XAPP1204 (v1.0) June 18, 2014

Methods for Integrating AXI4-based IP Using Vivado IP Integrator

Author: Christopher Stillo, Duncan Mackay, Mike Mitchell

Summary

Vivado® IP Integrator is a next-generation high-level graphical design tool that can be used to integrate various IP blocks. Occasionally, logic adapters, sometimes referred to as glue logic or shims, are required for integrating IP blocks with other IP blocks, or with interconnect logic. Two methods are presented to ease the task of creating these shims: Vivado High-Level Synthesis (HLS) and the AXI4 peripheral creation feature. An example design is included, to illustrate these methods.

The reference designs used in this application note can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=359071

Introduction

The AXI4 integration methodology described here is applicable to many designs, but a specific example design is presented to illustrate these methods. The example design is an autonomous (no microprocessor interaction required) system for generating a sine-wave data pattern for the digital-to-analog (DAC). The data is then used as input data for the analog mixed-signal (AMS) evaluation card (AMS101) [Ref 1]. The Zynq®-7000 All Programmable SoC ZC702 Evaluation Kit [Ref 2] is the target hardware platform. The block diagram for the example design is shown in Figure 1.

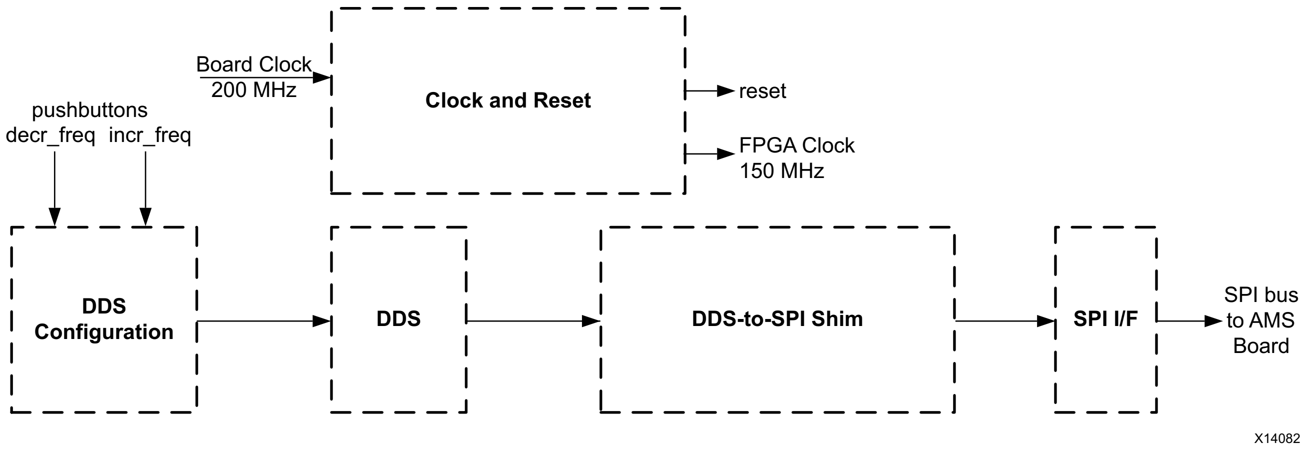


Figure 1: **Example Design Block Diagram**

An SPI bus is used as the communications link between the Zynq-7000 AP SoC on the ZC702 evaluation kit and the AMS evaluation card. A pushbutton interface is included to support dynamically increasing or decreasing the DDS frequency.

The AMS board features an Analog Devices AD5065 DAC [Ref 9]. The maximum SPI clock for the AD5065 is 50 MHz. An SPI clock frequency of 37.5 MHz is used in the design. The AD5065 requires a 2 μs wait time between DAC data updates; this wait time must be accounted for in the design. Each SPI access to the AD5065 is 32 bits, which takes approximately 0.850 μs at the chosen clock rate. A 3 μs DAC update rate was chosen for this design, based on the above (2 + 0.850 + margin = 3). Thus, the supported DAC sample rate is 333 KS/s.

At the center of this system is the Direct Digital Synthesizer (DDS) block, which generates sine wave data using a lookup table scheme. The DDS Configuration block configures the DDS at initialization (reset deassert), de-bounces the pushbutton inputs, and streams out a new word when a button push is detected. The DDS-to-SPI Shim block provides any data manipulation required for the sine-wave data, as well as handling the configuration of the SPI I/F.

The Clock and Reset block creates a 150 MHz FPGA clock, synthesized from the ZC702 200 MHz onboard clock. This block also generates a reset for the FPGA logic.

This application note illustrates the process of implementing and integrating this example design, using IP Integrator, System Generator, High-Level Synthesis (HLS), and the AXI4 Peripheral Creation wizard. As the design is fleshed out, an updated example block diagram is presented.

Using Vivado HLS for creating logic shims eliminates the need for you to implement the details of bus-level handshaking. With the HLS approach, you only need to describe (in C, C++, or SystemC language) the required data and address for each bus transaction.

An alternate to the HLS approach for implementing AXI shims is the RTL-based AXI4 Peripheral Creation wizard. In some instances, as in the pushbutton de-bouncer required in this design, RTL (synthesizable VHDL or Verilog) is a more appropriate choice for design entry. Using this wizard gives you a head start on the AXI4 interface design by providing example interface code and assistance with packaging the design for use in IP Integrator.

Implementation

This section describes the steps for implementing and integrating the example design are now described. The direct digital synthesizers (DDS), SPI I/F, and Clock and Reset blocks are fairly straightforward to implement because their logic can be generated using ready-made Xilinx® IP.

DDS Compiler

For DDS implementation, the Xilinx DDS compiler [Ref 3] provides a solution. The DDS compiler is available in the Vivado IP catalog, in IP Integrator, and in System Generator. For the example design, the System Generator DDS compiler was chosen to leverage the Mathworks simulation environment for functional verification and to illustrate the integration path between Xilinx System Generator and IP Integrator. The implemented System Generator model for this design is shown in Figure 2.

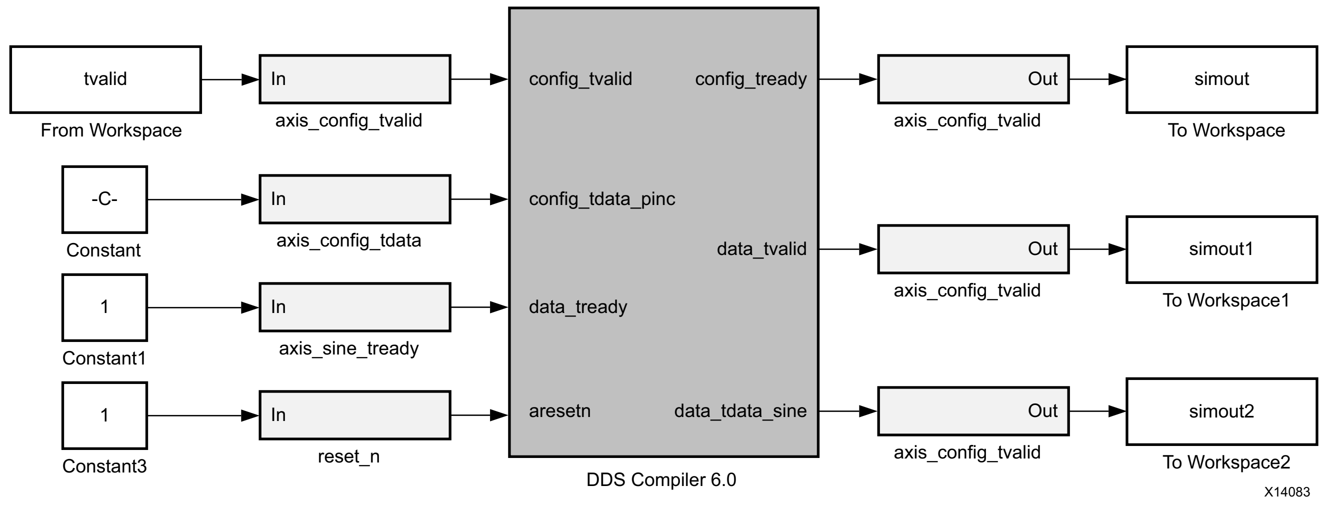


Figure 2: **DDS System Generator Model**

Once the System Generator model has been simulated with satisfactory results, the model is ready for integration. For a detailed example of how to include a System Generator design as a module in an IP Integrator design, see UG948, Lab 9 [Ref 4].

