Update of the register size used in the NeXtRad Synchronization Controller

Dominic Manthoko

March 31, 2017

1 Introduction

1.1 Subject and Motivation of Report

The author was requested to make modifications to the NeXtRad Synchronization Controller developed by Johann Burger. In particular, the register size of the Synchronization Controller needed to be increased from 16 bits to 32 bits.

1.2 Background of the NeXtRad Synchronization Controller

1.3 Methodology

2 The Synchronization Controller

2.1 Modifications Made to the Original Synchronization Controller

It is highly recommended that you read the document about the Timing Control Unit(TCU). This document specifies how to set up the synchronization

Parameter	Size	Description
MB offset	16 bits	Main Bang offset in 10ns intervals
DIG offset	16 bits	Digitisation offset in 10ns intervals
Next PRI (upper 2 bytes)	16 bits	Next PRI offset in 10ns intervals
Freq	16 bits	Frequency of operation
Pol Mode	3 btis	From table 1.2
Next PRI (lower 2 bytes)	16 bits	Next PRI offset in 10ns intervals

Table 1: Table of all the parameters necessary for each pulse

controller, the commands used to operate it and so forth. In this section, only the parts that were modified will be mentioned.

2.2 Setting up the Synchronization Controller

All the available registers can be accessed in the proc directory:

```
/proc/<PID>/hw/ioreg/
```

where <PID> must be replaced by the process ID of the running firmware. This can be identified by running the 'ps' command in the terminal.

Registers can be written to by using the echo command. The following command writes a 1 to the n register:

```
echo -e -n "\x01\x00" > /proc/<PID>/hw/ioreg/n
```

The registers can then be read back by using the 'od' command:

```
od -x /proc/<PID>/hw/ioreg/n
```

With the modified TCU, reading and writing from the n, m and reg_led registers found in the proc directory was left unchanged. The reg_pulse register was altered such that it would be able to handle 32 bit values for the PRI offset.

The original developer catered for the future improvement of the controller and had 16 bits of unused bits available for use. This free space was made use of in order to increase the register size of the PRI from 16- to 32-bits. In table 1, the various parameters required to set up a pulse are shown. Figure

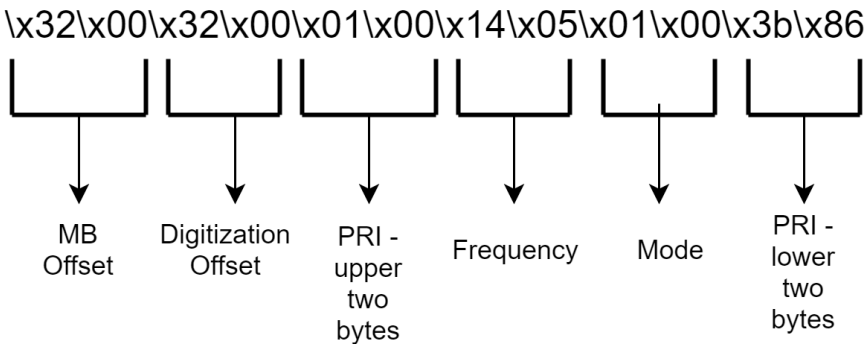


Figure 1: Visual depiction of how to order the parameters seen in Table 1

1 shows how the parameters seen in table 1 would be setup using the echo command.

The parameters seen in table 1 are set up by writing to the reg_pulses register. If you wanted to set up a pulse that triggers every 1 ms (1Khz), has a main bang and digitisation offset of 500ns, a band frequency of 1300 MHz and operating at L-band, you would write the following command.

```
echo -e -n "\x32\x00\x32\x00\x01\x00\x14\x05\x01\x00\x3b\x86"
> /proc/<PID>/hw/ioreg/reg_pulses
```

The Rhino operates using a 100MHz clock, which is 10ns cycles. For the main bang and digitization offset, we divide the desired offset by 10ns to get the number of cycles. Since we want an offset of 500ns for both of them, the number of cycles is equal to 50 (x0032 in hexadecimal). The value must be in little endian thus we set these to \x32\x00.

