SAN DIEGO STATE UNIVERSITY

Fourier Interpolation on an FPGA

Department of Electrical and Computer Engineering
Signal Processing Research Laboratory

Danny Toma and Ashkan Ashrafi

Table of Contents

Introduction

This paper describes the implementation of a real-time interpolator on an FPGA. The interpolator utilizes a method based on the 1-D Discrete Cosine Transform (DCT). The software required for this includes Xilinx ISE 14.4 and MATLAB. Knowledge of Verilog HDL and familiarity with MATLAB are required. In addition, experience with the Discrete Fourier Transform is required to understand the design. The physical requirements for this project are,

- Virtex 5 ML506 *with Xilinx JTAG programmer*
- Diligent Pmod1 AD1
- Diligent Pmod1 DA2
- Oscilloscope
- Function Generator

Theory

Interpolation is the construction of new data points, based on the given discrete points of a signal. The method used to interpolate in this design is based on the Discrete Cosine Transform (DCT). A signal that is band limited can be exactly interpolated by taking the Discrete Fourier Transform (DFT) and padding zeroes to the end of the spectrum. If one mirror extends the data points prior to taking the DFT, it would be equivalent to the Type-II DCT. The equations for the DCT and the inverse DCT are

$$X_c[k] = 2\sum_{n=0}^{M-1} x[m] * cos\left(\frac{\pi k(2m+1)}{2M}\right) ,\ \mathrm{k} = 0, 1, \ldots \mathrm{M} - 1 \quad (1)$$

$$x[m] = \frac{1}{M}\sum_{k=0}^{M-1} X_c[k] * b[k] * cos\left(\frac{\pi k(2m+1)}{2M}\right),\ \mathrm{m} = 0, 1, \ldots \mathrm{M} - 1 \quad (2)$$

where $b[0] = \frac{1}{2}$ and $b[k] = 1$ for $k = 1, 2 \ldots M - 1$.

In some texts, these formulas rearrange the constants at the beginning of the equation to make the transform orthogonal; however, for this design, it is not necessary. In order to interpolate, we must take the forward DCT of a signal, and then take the inverse DCT of the zero-padded transform. We can represent the forward DCT as a matrix with m in the columns and k in the rows,

$$2\begin{bmatrix} \frac{cos(0)}{2} & \frac{cos(0)}{2} & . & . & \frac{cos(0)}{2} \\ cos(\frac{\pi}{2M}) & cos(\frac{3\pi}{2M}) & cos(\frac{5\pi}{2M}) & . & . \\ . & cos(\frac{6\pi}{2M}) & . & . & . \\ . & . & . & . & . \\ cos(\frac{(M-1)\pi}{2M}) & . & . & . & cos(\frac{\pi(M-1)(2(M-1)+1)}{2M}) \end{bmatrix}$$

By multiplying this matrix by the extended inverse DCT matrix, the outcome is a matrix that may be multiplied by the sampled sequence to achieve an interpolated sequence (as will be seen in (4)). First, however, we must modify the inverse DCT to fit our application. We can zero-pad the DCT by only utilizing a summation of M points and multiplying the denominator of the cosine by N. Also, we apply a shift to the cosines in order to have x[n] = x[mN]. The resultant IDCT equation is

$$x[n] = \frac{1}{M}\sum_{k=0}^{M-1} X_c[k] * cos\left(\frac{\pi k(2n+N)}{2*N*M}\right),\ n = 0, 1, \ldots\ N * M - 1 \quad (3)$$

where N is an integer and the scaling factor.

The matrix for the inverse transform is

$$\frac{1}{M}\begin{bmatrix} cos(0) & cos(\frac{N\pi}{2NM}) & cos(\frac{2N\pi}{2NM}) & . & cos(\frac{(M-1)N\pi}{2NM}) \\ cos(0) & cos(\frac{(2+N)\pi}{2NM}) & cos(\frac{(2+N)2\pi}{2NM}) & . & . \\ . & cos(\frac{(4+N)\pi}{2NM}) & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ cos(0) & . & . & . & cos(\frac{\pi(M-1)(2(MN-1)+N)}{2NM}) \end{bmatrix}$$

If **A** is the DCT matrix and **B** is the inverse DCT matrix, then x[m], a row vector of M samples, can be interpolated by doing the multiplication in Equation 4 and x[n] becomes an M*Nx1 row vector.

$$x[n] = \frac{2}{M}(\mathbf{BA})x[m] \qquad (4)$$

Design Considerations

In order to interpolate in real time, the processing is done in frames. As explained in the previous Theory section, the size of the frame is M. For this implementation, M is fixed to four. Another aspect of the design that must be considered is the upscaling factor, N. In this design, the number N is user defined and parametric. It should be noted that N must be an integer. For the demonstration, we assign the upscaling factor, N, to four. If a different upscaling factor is desired, simply assign it to N in the top module and carry out the rest of the procedure with the corresponding factor in mind. An example of the processing frame is illustrated in the **Figure 1**.

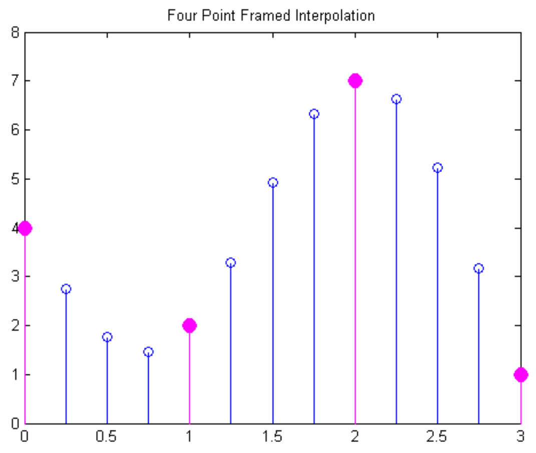


Figure 1:Four point processing frame using DCT/IDCT

In the design from **Figure 1**, four DCT coefficients are needed for every newly constructed data point. The design implements N*3 impulse responses, each of which are specifically dedicated to a point on the new, upsampled sequence. These sub-responses can be represented by a row of four points from the resulting matrix of Equation 4. The responses are as follows,

$$H(z) = \begin{array}{ccccccc} h_0[0] & + & h_0[1]z^{-1} & + & h_0[2]z^{-2} & + & h_0[3]z^{-3} \\ h_1[0] & + & h_1[1]z^{-1} & + & h_1[2]z^{-2} & + & h_1[3]z^{-3} \\ \vdots & & \vdots & & \vdots & & \vdots \\ h_{3N-1}[0] & + & h_{3N-1}[1]z^{-1} & + & h_{3N-1}[2]z^{-2} & + & h_{3N-1}[3]z^{-3} \end{array}$$

The 4 point processing frame requires 4 coefficients for every row; thus, a total of N*3*4 coefficients are needed. We can reduce the number of coefficients by refreshing the frame every sample and only interpolating on the final data points. Specifically, interpolants are only inserted between x[2] and x[3] (the original third and fourth samples). To do this, we can utilize rows 2*N to 3*N-1. The rows for the processing frame are as follows,

$$H(z) = \begin{array}{ccccccc} h_{2N}[0] & + & h_{2N}[1]z^{-1} & + & h_{2N}[2]z^{-2} & + & h_{2N}[3]z^{-3} \\ h_{2N+1}[0] & + & h_{2N+1}[1]z^{-1} & + & h_{2N+1}[2]z^{-2} & + & h_{2N+1}[3]z^{-3} \\ \vdots & & \vdots & & \vdots & & \vdots \\ h_{3N-1}[0] & + & h_{3N-1}[1]z^{-1} & + & h_{3N-1}[2]z^{-2} & + & h_{3N-1}[3]z^{-3} \end{array}$$

This means that only N*4 coefficients are needed rather than N*4*3. Basically, the design is a moving version of the illustration in **Figure 2**.

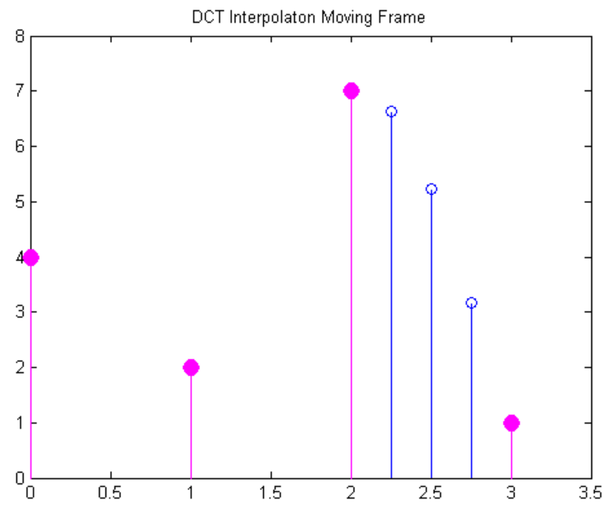


Figure 2:Processing frame between x[2] and x[3]

To verify, we have applied a test signal and interpolated based on the moving four point frame from **Figure 2**. **Figure 3** shows an example of the original signal, x[m] and the interpolated version, x[n].