Note: The DDS block will generate 150 MS/s data, based on the 150 MHz core clock provided.

AXI Quad SPI

The AXI Quad SPI core [Ref 5] is a logical choice to act as the SPI I/F for handling communication with the SPI port of the AD5065 DAC. The Analog Mixed Signal (AMS) Targeted Reference Design (TRD) [Ref 6] provides an example of using this core for that purpose.

For the clock and reset blocks, the IP Integrator Clocking wizard [Ref 7] and the Processor System Reset [Ref 8] blocks can fill the needs.

Note: A processor is not required to use the Processor System Reset block.

Based on the these choices, you now have a partially implemented design, shown in Figure 3. The remainder of the section details the implementation of the remaining blocks. There are no ready-made IP blocks to perform the DDS configuration and DDS-to-SPI shim functions, so custom logic is required.

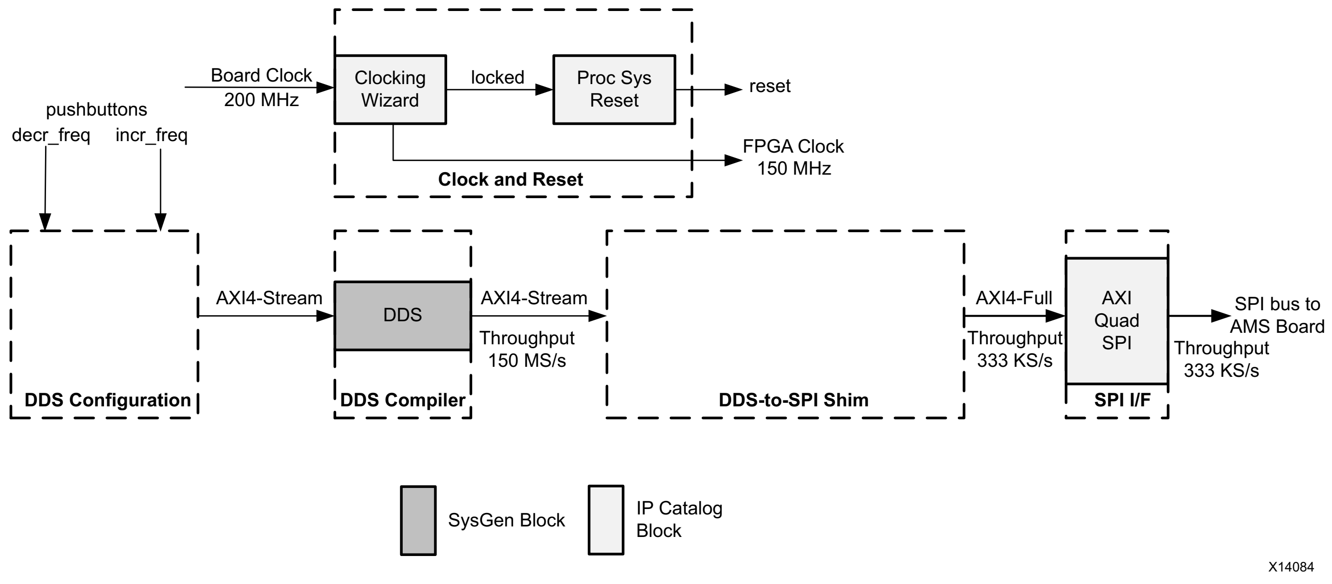


Figure 3: **Partially Implemented Example Design**

DDS Configuration Block

First consider the DDS Configuration block. The DDS core features an AXI4-Stream slave input for configuration. You can use this configuration port to dynamically change the frequency of the DDS core. As mentioned earlier, it is best to control the frequency using push-buttons. You can create an IP block with pushbutton inputs and an AXI-Stream master output to implement the DDS configuration function. An RTL-based design is chosen, as RTL is a good fit for implementing the desired de-bouncing of the pushbutton inputs. The AXI4 Peripheral Creation wizard is the tool of choice for generation of an RTL template for blocks containing AXI interfaces.

Figure 4 shows the dds_config_v1_0 IP Integrator block after it has been added to the IP Integrator block diagram. The config_register_value[31:0] input port specifies the starting value of the frequency configuration data. The config_register_delta[31:0] port specifies the value that is added/subtracted from the frequency configuration data each time a button push is detected. The incr_freq_pb and decr_freq_pb are from the pushbutton inputs. The M00_AXIS port is the AXI stream used to configure the DDS IP. The incr_freq_det and decr_freq_det are outputs of the debouncer and are for debug only.

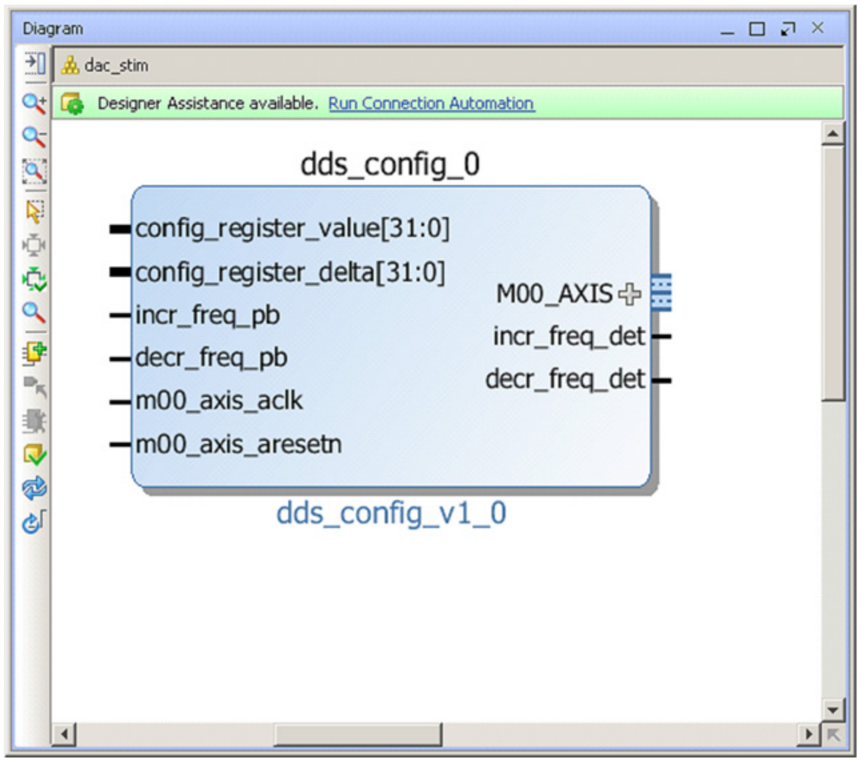


Figure 4: **dds_config_v1_0 IPI Block**

The example design with DDS configuration added is shown in Figure 5.

