XILINX

ALL PROGRAMMABLE™

Implementing Direct Digital Synthesizer (DDS) using Vivado HLS

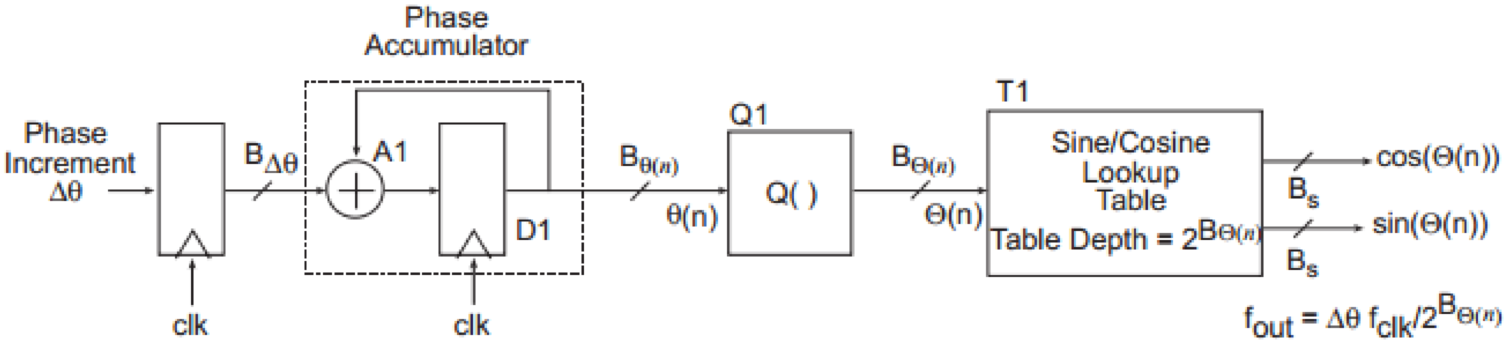
Alex Paek
Aug 2014

Overview

- **This document describes the implementation of a DDS using Vivado HLS tool**
- **The initial target device is K7 -1, but the design – described in C++, can be targeted to most of Xilinx FPGAs.**
- **Specification:**
  - Frequency resolution: clock rate / 2^32 (ie., 0.07 Hz for 300 MHz clock)
  - >120 dB SFDR
  - Clock rate = 400 MHz
  - Resource: 2 DSP48, 1110 LUT, 1069 FF, 0 BRAM(36kb)
- **Tools:**
  - Vivado Tool suite 2015.3
  - Vivado HLS 2015.3
- **Integrated with ADI RF frontend using SDSoC. Shown at Embedded World**
  - https://wiki.analog.com/resources/eval/user-guides/targetting/xilinx_sdsoc

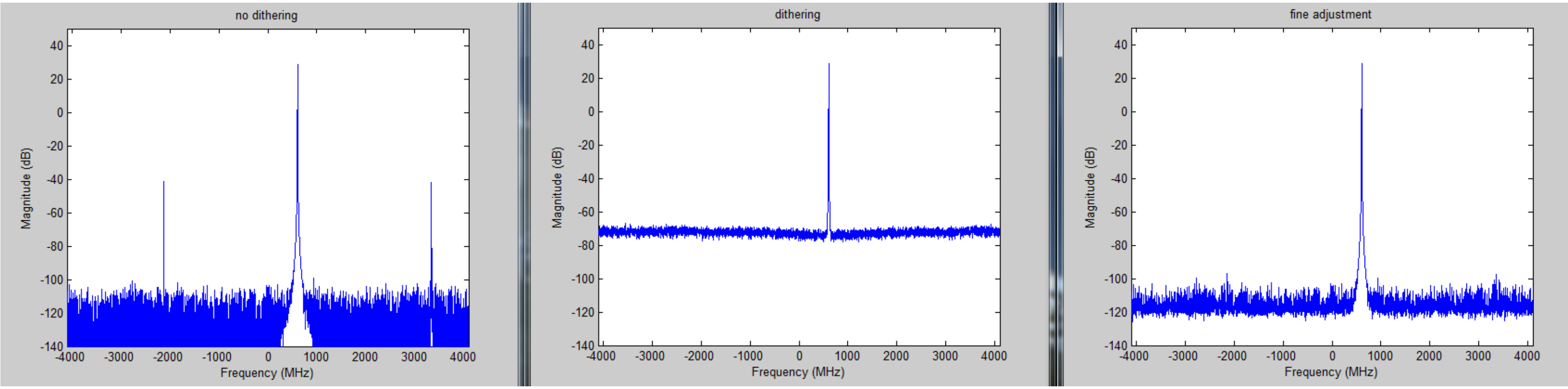
Design Criteria

- The basic theory and operation of DDS is described well in [1]
- A simple design - two main components of a DDS are phase accumulator and sine/cosine look up table - shown below
- Two most important parameters in designing DDS are:
  - frequency/phase resolution: determines how fine adjustment frequency can be made. The higher the resolution the bigger the look up table would be. Easy to meet.
  - Spurious free dynamic range (SFDR): How to achieve the maximum SFDR with the least amount of resource – mostly the size of the lookup table, is the design goal.



