FPGA-Based Error Rate Measurement

Introduction[1]

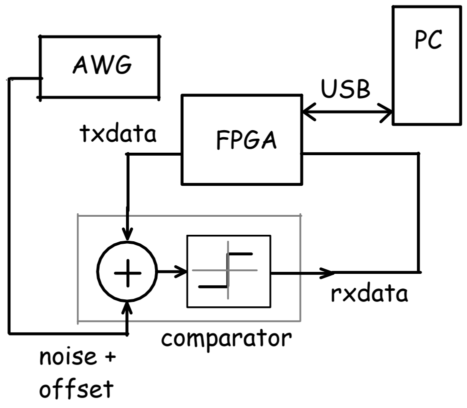
A Field-Programmable Gate Array (FPGA) is a logic device that can be programmed to implement any desired digital logic function.

FPGAs are useful for implementing small-volume products and to set up test fixtures, particularly when the signal frequencies are too high to process signals in software.

Logic analyzers are instruments that recognize patterns in digital signals and capture and display them. FPGAs can be configured to include logic analyzer features. This approach is often more flexible than using stand-alone logic analyzers.

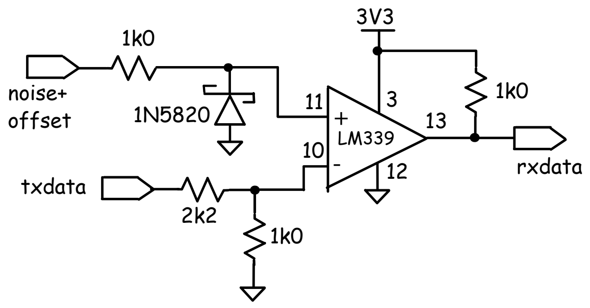
In this lab you will use an FPGA to implement a circuit that measures the error rate of data transmitted over a channel. The test circuit on the FPGA generates and outputs random data at 100 kbps. It compares the received data with the transmitted data and counts the number of errors and number of transmitted bits. The ratio is the bit error rate (BER). The values of the two counters will be read using a logic analyzer embedded in the FPGA design.

You will also build a simple circuit that adds additive white Gaussian noise (AWGN) to the transmitted data (`txdata`). The Arbitrary Waveform Generator (AWG) outputs samples of pre-computed AWGN plus a DC offset voltage. The comparator generates the received data signal (`rxdata`):



[1]Revised November 5, 2015.

The schmatic of the comparator circuit is shown below:



The noise plus a DC offset is applied to the non-inverting input. The 1 kΩ current-limiting resitor and the 1N5820 Schottky diode prevent the input going more negative than about -0.3 V and so prevent damage to the comparator.

The 0 to 3.3 V txdata signal is applied to the inverting input through a voltage divider. This keeps the inverting input in the range of approximately 0–1 V where the comparator can provide a valid output regardless of the level at the non-inverting input (see datasheet for details).

The comparator has an open-collector output so a pull-up is required on the output.

The AWG is configured to add a DC offset of approximately 0.5 V so that $V_+ = 0.5 + N$ where N is the noise voltage. V_- is a data signal (`txdata`) of approximately 0-1 V. The comparator output will be high when $V_+ > V_-$ or when $N + 0.5 >$`txdata` and low when $V_+ < V_-$ or when $N + 0.5 <$`txdata`. In the first case an error will happen when txdata is +1 and $N > 0.5$. In the second case an error will happen when txdata is 0 and $N < -0.5$. For zero-mean Gaussian noise both probabilities are the same. Thus the BER can be predicted as the probability that the noise is less than the threshold value of ≈ -0.5.

Note that the data also appears inverted at the output (txdata=1 results in a low output). However, the FGPA can easily invert `rxdata`.

In the lab you will measure the BER for various levels of noise power (standard deviation) and compare the results to predicted values.

You will use an Altera *DE-0 Nano* FPGA board that includes a Cyclone IV FPGA, 8 LEDs, two pushbuttons and two connectors with 40 I/O pins each.

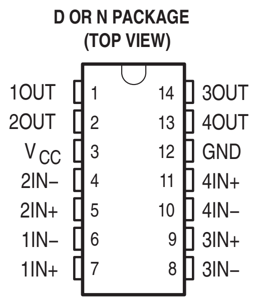
The FPGA is configured with a 100 kHz clock driving a pseudo-random bit sequence (PRBS) generator and two counters. One counter counts up on every clock cycle, the other counts up only when the received data does not match the transmitted data (the XOR gate does the comparison).