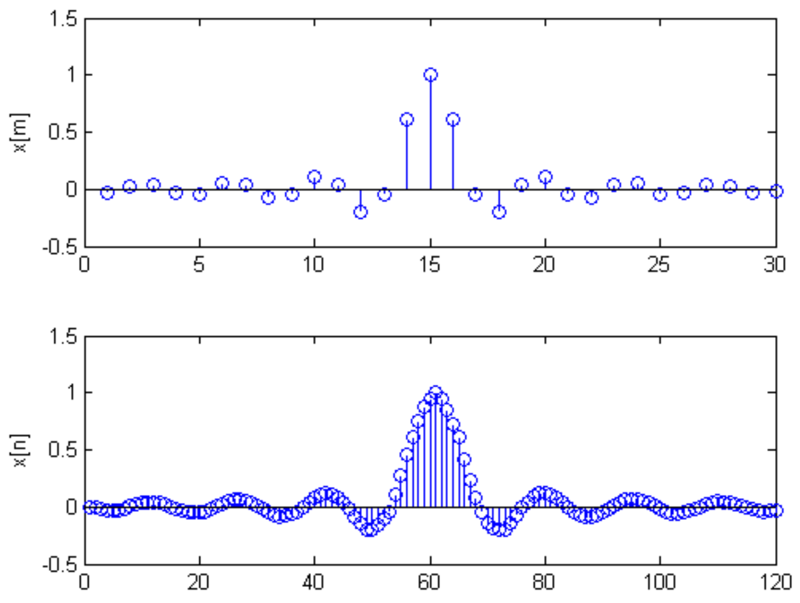


Figure 3:Test signal (above) and interpolated signal using moving frame (below)

Another parameter that must be set is SIZE. This is equal to the number of bits needed to represent the integer N. For the demonstration, N is four; thus, we only need SIZE to be equal to 2 bits. Again, this is set in the top module.

Conversion and Sampling Rates

Two external devices interface with the ML506 board. The PMOD AD1 and PMOD DA2 act as the input and output of the overall system. In order to digitize a signal, the PMODAD1 is used. It contains two AD7476A chips from Analog Devices. The DAC121S101 chip from National Semiconductor is used on the PMODDA2 to output the signal and the new data points. Both devices use Serial Peripheral Interface (SPI) and the module DAC_OUT is used to interface with the external boards.

The clocking speed on board of the ML506 is 100MHz. Both PMOD boards can only be clocked at a maximum frequency of 20 MHz. To be well within spec, we can bring the clock speed down to the 10 MHz. This will be done using the clock manager in a following section. A single conversion for both the ADC and the DAC takes at least 28 external clock cycles in this design. The DAC_OUT module gives the user the ability to change the conversion frequency by setting the parameter WAIT_TIME in the top module. This parameter must be calculated by the formula below. It should be noted that the sampling frequency of the output is the *Desired Frequency* from the equation below. For the ADC frequency, simply divide the *Desired Frequency* by the scaling factor, N. For the demonstration, the desired frequency is 192 KHz so the parameter WAIT_TIME is set to 24.

$$Desired\ Frequency = \frac{10\times10^{6}}{28+\text{WAIT\_TIME}}\ Hz \tag{5}$$

Setup

At this point, we are ready to implement the design. Power up the ML506 and find a Xilinx JTAG programmer to connect to the desktop and the FPGA. The PMODs will be placed on the J6 header pins. The AD1 will have pin 1 going to pin 38 on the ML506 and the DA2 will have pin 1 going to pin 52. This is illustrated in **Figure 4**.

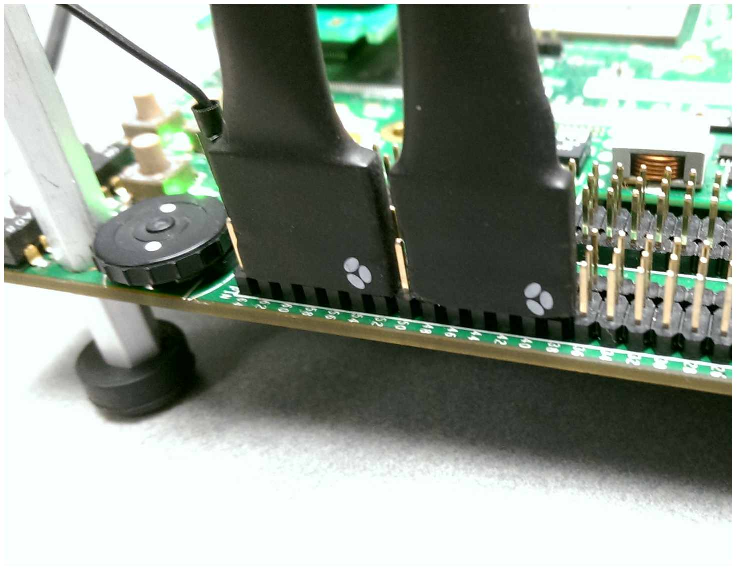


Figure 4:J6 headers on ML506 and PMOD connectors

Connect the ground to the designated pins on both boards. The ML506 has a ground pin in the last row, of the center column on the J6 header. Also, there are pins that output 3.3V. Jump these to the circuit to make the correct connections to the ADC and DAC. Next, connect the function generator and scope probes to A0 (first pin of the ADC), D0 (first pin of DAC), and D1 (third pin of the DAC) respectably. The setup is illustrated in **Figure 5** and **Figure 6**.

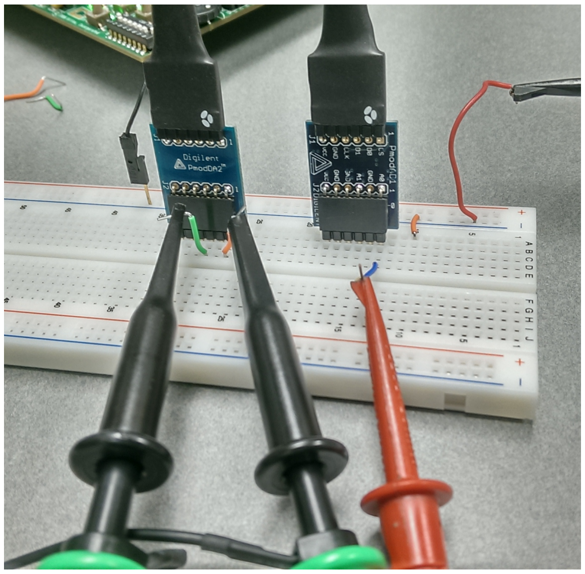


Figure 5:Scope probes (black) connected to DAC and function generator (red) connected to ADC

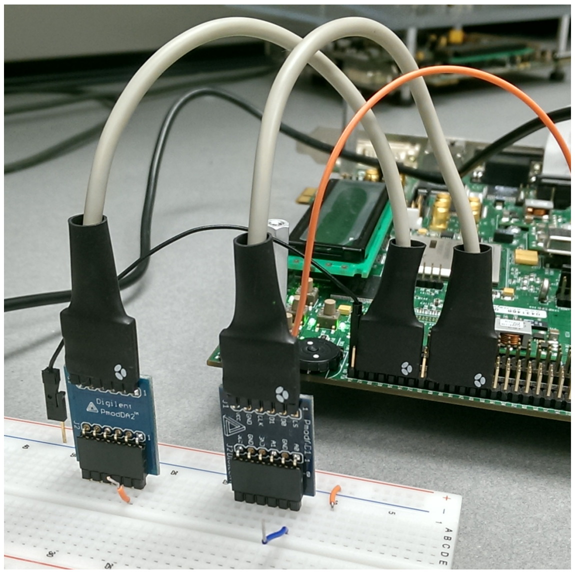
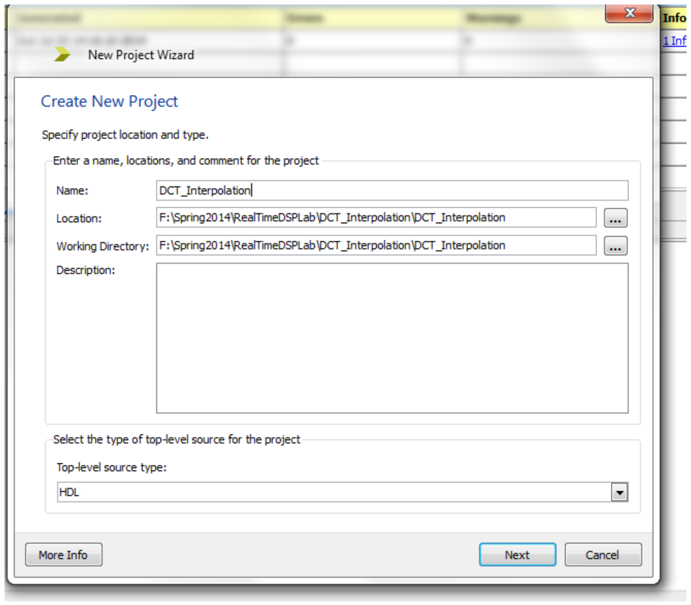


Figure 6:Board setup and ML506 orientation

Implementation

Now that we have the physical setup, we open up Xilinx ISE to implement the design. Create a designated folder for the modules in any appropriate directory. In Xilinx, select File, New Project. The window in **Figure 7** will pop up.