Maximizing SFDR

- **Techniques for maximizing SFDR:**
  - Inject noise into the phase output of the accumulator before looking up the table, commonly known as dithering technique – implemented from a simple LFSR. The spurs are lowered but overall noise floor is increased – center figure below
  - Interpolation of the sine/cosine LUT to get the more phase angle resolution; thus lowering the SFDR without the expense of bigger LUT. Not explored here.
  - Split the LUT into coarse table (phase angle resolution = pi/N, N=number of words, like 1024) and fine table (phase angle resolution = (pi/N)/M, M=number of words in the fine table), and do complex multiplication of two complex words out of both table (see the 3rd figure)



Splitting the table

- **The phase output of the accumulator (32bit) is truncated to 21 bit - 12 MSB looks up the coarse table, and 9 LSB looks up the fine table**
  - – Instead of 2^21 words (2M), you use 2^12 + 2^9 words (4096+512) table
- **The coarse table stores the single quadrant - 0 to pi/2 angle. 1024 words of 18 bit each.**
- **The derivation of this techniques is shown below:**

$$e^{j*\theta} = e^{j*(\phi+\varepsilon)} = e^{j*\phi}e^{j*\varepsilon} = (\cos(\phi) + j*\sin(\phi)) \; * \; (\cos(\varepsilon) + j*\sin(\varepsilon))$$

$$= (\cos(\phi)\cos(\varepsilon) - \sin\phi\,\sin(\varepsilon)) + \mathrm{j}(\sin\phi * \cos(\varepsilon) + \cos(\phi)\sin(\varepsilon))$$

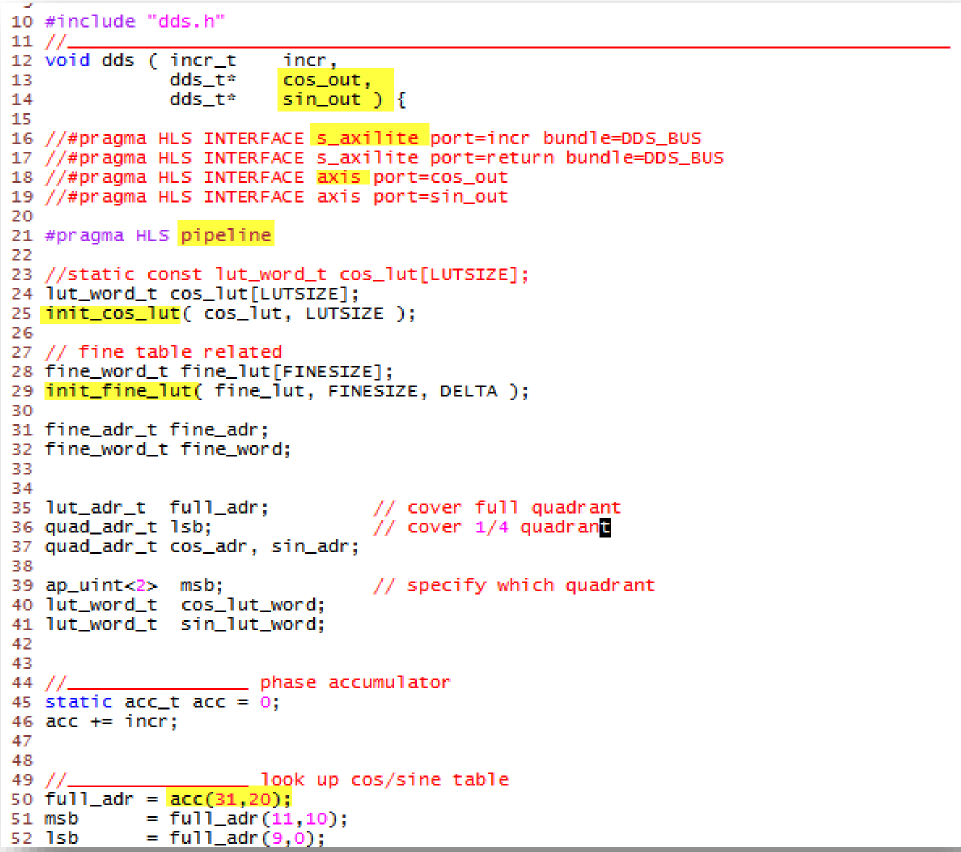
$$\cong (\cos(\phi) \; - \sin\phi\,\sin(\varepsilon)) + \mathrm{j}(\sin\phi + \cos(\phi)\sin(\varepsilon)),$$

since $\cos(\varepsilon)$ is almost unity

- **This solution is slightly more expensive (BRAM, 2 mults, 2 add/sub) than the dithering technique (small LUT/FF for LFSR, adder) but it does not raise the overall noise floor as the dithering does.**
- $\sin(\varepsilon)$ **is almost linear. This table might be replaced with a constant multiplier.**