The desired PRI is calculated using the following equation:

$$PRI = \frac{\frac{1}{PRF}}{10ns} - MB - D$$

Where

- PRF is the frequency we want at the output i.e coming out of the main bang pin on the Rhino board
- MB is the main bang offset in cycles
- D is the digitization offset in cycles

Since MB = D = 50 and we want a PRF of 1kHz, we get a value of 99899 for the PRI. Converting the value to hexadecimal we get x0001 863B. Then you would write \x01\x00 to PRI upper two bytes and \x3b\x86 to PRI lower two bytes.

3 Experimental setup

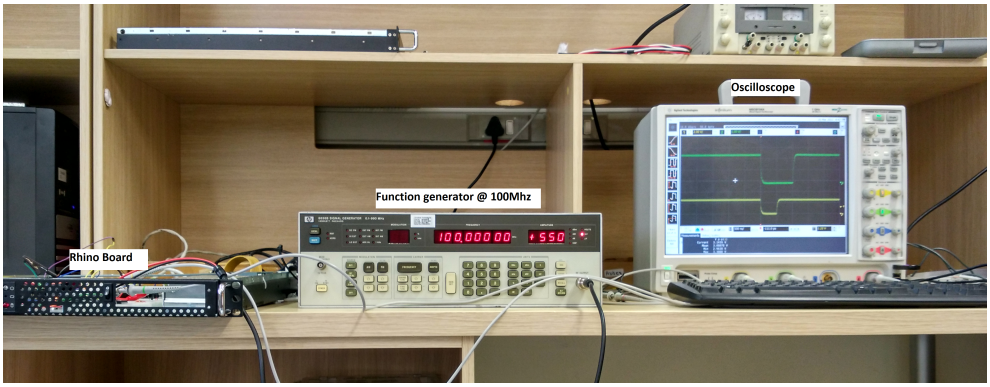


Figure 2: Picture of the equipment used to test the 32 bit version of the synchronization controller

The experimental setup used to test the modified controller can be seen in figure 2. A signal generator operating at 100MHz with an amplitude of +550mV was used as the input clock. This clock signal was passed through a diode then fed into the rhino. An oscilloscope was used to measure the Main bang offset and Digitisation signals coming out of the rhino.

A computer running the 64 bit version of xubuntu 16.04 LTS; which is a linux distribution based on ubuntu; was used to modify the VHDL code. Xilinx IDE verison 14.7 was used to edit the original TCU project which can be found here.

4 Results of the Experiment

4.1 Using the Rhino Board Internal 100MHz clock

Testing of the modified synchronization controller began with using the internal clock of the Rhino board. Using the internal clock, 1kHz, 2kHz, 3kHz and 4kHz signals were produced.