New Project Wizard

Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name: DCT_Interpolation

Location: F:\Spring2014\RealTimeDSPLab\DCT_Interpolation\DCT_Interpolation

Working Directory: F:\Spring2014\RealTimeDSPLab\DCT_Interpolation\DCT_Interpolation

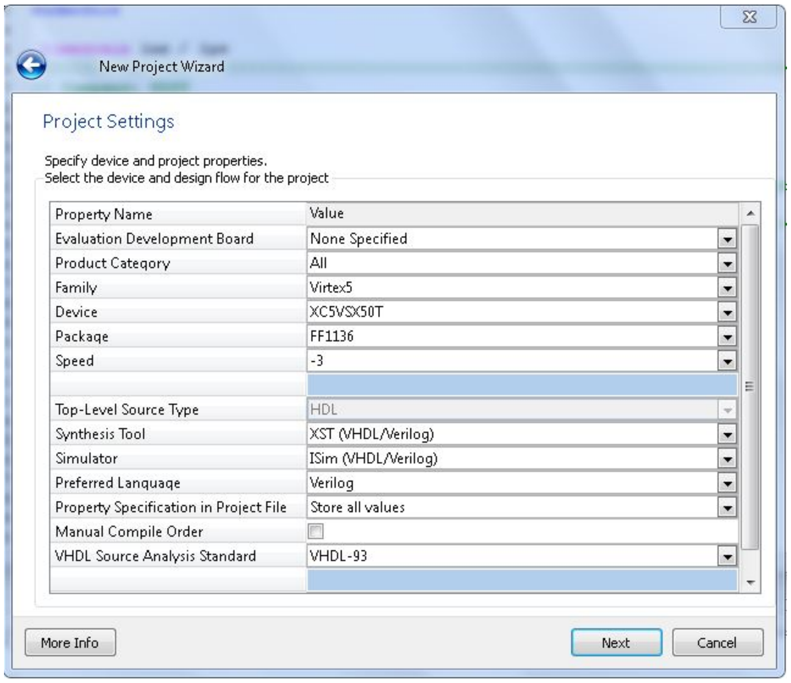
Description:

Select the type of top-level source for the project

Top-level source type: HDL

Figure 7:New Project Wizard

Name the project DCT_Interpolation. Match the project settings with the ones illustrated in **Figure 8.**


New Project Wizard

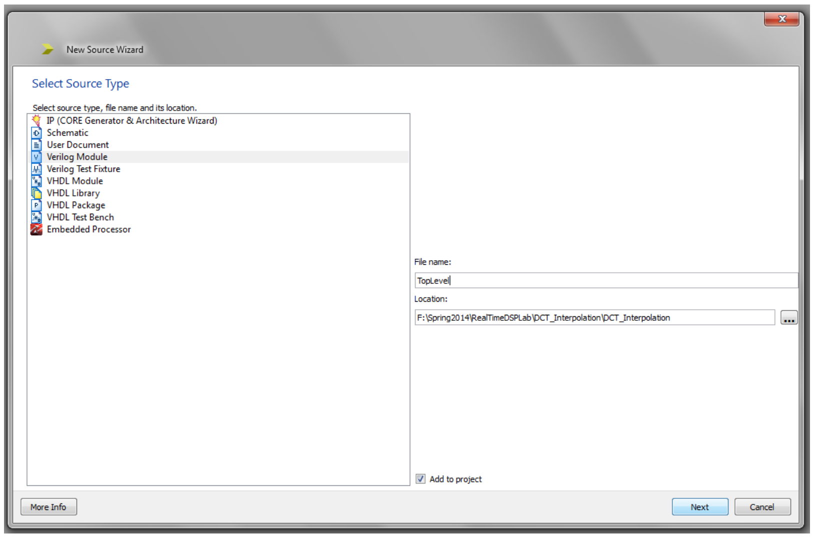
Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Virtex5
Device	XC5VSX50T
Package	FF1136
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93

Figure 8:Project settings

Add a Verilog Module by selecting Project, New Source. A window will appear. Select Verilog Module and name it TopLevel as shown in **Figure 9**.



Select source type, file name and its location.

File name: TopLevel

Location: F:\Spring2014\RealTimeDSPLab\DCT_Interpolation\DCT_Interpolation

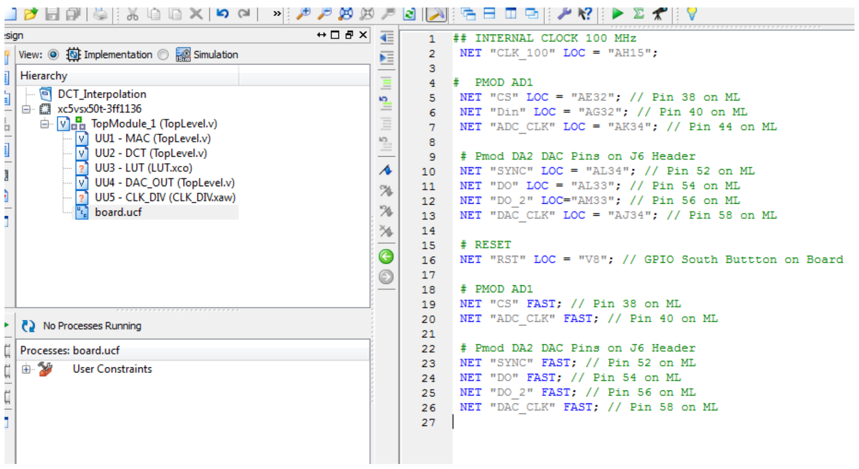
☑ Add to project

Figure 9:New verilog source for top module

Copy the **DCT INTERPOLATION** code from the Appendix and paste it into the module. After saving the code, it is evident that there seems to be missing modules from the top level. This is where the coefficient look up table and clock divider will be. Prior to inserting these, however, we can map the IO's to our design by adding a User Constraints File. Select Project, New Source, and select Implement Constraints File. Name the file "board". Copy and paste the following code for the .ucf file.

```
## INTERNAL CLOCK 100 MHz
 NET "CLK_100" LOC = "AH15";

#  PMOD AD1
 NET "CS" LOC = "AE32"; // Pin 38 on ML
 NET "Din" LOC = "AG32"; // Pin 40 on ML
 NET "ADC_CLK" LOC = "AK34"; // Pin 44 on ML

 # Pmod DA2 DAC Pins on J6 Header
 NET "SYNC" LOC = "AL34"; // Pin 52 on ML
 NET "DO" LOC = "AL33"; // Pin 54 on ML
 NET "DO_2" LOC="AM33"; // Pin 56 on ML
 NET "DAC_CLK" LOC = "AJ34"; // Pin 58 on ML

 # RESET
 NET "RST" LOC = "V8"; // GPIO South Buttton on Board

 # PMOD AD1
 NET "CS" FAST; // Pin 38 on ML
 NET "ADC_CLK" FAST; // Pin 40 on ML

 # Pmod DA2 DAC Pins on J6 Header
 NET "SYNC" FAST; // Pin 52 on ML
 NET "DO" FAST; // Pin 54 on ML
 NET "DO_2" FAST; // Pin 56 on ML
 NET "DAC_CLK" FAST; // Pin 58 on ML
```

When saved, the project should look like **Figure 10**.

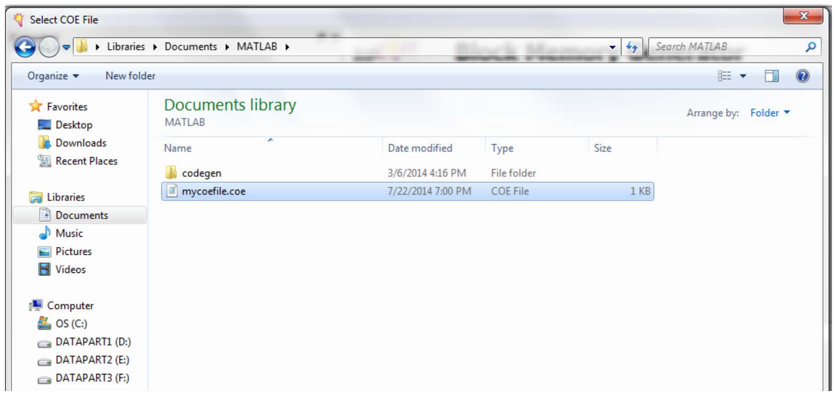


```
## INTERNAL CLOCK 100 MHz
 NET "CLK_100" LOC = "AH15";

#  PMOD AD1
 NET "CS" LOC = "AE32"; // Pin 38 on ML
 NET "Din" LOC = "AG32"; // Pin 40 on ML
 NET "ADC_CLK" LOC = "AK34"; // Pin 44 on ML

 # Pmod DA2 DAC Pins on J6 Header
 NET "SYNC" LOC = "AL34"; // Pin 52 on ML
 NET "DO" LOC = "AL33"; // Pin 54 on ML
 NET "DO_2" LOC="AM33"; // Pin 56 on ML
 NET "DAC_CLK" LOC = "AJ34"; // Pin 58 on ML

 # RESET
 NET "RST" LOC = "V8"; // GPIO South Buttton on Board

 # PMOD AD1
 NET "CS" FAST; // Pin 38 on ML
 NET "ADC_CLK" FAST; // Pin 40 on ML

 # Pmod DA2 DAC Pins on J6 Header
 NET "SYNC" FAST; // Pin 52 on ML
 NET "DO" FAST; // Pin 54 on ML
 NET "DO_2" FAST; // Pin 56 on ML
 NET "DAC_CLK" FAST; // Pin 58 on ML
```