Source Code

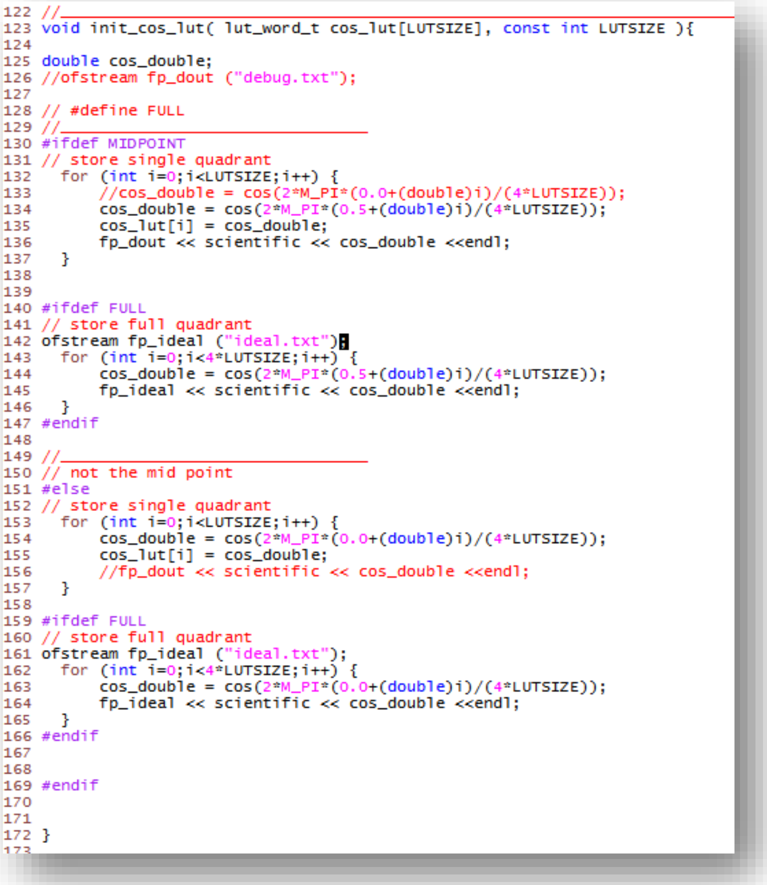
- Sample based input (vs. packet/array based)
- Calls routine to initialize look up table
- Check out the bit extraction



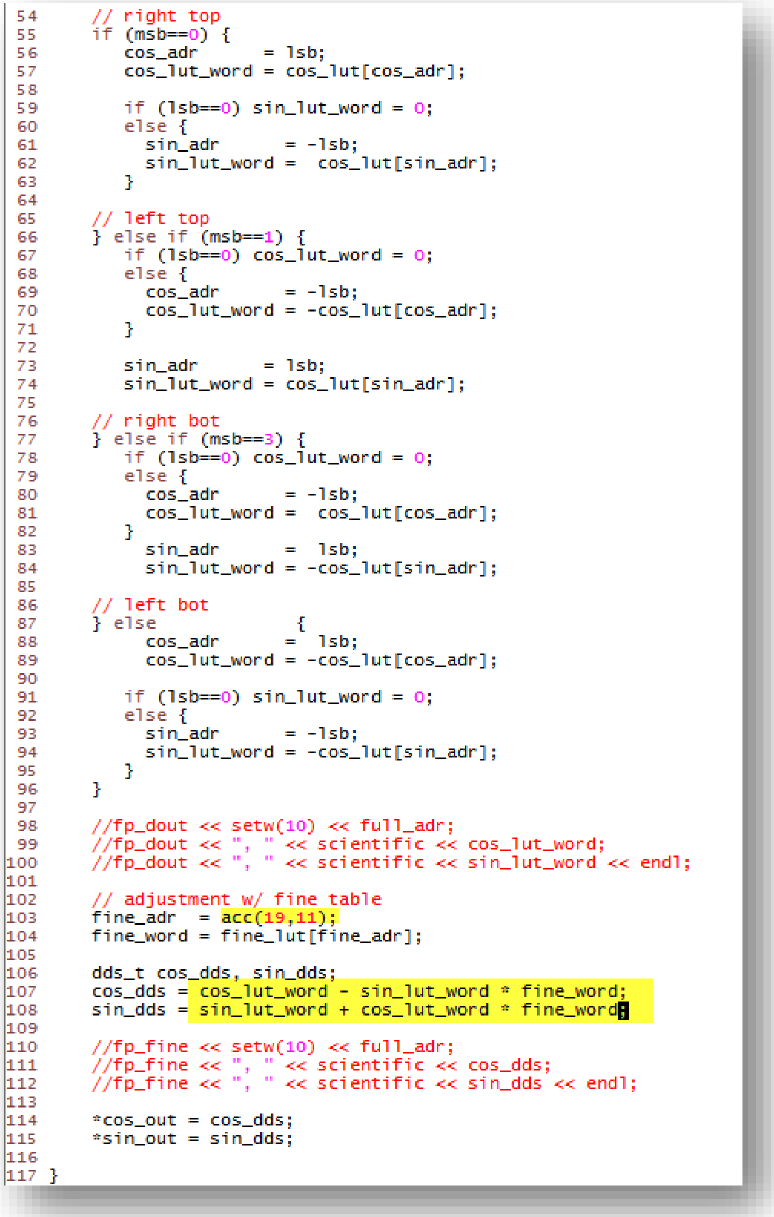
```
#include "dds.h"
//________________________________________________________________________________
void dds ( incr_t    incr,
           dds_t*    cos_out,
           dds_t*    sin_out ) {

//#pragma HLS INTERFACE s_axilite port=incr bundle=DDS_BUS
//#pragma HLS INTERFACE s_axilite port=return bundle=DDS_BUS
//#pragma HLS INTERFACE axis port=cos_out
//#pragma HLS INTERFACE axis port=sin_out

#pragma HLS pipeline

//static const lut_word_t cos_lut[LUTSIZE];
lut_word_t cos_lut[LUTSIZE];
init_cos_lut( cos_lut, LUTSIZE );

// fine table related
fine_word_t fine_lut[FINESIZE];
init_fine_lut( fine_lut, FINESIZE, DELTA );

fine_adr_t fine_adr;
fine_word_t fine_word;


lut_adr_t  full_adr;            // cover full quadrant
quad_adr_t lsb;                 // cover 1/4 quadrant
quad_adr_t cos_adr, sin_adr;

ap_uint<2>  msb;                // specify which quadrant
lut_word_t  cos_lut_word;
lut_word_t  sin_lut_word;


//________________ phase accumulator
static acc_t acc = 0;
acc += incr;


//________________ look up cos/sine table
full_adr = acc(31,20);
msb      = full_adr(11,10);
lsb      = full_adr(9,0);
```