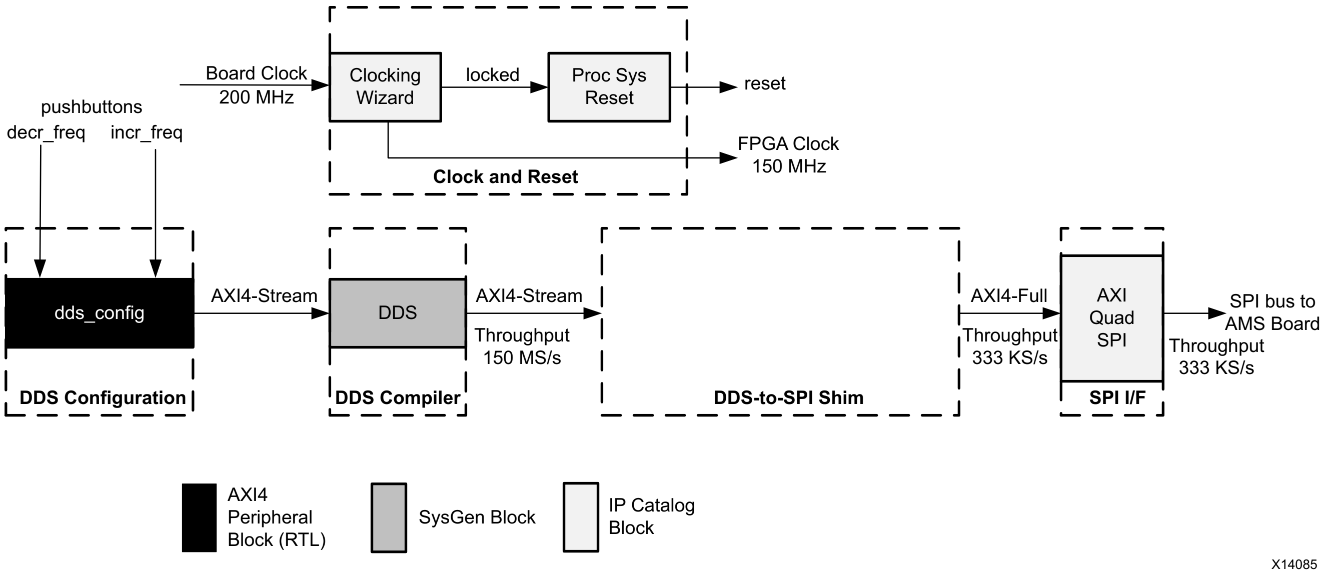


Figure 5: **Example Design with DDS Configuration Added**

DDS-to-SPI Shim Block

The last remaining block to implement is the DDS-to-SPI shim. The requirements for the DDS-to-SPI shim block as are follows:

- Configure the Quad SPI core. The AXI Quad SPI uses an AXI slave interface for both configuration and data, so this block must “virtually multiplex” the configuration and sine-wave data sources.
- The register settings are based on the C-code included in the Analog Mixed Signal (AMS) Targeted Reference Design (TRD) [Ref 6]. The configuration data is sent following the de-assertion of reset.
- Decimate the DDS data. The decimation is necessary because the DDS generates data at a 150 MS/s rate, and the maximum data rate that the DAC can accept, based on the chosen 3 μs update rate, is 333 KS/s. This decimation could be handled by the System Generator module but is performed in the HLS shim for illustrative purposes.
- Bridge an AXI4-Stream slave (DDS output) to an AXI4-Lite master (SPI Input).
- Convert bipolar DDS data to the DAC unipolar data format.
- Construct the DAC data word format by shifting the data from the input stream and adding the control code. The format of the data word is shown in the AD5065 DAC Specification. [Ref 9].

You can use Vivado HLS to create this IP Block. Creating an HLS function involves coding a C, C++, or SystemC module. C++ has been chosen for this example. The coding constructs for inferring various types of AXI4 ports are detailed in *Vivado Design Suite User Guide High-Level Synthesis* (UG902) [Ref 10] in the Interface Synthesis section.

Source Code

The source code for the HLS function is shown below.

The functionality is primarily described by the code in the switch statement. In addition, some internal counters and state variables are defined. These use data types defined in the header file `spi_axi_merge.h` which is included at the start of the code. Details on these data types can be found in the example design.

In addition to the functionality, the code contains several pragmas to direct Vivado HLS optimizations. In summary:

- The INTERFACE pragmas specify the I/O protocols for the design.
  - Some ports are specified to have no I/O protocol (simply be data ports).
  - The interface for variable *m* is specified to have an AXI-Master.
  - The interface for variable *data_i* is specified to be an AXI-Stream.
- The PIPELINE pragma ensure the design is pipelined and can accept a new input before the previous input has been output.
- The RESET pragma specifies that specific static variables be connect to the reset port (this is not the default).

Full details on these optimizations are available in the *Vivado Design Suite User Guide High-Level Synthesis* (UG902) [Ref 10].

```
//
// spi_axi_merge.c
//
// This HLS function generates AXI-4 master transactions to initialize and
// feed data to Xilinx IP core Quad SPI via its AXI-4-Lite port. See PG153
// for info on the core.
//
// The SPI port for this application is connected to AD5065 DAC on AMS Eval.
Board.
// After initializing the SPI core, this function reads an input AXI data
stream, formats
```

```
// the data word for use by AD5065 on AMS Eval. Board. and sends the 
formatted
// data word to the SPI core.

#include "spi_axi_merge.h"

void spi_axi_merge (volatile DT *m, STREAM_DT data_i[N],
    unsigned short decimation_factor, unsigned short *sample_count_out ) {
#pragma HLS PIPELINE II=1 enable_flush

// Define the RTL interfaces
#pragma HLS interface ap_stable port=sample_count_out
#pragma HLS interface ap_stable port=decimation_factor
#pragma HLS interface ap_ctrl_none port=return
#pragma HLS interface m_axi port=m

// AXI4-Stream slave interface
#pragma HLS interface axis port=data_i

STREAM_DT ddfs_data_bipolar;
DT ddfs_data_unipolar;
DT ddfs_data_unipolar_fmt;

static unsigned short sample_count = 1;
#pragma HLS RESET variable=sample_count
static unsigned char state = 0;
#pragma HLS RESET variable=state

switch (state)
{
  case 0: // SPI Control Register setup
    *(m+SPICR_OFFSET) = 0x4 | 0x8 | 0x2; // Master, CPOL, SPE
    state++;
    break;
  case 1: // SPI Slave select Register setup (active low)
    *(m+SPISSR_OFFSET) = 0xFFFE;
    state++;
    break;
  default: // read DDFS stream and decimate to meet DAC SPI bandwidth limit
    ddfs_data_bipolar = data_i[0];
    // convert bipolar DDFS output to unipolar for DAC
    ddfs_data_unipolar = ddfs_data_bipolar + 32768;
    // shift by 4 for DAC SPI format
    ddfs_data_unipolar_fmt = ddfs_data_unipolar << 4;

    if (sample_count < decimation_factor) {
      sample_count++;
    } else {
      sample_count = 1;
      *(m+SPIDTR_OFFSET) = (0x03000000 | ddfs_data_unipolar_fmt); // format 
input stream for DAC: write to channel A
    }

    break;
} // end switch

*sample_count_out = sample_count;

} // end of function
```

Note: The DDS-to-SPI shim requires an AXI protocol converter to convert the spi_axi_merge block AXI4-Full Master interface to an AXI4-Lite interface, to interface with the Quad SPI core. HLS does not currently support an AXI4-Lite master interface.

The fully implemented example design is shown in Figure 6.