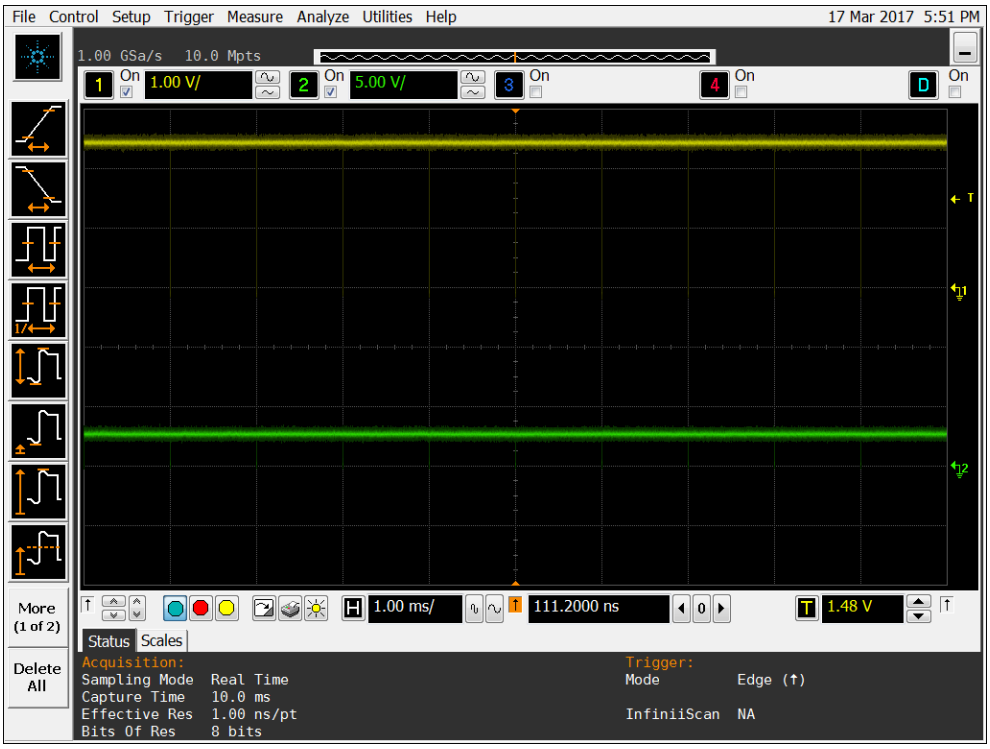


Figure 3: 1kHz signal produced using the internal clock of the rhino

Figure 3 shows the output of the main bang and the digitisation signals measured from the Rhino board. A time division of 1.0 ms was used to display the output. Observing the yellow signal, it can seen that a 1kHz signal is indeed being produced by the Rhino board.

Figure 4 shows a zoomed in image of Figure 3. The yellow signal depicts the main bang signal and the green signal depicts the digitization signal. The yellow signal goes high after 500ns and the green signal triggers 500ns after the yellow signal. These results are as desired.

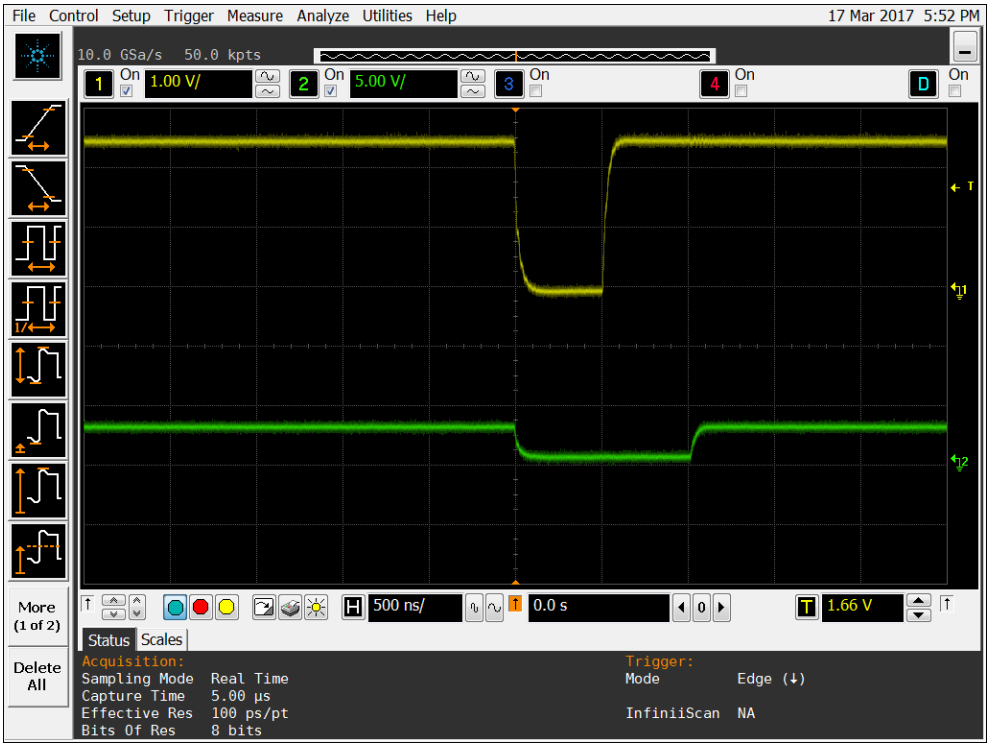


Figure 4: Zoomed in image of Figure 3

To change the main bang offset and the digitization offset to 1000ns without changing the other parameters, the following command was used:

```
echo -e -n "\x64\x00\x64\x00\x01\x00\x14\x05\x01\x00\x3b\x86"
> /proc/<PID>/hw/ioreg/reg_pulses
```

Figure 5 shows a picture of the main bang and digitization signals with 1000ns offsets. Using the 500ns time division on the horizontal axis, it can be seen that it takes two time divisions for the main bang signal(yellow signal seen at the top of figure 5) to go high. This equates to 1000ns, which is the desired result. Similarly for the digitization signal, we see that it also takes 2 time divisions to trigger, thus meeting our desired result.