Figure 10: Project after saving .ucf file

Generating Coefficients

The implementation is similar to a four tap FIR filter; however, we need to have coefficients for each index rather than one set impulse response. These coefficients are based on Equation 5 and can be placed into a ROM. To generate the coefficients, we must first use MATLAB. Create a new script in MATLAB and insert the code provided in the Appendix. Set the variable named SCALE to the integer N that was chosen for this design. Run the script. A new .coe file named mycoefile will be saved in the MATLAB directory as illustrated in **Figure 11.**



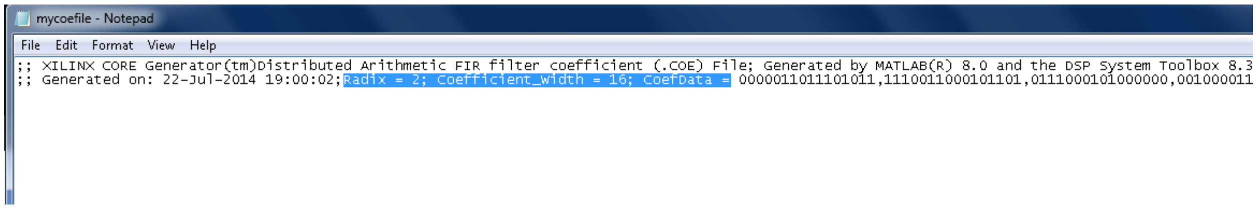
Name	Date modified	Type	Size
codegen	3/6/2014 4:16 PM	File folder	
mycoefile.coe	7/22/2014 7:00 PM	COE File	1 KB

Figure 11:Path to .coe file from MATLAB

Due to MATLAB generating coe files that are not compatible with any recent versions of Xilinx, we need to manually alter the file. When selected, the coe file will open in a text editor. Remove the text “Radix = … CoefData =” and replace it with the following text as shown in **Figure 12** and **Figure 13.**

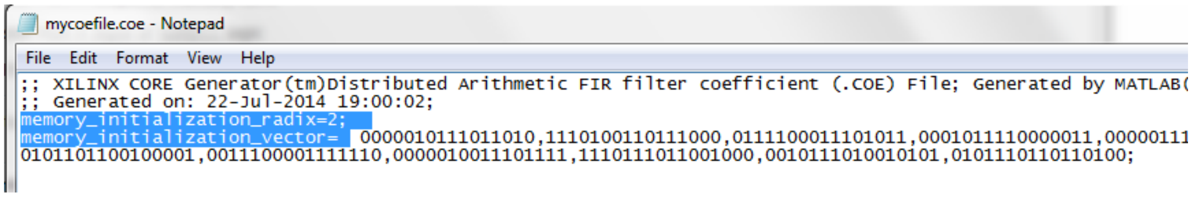
memory_initialization_radix=2;
memory_initialization_vector=

Verify that the beginning of the file looks identical to the **Figure 13.** Save the file and exit MATLAB. Do not rerun the script. That would reformat the file back to the incorrect form.



```
;; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File; Generated by MATLAB(R) 8.0 and the DSP System Toolbox 8.3
;; Generated on: 22-Jul-2014 19:00:02;Radix = 2; Coefficient_Width = 16; CoefData = 0000011011101011,1110011000101101,0111000101000000,001000011
```

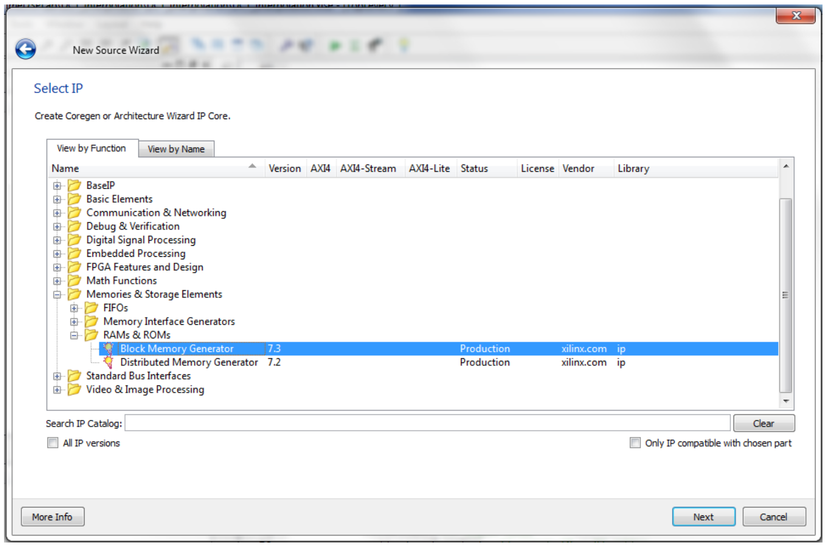
Figure 12: Incorrect format for .coe file



```
;; XILINX CORE Generator(tm)Distributed Arithmetic FIR filter coefficient (.COE) File; Generated by MATLAB(
;; Generated on: 22-Jul-2014 19:00:02;
memory_initialization_radix=2;
memory_initialization_vector= 0000010111011010,1110100110111000,0111100011101011,0001011110000011,00000111
0101101100100001,0011100001111110,0000010011101111,1110111011001000,0010111010010101,0101110110110100;
```

Figure 13:Correct format for .coe file

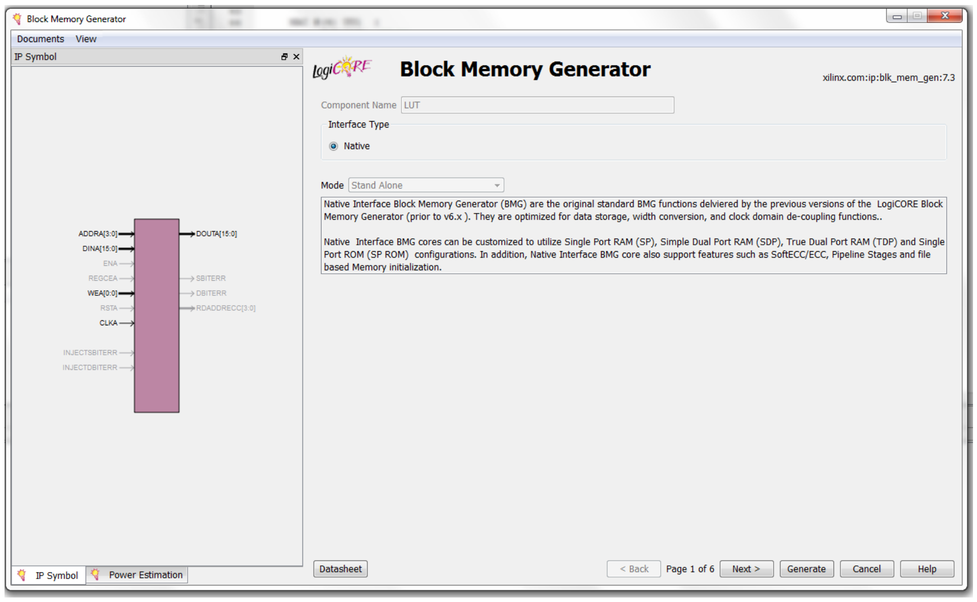
Next, we can insert this ROM by using the Xilinx IP tool. Select Project and New Source. On the window that pops up, select IP (CORE Generator and Architecture Wizard) and name the file LUT. The name of the module is significant as it is called in the top level of the project. In Memories and Storage, RAMs & ROMs, select Block Memory Generator (**Figure 14**).



Name	Version	AXI4	AXI4-Stream	AXI4-Lite	Status	License	Vendor	Library
BaseIP								
Basic Elements								
Communication & Networking								
Debug & Verification								
Digital Signal Processing								
Embedded Processing								
FPGA Features and Design								
Math Functions								
Memories & Storage Elements								
FIFOs								
Memory Interface Generators								
RAMs & ROMs								
Block Memory Generator	7.3				Production		xilinx.com	ip
Distributed Memory Generator	7.2				Production		xilinx.com	ip
Standard Bus Interfaces								
Video & Image Processing								

Figure 14:IP Core Wizard directory

A wizard will appear (**Figure 15**). Go to the second page of the wizard and select Single Port ROM as illustrated in **Figure 16**.



