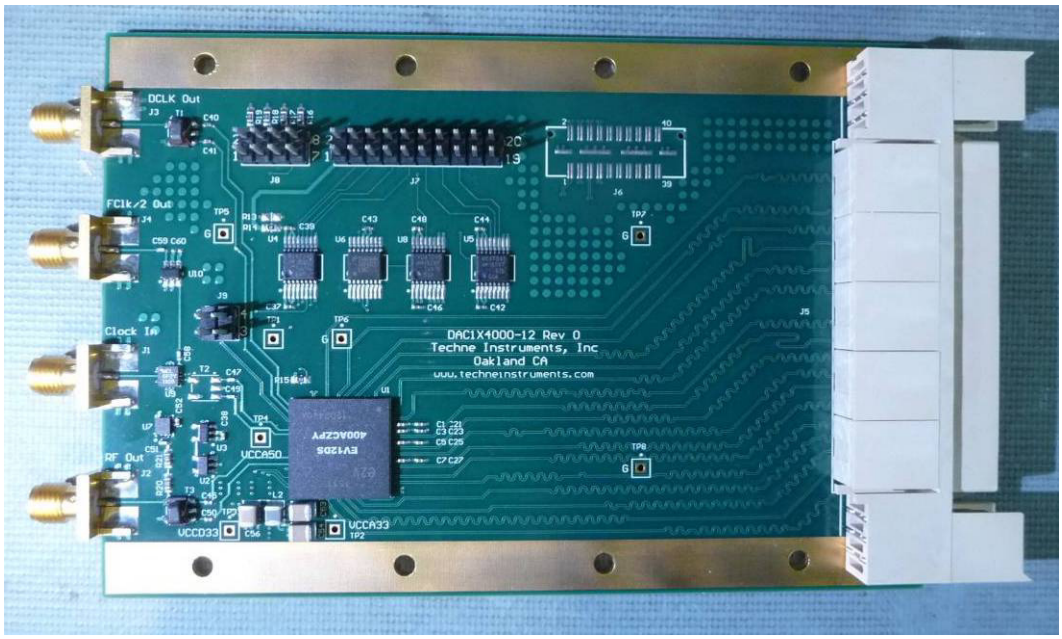
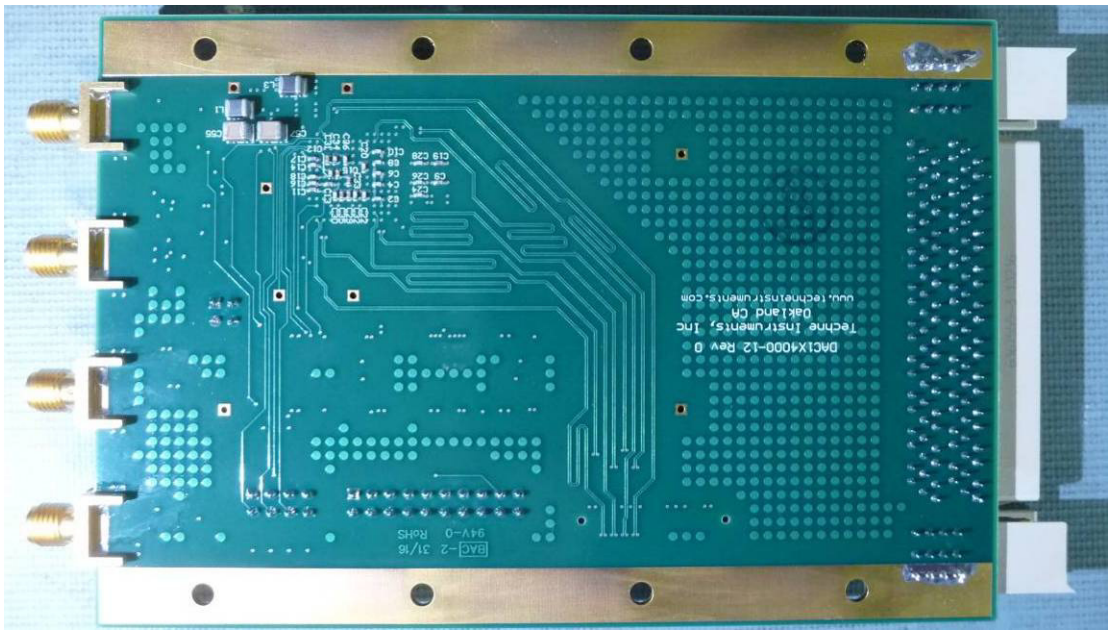
Test Report, DAC1X4500-10

Rick Raffanti
Sept 6, 2016

Design Overview
The ROACH-compatible board contains a single E2V EV12DS400AZP 4.5GSPS Digital-Analog Converter, plus support circuitry.

