier_fixed');

%% Model HDL Parameters
%% Set Model 'bidir_rectifier' HDL parameters
hdlset_param('bidir_rectifier', 'Workflow', 'Generic ASIC/FPGA');
hdlset_param('bidir_rectifier', 'Backannotation', 'off');
hdlset_param('bidir_rectifier', 'ClockHighTime', 10);
hdlset_param('bidir_rectifier', 'ClockLowTime', 10);
hdlset_param('bidir_rectifier', 'DistributedPipeliningPriority', 'Performance');
hdlset_param('bidir_rectifier', 'FloatingPointTargetConfiguration', hdlcoder.createFloatingPointTargetConfig('NativeFloating
, 'LatencyStrategy', 'ZERO') ...
);
```

Running the HDL workflow script generates the `hdl_prj_c5/quartus_prj` for Cyclone V or `hdl_prj_m10/quartus_prj` (for Intel MAX 10) inside the `matlab/Simulink` directory. These newly created directories contain a full Intel Quartus Prime software project similar to the one delivered in this design example. This flow helps regenerate the HDL if you want to modify the model. The directory named `hdl_prj_x/hdlscr` is the newly generated HDL for the three-phase boost bidirectional AC/DC converter for EV charging based on the fixed-point Simulink model.

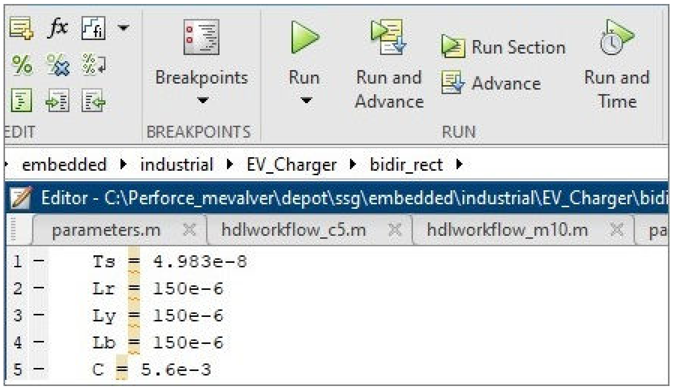
6.2. Generating HDL and QPF Using MATLAB GUI Tools

This section provides instructions to generate an Intel Quartus Prime software project and VHDL code for the fixed-point model described in the `bidir_rectifier.slx` file.

Follow these steps:

1. Launch the MATLAB model in the `matlab/Simulink/bidir_rectifier.slx` directory. The **MATLAB and Simulink** window launches.
2. In the **MATLAB** interface, select the `parameters.m` file and click **Run**. This sets some necessary variables to synthesize the model.

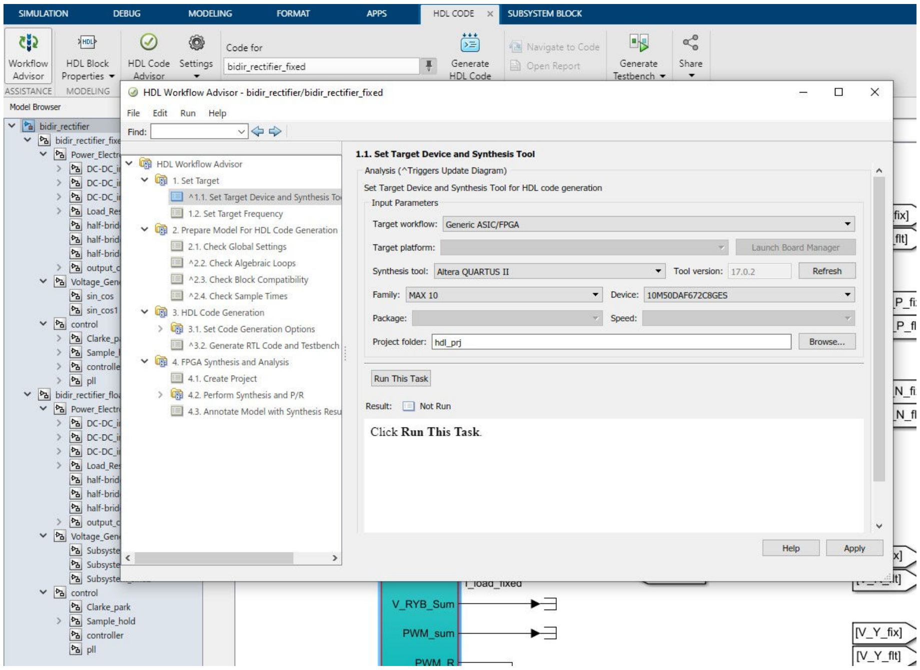
Figure 20. Setting Up Model Variables


EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
embedded ▸ industrial ▸ EV_Charger ▸ bidir_rect ▸
Editor - C:\Perforce_mevalver\depot\ssg\embedded\industrial\EV_Charger\bidi
parameters.m
hdlworkflow_c5.m
hdlworkflow_m10.m
pa