Block Memory Generator

xilinx.com:ip:blk_mem_gen:7.3

Component Name: LUT

Interface Type

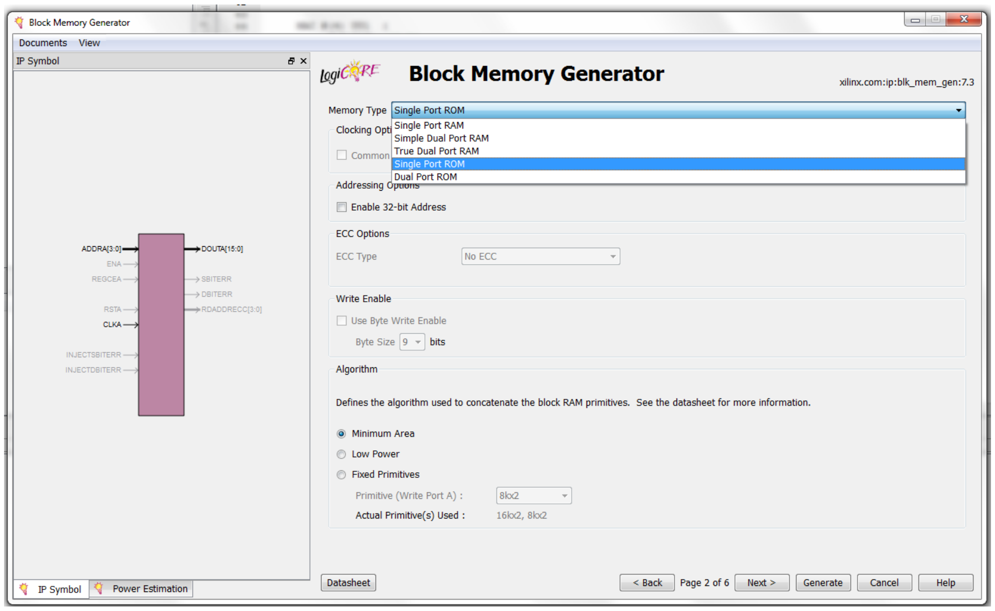
Native [✓]

Mode: Stand Alone

Native Interface Block Memory Generator (BMG) are the original standard BMG functions delvered by the previous versions of the LogiCORE Block Memory Generator (prior to v6.x). They are optimized for data storage, width conversion, and clock domain de-coupling functions..

Native Interface BMG cores can be customized to utilize Single Port RAM (SP), Simple Dual Port RAM (SDP), True Dual Port RAM (TDP) and Single Port ROM (SP ROM) configurations. In addition, Native Interface BMG core also support features such as SoftECC/ECC, Pipeline Stages and file based Memory initialization.

Figure 15:Block Memory Generator initial window



Block Memory Generator

xilinx.com:ip:blk_mem_gen:7.3

Memory Type: Single Port ROM

Single Port RAM
Simple Dual Port RAM
True Dual Port RAM
Single Port ROM
Dual Port ROM

Clocking Opti

Common

Addressing Options

Enable 32-bit Address [ ]

ECC Options

ECC Type: No ECC

Write Enable

Use Byte Write Enable [ ]

Byte Size: 9 bits

Algorithm

Defines the algorithm used to concatenate the block RAM primitives. See the datasheet for more information.

[✓ Minimum Area]

Low Power

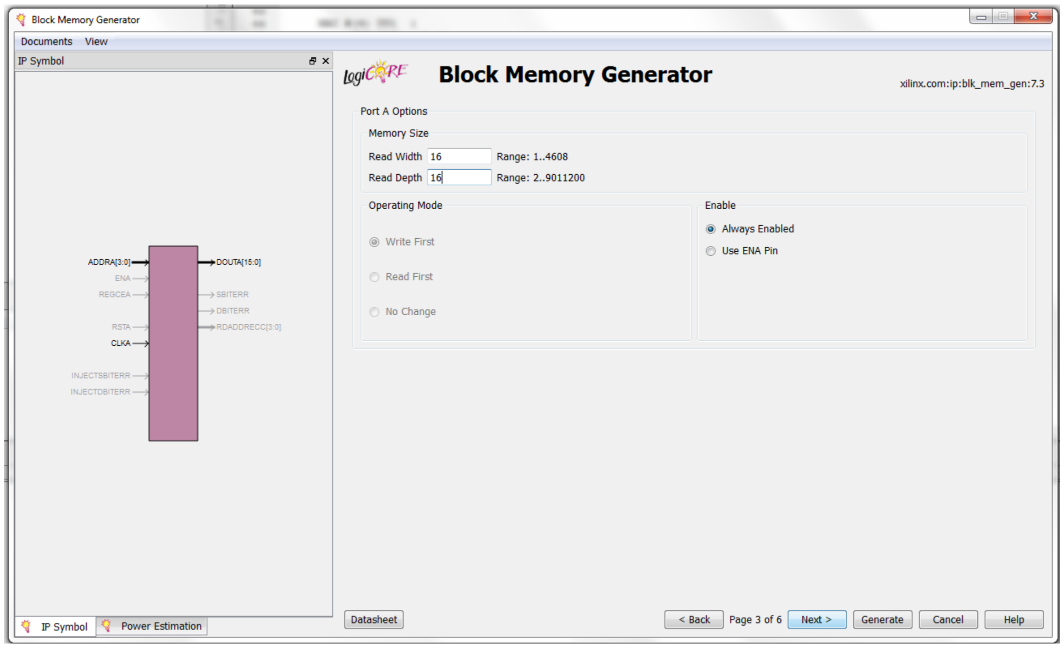
Fixed Primitives

Primitive (Write Port A) : 8kx2

Actual Primitive(s) Used : 16kx2, 8kx2

Figure 16:Single Port ROM

On the next page, input 16 for the Read Width. The Read Depth is equal to the parameter N*4.



Block Memory Generator

xilinx.com:ip:blk_mem_gen:7.3

Port A Options

Memory Size

Read Width 16 Range: 1..4608

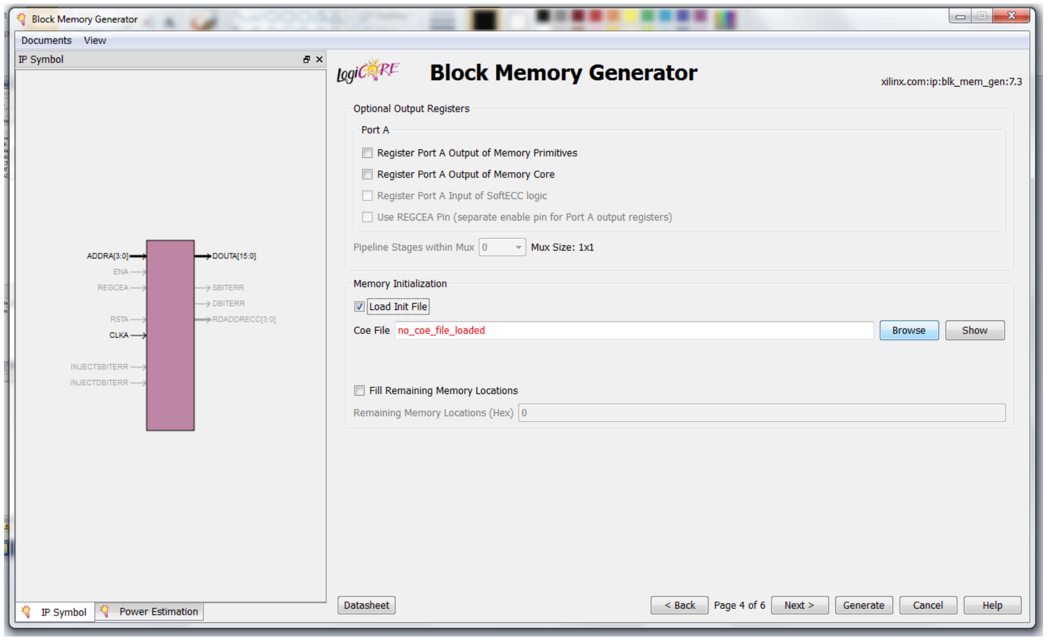
Read Depth 16 Range: 2..9011200

Operating Mode: [✓ Write First] Read First No Change

Enable: [✓ Always Enabled] Use ENA Pin

Figure 17:Size of Memory

Finally, load the .coe file on page 4 by selecting browse and finding the correct file from the MATLAB directory. It is placed in DOCUMENTS/MATLAB folder.



Block Memory Generator

xilinx.com:ip:blk_mem_gen:7.3

Optional Output Registers

Port A

- [ ] Register Port A Output of Memory Primitives
- [ ] Register Port A Output of Memory Core
- [ ] Register Port A Input of SoftECC logic
- [ ] Use REGCEA Pin (separate enable pin for Port A output registers)

Pipeline Stages within Mux 0 Mux Size: 1x1

Memory Initialization

- [x] Load Init File

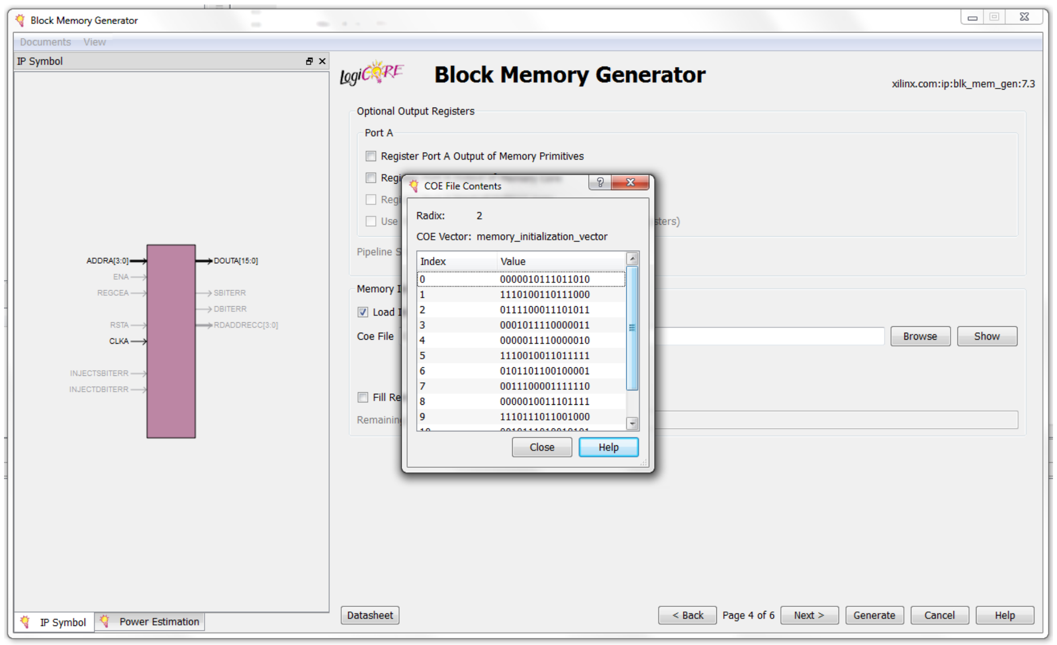
Coe File no_coe_file_loaded

- [ ] Fill Remaining Memory Locations

Remaining Memory Locations (Hex) 0

Figure 18: Loading of .coe file

To verify that the coefficients are properly loaded into the ROM, select Show to view the contents. After verification, select Generate.