There are four 12-bit data buses bringing digital data into the part, but the ROACH ZDOK connector has only 40 bits, so I've brought only the 10 MS bits of each from the ZDOK. The remaining 8 pairs are brought to another QSH connector (not installed), which could be used to achieve full 12-bit resolution, though it would involve bringing in the pairs from a separate connector (the second ZDOK, or the QSH connector on a ROACH-1, for instance) and adjusting the delay for these pairs separately.





A 4 GHz sample clock is input on J1, analog output appears on J2.

Because of the limited number of pairs, the data clock is converted to a single-ended signal via a balun and brought out on J3. This will be brought into the FPGA board on one of the available SMAs. The three FPGA boards, mini-ROACH, ROACH-1, and ROACH-2 differ in the details of the available FPGA pair.

mini-ROACH:

SMAs J2 and J3 both are received by global clock pairs on the Virtex-5 FPGA

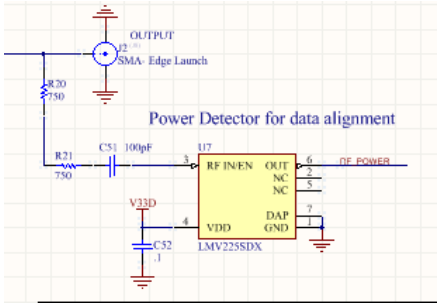
ROACH-1:

SMAs J12 and J13 are also both received by GC pairs

ROACH-2:

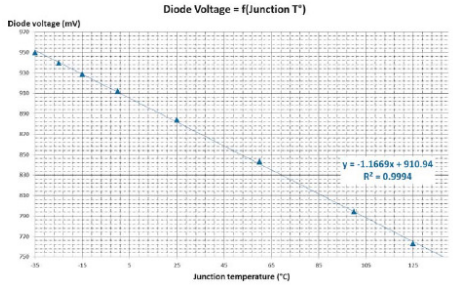
SMAs are J9 and J10, but only J9 ends up on a Single-Region Clock Capable pin, which can drive an MMCM, so this is the one to use.

Also because of the limited number of pairs, we are unable to use the IDC/TVF function of the DAC, where the IDC pair would be toggled along with the data, and the TVF signal would be monitored while varying the data clock phase, to find the phase setting that provides reliable data transfer. Instead, I've implemented a power detector, read by an ADC.

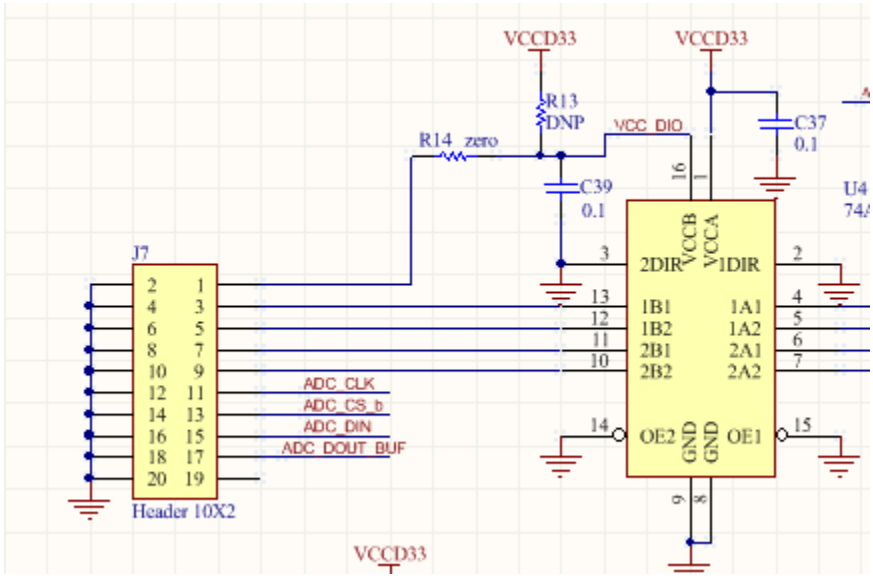


The DAC data is toggled between 0x3FF and 0x400 (one LSB around mid-scale) while the data clock phase is varied; when it passes through bad phase settings the output power rises, allowing us to know where the reliable zone is.
The ADC is also used to read out the temperature diode on the DAC die:

