ACTIVE OPTICAL SYSTEMS

AN035

Analysis of Bit Precision Requirements for Matrix AO Control

Author: **Justin Mansell, Ph.D., Brian Henderson, and Robert Praus**
Revision: **1/6/12**

1 Abstract

As we move from microprocessor based control to FPGA control, we have identified a need to determine the bit depth requirements for AO control since the FPGA routines we're using operate in fixed point or integer mode only. This document describes analysis we did to determine the number of bits of precision we required to achieve good results from the control system.

2 Analytical Approach

The basic equation we're implementing in the integrator AO control is

$$c_{n+1} = c_n + g \cdot [\Gamma] \cdot \nabla\phi$$

where c_{n+1} is the next command, c_n is the current command, g is the gain factor Γ is the control matrix (Nact x 2*Nsub elements where Nact is the number of actuators and Nsub is the number of Hartmann sub-apertures), and $\nabla\phi$ is the current measured wavefront slope column vector (which we implemented as xx…yy… not xyxy…xy). We will focus on the matrix-vector multiply here because it is the most computationally intensive calculation. We assumed that each row-vector multiply would take place on an individual DSP48 slice of the FPGA. For this we need at least one DSP slice for each actuator. The following table shows the specifications of the Xilinx Virtex 5 FPGA family.[1] Fourteen of the twenty-six FPGAs in this family (53%) have at least 128 DSP48 slices, but only six (23%) have more than 256 DSP48 slices. By making sure that we can fit our calculation into a single DSP48 (48-bit wide output), we open our calculations up to more of the Virtex 5 chip family.

© Active Optical Systems, LLC

Table 1: **Virtex-5 FPGA Family Members**

Device	Configurable Logic Blocks (CLBs)			DSP48E Slices[2]	Block RAM Blocks			CMTs[4]	PowerPC Processor Blocks	Endpoint Blocks for PCI Express	Ethernet MACs[5]	Max RocketIO Transceivers[6]		Total I/O Banks[8]	Max User I/O[7]
	Array (Row x Col)	Virtex-5 Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)					GTP	GTX		
XC5VLX30	80 x 30	4,800	320	32	64	32	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120 x 30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120 x 54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160 x 54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160 x 76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160 x 108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240 x 108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60 x 26	3,120	210	24	52	26	936	1	N/A	1	2	4	N/A	7	172
XC5VLX30T	80 x 30	4,800	320	32	72	36	1,296	2	N/A	1	4	8	N/A	12	360
XC5VLX50T	120 x 30	7,200	480	48	120	60	2,160	6	N/A	1	4	12	N/A	15	480
XC5VLX85T	120 x 54	12,960	840	48	216	108	3,888	6	N/A	1	4	12	N/A	15	480
XC5VLX110T	160 x 54	17,280	1,120	64	296	148	5,328	6	N/A	1	4	16	N/A	20	680
XC5VLX155T	160 x 76	24,320	1,640	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX220T	160 x 108	34,560	2,280	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX330T	240 x 108	51,840	3,420	192	648	324	11,664	6	N/A	1	4	24	N/A	27	960
XC5VSX35T	80 x 34	5,440	520	192	168	84	3,024	2	N/A	1	4	8	N/A	12	360
XC5VSX50T	120 x 34	8,160	780	288	264	132	4,752	6	N/A	1	4	12	N/A	15	480
XC5VSX95T	160 x 46	14,720	1,520	640	488	244	8,784	6	N/A	1	4	16	N/A	19	640
XC5VSX240T	240 x 78	37,440	4,200	1,056	1,032	516	18,576	6	N/A	1	4	24	N/A	27	960
XC5VTX150T	200 x 58	23,200	1,500	80	456	228	8,208	6	N/A	1	4	N/A	40	20	680
XC5VTX240T	240 x 78	37,440	2,400	96	648	324	11,664	6	N/A	1	4	N/A	48	20	680
XC5VFX30T	80 x 38	5,120	380	64	136	68	2,448	2	1	1	4	N/A	8	12	360
XC5VFX70T	160 x 38	11,200	820	128	296	148	5,328	6	1	3	4	N/A	16	19	640
XC5VFX100T	160 x 56	16,000	1,240	256	456	228	8,208	6	2	3	4	N/A	16	20	680
XC5VFX130T	200 x 56	20,480	1,580	320	596	298	10,728	6	2	3	6	N/A	20	24	840
XC5VFX200T	240 x 68	30,720	2,280	384	912	456	16,416	6	2	4	8	N/A	24	27	960