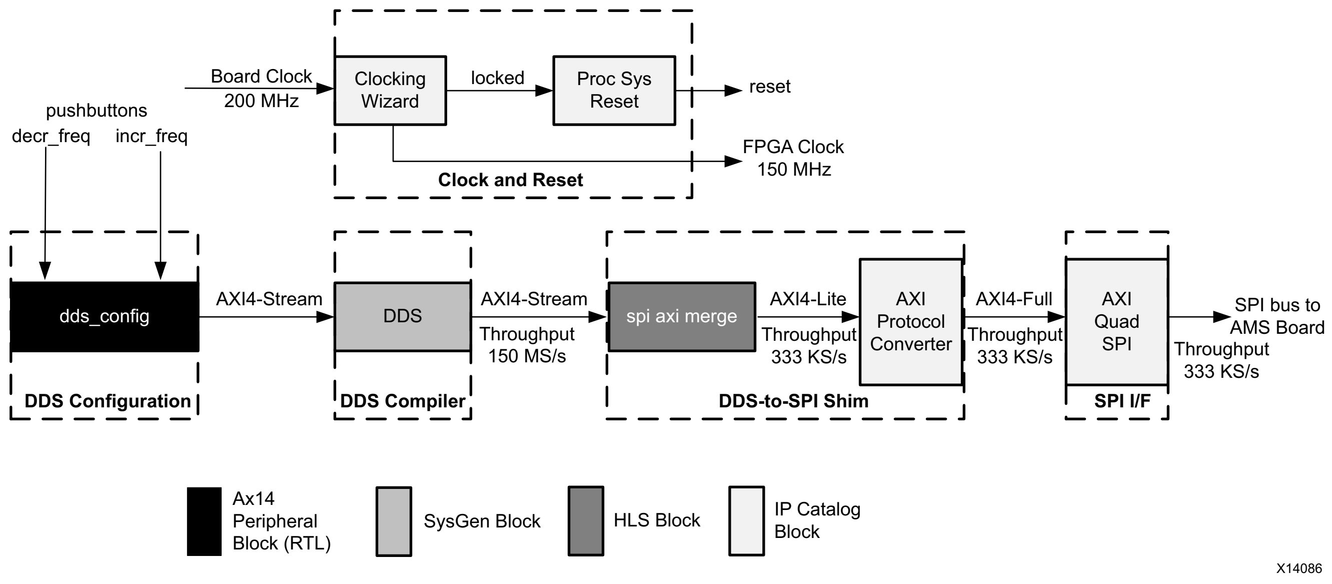


Figure 6: **Fully Implemented Example Design**

Reference Design or Sample Application

The accompanying ZIP file for this application note contains the files for the implemented example design, which targets the ZC702 Evaluation Kit and AMS101 evaluation card. The reference design is available at: https://secure.xilinx.com/webreg/clickthrough.do?cid=359071

Table 1 shows the reference design matrix.

Table 1: **Reference Design Matrix**

Parameter	Description
General	
Developer name	Christopher Stillo
Target devices (stepping level, ES, production, speed grades)	Zynq-7000 AP SoC (specifically ZC702)
Source code provided	Yes
Source code format	VHDL/Verilog; HLS; SysGen
Design uses code/IP from existing Xilinx application note/reference designs or third-party sources.	N/A
Simulation	
Functional simulation performed	Yes
Timing simulation performed	Simulation not supported
Test bench used for functional and timing simulations	Yes
Test bench format	Verilog
Simulator software/version	Vivado Simulator 2014.1

Table 1: **Reference Design Matrix *(Cont'd)***

Parameter	Description
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/version	Vivado Design Suite 2014.1
Implementation software tools/versions used	Vivado Design Suite 2014.1
Static timing analysis performed?	Yes (passing timing with Vivado implementation tools)
Hardware Verification	
Hardware verified?	Yes
Hardware platform used for verification	ZC702 Evaluation Kit

The top-level of the design is captured in an IP Integrator block diagram. The block diagram is shown in the Figure 7. The steps outlined below describe the method for re-generating the example design.

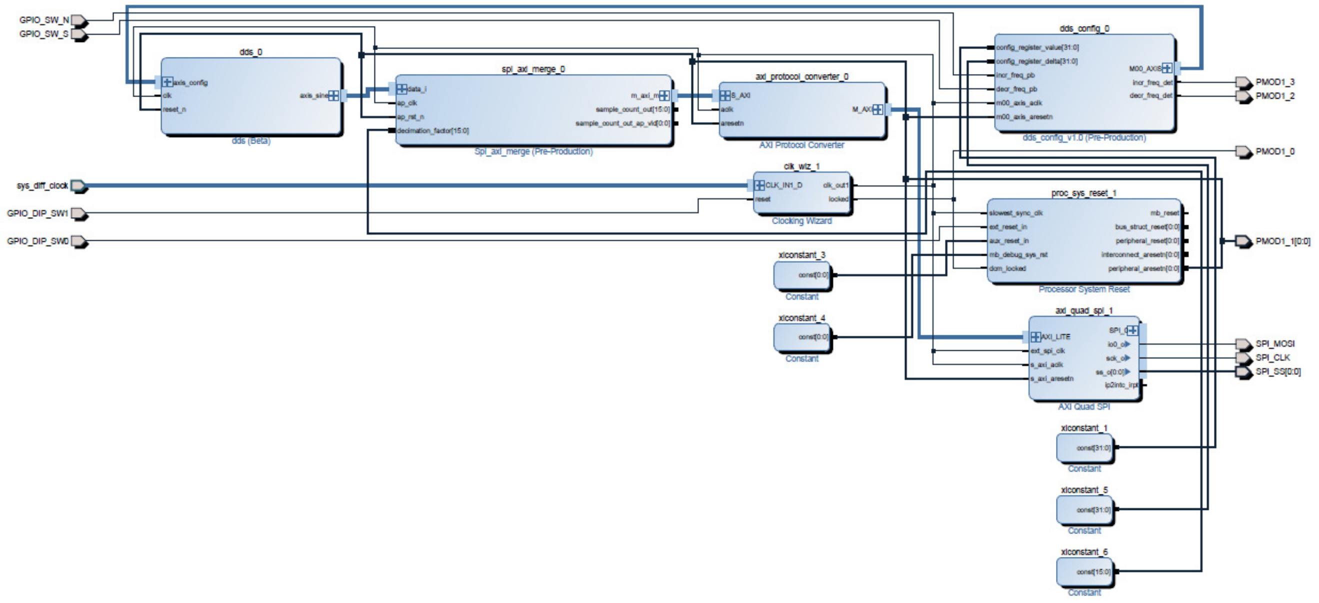


Figure 7: **Fully Implemented Example Design in IPI**

1. Generate three IP cores as follows:
   a. Appendix A details a step-by-step process for creating the DDS Configuration AXI4 Peripheral. An example of the completed IP for this block is located in the `./vivado/ip_repo` directory. The resulting IP is found in the `./axi_peripheral/dds_config_1.0` directory.
   b. To generate the System Generator DDS IP, go to the `./SysGen/dds.slx` model in MATLAB® Simulink® R2013a. Once the model is open, double-click the System Generator token and click **Generate**. The resulting IP is found in the `./SysGen/dds_netlist/ip` directory.
   c. To generate the HLS spi_axi_merge IP, go to the `./HLS` directory and launch Vivado HLS in interactive mode. Once at the Vivado HLS command prompt, source the `spi_axi_merge` Tcl script. The resulting IP is found in the `./HLS/spi_axi_merge_prj/solution1/impl/ip` directory.

2. Create the Vivado Design Suite project. To generate the project and IP Integrator block diagram:

   a. Open Vivado Design Suite 2014.1.

   b. From the Tcl console, change the directory to `./vivado`.