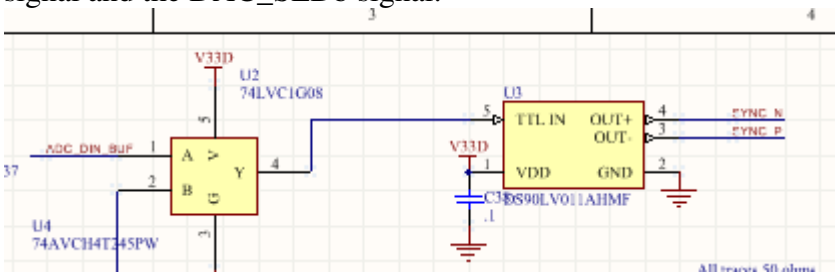
Figure 5-28. Diode Characteristics for Die Junction Temperature Monitoring



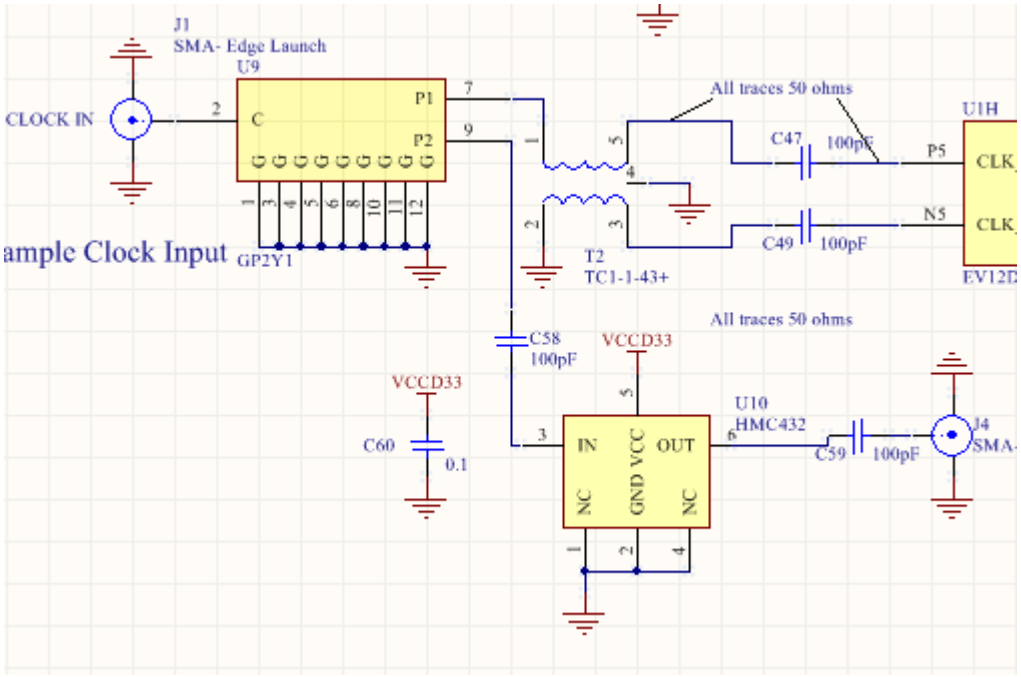
There is a 2-by-10 header to interface the DAC serial register IF and the serial ADC to the FPGA board:



There are many internal DAC registers which will need to be accessed.
Because the various FPGA boards have different IO voltages on the available headers, the levels are translated to 3.3v before being used. The SYNC pair on the DAC serves as a reset for the part and must be asserted after power up. (It's also necessary to apply the clock signal to the DAC before power up, according to the data sheet). Due to a lack of additional single-ended signals, the SYNC pulse is asserted as the AND of the ADC_DIN signal and the DAC_SLDb signal.