The matrix multiply calculation implemented on each DSP slice is a multiply-and-accumulate (MAC) step implementing the formula

$$\Delta_j = \sum_{i=0}^{2N_{sub}} \Gamma_{i,j} \cdot \nabla\phi_i \, .$$

When multiplying two elements in fixed precision, the product needs at least the sum of the bit precision of the two numbers being multiplied. For example, multiplying two 8-bit numbers will produce a 16-bit number (0xFF times 0xFF = 0xFE01 in hexadecimal). When adding two numbers, one additional bit is needed ('0001 + '0001 = '0010 in binary), but another bit is not needed for 3 additions ('0001+'0001+'0001 = '0011). The number of bits needed for N additions of numbers of M bits is $\log_2(N)+M$. For example, the sum of 8 8-bit numbers has a maximum value of 0x7F8 which is $\log_2(8) + 8 = 11$ bits. Given these rules, the maximum number of bits required for the matrix multiply is given by

$$N_{bits} = \log_2\left(2N_{sub}\right) + P\left(\Gamma\right) + P\left(\nabla\phi\right)$$

where P(x) is the precision of the element x. For a 16-bit control matrix and a 16-bit slope vector, and 128 sub-apertures, the maximum precision required is 40 bits. For certain cases, the maximum precision could be less than this due to factor like the sparcity of the matrices.

© Active Optical Systems, LLC

3 Numerical Simulations

3.1 Setup

After completing the analytical approach, we performed several numerical simulations of an AO system to verify the answer above. We created a model of a DM with an 11x11 actuator grid with the corner actuators removed to create a 112 actuator count. We then employed a version of the Fried geometry for the wavefront sensor making 97 sub-apertures. Figure 3.1 shows a visualization of the overlay of actuator pattern and the sub-aperture grid. The analytical approach above indicates that we need a maximum of 7.6 bits beyond the precision of the control and poke matrices.

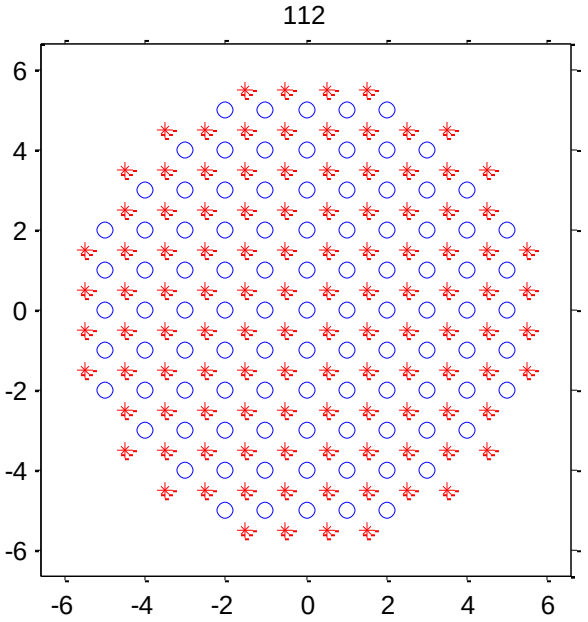


Figure 3.1 – Overlay of the actuator locations (red asterisks) and the sub-aperture centers (blue circles)

For simplicity, we modeled the poking of an actuator as slopes in the four sub-apertures surrounding an actuator pointing away from the poked actuator. Figure 3.2 shows an example ideal poke where the blue quiver plot represents the wavefront gradient (slopes) and the red asterisks are the actuator locations.