Radix: 2

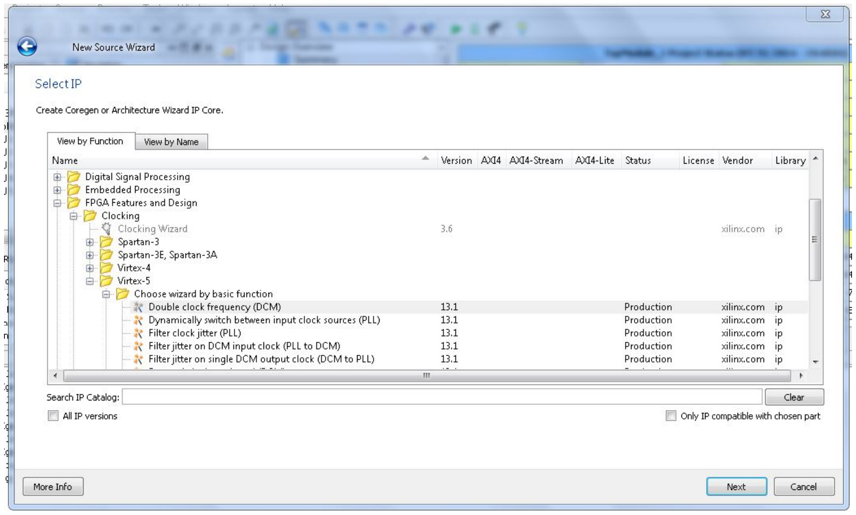
COE Vector: memory_initialization_vector

Index	Value
0	0000010111011010
1	1110100110111000
2	0111100011101011
3	0001011110000011
4	0000011110000010
5	1110010011011111
6	0101101100100001
7	0011100001111110
8	0000010011101111
9	1110111011001000

Figure 19: Coefficients correclty inserted into memory

Clock Divider

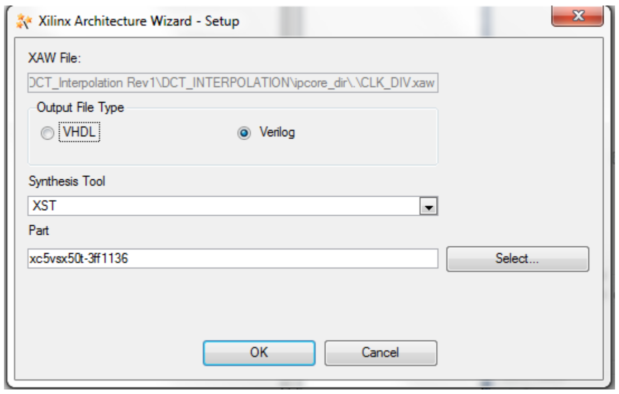
The final component that must be added to the design is the DCM clocking manager. Create a new IP Core source as done in the previous section, but this time, name it CLK_DIV. Again the name should be exactly the same in order to instantiate it from the top module. Select next and find the “Double clock frequency (DCM)” under FPGA Features and Designs, Clocking, Virtex 5, and Choose wizard by basic function (**Figure 20**).



Name	Version	AXI4	AXI4-Stream	AXI4-Lite	Status	License	Vendor	Library
Digital Signal Processing								
Embedded Processing								
FPGA Features and Design								
Clocking								
Clocking Wizard	3.6						xilinx.com	ip
Spartan-3								
Spartan-3E, Spartan-3A								
Virtex-4								
Virtex-5								
Choose wizard by basic function								
Double clock frequency (DCM)	13.1				Production		xilinx.com	ip
Dynamically switch between input clock sources (PLL)	13.1				Production		xilinx.com	ip
Filter clock jitter (PLL)	13.1				Production		xilinx.com	ip
Filter jitter on DCM input clock (PLL to DCM)	13.1				Production		xilinx.com	ip
Filter jitter on single DCM output clock (DCM to PLL)	13.1				Production		xilinx.com	ip

Figure 20: Clock Manager in IP Core Wizard directory

Next, a window will appear asking for the HDL that is in use. Select Verilog as shown below.


Xilinx Architecture Wizard - Setup

XAW File: DCT_Interpolation Rev1\DCT_INTERPOLATION\ipcore_dir\.\CLK_DIV.xaw

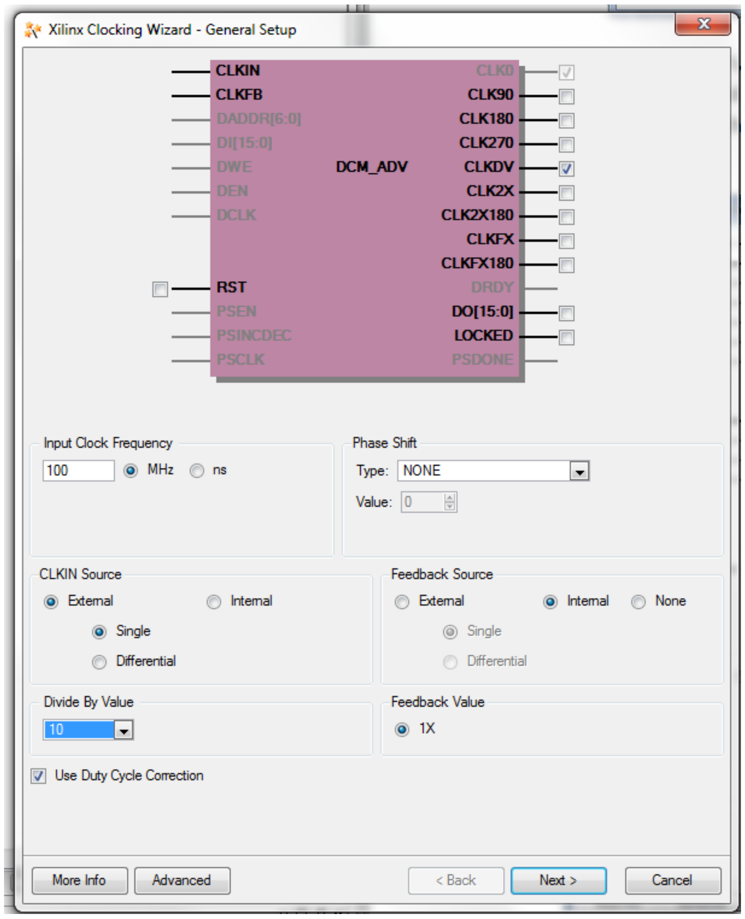
Output File Type: VHDL [✓ Verilog]

Synthesis Tool: XST

Part: xc5vsx50t-3ff1136

Figure 21: HDL Selecton

On the pop-up window that appears, select the CLKDV component, input 100 to the Input Clock Frequency, and select 10 for the Divide By Value (**Figure 22**). Verify that the wizard looks identical to **Figure** 22. Generate the clock divider by clicking next for the remaining windows. When the core is done loading, all modules instantiated in the top level should be present.


Xilinx Clocking Wizard - General Setup

Input Clock Frequency: 100 [✓ MHz] ns

Phase Shift: Type: NONE; Value: 0

CLKIN Source: [✓ External] Internal; [✓ Single] Differential

Feedback Source: External [✓ Internal] None; [✓ Single] Differential

Divide By Value: 10

Feedback Value: [✓ 1X]

[✓] Use Duty Cycle Correction

Figure 22: Clocking options for DCM

Test and Verification

With all parameters set and all modules ready, we can select Generate Programming File (**Figure 23**). When the Xilinx is done loading, program the FPGA using IMPACT and assign the corresponding configuration file to the device.