Source Code

- Stores only 1 quadrant (90 degree) out of 4 quadrants
- “init_cos_lut” does not build any logic



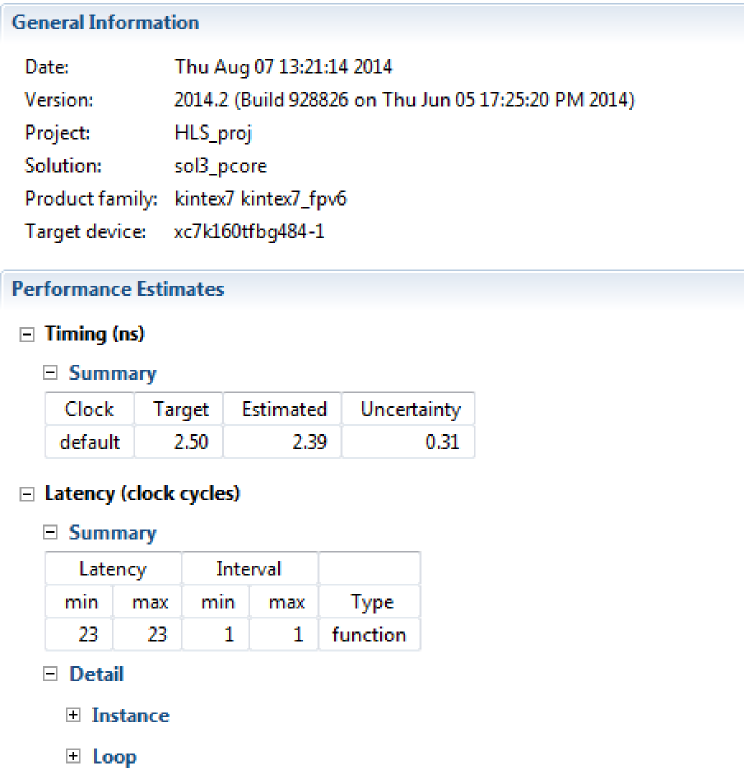
```
//______________________________________________________________________
void init_cos_lut( lut_word_t cos_lut[LUTSIZE], const int LUTSIZE ){

double cos_double;
//ofstream fp_dout ("debug.txt");

// #define FULL
//________________________________
#ifdef MIDPOINT
// store single quadrant
  for (int i=0;i<LUTSIZE;i++) {
      //cos_double = cos(2*M_PI*(0.0+(double)i)/(4*LUTSIZE));
      cos_double = cos(2*M_PI*(0.5+(double)i)/(4*LUTSIZE));
      cos_lut[i] = cos_double;
      fp_dout << scientific << cos_double <<endl;
  }


#ifdef FULL
// store full quadrant
ofstream fp_ideal ("ideal.txt");
  for (int i=0;i<4*LUTSIZE;i++) {
      cos_double = cos(2*M_PI*(0.5+(double)i)/(4*LUTSIZE));
      fp_ideal << scientific << cos_double <<endl;
  }
#endif

//________________________________
// not the mid point
#else
// store single quadrant
  for (int i=0;i<LUTSIZE;i++) {
      cos_double = cos(2*M_PI*(0.0+(double)i)/(4*LUTSIZE));
      cos_lut[i] = cos_double;
      //fp_dout << scientific << cos_double <<endl;
  }

#ifdef FULL
// store full quadrant
ofstream fp_ideal ("ideal.txt");
  for (int i=0;i<4*LUTSIZE;i++) {
      cos_double = cos(2*M_PI*(0.0+(double)i)/(4*LUTSIZE));
      fp_ideal << scientific << cos_double <<endl;
  }
#endif


#endif


}
```



```
    // right top
    if (msb==0) {
       cos_adr      = lsb;
       cos_lut_word = cos_lut[cos_adr];

       if (lsb==0) sin_lut_word = 0;
       else {
         sin_adr      = -lsb;
         sin_lut_word =  cos_lut[sin_adr];
       }

    // left top
    } else if (msb==1) {
       if (lsb==0) cos_lut_word = 0;
       else {
         cos_adr      = -lsb;
         cos_lut_word = -cos_lut[cos_adr];
       }

       sin_adr      = lsb;
       sin_lut_word = cos_lut[sin_adr];

    // right bot
    } else if (msb==3) {
       if (lsb==0) cos_lut_word = 0;
       else {
         cos_adr      = -lsb;
         cos_lut_word =  cos_lut[cos_adr];
       }
         sin_adr      =  lsb;
         sin_lut_word = -cos_lut[sin_adr];

    // left bot
    } else                {
         cos_adr      =  lsb;
         cos_lut_word = -cos_lut[cos_adr];

       if (lsb==0) sin_lut_word = 0;
       else {
         sin_adr      = -lsb;
         sin_lut_word = -cos_lut[sin_adr];
       }
    }

    //fp_dout << setw(10) << full_adr;
    //fp_dout << ", " << scientific << cos_lut_word;
    //fp_dout << ", " << scientific << sin_lut_word << endl;

    // adjustment w/ fine table
    fine_adr  = acc(19,11);
    fine_word = fine_lut[fine_adr];

    dds_t cos_dds, sin_dds;
    cos_dds = cos_lut_word - sin_lut_word * fine_word;
    sin_dds = sin_lut_word + cos_lut_word * fine_word;

    //fp_fine << setw(10) << full_adr;
    //fp_fine << ", " << scientific << cos_dds;
    //fp_fine << ", " << scientific << sin_dds << endl;

    *cos_out = cos_dds;
    *sin_out = sin_dds;

}
```