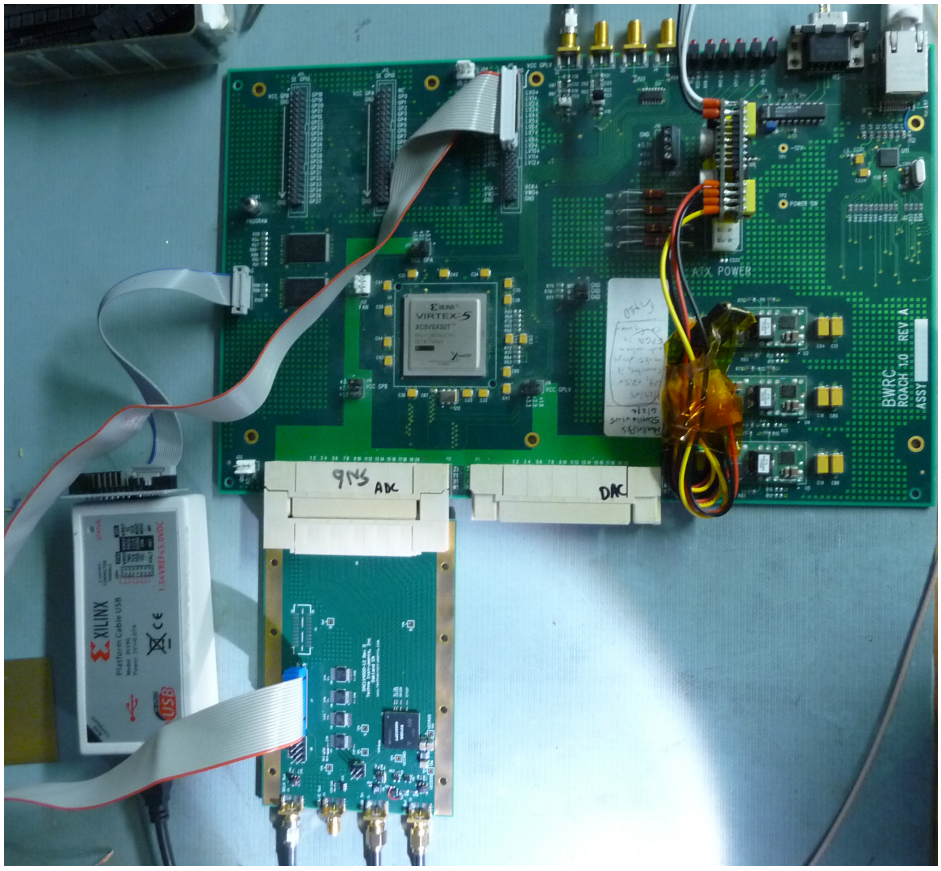


A clock divider chip, HMC432, is provided to produce an Fsample/2 output on J4, for clocking an ADC at ½ the 4GHz DAC sample rate. Note that the GP2Y1 splitter employed here is rated 1.55 to 4.4 GHz, so sample rates much less than 1.5GHz can't be used.



Test Configuration

The first board was tested in a "mini-roach" board, which contains a Virtex 5 FPGA. This board has a pair of ZDOK connectors, like ROACH-1 and ROACH-2, and 100Mbit Ethernet connectivity. A ribbon cable was used to connect a GPIO header on the mini-roach to the J7 control header of the DAC board. The DSPCLK output on J3 of the DAC board was connected to the SMA input J2 of the mini-roach.