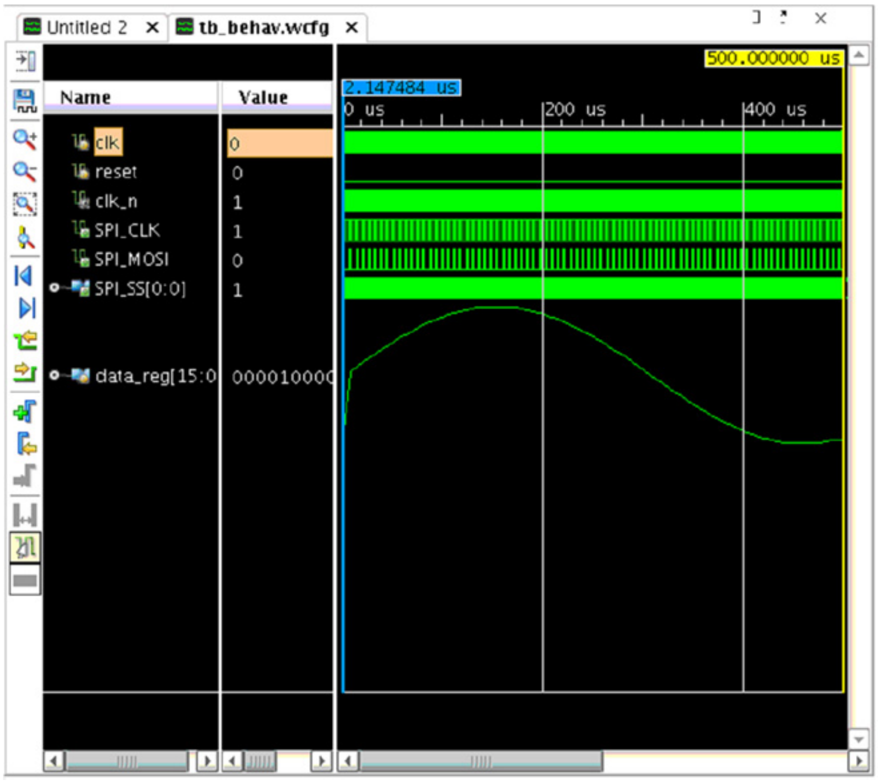
   c. Source the `./tcl/create_and_sim.tcl` script.

   This script performs the following tasks:

   - Creates a Vivado Design Suite project targeting the ZC702 Evaluation Kit.
   - Adds the IP repositories for the three IP blocks generated above to the Vivado Design Suite project.
   - Creates the IP Integrator block diagram and HDL wrapper file (by calling the `dac_stim_bd.tcl` script).
   - Adds test bench files.
   - Compiles and runs the design for Vivado simulation.

   The test bench contains an SPI bus slave model (`spi_slave.v`) which models the SPI interface of the DAC on the AMS Evaluation Card. This model contains a register 'data_reg' which represents the value of the data word passed to the DAC. Tracing the value of data_reg in simulation approximates the DAC action which occurs in hardware.

   A snapshot of the simulation output is shown in the Figure 8. The SPI bus signals are shown, as well as the value of the data_reg. Note the sine-wave shaped output of the data_reg signal. When the simulation run completes, make sure to apply **Zoom Fit** to the waveform, so that you can see the sine wave.



Name	Value
clk	0
reset	0
clk_n	1
SPI_CLK	1
SPI_MOSI	0
SPI_SS[0:0]	1
data_reg[15:0	00001000(

Figure 8: **Simulation Waveforms**

3. Creating a bitfile and testing it on the ZC702 platform is the final step in the implementation of the example design. In the Vivado project created above, you call the Generate Bitstream step to create the bitfile, and you use the Hardware Manager to program the ZC702 FPGA bitfile.

Table 2 illustrates the relevant ZC702 buttons, switches and LEDs.

Table 2: **ZC702 Evaluation Kit Pins, Buttons, and Switches**

FPGA Pin Name	ZC702 Reference Designator	Component Type	Example Design Function
GPIO_DIP_SW0	U12:1	DIP switch	System reset (active-High)
GPIO_DIP_SW1	U12:2	DIP switch	MMCM reset (active-Low)
GPIO_SW_N	SW5	Pushbutton switch	Frequency Increase
GPIO_SW_P	SW7	Pushbutton switch	Frequency Decrease
PMOD1_0	DS19	User LED	MMCM locked (active-High)
PMOD1_1	DS20	User LED	System reset active (active-Low)
PMOD1_2	DS21	User LED	Toggles when Frequency Decrease button push detected
PMOD1_3	DS22	User LED	Toggles when Frequency Increase button push detected

For normal operation of the example design, set reset switches to their inactive states (U12:1 = OFF and U12:2 = ON).

For hardware validation of the example design, probe the DAC output of the AMS101 evaluation card pin J5-1 with an oscilloscope, as shown in the Figure 9. Once you can see the output sine wave on the oscilloscope, you can make small adjustments to the frequency. Press the SW5 button to increase the frequency or press the SW7 button to decrease it.

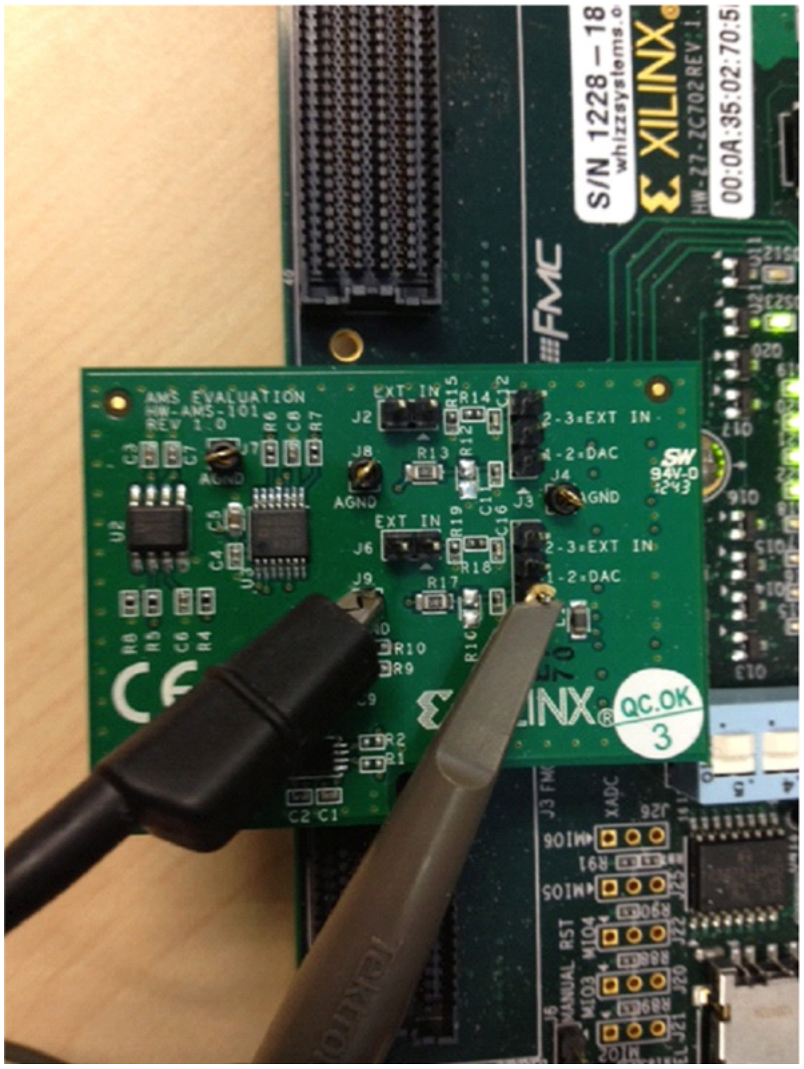


Figure 9: **Probing DAC Output on AMS Evaluation Card**

The DAC output on the oscilloscope is shown in Figure 10.