Figure 1 shows the schematic of the FPGA. It includes a 100 kHz clock generator, a pseudo-random bit sequencer (PRBS) generator and the bit and error counters.

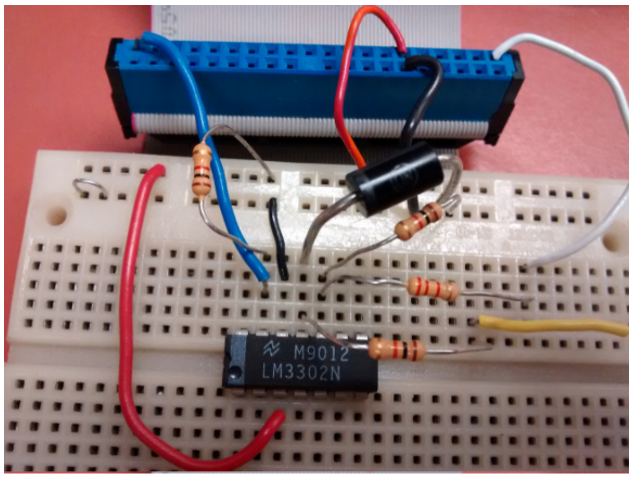
The FPGA board has two pushbuttons. One is used to reset both counters and the PRBS generator. The other is used as a trigger by the embedded logic analyzer. The comparator power is supplied by 3.3VDC from the FPGA board. The FPGA is not tolerant of 5V signals so external power supplies must not be used (50% of your lab mark will be deducted if you connect or turn on a power supply during this lab!). The FPGA is programmed over a JTAG interface plugged into one of the PC's USB ports. The JTAG interface is also used for the logic analyzer interface.

Threshold Comparator Circuit

You can substitute any comparator that operates at 3.3V and has an open-collector output (for example the LM3302 which has the same pin-out as the LM339). The pin-out of these comparators is:



The following diagram shows how the circuit can be assembled on a prototyping board and connections made to a cable that connects to the FPGA board. The red stripe (on the left of the photograph below) marks the side of the ribbon cable connected to pin 1. The leads of the Schottky diode are too large to fit in the breadboard and must be cut down. The white band marks the cathode.

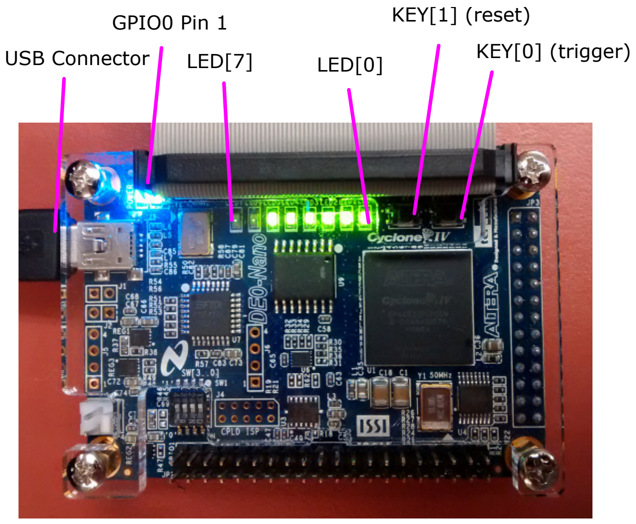


FPGA Board

The connections to the FPGA are made through a supplied 40-pin ribbon cable plugged into the GPIO0 connector on the FPGA board (see photo below).

Obtain 3.3V from the ribbon cable connector pin 29 and ground from pin 30 for the comparator circuit.

The following photograph shows the LEDs and pushbuttons on the board:



You will use the `reset` pushbutton (labelled `KEY1` on the board) to reset the error counter. You will use the `trigger` pushbutton to trigger the logic analyzer and display the currrent bit and error counts.