C Synthesis

- VHLS is smart enough to implement a function for computing a cosine table into ROM
- “incr” input is implemented into AXI Lite interface
- For future improvement, the code can become a **template** class with parameters



General Information

Date: Thu Aug 07 13:21:14 2014
Version: 2014.2 (Build 928826 on Thu Jun 05 17:25:20 PM 2014)
Project: HLS_proj
Solution: sol3_pcore
Product family: kintex7 kintex7_fpv6
Target device: xc7k160tfbg484-1

Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
default	2.50	2.39	0.31

⊟ Latency (clock cycles)

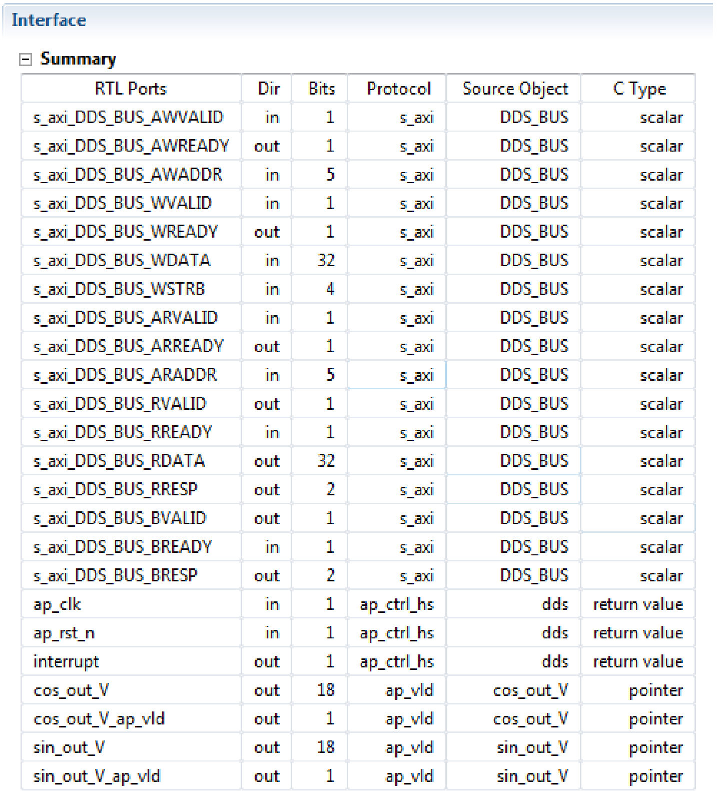
⊟ Summary

Latency		Interval		
min	max	min	max	Type
23	23	1	1	function

⊟ Detail

⊞ Instance

⊞ Loop



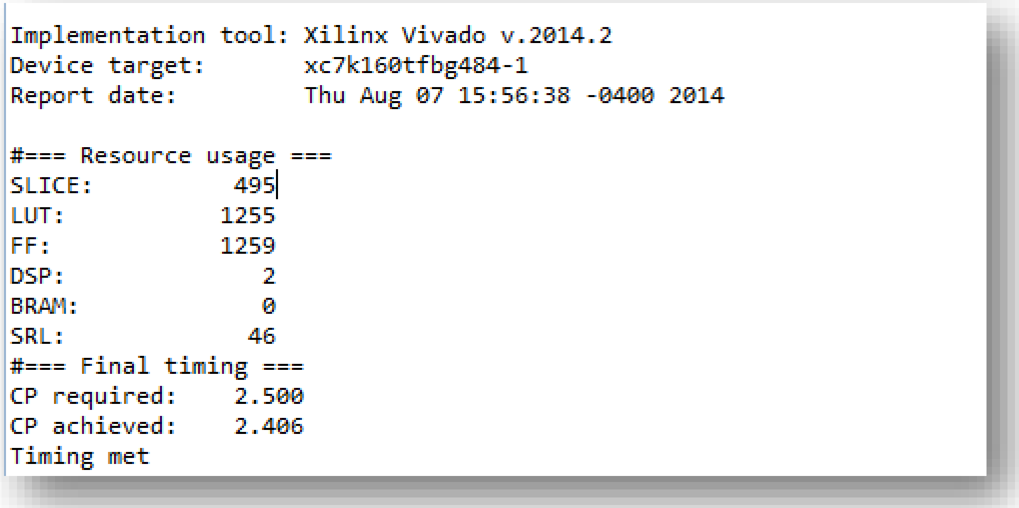
Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
s_axi_DDS_BUS_AWVALID	in	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_AWREADY	out	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_AWADDR	in	5	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_WVALID	in	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_WREADY	out	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_WDATA	in	32	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_WSTRB	in	4	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_ARVALID	in	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_ARREADY	out	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_ARADDR	in	5	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_RVALID	out	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_RREADY	in	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_RDATA	out	32	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_RRESP	out	2	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_BVALID	out	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_BREADY	in	1	s_axi	DDS_BUS	scalar
s_axi_DDS_BUS_BRESP	out	2	s_axi	DDS_BUS	scalar
ap_clk	in	1	ap_ctrl_hs	dds	return value
ap_rst_n	in	1	ap_ctrl_hs	dds	return value
interrupt	out	1	ap_ctrl_hs	dds	return value
cos_out_V	out	18	ap_vld	cos_out_V	pointer
cos_out_V_ap_vld	out	1	ap_vld	cos_out_V	pointer
sin_out_V	out	18	ap_vld	sin_out_V	pointer
sin_out_V_ap_vld	out	1	ap_vld	sin_out_V	pointer