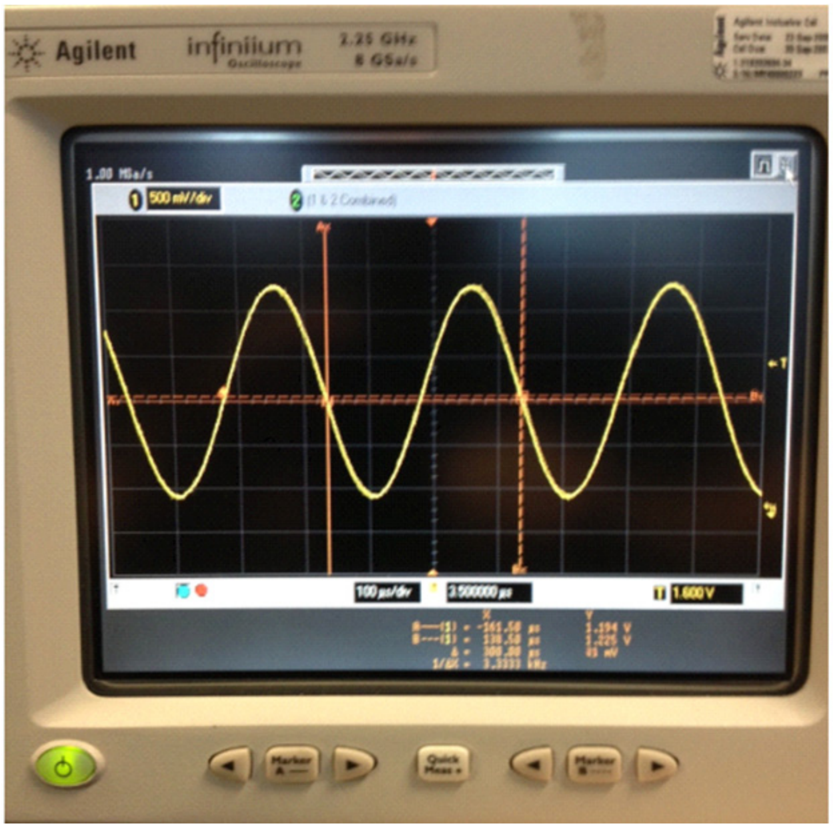


Figure 10: **AMS Evaluation Card DAC Output on Oscilloscope**

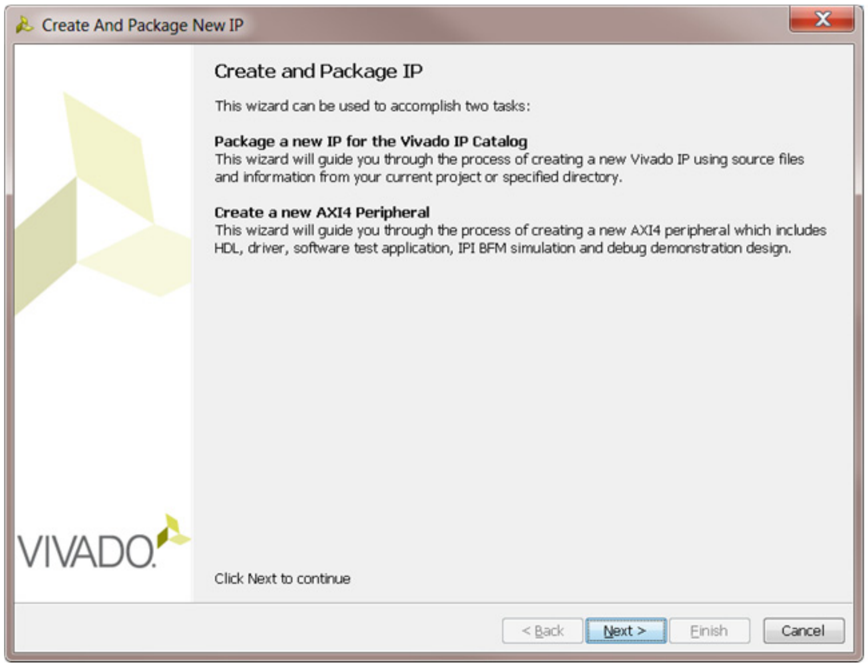
Conclusion

The application note describes the integration of AXI-based IP from Vivado HLS, System Generator, and AXI Peripheral Creation wizard. The methods outlined for generation of the example design demonstrate how you can use Vivado IP Integrator to integrate IP from many different sources. The Vivado System Generator, HLS, and AXI Peripheral Creation wizard provide turnkey methods for creating IP with AXI interfaces, and the ease of connecting these various AXI interfaces with IP Integrator provides a significant productivity boost versus hand-coding these connections in RTL.

Appendix A: AXI4 Peripheral Creation Wizard Example

The following example shows how to create an AXI4 Peripheral.

1. Open the AXI4 Peripheral Creation wizard from the **Vivado** > **Tools** > **Create and Package IP** menu. Figure 11 shows the first wizard screen.



Create and Package IP

This wizard can be used to accomplish two tasks:

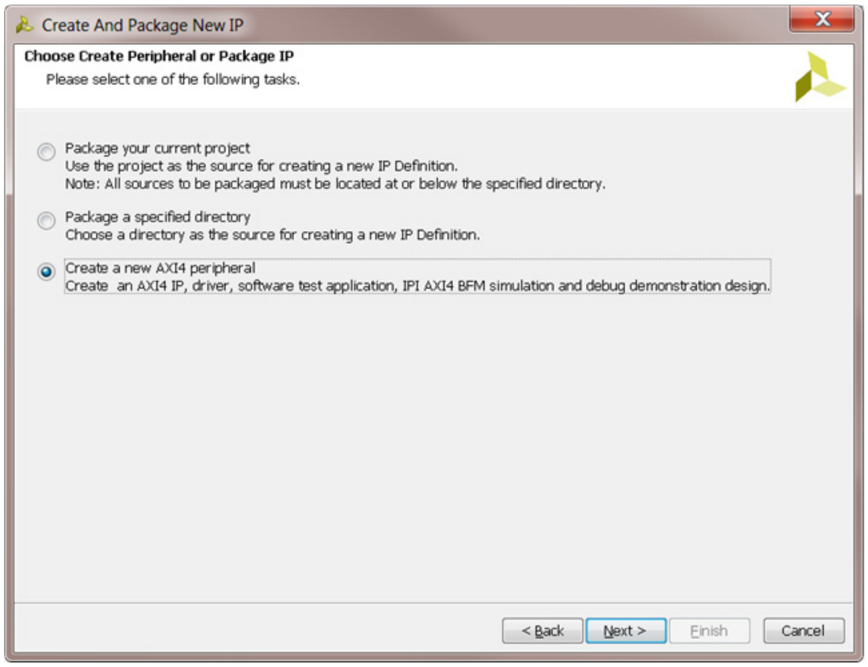
Package a new IP for the Vivado IP Catalog
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project or specified directory.

Create a new AXI4 Peripheral
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IPI BFM simulation and debug demonstration design.

Click Next to continue

Figure 11: **AXI4 Peripheral Creation**

2. Choose **Create new AXI4 Peripheral** on the next screen, as shown in Figure 12.



Choose Create Peripheral or Package IP
Please select one of the following tasks.

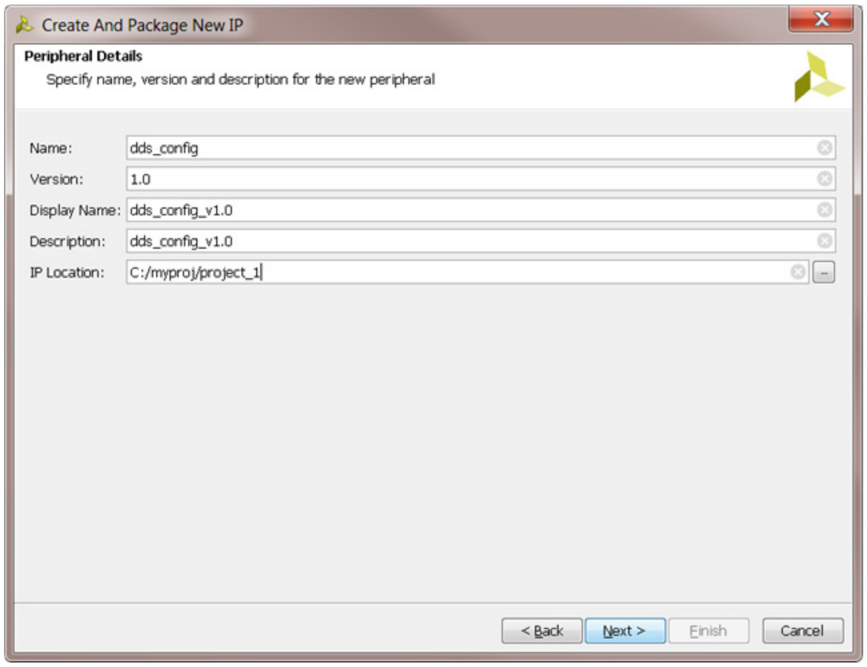
( ) Package your current project
Use the project as the source for creating a new IP Definition.
Note: All sources to be packaged must be located at or below the specified directory.