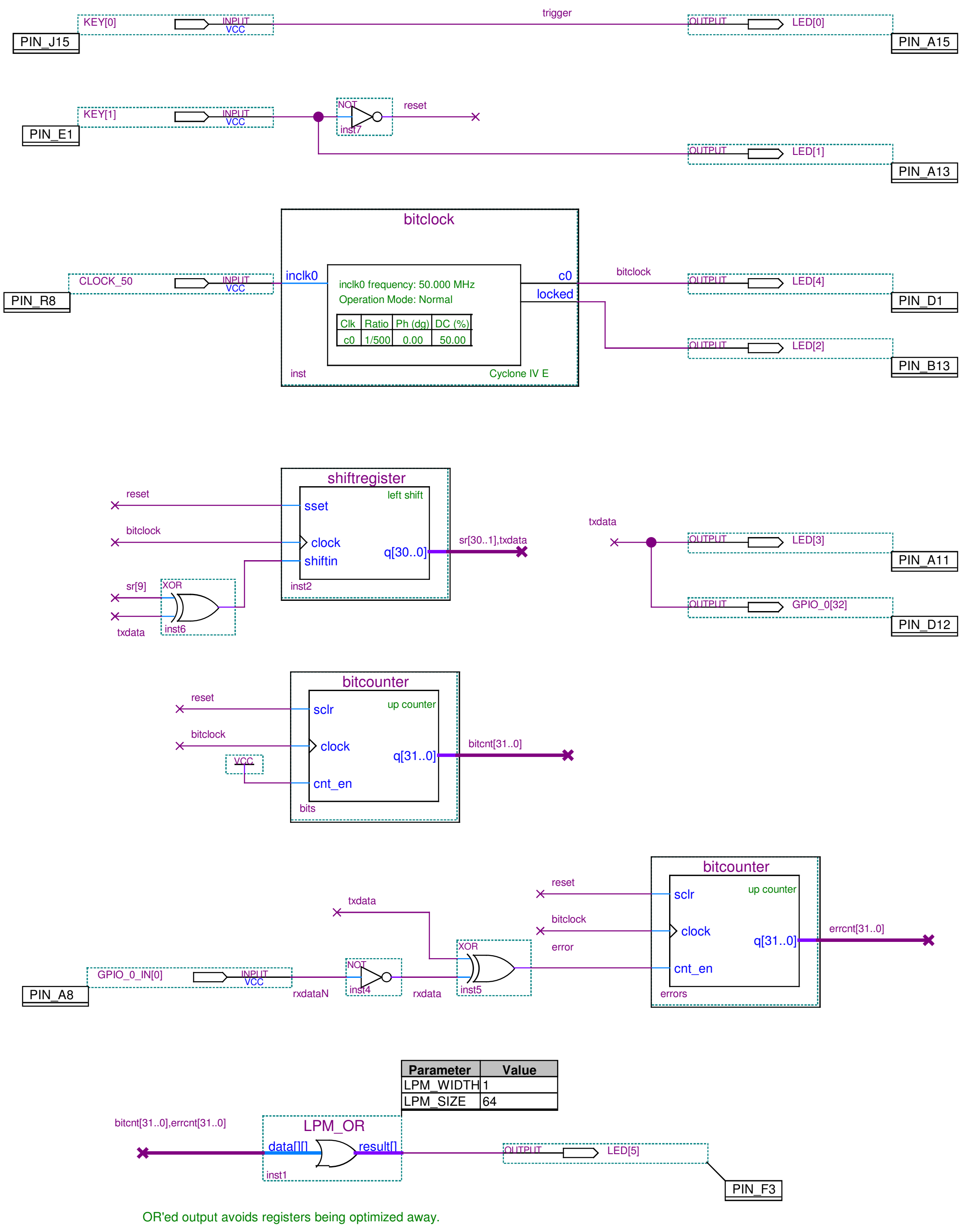


Figure 1: Bit and Error Counters.

The following diagram labels the connection points on the ribbon cable connector with pin 1 (red conductor) on top. The letter-number pairs are the row and column respectively on the 256-pin (16×16) BGA (ball-grid array) FPGA package; numbers beginning with zero (0) are signal names used in the the DE-0 documentation (prefixed with `GPIO_`), the other numbers are the 40-pin "header" connector numbers. For example, the signal on FPGA pin A2 has signal name GPIO_02 and is on pin 5 of the connector (third pin from the top on the right side).