FPGA Implementation

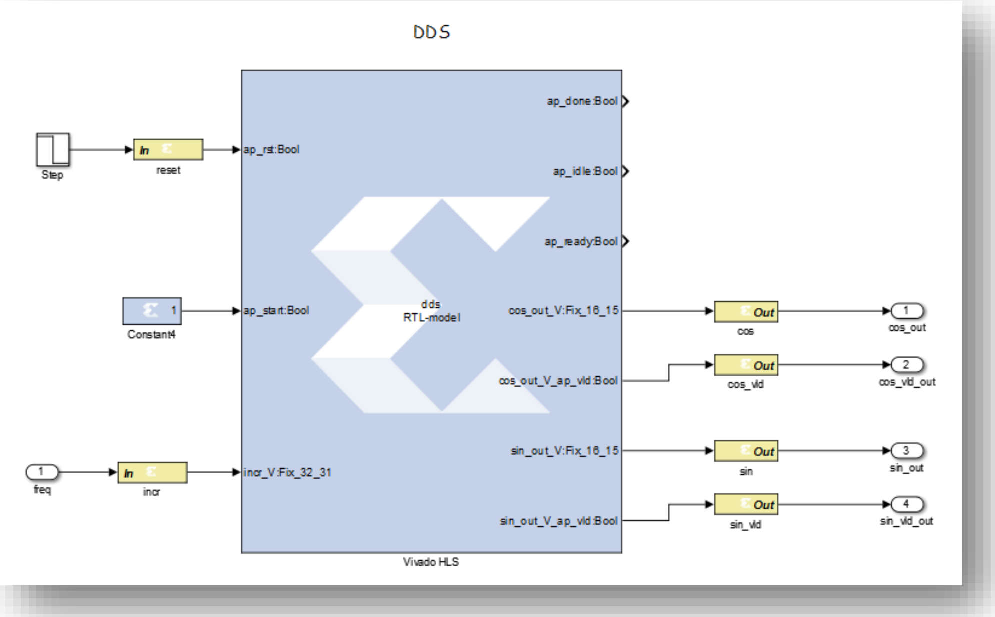
- .

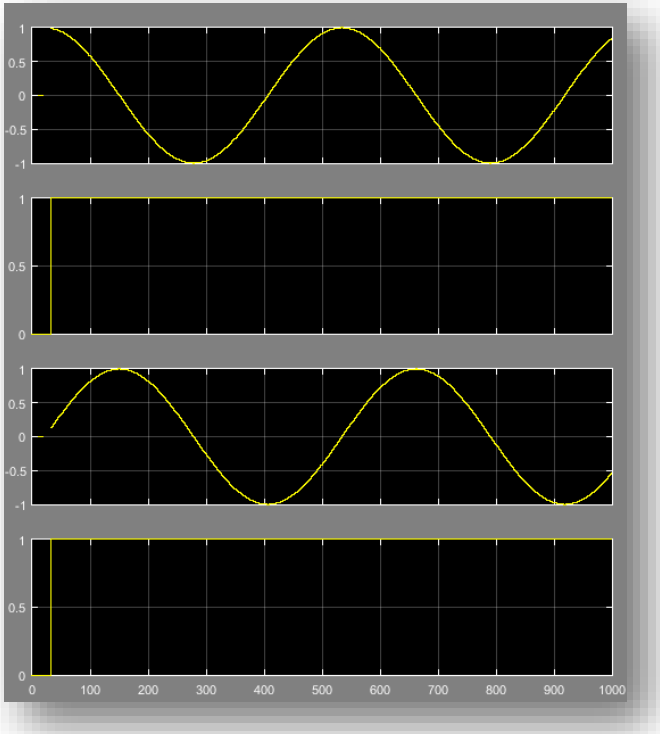


```
Implementation tool: Xilinx Vivado v.2014.2
Device target:       xc7k160tfbg484-1
Report date:         Thu Aug 07 15:56:38 -0400 2014

#=== Resource usage ===
SLICE:            495
LUT:             1255
FF:              1259
DSP:                2
BRAM:               0
SRL:               46
#=== Final timing ===
CP required:    2.500
CP achieved:    2.406
Timing met
```

Export

- The HLS generated RTL can be exported to Sysgen
- Exported to SDSoC and integrated with ADI RF Frontend for real time HW demo





Design Review Question

- Is this a sample based design or packet based design?
- Where is pipeline directive placed in the code?
- Does init_cos_lut results in ROM table – no logic built?
- How about init_fine_lut?
- Interesting that there is no BRAM!
- Can we make it to utilize BRAM and save LUT?
- Can you make this a template function? What would be a good template parameters?