To show that the main bang offset and digitization offset can be set to different values, the main bang offset was set to 1650 ns and the digitization offset was set to 2950ns. The following command was used to set these values:

```
echo -e -n "\xa5\x00\x27\x01\x01\x00\x14\x05\x01\x00\x3b\x86"
> /proc/<PID>/hw/ioreg/reg_pulses
```

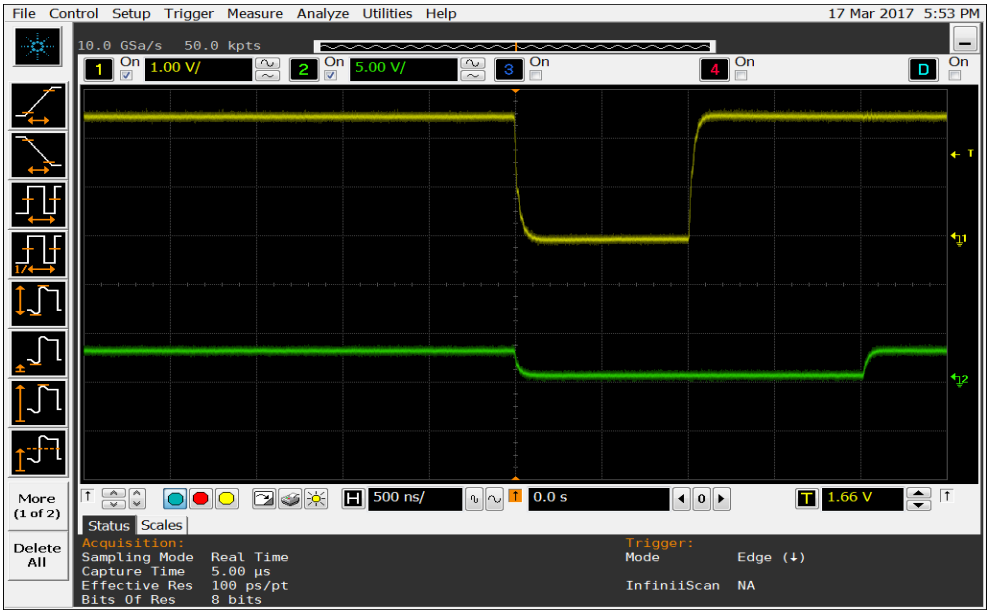


Figure 5: Picture of the main bang signal and digitization with 1000ns offsets

Figure 6 shows the plots of the main bang signal and the digitization signal together with the measurements. Making use of cursors on the oscillscope, which allow you to measure a value on the horizontal and vertical axis, the offsets were measured. In the markers tab seen in figure 6a, there is a delta value. From this plot, we see that $\Delta = 1.6555556\mu$. Observing the delta seen in figure 6b, we see that $\Delta = 2.9541483\mu$.

A main bang offset of 1650ns is equivalent to 1.65μ. The value of delta seen in figure 6a is thus approximately equal to the value that was set using the echo command. For the digitization offset, we set a value of 2950ns (2.95μ). The delta value seen in figure 6b is also approximately equal to the value that was set. This proves that the main bang and the digitization offsets can be set to two different values.

Figures 7, 8 and 9 show pictures of the output produced when a signal with either a 2kHz, 3kHz or 4kHz frequency is set using the echo command. Each of the plots was set using a 500ns offset for the main bang and digitization offsets, using a band frequency of 1300 Mhz, operating at L-band. The only that changes when switching between frequency is the PRI that is set.