D3	00	2	□	□	1	0_IN0	A8
C3	01	4	□	□	3	0_IN1	B8
A3	03	6	□	□	5	02	A2
B4	05	8	□	□	7	04	B3
B5	07	10	□	□	9	06	A4
GND	–	12	□	□	11	–	5V
D5	09	14	□	□	13	08	A5
A6	011	16	□	□	15	010	B6
D6	013	18	□	□	17	012	B7
C6	015	20	□	□	19	014	A7
E6	017	22	□	□	21	016	C8
D8	019	24	□	□	23	018	E7
F8	021	26	□	□	25	020	E8
E9	023	28	□	□	27	022	F9
GND	–	30	□	□	29	–	3.3V
D9	025	32	□	□	31	024	C9
E10	027	34	□	□	33	026	E11
B11	029	36	□	□	35	028	C11
D11	031	38	□	□	37	030	A12
B12	033	40	□	□	39	032	D12

FPGA Configuration

The procedure to design the error rate measurement circuit and configure the FPGA using the (free) Quartus II software is as follows:

- start Quartus II (64 bit)
- File -> New -> New Quartus II Project
- create or select a folder, project name (e.g. lab7), the Cyclone IV E Family, and the EP4CE22F17C6 device, and leave other settings set to defaults
- select Assignments -> Import Assignments -> DE0-Nano.qsf to assign FPGA pin numbers to the signals on the DE0-Nano board. This file is available on the course web site
- Remove the assignment that sets the drive levels of all output pins to "Minimum". Under Assignments -> Assignment Editor, Sort on the Assignment Name column to find the line that sets "Current Strenght" for all pins ("*") to "Minimum Current". This setting should be disabled (click on the"Enabled" column and select "No") or deleted (right-click on the line and select "Delete").
- File -> New -> Block Diagram/Schematic File
- if necessary, select View -> Utility Windows -> IP Catalog to show the list of configurable intellectual property (IP) blocks
- select Library -> Basic Functions -> Arithmetic -> LPM_COUNTER
- define the counter component by specifying a name (e.g. bitcounter), VHDL source, 32-bit up-only counter with count enable and synchronous clear inputs. Enable generation of a symbol (.bsf) file.
- select Library -> Basic Functions -> Miscellaneous -> LPM_SHIFTREG
- define the shift register by specifying a name (e.g. shiftregister), 31 bits, left shift, [parallel] Data output and a Serial shift data input. Add a synchronous set input and enable generation of a symbol file.
- similarly, define a clock component by selecting Library -> Basic Functions -> Clocks; PLLs and Resets -> PLL -> ALTPLL
- specify a name (e.g. clock), a 50 MHz input, locked and a 100 kHz clock outputs (c0), and no additional inputs or outputs. As before, generate a symbol file.
- use the "Symbol Tool" and insert two counters, one clock and one shift register from the Project library as shown in the schematic. Give each component instance a unique label (e.g. bits, errors, clock).

- use the "Symbol Tool" and insert the required `xor` gates (one for LFSR generator and one for error detection), inverters (`not`), one Vcc constant (to enable the bit counter), and one 64-input OR gate (Megafunctions -> gates -> `lpm_or`) as shown in the schematic. *Hints: Use the search box to find components. It is often easier to give signals names (right-click -> Properties) rather than connecting them.* Assign names to the bit and error count signals so the logic analyzer can select them.

- use the "Pin Tool" to insert the output and input ports labeled with the exact pin names shown in the schematic[2] Although the pushbutton and counter outputs are only used by the logic analyzer, they must drive output pins so that they are not optimized away. The error and trigger pushbuttons also drive two LEDs to verify that the FPGA has been properly configured.

- use the "Bus", "Node" and "Selection" tools to connect the components as shown in the schematic. Note that the bus pin numbers in the schematic below should be `[31..0]`.

- click on the "Start Compilation" icon. Correct any errors and recompile as necessary.

- connect the FPGA to the PC's USB port and select Tools -> Signal Tap II Logic Analyzer

- under JTAG Chain Configuration -> Setup select USB Blaster

- under SOF select the correct `.sof` (FPGA serial programming) file in the output_files folder

- click on the download ("Program Device") icon to make sure the device is recognized and can be programmed.

- under Signal Configuration use the node finder to list all pins using the SignalTapII:pre-synthesis filter and add the `bitclock` as the clock

[2]Signal names must match the symbol names in the DE0-Nano settings file or else they will not be connected to the correct FPGA pins.

- in the logic analyzer window double-click in the indicated area to add signals. As before, list the pins and add the GPIO_0[32] (`txdata`), GPIO_0_IN[0] (`rxdata`), KEY[0] (trigger) and KEY[1] (reset) pins to the list of signals being monitored in the Setup tab. Add the bit and error count signals.