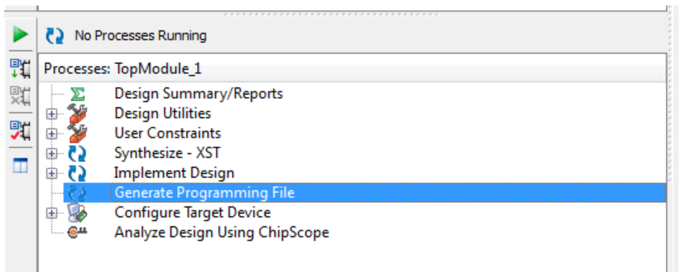


Figure 23: Generate programming file for target device

By connecting the probes to pins 38 and 52 on the J6 header, you are able to view the enable for the ADC and DAC. Notice that the ADC and DAC are both active low. As illustrated on the scope, SYNC, the yellow signal, is enabled N times more than CS, the blue signal. Also, the frequency of CS is approximately 48 KHz, four times less than the desired frequency of 192 KHz. Prior to inserting a signal into the system, it should be mentioned that the ADC and DAC only operate with positive voltages; thus, if one inserts a signal straight from the function generator, an offset must be used to place the signal in the proper range (0-3.3V). Input a signal to the ADC.

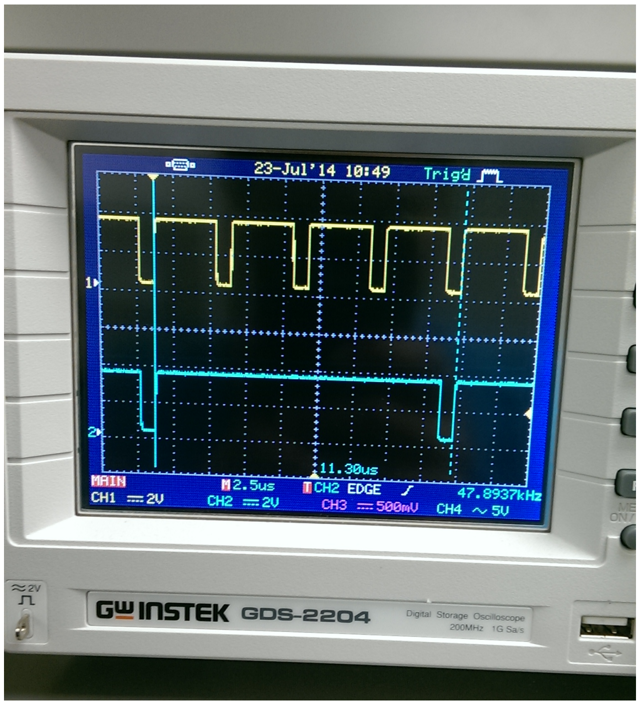


Figure 24: Enable for DAC (yellow) and ADC (blue)

When observing signals at the input and the output, the signal should be uninterrupted. Again, the intention is not to alter the content of the signal; rather, the goal is simply to change the resolution of the output. Raise the frequency to near half of the sampling frequency. If the scope is used to runstop and zoom in on the signal, as illustrated in **Figure 26**, cursors may be used to view the sampling period. Notice the stair-like nature of the signal in **Figure 26**. The cursors show that the width of the steps in time is approximately 5.2µs. This translates to an output rate of approximately 192 KHz and verifies that that the sampled signal is being interpolated.

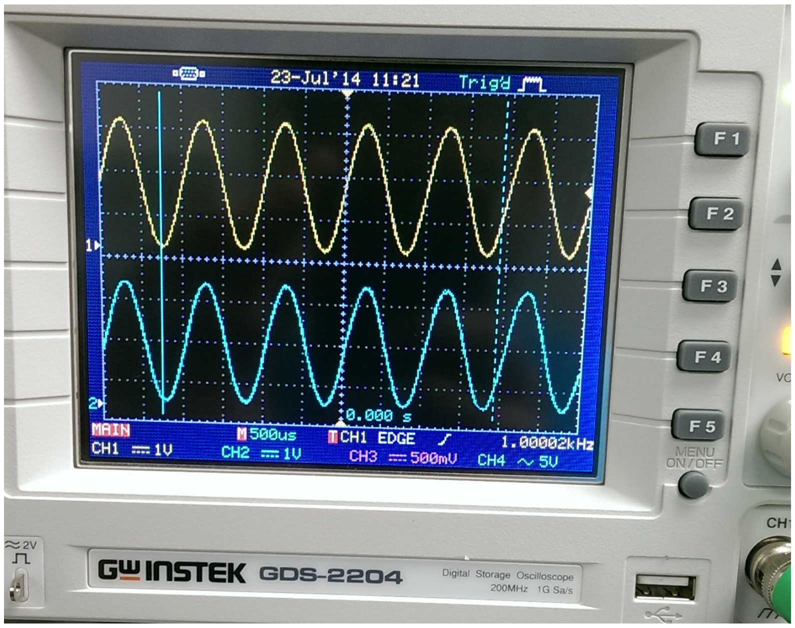


Figure 25:Signal at the input and output

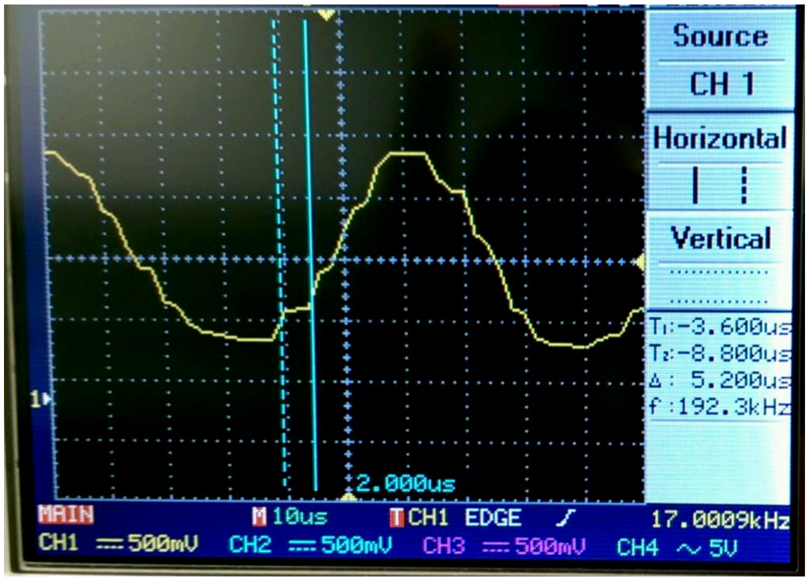


Figure 26: Zoomed in scope shot

Appendix

//DCT INTERPOLATION in Verilog HDL

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Create Date:    13:15:07 06/28/2014
// Design Name:            DCT INTERPOLATION
// Module Name:          TopModule_1
// Target Devices: Virtex 5, ML506
// Tool versions: Xilinx ISE 14.4
// Description: This design is a real-time interpolator that is based on the DCT
// Parameters:  N is the upsampling integer.
//                    SIZE must be set to the number of bits needed to suppord the integer N
//                    WAIT_TIME allows the user to change the sampling frequency
////////////////////////////////////////////////////////////////////////////////
module TopModule_1#(parameter N=4, WAIT_TIME=78, SIZE=2)
                    (Din, CLK_100, RST,CS, ADC_CLK, SYNC, DO, DO_2, DAC_CLK);

                    input Din;
                    input CLK_100;
                    input RST;
                    output CS;
                    output ADC_CLK;
                    output SYNC;
                    output DO;
                    output DO_2;
                    output DAC_CLK;


                    wire [11:0] value_out;
                    wire [11:0] value_in;
                    wire [SIZE+1:0] coef_index;
                    wire [11:0] sample;
                    wire [15:0] dct_coeff;

                    wire [16:0] mat;
                    wire mult_en;
                    wire mult_done;

                    wire CLK;

//                  wire CLK; // TEST
//                  assign CLK=CLK_100; // TEST

                    assign ADC_CLK=CLK;
                    assign DAC_CLK=CLK;
```

```
MAC #(4) UU1  (
            .CLK(CLK), // in from ML506
            .en(mult_en),// in from DCT
            .A(sample),// in 12 bits from DCT
            .B(dct_coeff), // in 16 bits from LUT
            .MULT_COMPLETE(mult_done),//  out to DCT
            .FINAL(mat)// out 16 bits  to DCT
            );


DCT #(.N(N),.RANGE(SIZE)) UU2 (
            .CLK(CLK), // in from ML506
            .SAMPLE(value_in), // in 12 bits from DAC_OUT
            .RST(RST), //in
            .DAC_done(dac_done),// in from DAT_OUT
            .MULT_done(mult_done), // in from MAC
            .COEF(mat), // in 16 bits from MAC
            .MULT(sample), // out 12 bits to MAC
            .en_MULT(mult_en), // out to MAC
            .COEFFICENT(coef_index), // out to LUT
            .ADC_en(adc_en), // out to DAC_OUT
            .UPSAMPLE(value_out)// out 12 bitsto DAC_OUT
            );

(* BOX_TYPE = "user_black_box" *)

LUT UU3 (
             .clka(CLK), // input clka
             .addra(coef_index), // input
             .douta(dct_coeff) // output [15 : 0] douta
            );


DAC_OUT #(WAIT_TIME)
            UU4 (
 .CLK(CLK),// 100 MHz Clock
 .RST(RST),
 .VALUE_OUT(value_out), // output 12 bit from DCT to DO
 .DI(Din), // input
 .ADC_EN(adc_en), // input
 .VALUE_IN(value_in), // input 12 bit to DCT
 .CS(CS), // output
 .DCLK1(CLK), // output
 .SYNC(SYNC), // output
 .DO(DO), // output
 .DO_2(DO_2), // output
 .DAC_DONE(dac_done) // output to DCT
```