( ) Package a specified directory
Choose a directory as the source for creating a new IP Definition.

[✓ Create a new AXI4 peripheral]
Create an AXI4 IP, driver, software test application, IPI AXI4 BFM simulation and debug demonstration design.

Figure 12: **AXI4 Peripheral Creation**

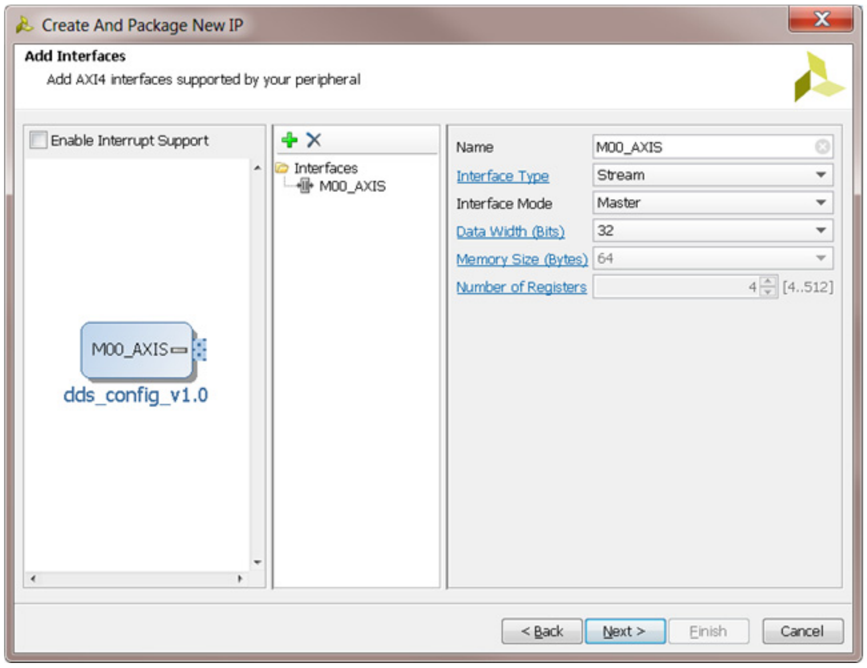
3. Peripheral details are shown in Figure 13.


Create And Package New IP
Peripheral Details
Specify name, version and description for the new peripheral
< Back
Next >
Finish
Cancel

Name: dds_config
Version: 1.0
Display Name: dds_config_v1.0
Description: dds_config_v1.0
IP Location: C:/myproj/project_1

Figure 13: **IP Packager Data for AXI4 Peripheral Creation**

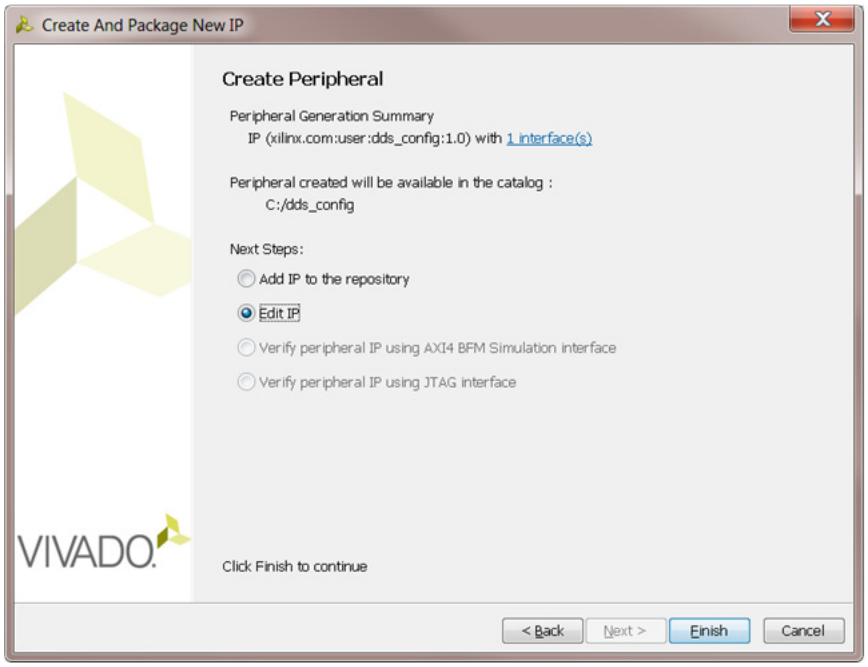
4. Change the default interface shown to Master AXI-Stream interface, as shown in Figure 14.



Name: M00_AXIS
Interface Type: Stream
Interface Mode: Master
Data Width (Bits): 32
Memory Size (Bytes): 64
Number of Registers: 4 [4..512]

Figure 14: **Interface Configuration for AXI4 Peripheral Creation**

5. Select **Edit IP** and select **Finish** as shown in Figure 15.



Create And Package New IP

Create Peripheral

Peripheral Generation Summary
IP (xilinx.com:user:dds_config:1.0) with 1 interface(s)

Peripheral created will be available in the catalog :
C:/dds_config

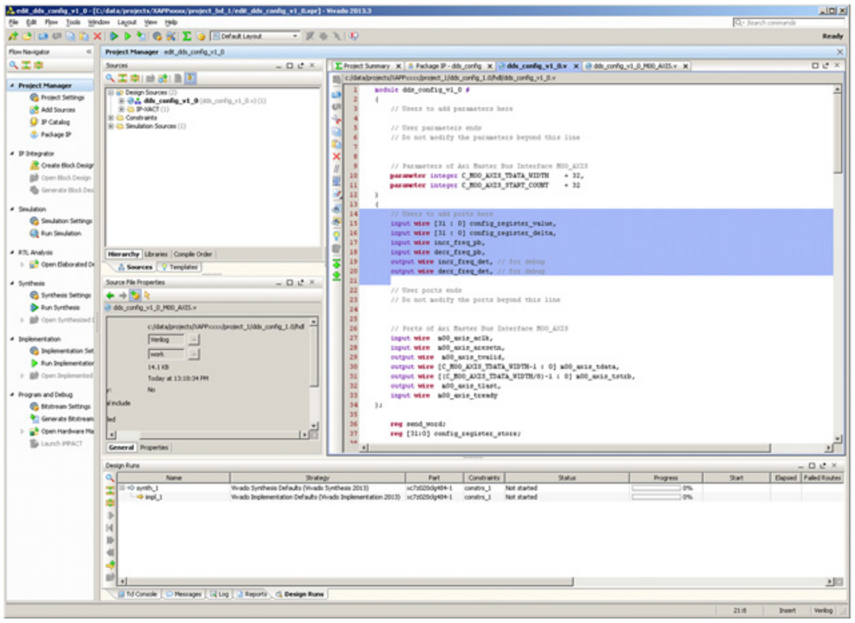
Next Steps:

( ) Add IP to the repository
(●) Edit IP
( ) Verify peripheral IP using AXI4 BFM Simulation interface
( ) Verify peripheral IP using JTAG interface