Design files

- **dds.cpp: has several functions. dds() is the design to synthesize**
  - void init_cos_lut(lut_word_t cos_lut[LUTSIZE], const int LUTSIZE);
  - void read_cos_lut(lut_word_t cos_lut[LUTSIZE], const int LUTSIZE);
  - void read_sine_lut(lut_word_t cos_lut[LUTSIZE], const int LUTSIZE);
  - void init_fine_lut(fine_word_t fine_lut[FINESIZE], const int FINESIZE, const double DELTA);
  - void dds (incr_t incr, dds_t* cos_out, dds_t* sin_out);
- **dds_test.cpp: testbench**
- **matlab file:**
  - test_dds.m: to plot the FFT or PSD of the output from the C simulation
  - plot_psd.m: utility to plot PSD
- **run_hls.tcl:TCL file for HLS**

Reference

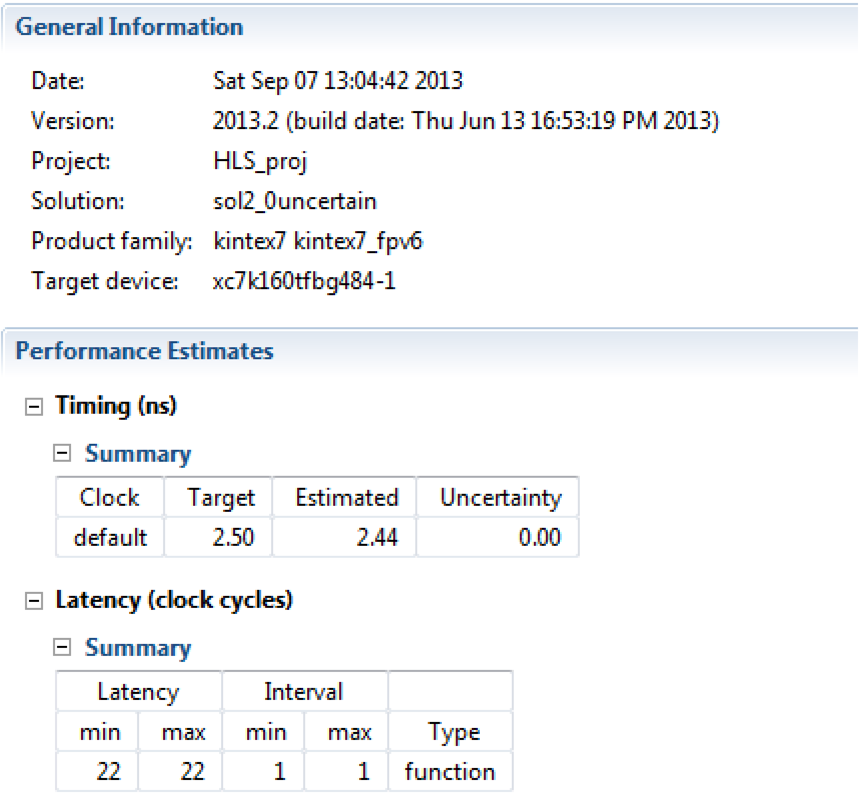
1. DDS Compiler v6.0 Product Guide, PG141
2. Vivado HLS User's Guide, UG902
3. Consultation with Gordon Old

Appendix

Results using previous HLS version – 2014.1 and 2013.2

C Synthesis

- VHLS is smart enough to implement a function for computing a cosine table into ROM
- The only HLS pragma used for this design is pipeline at the top level
- For future improvement, the code can become a template-ized class with passing parameters



General Information

Date:	Sat Sep 07 13:04:42 2013
Version:	2013.2 (build date: Thu Jun 13 16:53:19 PM 2013)
Project:	HLS_proj
Solution:	sol2_0uncertain
Product family:	kintex7 kintex7_fpv6
Target device:	xc7k160tfbg484-1

Performance Estimates

⊟ Timing (ns)