```
                        );

                        CLK_DIV UU5 (
   .CLKIN_IN(CLK_100),
   .CLKDV_OUT(CLK),
   .CLKIN_IBUFG_OUT(), // Unconnected
   .CLK0_OUT() // Unconnected
   );



endmodule



`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Create Date:    20:27:34 05/01/2014
// Description: This is a MAC that only Adds four levels at a time. "en" initiates
//          the process. MULT_COMPLETE Goes high when MAC is done and the output
//          retains the correct value until reset.
////////////////////////////////////////////////////////////////////////////////
module MAC #(parameter SIZE=4)
                                        (input CLK, en,
                                        input signed [11:0] A,
                                        input signed [15:0] B,
                                        output reg MULT_COMPLETE,
                                        output signed [16:0] FINAL
                                        );

                        reg [4:0] COUNT;
                        wire signed [28:0] Temp1;
                        reg signed [30:0] Temp2;


                        assign Temp1=$signed({1'b0,A})*$signed(B);// Multiply
                        assign FINAL=Temp2[30:14];

                        always@( posedge CLK )
                        begin

                              if( en )
                              begin

                                        if( COUNT < (SIZE) )
                                        begin
                                              COUNT<=COUNT+1;
                                              Temp2<=Temp1+Temp2; // Accumulate
```

```
                    end

                    else
                // raise COMPLETE flag and output the high 16 bits
                                MULT_COMPLETE<=1;

                end // end if(en)

                else
                begin
                    // RESET
                    MULT_COMPLETE<=0;
                    COUNT<=0;
                    Temp2<=0;
                end

            end // always

            initial
            begin
                // Initial Conditions
                COUNT=0;
                Temp2=0;
                MULT_COMPLETE=0;
            end

endmodule


////////////////////////////////////////////////////////////////////////////////
// Create Date:    10:23:17 04/28/2014
// Module Name:    DAC_OUT
// Description: The DAC_OUT module is the front end of the design. It interfaces
//                                          directly with the PMOD boards.
////////////////////////////////////////////////////////////////////////////////
module DAC_OUT#(parameter ADDED_TIME=12)
                    (input CLK, RST,
                    input [11:0] VALUE_OUT, //TO DAC
                    input DI,
                    input ADC_EN,
                    output reg [11:0] VALUE_IN, // TO FPGA for Processing
                    output reg CS,
                    output DCLK1,
                    output reg SYNC, DO, DO_2, DAC_DONE );

    // Used to input and output when enabled
    reg [4:0] COUNT;
```

```
// temp register for original samples
reg [11:0] Temp;

integer TIMER=0;

assign DCLK1=CLK;

always@( posedge CLK )
begin
        if( ~RST )
        begin
                COUNT<=COUNT+1;

                if(COUNT<11 )
                begin
                        DO<=0;
                        DO_2<=0;
                        DAC_DONE<=0;
                end

                // Begin data conversion by  setting ADC and DAC to '0'
                else if( COUNT >= 11  && COUNT <= 14 )
                begin
                        SYNC<=1'b0;
                                if( ADC_EN )
                                        CS<=1'b0;
                        DO<=0;
                        DO_2<=0;
                end

                // 12 bit words are sent and recieved
                else if( COUNT<=26 )
                begin
                        DO<=VALUE_OUT[26-COUNT];
                        DO_2<=Temp[26-COUNT];

                        if( ADC_EN )
                        begin
                                VALUE_IN[0]<=DI;
                                //Shift Data Into ADC
                                VALUE_IN[11:1]<=VALUE_IN[10:0];
                        end
                end

                // Turn off enablers
                else if( COUNT==27 )
                begin
                        Temp<=VALUE_IN;
```

```
                                        CS<=1'b1;
                                        SYNC<=1'b1;

                                        if( TIMER==ADDED_TIME )
                                        begin
                                                COUNT<=0;
                                                TIMER<=0;
                                                DAC_DONE<=1;
                                        end

                                        else
                                        begin
                                                COUNT<=COUNT;
                                                TIMER<=TIMER+1;
                                                DAC_DONE<=0;
                                        end

                                end

                        end //RST

                        else
                                COUNT<=0;


                end

                initial
                begin
                        CS=1;
                        DAC_DONE=0;
                        COUNT=0;
                        SYNC=1;
                end

endmodule




`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Create Date:    01:01:16 05/15/2014
// Module Name:    DCT
// Description: This module acts as the controller for the design.
////////////////////////////////////////////////////////////////////////////////
module DCT #(parameter N=4,  RANGE=2) //N interpolants
                                (input CLK,
		input [11:0] SAMPLE,
		input RST,
		input DAC_done,
		input MULT_done,
		input [16:0] COEF,
		output reg [11:0] MULT,
		output reg en_MULT,
		output [1+RANGE:0] COEFFICENT,
		output reg ADC_en,
		output reg [11:0] UPSAMPLE );

	integer j;

	// Neg Edge Flag
	reg CONVERSION_flag;

	reg [11:0] A [0:3]; //Sample Storage
	reg [RANGE-1:0] index; // interpolant index
	reg [1:0] Xc; // coefficient

	assign COEFFICENT ={index,Xc};

	// Next STATE Logic

	always@( posedge CLK )
	begin

		// RESET
		if( RST )
		begin
			UPSAMPLE<=0;
			A[0]<=0;
			A[1]<=0;
			A[2]<=0;
			A[3]<=0;
			index<=0;
			Xc<=0;
			CONVERSION_flag<=0;
		end // RESET

		else
		begin

			// Conversion NOT in progress
			if( ~CONVERSION_flag )
			begin
				MULT<=A[Xc];
```

```
        // Only increment up to three
        if( Xc==2'b11 )
                Xc<=Xc;
        else
                Xc<=Xc+1;

        en_MULT<=1'b1;

        if( MULT_done )
        begin
                en_MULT<=0;
                Xc<=0;
                CONVERSION_flag<=1;

                if( index==N-1 )
                        ADC_en<=1;
                else
                        ADC_en<=0;

                // Roof and floor protection
                if( COEF[16] )
                        UPSAMPLE<=0;
                else if( |COEF[15:12] )
                        UPSAMPLE<=12'b111_111_111_111;
                else
                        UPSAMPLE<=COEF[11:0];
        end // MULT_done
end // Conversion Flag

else
begin

        if( DAC_done )
        begin

                if( ADC_en )
                begin
                        A[3]<=SAMPLE;
                        A[2]<=A[3];
                        A[1]<=A[2];
                        A[0]<=A[1];
                        index<=0;
                        Xc<=0;
                        ADC_en<=0;
                end // ADC_en

                else
                begin
```

```
                                        index<=index+1;
                                        Xc<=0;
                                end // ADC_en/else

                                CONVERSION_flag<=0;
                        end // DAC_done

                        else
                                CONVERSION_flag<=1;
                end // else


        end // else (~reset)

    end // always


    initial
    begin

        CONVERSION_flag=0;
        Xc=0;
        index=0;
        for( j=0;j<=3;j=j+1)
        begin
                A[j]=0;
        end
    end

endmodule
```

%%MATLAB Script

```
clc
clear all
close all
%%%%%%%%%%%%%%%%%%%%%
% This script is used to generate coefficients that
% are needed to interpolate. It is important that
% the SCALE variable below is set to the number N from the
% top level module from the design in Xilinx.
%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%
%%%%PARAMETER%%%%%%%%
SCALE=4;
%%%%%%%%%%%%%%%%%%%%%

% Indexes
n=0:3;
k=0:3; k=k';
m=0:4*SCALE-1; m=m';

% % Forward DCT
Forward=cos(pi.*k*(2*n+1)/8);
Forward(1,1:end)=ones(1,4)*0.5;

k=k';
% % INVERSE DCT
Inverse=cos(pi.*(2*m+SCALE)*k/(SCALE*8));
% % Matrix Multiplication of full system
DCTINTERP=Inverse*Forward/2;

% % Isolate needed coefficients
h1=DCTINTERP(2*SCALE+2:3*SCALE+1,:);
% % Reshape to fit one dim array
h1=reshape(h1',4*(SCALE),1);
h=dfilt.dffir(h1);
% % Quantize Coeff.
set(h,'arithmetic','fixed');
h.CoeffWordLength  = 16;
coewrite(h,2,'mycoefile');
```

References

Analog Devices, “2.35 V to 5.25 V, 1 MSPS, 12-/10-/8-Bit ADCs in 6-Lead SC70,” D02930-0-1/11(F) datasheet, January 2011.

Diligent, “Diligent PmodAD1 Analog to Digital Module Converter Board Reference Manual,” 502-064 datasheet, December 6, 2011.

Diligent, “Diligent PmodDA1 Diligent to Analog Module Converter Board Reference Manual,” 502-113 datasheet, September 28, 2006.

National Semiconductor, “DAC121S101 12-Bit Micro Power Digital-to-Analog Converter with Rail-to-Rail Output,” DS201149 datasheet, June 2005.

Oppenheim, Alan V., and Ronald W. Schafer. Discrete-time Signal Processing. Upper Saddle River, NJ: Prentice Hall, 2010. Print