To produce a 2kHz signal, we would need a PRI of 49899 (calculated using the equation for PRI found in section 2.2). The following command was used

to set the parameters for the 2kHz signal:

```
echo -e -n "\x32\x00\x32\x00\x00\x00\x14\x05\x01\x00\xeb\xc2"
> /proc/<PID>/hw/ioreg/reg_pulses
```

To produce the 3kHz and 4kHz signals, the same command as above was used, the only difference being that a new PRI value was calculated and this new value plugged into the appropriate position in the command.

Observing figure 7, we see that a 2kHz signal is being produced. A 1.00ms time division is being used and in 1 time division, the signal goes high twice. This equates to the signal going high every 2ms which is equivalent to 2kHz.

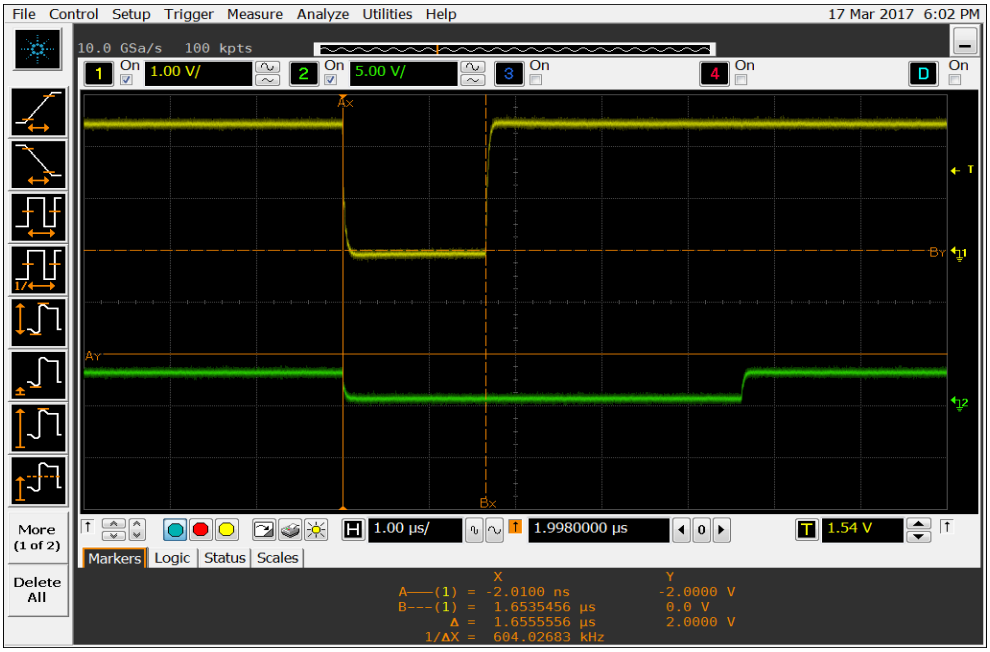
Observing figure 8, which has a time division of $100\mu s$, we see that a 3kHz is being produced. The main bang signal (yellow signal) goes high at about 3.3 time divisions. When you multiply the number of time divisions with the value of 1 time division, we get $330\mu s$. This is equivalent to 3z, which is exactly what we wanted. From figure 9 we can see that a 4kHz signal is being produced. A time division of $500\mu s$ is being used. Thus 1 time division is equal to 2kHz. In this picture, we see that the yellow signal goes high twice for every time division. Thus, we have twice the frequency observed in a single time division. i.e. a 4khz signal is being produced.