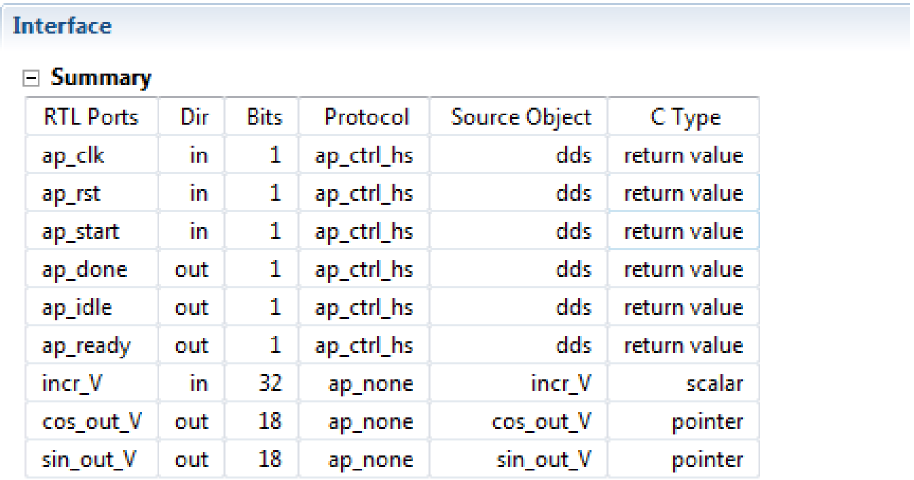
⊟ Summary

Clock	Target	Estimated	Uncertainty
default	2.50	2.44	0.00

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
22	22	1	1	function

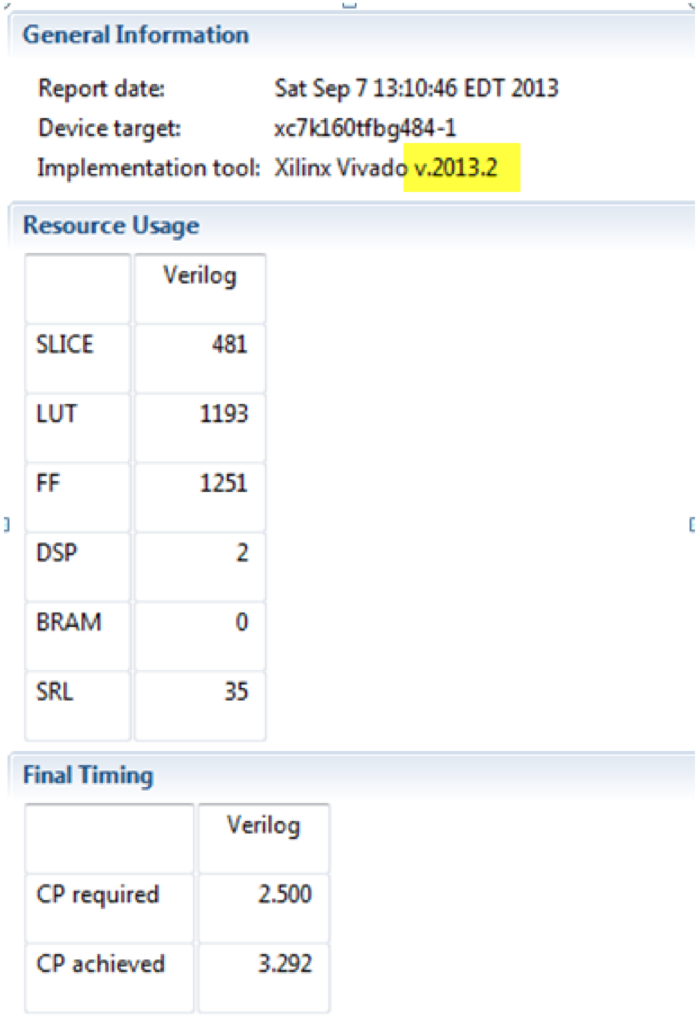


Interface

⊟ Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	dds	return value
ap_rst	in	1	ap_ctrl_hs	dds	return value
ap_start	in	1	ap_ctrl_hs	dds	return value
ap_done	out	1	ap_ctrl_hs	dds	return value
ap_idle	out	1	ap_ctrl_hs	dds	return value
ap_ready	out	1	ap_ctrl_hs	dds	return value
incr_V	in	32	ap_none	incr_V	scalar
cos_out_V	out	18	ap_none	cos_out_V	pointer
sin_out_V	out	18	ap_none	sin_out_V	pointer

FPGA Implementation



General Information

Report date: Sat Sep 7 13:10:46 EDT 2013

Device target: xc7k160tfbg484-1

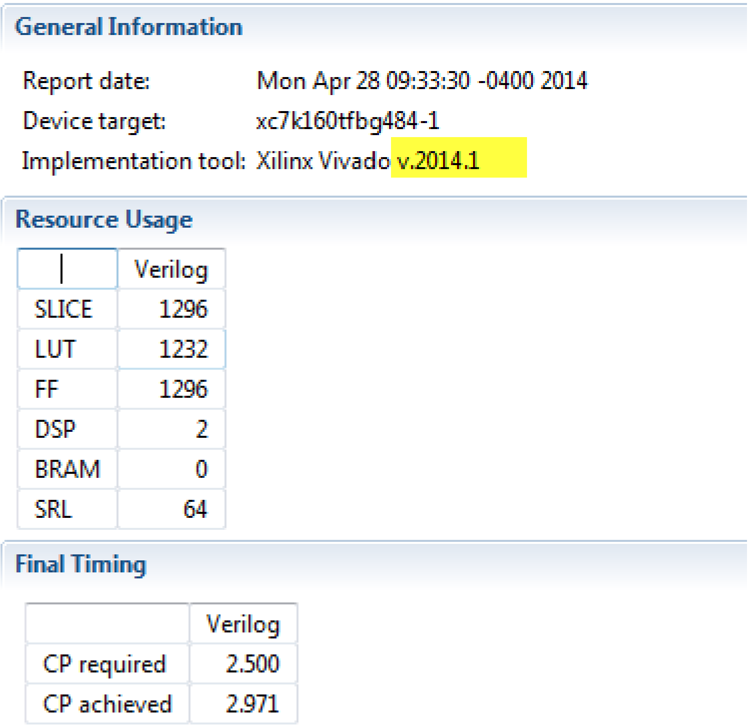
Implementation tool: Xilinx Vivado v.2013.2

Resource Usage

	Verilog
SLICE	481
LUT	1193
FF	1251
DSP	2
BRAM	0
SRL	35

Final Timing

	Verilog
CP required	2.500
CP achieved	3.292



General Information

Report date: Mon Apr 28 09:33:30 -0400 2014

Device target: xc7k160tfbg484-1

Implementation tool: Xilinx Vivado v.2014.1

Resource Usage

	Verilog
SLICE	1296
LUT	1232
FF	1296
DSP	2
BRAM	0
SRL	64

Final Timing

	Verilog
CP required	2.500
CP achieved	2.971