© Active Optical Systems, LLC

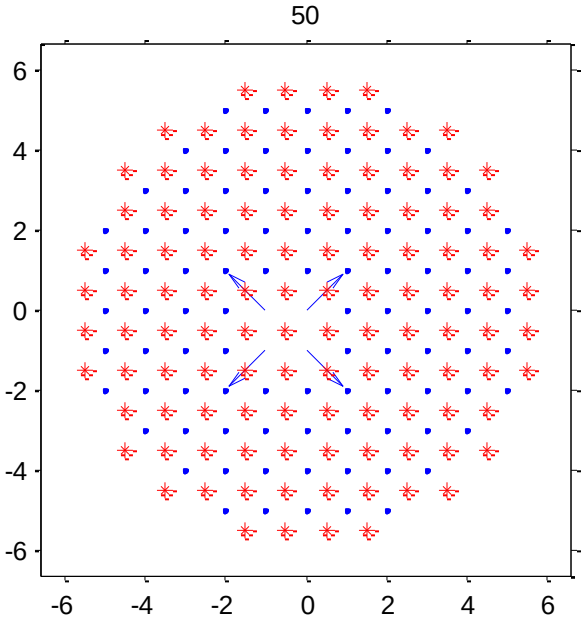
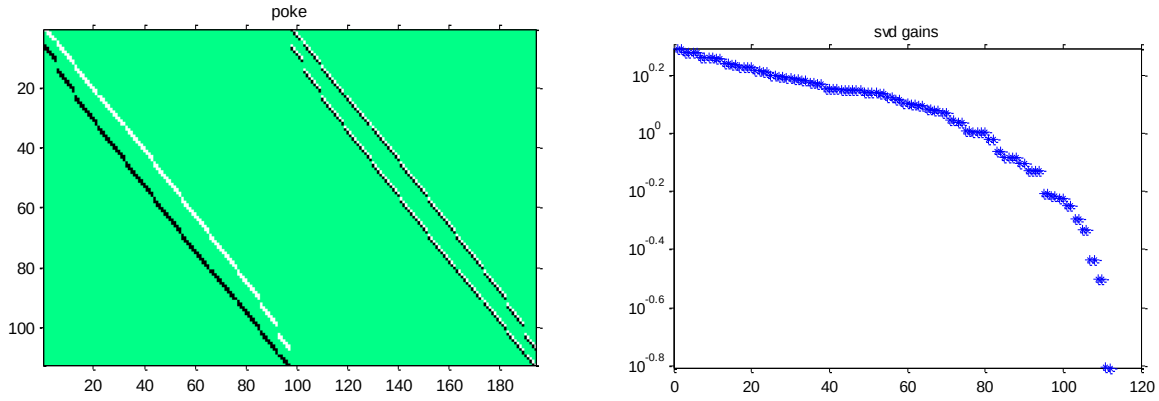


Figure 3.2 - Illustration of an ideal poke from this actuator geometry where the wavefront slopes are a blue quiver plot and the actuator locations are shown as red asterisks

We made an ideal poke matrix by combining the slope vectors from the poke on each actuator and then inverted it using single value decomposition all in double numerical precision. Figure 3.3 shows the results from this process.



© Active Optical Systems, LLC

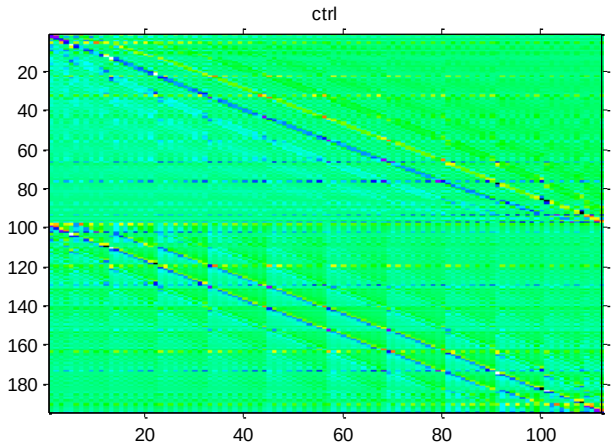


Figure 3.3 - Poke Matrix, SVD Gains, and Control Matrix for the system

3.2 Expected Slope Precision

In many cases, a Shack-Hartmann wavefront sensor is manufactured for AO with 8 pixels across each sub-aperture. The slope vector can be represented in pixel deltas since the normal division by the lens array focal length is a simple scaling factor that can be introduced later. Thus, the range of numbers coming from a slope measurement is +4 to -4 (3 bits + 1 sign bit = 4 bits). If we assume that we can achieve ~1/20 of a pixel accuracy with centroiding, we can almost represent that with 4 bits of decimal precision. Therefore, it is likely that we will only achieve 8 bits of slope precision from a typical Shack-Hartmann wavefront sensor.