```
Ts = 4.983e-8
Lr = 150e-6
Ly = 150e-6
Lb = 150e-6
C = 5.6e-3
```

3. Launch the Simulink window where you can see the model that is synthesized and used to generate an Intel Quartus Prime software project and VHDL code (`bidir_rectifier_fixed`).
4. In the Simulink window, navigate to the **HDL CODE** tab and click **Workflow Advisor**. A warning of a missing `./top.vhd` appears. Ignore the warning.

Figure 21. Workflow Advisor and Simulink GUI



1.1. Set Target Device and Synthesis Tool

Analysis (^Triggers Update Diagram)

Set Target Device and Synthesis Tool for HDL code generation

Input Parameters

Target workflow: Generic ASIC/FPGA

Target platform:

Synthesis tool: Altera QUARTUS II Tool version: 17.0.2

Family: MAX 10 Device: 10M50DAF672C8GES

Package: Speed:

Project folder: hdl_prj

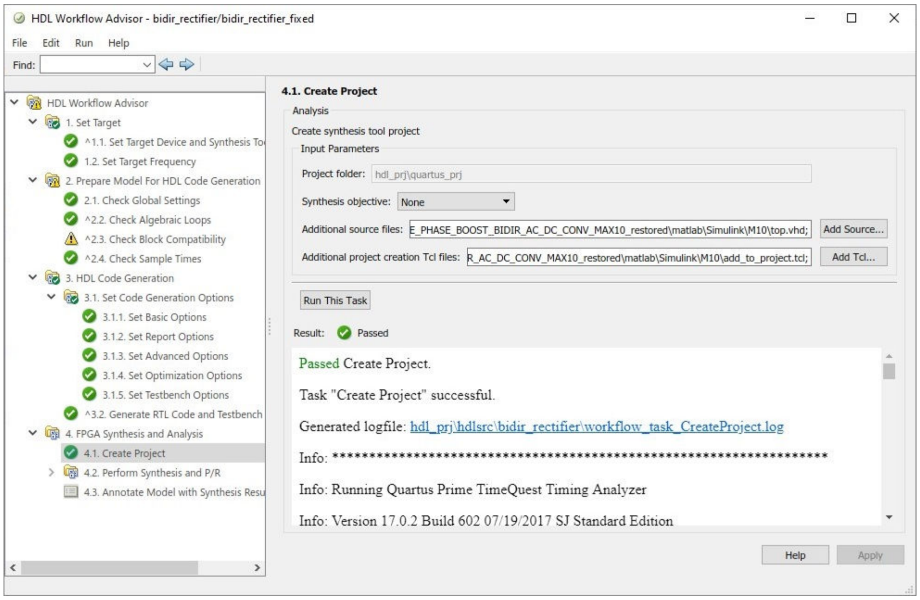
Result: Not Run

Click **Run This Task**.

5. Expand the **HDL Workflow Advisor** tasks menu in the left-hand pane.
6. Select **1.1 Set Target Device and Synthesis Tool** to configure the target FPGA device and perform one of the following:
   - **For Intel MAX 10 FPGA Development Kit**: Select the value for **Family** as `MAX 10` and **Device** as `10M50DAF484C6GES`.
   - **For Cyclone V SoC Development Kit**: Select the value for **Family** as `Cyclone V` and **Device** as `5CSXFC6D6F31C6`.

7. Change the **Project folder** to a desired project location and name.
8. In the left-hand pane, select **4.1 Create Project**.

Figure 22. Running Tasks in HDL Workflow Advisor


HDL Workflow Advisor - bidir_rectifier/bidir_rectifier_fixed
File
Edit
Run
Help
Find:
HDL Workflow Advisor
1. Set Target
^1.1. Set Target Device and Synthesis To
1.2. Set Target Frequency
2. Prepare Model For HDL Code Generation
2.1. Check Global Settings
^2.2. Check Algebraic Loops
^2.3. Check Block Compatibility
^2.4. Check Sample Times
3. HDL Code Generation
3.1. Set Code Generation Options
3.1.1. Set Basic Options
3.1.2. Set Report Options
3.1.3. Set Advanced Options
3.1.4. Set Optimization Options
3.1.5. Set Testbench Options
^3.2. Generate RTL Code and Testbench
4. FPGA Synthesis and Analysis
4.1. Create Project
4.2. Perform Synthesis and P/R
4.3. Annotate Model with Synthesis Resu
Add Source...
Add Tcl...
Run This Task
Help
Apply

4.1. Create Project

Analysis

Create synthesis tool project

Input Parameters

Project folder: hdl_prj\quartus_prj

Synthesis objective: None

Additional source files: E_PHASE_BOOST_BIDIR_AC_DC_CONV_MAX10_restored\matlab\Simulink\M10\top.vhd;

Additional project creation Tcl files: R_AC_DC_CONV_MAX10_restored\matlab\Simulink\M10\add_to_project.tcl;

Result: ✓ Passed