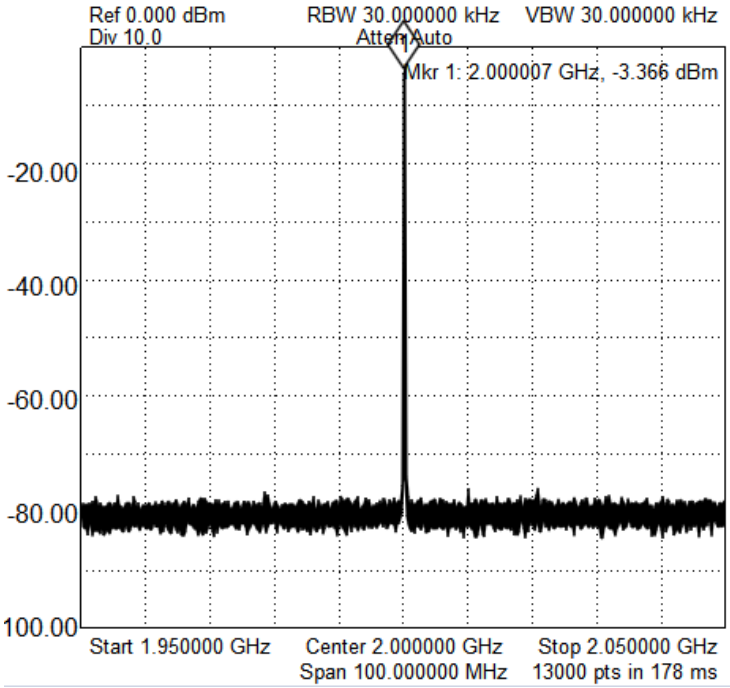
I wrote firmware to generate a sinewave of software-settable frequency. This uses a 1024-by-10 bit sine lookup table, replicated 16 times, to generate four phases of data for each of the four DAC data busses. The four phases are connected to a set of 4-bit input, 1 bit output, serializers (OSERDES's), which operate with a main clock rate of 250MHz to produce an output stream at 500MHz DDR, or 1 Gbit/second. The FPGA fabric runs at 250MHz. Ethernet UDP packets are used to control the system, allowing the setting of DAC registers, internal FPGA registers, etc.

The FPGA on the mini-roach is an XC5VSX50T -1 speed. This has a specified limit on the Digital Clock Manager (DCM) of 450MHz, rather than the 500MHz required to operate the system at 4GSPS (though the fabric runs at 250MHz, the OSERDES's require both a 250MHz and 500MHz clock to produce output data at 500MHz DDR). The design can't meet timing constraints for this reason, but I've found that the system works at 500MHz in many applications, including this one.

The host PC talks to the system via the Ethernet connection; a python (3.x) script allows setting of registers, etc.

Test Results

I drove the clock input J1 with a 4GHz signal at about +8dBm from a USB generator, DS Instruments SG6000L. I first looked at the FS/2 output on J4, using the spectrum analyzer Signal Hound USB-SA124A. Output is correctly 2.0GHz (at -3 dBm):



I programmed the FPGA with bitfile dac1x4000_top_r02.bit, corresponding to archive dac1x4000_top_r01.zip.

Running the python code allows system setup:

```
C:\rix\techne\DAC1X4500-10\python>python3 DAC1X4000_control.py
DAC1X4000 Control
Enter "I" to do complete system initialization,
    "D" to reset the FPGA DCM,
    "R" to pulse the serial reset line of the DAC,
    "S" to pulse the SYNC line of the DAC,
    "O" to pulse the OSERDES reset
    "W" to write a single DAC reg,
    "A" to read an ADC channel,
    "F" to set output freq,
    "T" to set to turn on/off Toggle mode (2GHz, 1LSBpp square wave out),
    "P" to sweep the PSS phase and report the RF detector output,
    "M" to log the voltage on the temperature diode,
    "L" to flash the LEDs,
    or "q" to quit
```

The “I” selection will do all of the system setup required to get the system going. The other selections allow exercising various functions individually. The series of steps needed to initialize the system are shown here in the python source:

```
    elif inp == 'I':
        sw_reg[0] = sw_reg[0] | 0b10000000  #the DAC_RST, to get to default settings
        sendit()
        sw_reg[0] = sw_reg[0] & ~0b10000000  #the DAC_RST
        sendit()
        write_DAC(0,0)  #Clear the ECDC bit, to use the 3WI for setting PSS and OCDC
        write_DAC(5,(PSS_default_setting<<2) + 1) #Set OCDC bit (to output 250MHz DSPCLK instead of 500)
        sw_reg[5] = sw_reg[5] | 0b00000010  #the DCM_RESET
        sendit()
        sw_reg[5] = sw_reg[5] & ~0b00000010  #the DCM_RESET
        sendit()
        sw_reg[4] = sw_reg[4] | 0b00001000  #the ADC_DIN
        sendit()
        sw_reg[4] = sw_reg[4] & ~0b00001000  #the ADC_DIN
        sendit()
        sw_reg[5] = sw_reg[5] | 0b00000001  #the OSERDES_RESET
        sendit()
        sw_reg[5] = sw_reg[5] & ~0b00000001  #the OSERDES_RESET
        sendit()
        #set default stepsize to 2 (freq = 2*4000/1024 = 7.8MHz)
        sw_reg[6] = 1
        sw_reg[5] = (sw_reg[5] & ~0b10000000) | 0b00000100
        sendit()
        #clear the change_freq  bit
        sw_reg[5] = sw_reg[5] & ~0b00000100
        sendit()
```

These are further discussed here.