4.2 Using an External 100MHz Clock Signal

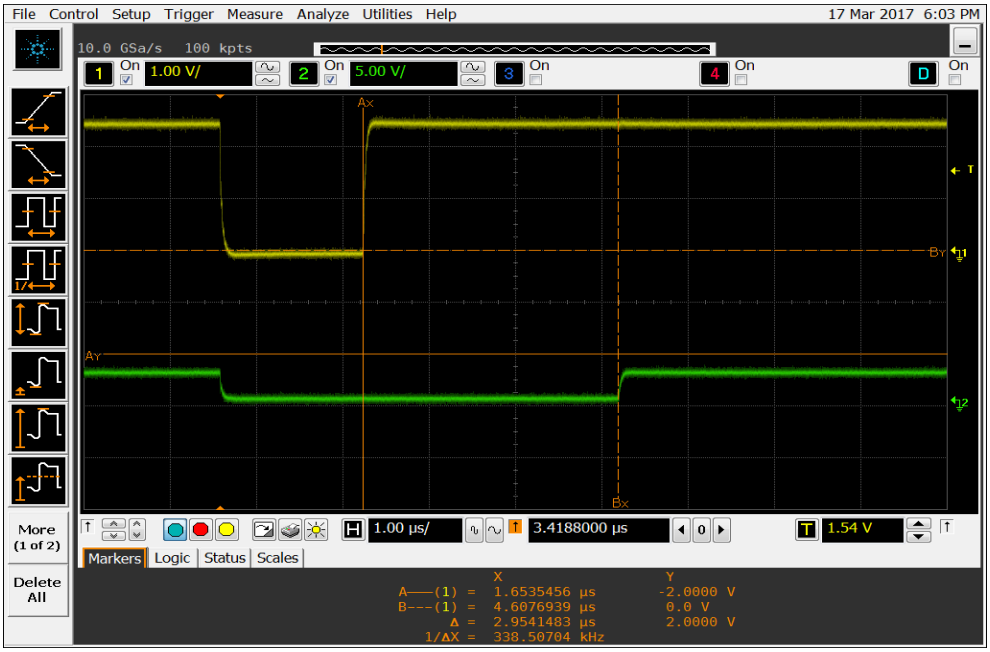
Initial testing was done by making use of the internal 100MHz clock of the rhino board as mentioned in the previous section. Since the final setup would be run using an external 100MHz clock, tests were conducted to ensure the updated synchronization controller would be able to operate correctly using an external clock.

5 Conclusions

6 Recommendations



(a)



(b)

Figure 6: (a) Main bang and digitization signal with the measurement of the main offset shown in the markers box toward the bottom of the image. (b) Same as figure 6a but with the measurement of the digitization offset shown in the markers box.

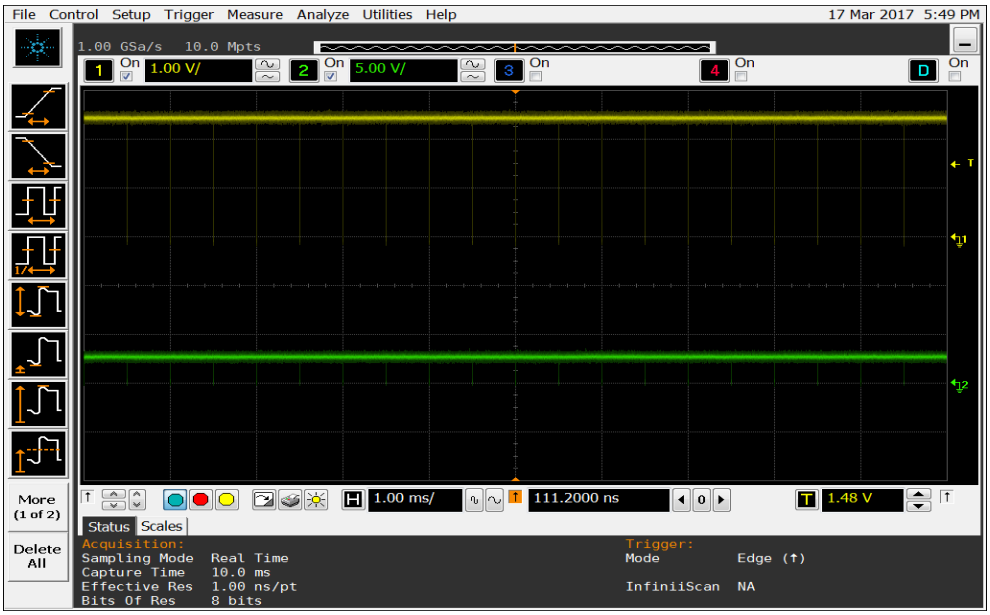


Figure 7: Picture showing a 2kHz signal produced using the internal clock of the Rhino board