3.3 Digitizing Poke and Control Matrices

With this setup, we digitized the poke matrix to only have 8 bits of precision in Matlab with:

```
poke_8bit = poke ./(max(abs(poke(:))));
poke_8bit = round(poke.*(2^8))./2^8;
poke_8bit = poke .*(max(abs(poke(:))));
```

We then created a control matrix in double precision using this digitized poke matrix using SVD. The RMS difference between the identity matrix and the product of the digitized poke and double-precision control, we obtained 2.4754e-016, which is comparable to the precision expected from double-precision calculations of this sort.

We then examined the quality of the control matrix as a function of the number of bits of rounding (precision) with the following Matlab code:

```
bitvec = 4:32; cnt=0;
for bits=bitvec;
    cnt=cnt+1;
    ctrl2 = ctrl_8bit ./ max(abs(ctrl_8bit(:)));
    ctrl2 = ctrl2 .* (2^bits);
    ctrl2 = round(ctrl2);
    ctrl2 = ctrl2 ./ (2^bits);
    ctrl2 = ctrl2 .* max(abs(ctrl_8bit(:)));
    errmatrix = poke_8bit*ctrl2 - eye(Nact);
    errrms2(cnt) = RMS(errmatrix(:));
    errmax(cnt) = max(errmatrix(:));
 end;
```


© Active Optical Systems, LLC

The bulk of this code is reducing the precision of the control matrix to the number of specified bits. We then looked at the RMS error of the difference between the 8-bit poke matrix and the precision-limited control matrix. This procedure is equivalent to looking at the error associated with generating a command for a single poke. We were particularly interested in the 2^{-16} answer since we are using 16-bit DACs. Figure 3.4 shows the results. Not surprisingly, to achieve 16-bits of output precision, we needed 16-bits of control matrix precision.

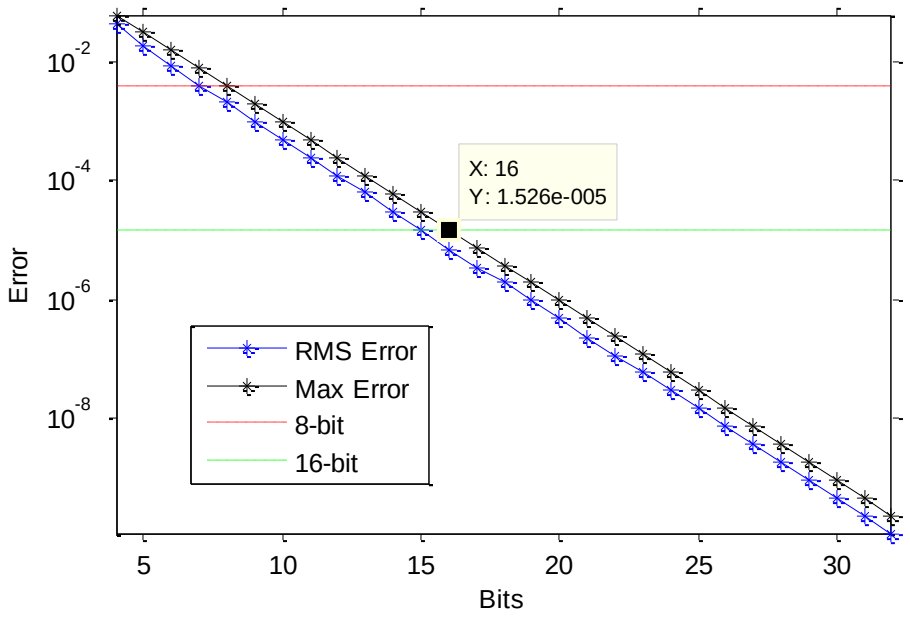


Figure 3.4 – RMS Error from identity versus bits of precision.