1) Pulse the DAC RESET line, which sets all internal registers to default values. Having done that, the DAC is set to have the PSS[2:0] (Phase Shift Select) and OCDC (Output Clock Divide Control) bits set from the J8 jumper block. The system can be operated this way, but the other registers (eg, the output mode control and the gain control) are only accessible through the 3-wire interface, so we will set these controls that way, too.

2) In order to set PSS and OCDC from the 3WI, we need to clear the ECDC bit in DAC register 0. So, Write_DAC(0,0)

3) Need to set the OCDC bit in order to get the DSPCLK divided by an additional factor of 2. With OCDC = 0, DSPCLK = Fsample/8, with OCDCPCLK = Fsample/16. On my mini-roach, the level translator which receives the DSPCLK (from SMA connector J2) is relatively slow, and would have a hard time with a 500MHz signal. On the other members of the ROACH family, the receivers are faster, and might be OK with 500MHz. I'm not sure if there's an advantage (for clock jitter, for instance).
The PSS setting is also loaded into DAC register 5 (more on this setting below). So write these values to register 5.

4) Now that there is a clock of the desired frequency entering the FPGA, we need to pulse the DCM_RESET line, to start up the DCM.

5) The DAC requires a pulse to its SYNC line; this is done by asserting the ADC_DIN line, while holding the DAC_SLDb line high.

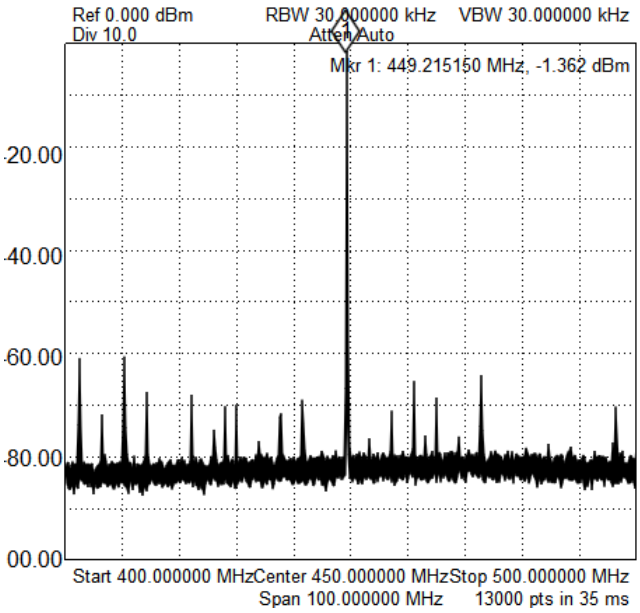
6) The OSERDES modules in the FPGA need to be reset once the clocks have been started, so pulse this line.

7) Now set the FPGA lookup table address generator to some sensible value. This can then be changed at any time without any of the other initialization being repeated.