```
Passed Create Project.

Task "Create Project" successful.

Generated logfile: hdl_prj\hdlsrc\bidir_rectifier\workflow_task_CreateProject.log

Info: ********************************************************************

Info: Running Quartus Prime TimeQuest Timing Analyzer

Info: Version 17.0.2 Build 602 07/19/2017 SJ Standard Edition
```

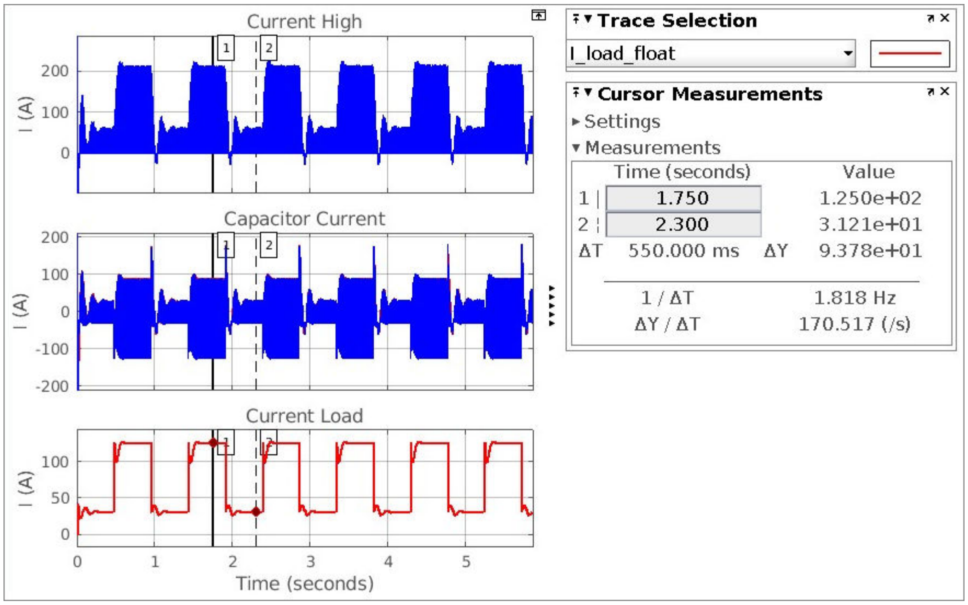
9. For **Additional source files**, select one of the following:
   - **For Intel MAX 10 FPGA Development Kit**: `<`*`local_path`*`>/M10/top.vhd`
   - **For Cyclone V SoC Development Kit**: `<`*`local_path`*`>/C5/top.vhd`
10. For **Additional project creation Tcl files**, select one of the following:
    - **For Intel MAX 10 FPGA Development Kit**: `<local>/matlab/Simulink/m10/add_to_project.tcl`
    - **For Cyclone V SoC Development Kit**: `<local>/matlab/Simulink/c5/add_to_project.tcl`
11. Run tasks 1, 2, 3, and 4.1 by clicking **Run This Task**.
12. Navigate to the project folder that you selected in step 7. The newly created Intel Quartus Prime software project is located in the `quartus_prj` directory and the generated VHDL code for the Simulink model is located in the `hdlsrc` directory.
13. Navigate to the `quartus_prj` directory and open the `bidir_rectifier_fixed_quartus.qpf` file with the Intel Quartus Prime software.
14. Compile the design, generate a SOF file, and program your board as described in Downloading and Installing the Design on page 8. Use Signal Tap logic analyzer to view the waveforms created by the generated VHDL code programmed in your board.