Examination of the difference between the poke-control product and the identity matrix showed similar patterns for each case. There were many zero values, but the rest were all +1 or -1 times some small number. The patterns changed for the varying precision, but the form did not. It was almost like this pattern corresponded to something about the bit near the round-off position. Figure 3.5 shows an example of this pattern at 32-bits of precision.

© Active Optical Systems, LLC

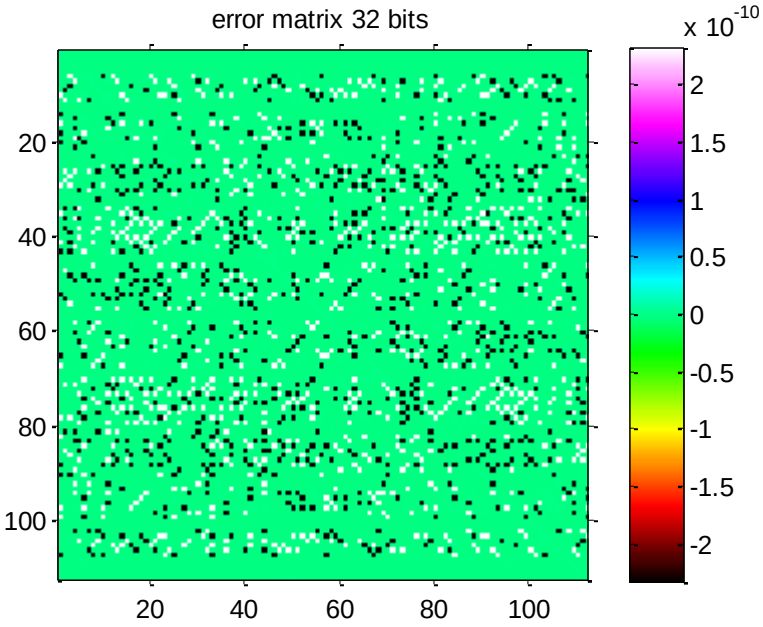


Figure 3.5 - Error Matrix at 32-bits control matrix precision

These results indicate that even if we have only 8-bits of precision in the poke matrix, you can generate results with 16-bits of output precision using a 16-bit precise control matrix. If we have 8-bits of precision in the slopes and 16-bits of precision in the control matrix, in this system we will need a maximum of 8+16+8=32 bits. The results shown in this section are optimistic though because the math was done using floating-point operations. We need to examine the effect of precision reduction due to non-floating-point math.

3.4 Adding Limited Precision to the Control-Matrix Slope-Vector Calculation

To add limits in the precision of the calculation of the control matrix, we leveraged Matlab's int64 data type. To make sure that the math was taking place in integer operations, we implemented the matrix multiply operation inside a for loop where we could control each multiplication in integer space. (This is much slower than leveraging Matlab's operations, but it will enforce the integer operations).

```
ctrlbits = 16;
slopebits = 8;
svec = ones(2*Nsub,1);
ctrl = ctrl ./max(abs(ctrl(:))) * 2^ctrlbits;
svec = svec ./max(abs(svec(:))) * 2^slopebits;
dcmd = int64(zeros(Nact,1));
pow2max=0;
for ii=1:size(ctrl,1); %2*Nsub
    for jj=1:size(ctrl,2); %Nact
        cval = int64(ctrl(ii,jj));
        sval = int64(svec(ii,1));
        dcmd(jj) = dcmd(jj) + cval*sval;
        pow2 = log(double(dcmd(jj)))./log(2.0);
        if (pow2>pow2max) pow2max = pow2; end;
    end;
end;
```

During each of these calculations, we looked at the maximum value of the accumulated command delta and found that the answer was 26.8 bits. This is less than the predicted 31.6 bits (8+16+7.6), which we believe is because the matrix was sparser than the worst case. The output delta command from this multiply (assuming a maximum positive slope case) was 26.7 bits.