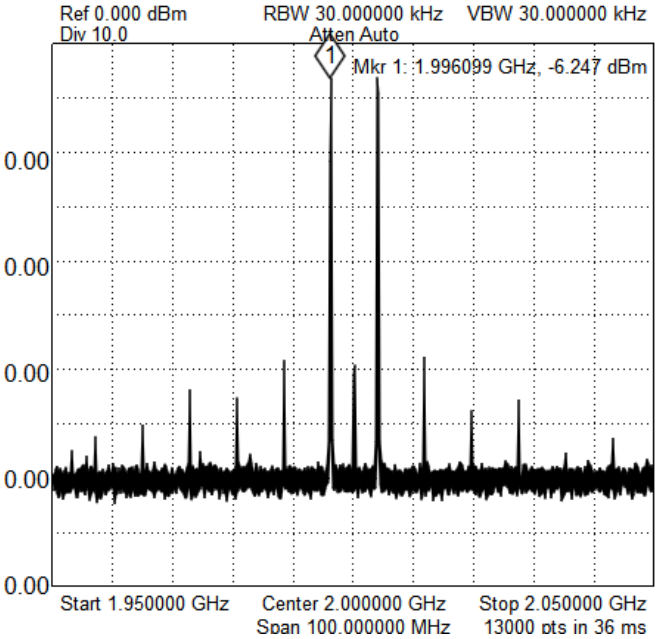
Output Waveforms

After setting up the system as described, we can set the frequency to any value Fout = N * Fsample/1024, for 1<= N <= 511, so from 3.9 to 1996 MHz.
Here are outputs at 7.8 and 449 MHz on my 500MHz scope

On the spectrum analyzer, the same 449 MHz waveform:



And 1996MHz:

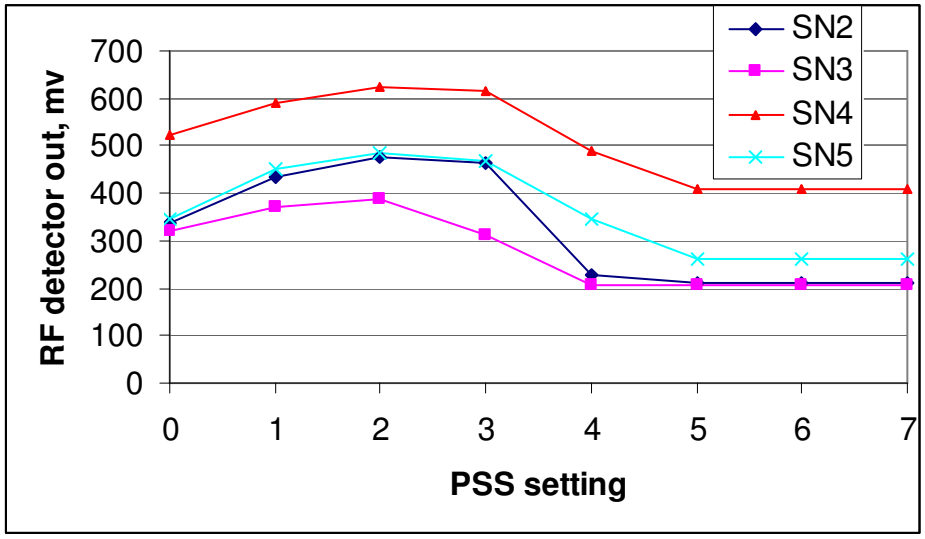


At Fsample/2 the output is down about 5 dB from its low frequency value; 3dB of this can be attributed to the sin(x)/x response of an ideal DAC.

Setting PSS

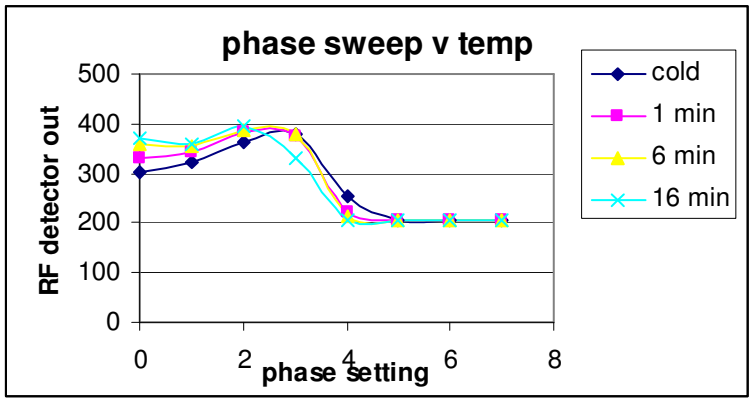
The PSS setting of 0 to 7 determines the sample phase of the input data, relative to the DSPCLK output. This can be varied from 0 to 7 half-cycles of the input clock, or 0 to 875ps at 4GHz in. I've tested five boards successfully, using a default setting of 6 for all. Of course, this setting will change depending on the length of the cable connecting the DAC clock output to the FPGA board, and delays through the receiver chip, etc.

The one-LSB toggle and RF detector test described above works well to find the proper setting, as described above. Here is a scan for four boards:



Data transfer robustness

To test the robustness of the phase setting, I did a sweep of the PSS value with the system cold, then at various points during warmup (with no heatsink).