Click Finish to continue

Figure 15: **AXI4 Peripheral Creation**

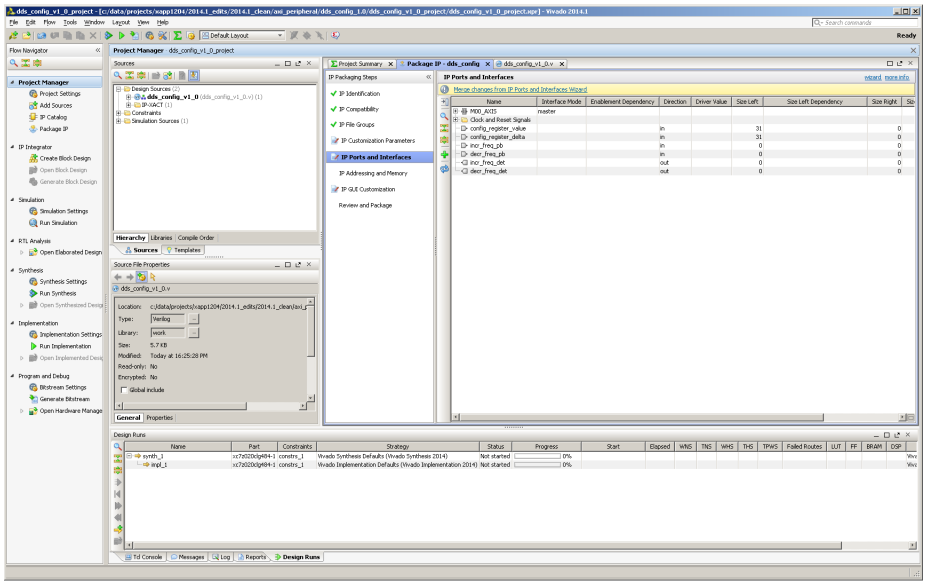
6. Figure 16 shows the opened IP project for the created AXI4 peripheral. Note the Sources window shows two created Verilog files: `dds_config_v1_0.v`, which is the top-level of the IP, and `dds_config_v1_0_M00_AXIS.v`, which contains template logic for the AXI-master stream. To accomplish the stated objectives for this block, edit `dds_config_v1_0.v` to add in the pushbutton ports and de-bouncers, and edit `dds_config_v1_0_M00_AXIS.v` to support sending a new data word on the stream each time a button push is detected. The included example design contains the completed Verilog files.



```
module dds_config_v1_0 #
(
    // Users to add parameters here

    // User parameters ends
    // Do not modify the parameters beyond this line


    // Parameters of Axi Master Bus Interface M00_AXIS
    parameter integer C_M00_AXIS_TDATA_WIDTH    = 32,
    parameter integer C_M00_AXIS_START_COUNT    = 32
)
(
    // Users to add ports here
    input wire [31 : 0] config_register_value,
    input wire [31 : 0] config_register_delta,
    input wire incr_freq_pb,
    input wire decr_freq_pb,
    output wire incr_freq_det, // for debug
    output wire decr_freq_det, // for debug

    // User ports ends
    // Do not modify the ports beyond this line


    // Ports of Axi Master Bus Interface M00_AXIS
    input wire  m00_axis_aclk,
    input wire  m00_axis_aresetn,
    output wire  m00_axis_tvalid,
    output wire [C_M00_AXIS_TDATA_WIDTH-1 : 0] m00_axis_tdata,
    output wire [(C_M00_AXIS_TDATA_WIDTH/8)-1 : 0] m00_axis_tstrb,
    output wire  m00_axis_tlast,
    input wire  m00_axis_tready
);

    reg send_word;
    reg [31:0] config_register_store;
```

Figure 16: **RTL Editing the AXI4 Peripheral**

7. After RTL editing is complete, package the design for use in IP Integrator. Since additional ports have been added to the IP block, revisit the Package IP settings for the ports. Select **Merge changes from IP Ports Wizard** to automatically update the ports list for the packager. Figure 17 shows the IP Ports tab.



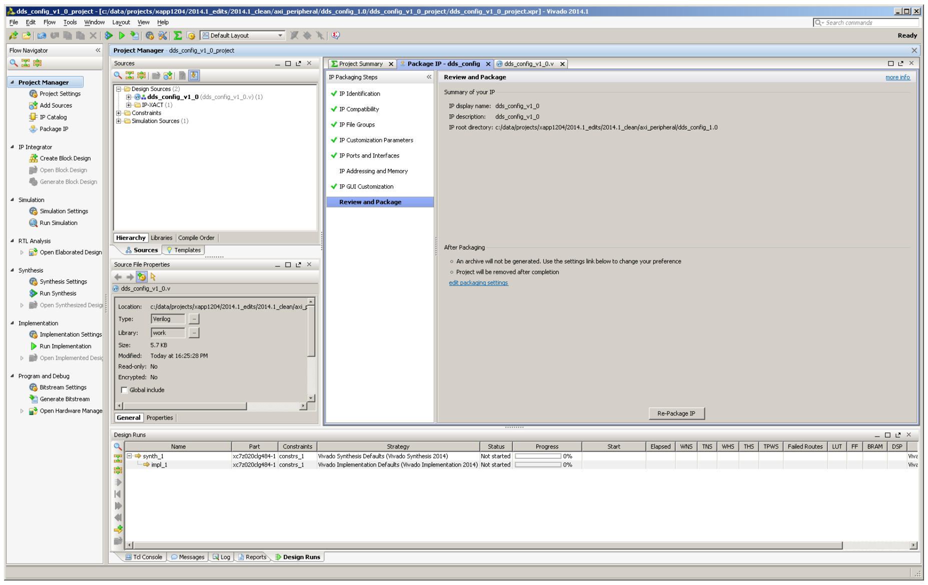
IP Ports and Interfaces

Merge changes from IP Ports and Interfaces Wizard

Name	Interface Mode	Enablement Dependency	Direction	Driver Value	Size Left	Size Left Dependency	Size Right
M00_AXIS	master						
Clock and Reset Signals							
config_register_value			in		31		0
config_register_delta			in		31		0
incr_freq_pb			in		0		0
decr_freq_pb			in		0		0
incr_freq_det			out		0		0
decr_freq_det			out		0		0

Figure 17: **Updating the Packaged Ports List for the AXI4 Peripheral**

8. Now that the AXI4 peripheral is ready for packaging, select **Review and Package** to create the IP block for IP Integrator. See Figure 18.



Review and Package

Summary of your IP

IP display name: dds_config_v1_0
IP description: dds_config_v1_0
IP root directory: c:/data/projects/xapp1204/2014.1_edits/2014.1_clean/axi_peripheral/dds_config_1.0

After Packaging

- An archive will not be generated. Use the settings link below to change your preference
- Project will be removed after completion

edit packaging settings

Figure 18: **Packaging the AXI4 Peripheral**

References

This document uses the following references.

1. AMS101 Evaluation Card User Guide (UG886)
2. ZC702 Evaluation Board User Guide (UG850)
3. LogiCORE IP DDS Compiler v6.0 Product Guide for Vivado Design Suite (PG141)
4. Vivado Design Suite Tutorial Model-Based DSP Design using System Generator (UG948)

5. LogiCORE IP AXI Quad SPI v3.1 Product Guide for Vivado Design Suite (PG153)
6. Analog Mixed Signal (AMS) Targeted Reference Design (RDF0277)
7. LogiCORE IP Clocking Wizard v5.1 Product Guide for Vivado Design Suite (PG065)
8. LogiCORE IP Processor System Reset Module v5.0 Product Guide for Vivado Design Suite (PG164)
9. AD5065 DAC specification (Analog Devices)
10. Vivado Design Suite User Guide High-Level Synthesis (UG902)

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
06/18/2014	1.0	Initial Xilinx release.

Notice of Disclaimer