© Active Optical Systems, LLC

3.5 Moving from Fixed Point to Integer

With proper scaling factors, the fixed-point operations are effectively equivalent to integer operations. For example, 0.1+0.1=0.2 is the same as 1+1=2 working in 10x space. Working in integer space can make thinking about the problem a bit easier, but does not change the results. To show this in an AO control example, we wrote the following Matlab code:

```
poke2 = round(poke./max(abs(poke(:))) .* (2^slopebits));
[ctrl,gains]=svdinverse(poke2,modesRemoved);
ctrl2 = round(ctrl./max(abs(ctrl(:))) .* (2^ctrlbits));
InvMatrix = poke2*ctrl2;
scalingBits = log(max(InvMatrix(:)))./log(2.0)
```

In this code, the double-precision poke matrix is scaled to a maximum value of 2^8 (slopebits = 8). The limited precision poke matrix is inverted with zero modes removed and the generated control matrix is scaled to a maximum precision of 2^{16} (ctrlbits=16). Using this procedure, the poke and control matrix product (InvMatrix) is nearly identity with a maximum value along the diagonal of 2^{25}. The few off-diagonal non-zero elements in the matrix were all ±512 (2^9). The variation in the on-diagonal elements was also ±512. Figure 3.6 shows the difference between the control and poke matrix product and the identity matrix times 2^{25}. The errors were consistent with the 16-bit precision on the control matrix (25-9=16).

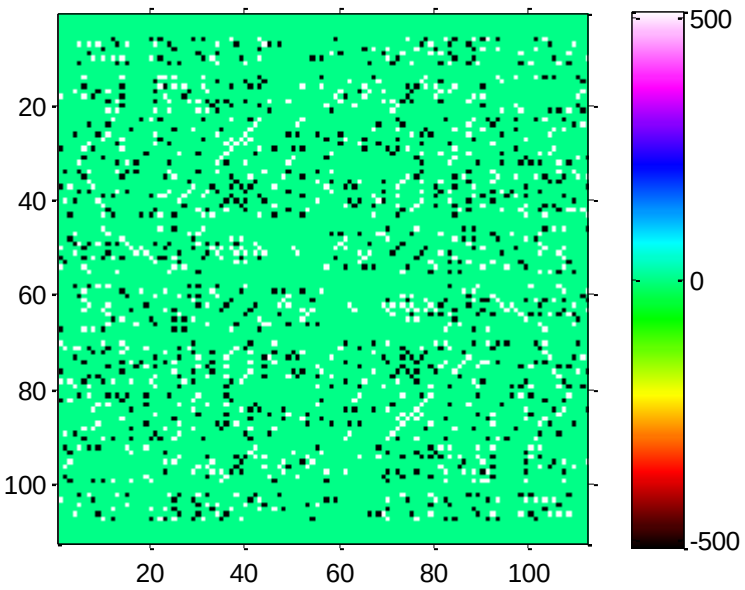


Figure 3.6 - The difference between the limited control and poke matrix product and the identity matrix times 2^{25}.

This demonstration shows that if the poke matrix is scaled to fit in an integer 8-bit range and the control matrix is scaled to be an integer 16-bit range using the procedure shown above, the resulting output vectors will be in the $2^{(16+8+1)}$ range with noise in only the 9th bit.

3.6 Suggested Operational Procedure 1

Based on this analysis, we recommend the following operational procedure:

1. Measure the influence functions with maximum poke strength (or such that they can be scaled to effective maximum strength).
2. Scale the poke matrix to a desired bit depth with the following procedure:

© Active Optical Systems, LLC

```
round(poke./max(abs(poke(:))) .* (2^slopebits));
```

3. Invert the poke matrix using SVD to create a control matrix.
4. Scale the control matrix to the maximum desired output bit depth (typically specified by the DACs):

```
round(ctrl./max(abs(ctrl(:))) .* (2^ctrlbits));
```

5. In AO control, scale your measured slopes (centroid motion) using the same factor in the poke matrix generation and multiply that vector times the new control matrix. Use only the upper most significant 16 bits of the output.

The only challenge with this approach is dealing with the scaling factor on the slopes.

4 References

[1] http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

© Active Optical Systems, LLC