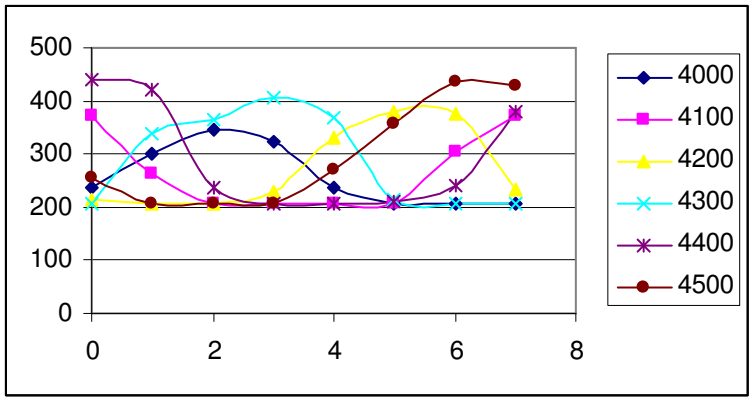
Optimal setting doesn't change much during warmup

Clock drive level

The testing described above was done with a clock of 4GHz, +8dBm. I attenuated the clock input down to -16dBm and the system continued to function normally, including the Fclk/2 output.

Overclocking

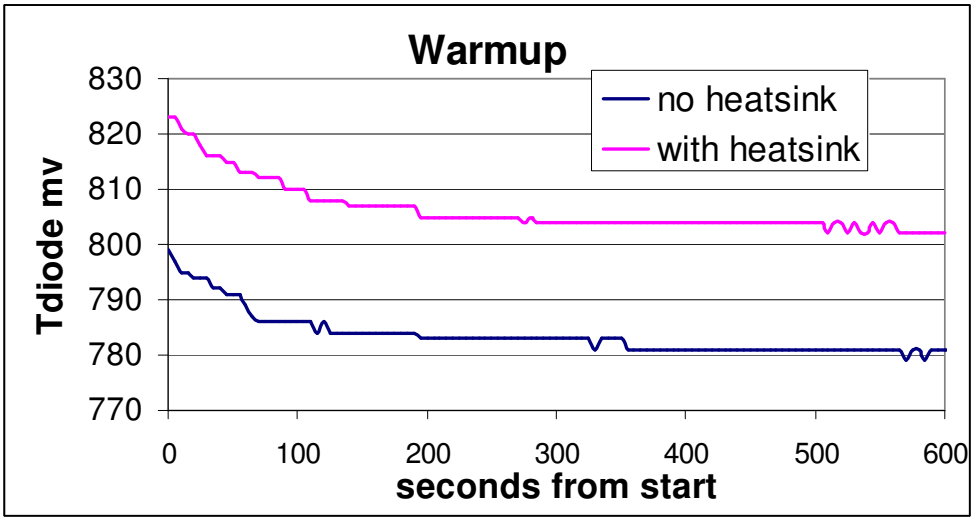
The DAC is specified to work at a sample rate of 4500MHz, while the FPGA is rated to only 3600, as discussed above. I increased the sample rate and swept the phase setting to see where the data transfer failed.



At all sample rates up to 4.5GHz there is a quiet zone (which shifts with clock frequency, as is expected). The system seems to function correctly at 4.5GHz when the proper PSS setting is made. Above this, I believe the FPGA internal logic fails, and although there is a quiet zone in the sweep test up to 5GHz, the output waveforms are not correct (the sweep test causes the data lines to toggle between 1 and 0, which only requires DC levels into the output OSERDES's, whereas the sine wave output requires data to be read from a series of ROMs at Fsample/16). If higher sample rates are beneficial, it seems like they may be achievable.

Thermal Considerations

The DAC chip dissipates 2.6W and gets hot. The chip is placed near the row of mounting holes to facilitate heat conduction through the PCB to a mounting rail. An adhesive-backed heatsink can also be used, if space allows. The warmup curves below show the temperature rise with and without heatsink. Die temperature reaches about 108C without a heatsink and 88C with (using the datasheet transfer curve for the temperature diode):



Absolute maximum junction temperature is 170C, but I think it'd be good to keep the die cooler by using the heatsink if space permits.