7. Simulink Simulation Results

This section shows the simulation results from the `bidir_rectifier.slx` model that you can find inside the `matlab/Simulink` directory.

The following figure shows how the variation in the output load immediately results in variation in the load current. By constantly changing the output load, you can evaluate the system response. This change is equivalent to having different vehicles connected to the charging station unit. In this simulation, the load current constantly changes from ~30 A to ~125 A to evaluate the output voltage response time.

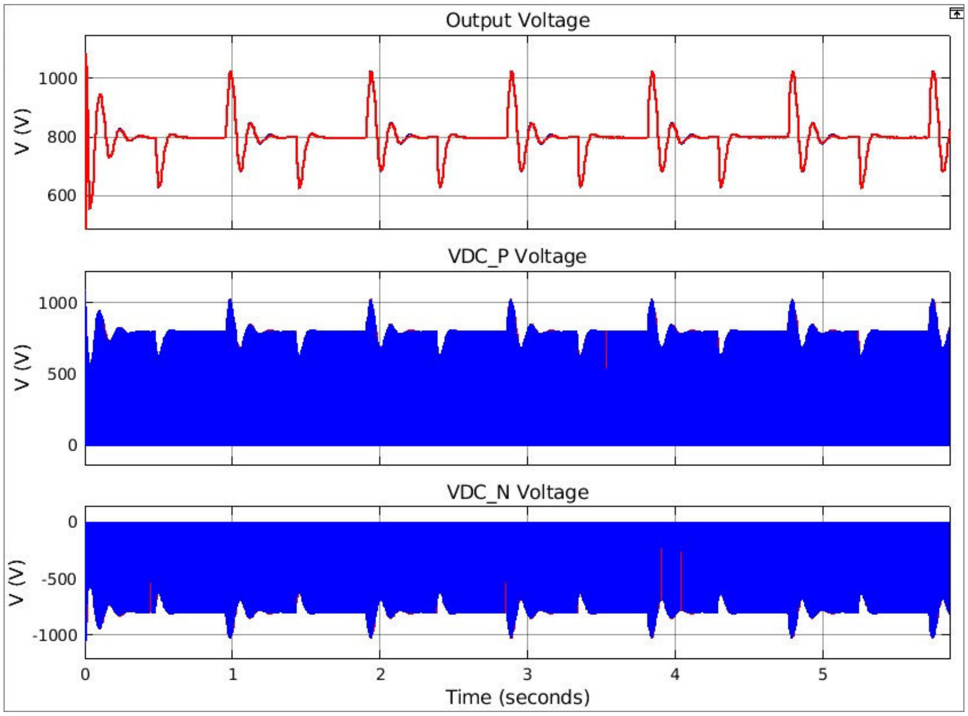
Figure 23. Load Changes in Simulink Simulation



	Time (seconds)		Value
1	1.750		1.250e+02
2	2.300		3.121e+01
ΔT	550.000 ms	ΔY	9.378e+01
	1 / ΔT		1.818 Hz
	ΔY / ΔT		170.517 (/s)

The following figure shows how the AC/DC converter keeps the output voltage around 800 V, as specified for the input reference voltage. Every time the load changes, the controller compensates for the changes and adjusts to meet the target reference voltage of 800 V. It shows that the three-phase boost bidirectional AC/DC converter for EV charger is well controlled when there is variation in the output load with reduced overshot and fast settling time.

Figure 24. AC/DC Converter Output Voltage Response



8. Document Revision History for AN 973: Three-phase Boost Bidirectional AC/DC Converter for EV Charging

Document Version	Intel Quartus Prime Version	Changes
2022.07.26	17.0	- Added links to Intel MAX 10 and Cyclone V design examples.
2022.06.23	17.0	- Initial version.