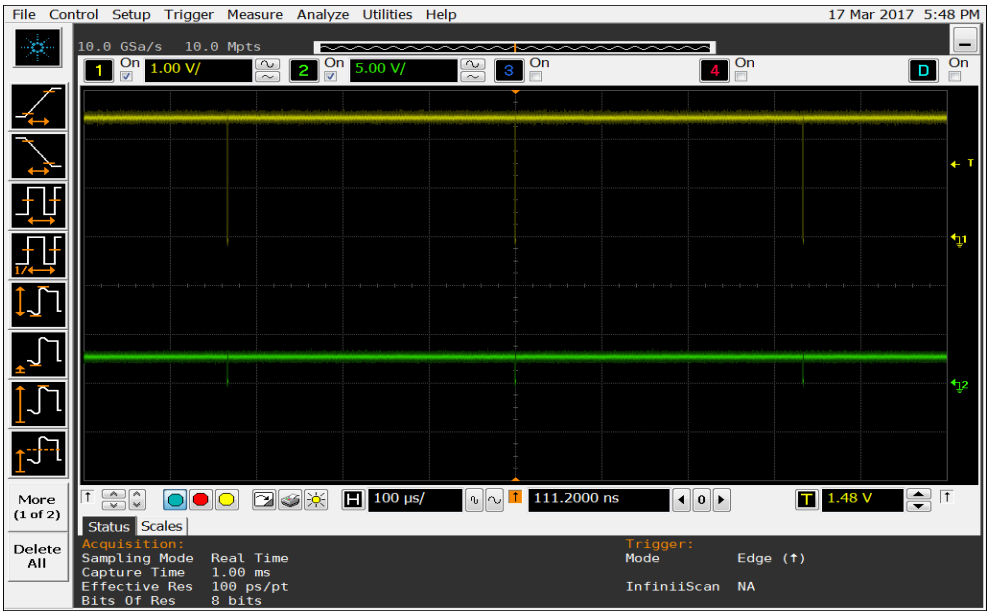


Figure 8: Picture showing a 3kHz signal produced using the internal clock of the Rhino board

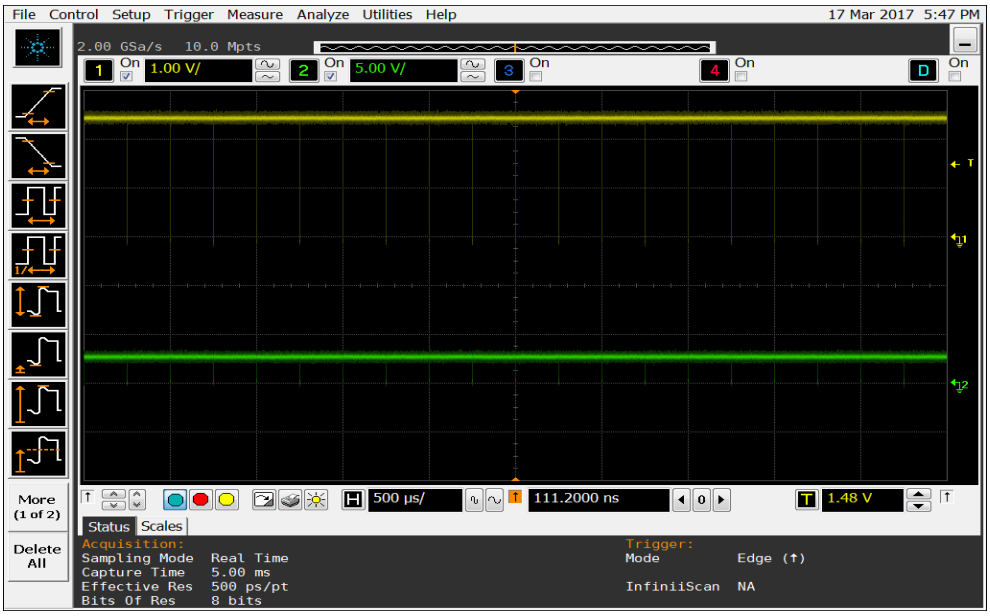


Figure 9: Picture showing a 4kHz signal produced using the internal clock of the Rhino board