- right-click on the trigger condition for KEY[0] (trigger) and select a falling edge

- recompile and reprogram the FPGA

- click on "Autorun Analysis" to start the logic analyzer

- the logic analyzer should display the error and bit counts in the Data tab each time you press on the trigger button

- you can change the display format to unsigned decimal to make it easier to calculate the error rates.

- check the bit error rate: if the input and output are not connected it should be 50% (why?); if connected it should be 100% because of the inverter on the data input

Data Analysis

The oscilloscope and DMM can be used to measure the average data voltage (which should correspond to the threshold voltage) and the noise RMS voltage at the comparator inputs. RMS voltages (standard deviations) must be measured using AC coupling and average voltages (threshold voltage) using DC coupling. From these the BER can be predicted. The logic analyzer can be used to read the bit and error counter values and their ratio is the measured BER. The predicted and measured BER values can then be plotted against each other for a range of SNR values. When the BER in log units is plotted against the SNR in dB the result is the familiar "waterfall" curve showing a rapid reduction in BER with increasing SNR.

Procedure

Use Quartus II to create the design shown in Figure 1. Check that you can program the FPGA and that pushing the trigger and reset buttons give you the expected results in the logic analyzer display.

Build the comparator circuit and connect it to the FGPA board using the supplied ribbon cable. Connect the comparator to the AWG. Check that the comparator output responds as expected as you change the threshold voltage. Check that the error count as displayed by the logic analyzer responds as expected.

Generate a `.RAF` file for the AWG using the Matlab code supplied on the course web site (run the command `berlabnoisegen` to generate `awgn.raf`). Set the AWG for Arb operation and load the `.RAF` file. Select SRate (sample rate) mode and set the sample rate to 100 kHz. Set the offset to the average data voltage (threshold).

Use the 'scope and DMM to measure the average and RMS voltages at the two comparator pins and use the logic analyzer to measure the BER.

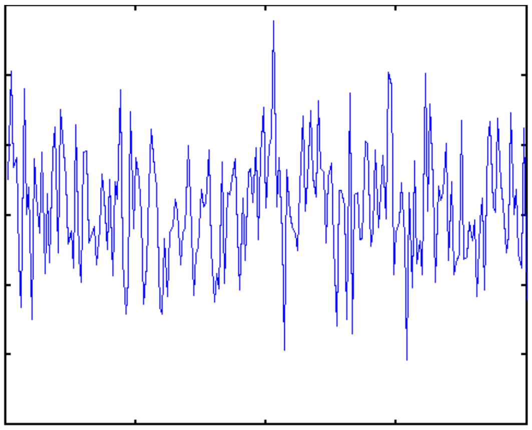
Record the threshold voltage, noise RMS voltage, error and bit counts and predicted and measured BER in a spreadsheet. In your spreadsheet you can use the erfc() function to compute the predicted error rate:

$$P(x < t) = \frac{1}{2}\left(1 + \text{erf}(\frac{t}{\sqrt{2}})\right)$$

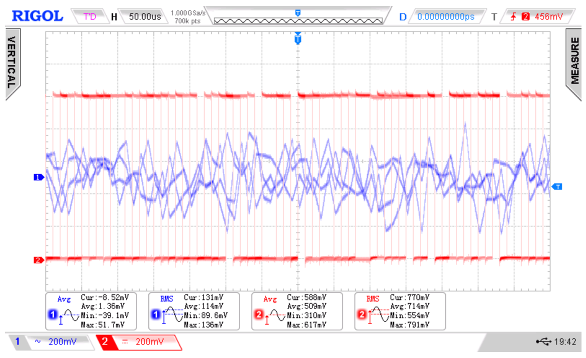
Lab Report

Submit the usual identification information and a report in PDF format containing the following:

- your FPGA block diagram (schematic)
- a plot of the first 200 samples of your noise waveform:

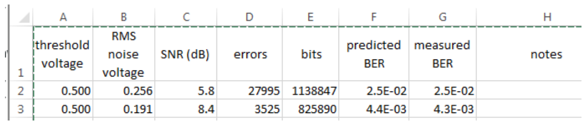


- a screen capture showing the signals on the two comparator inputs (note the measurements are made with AC coupling to measure the noise standard deviation and DC coupling to measure the signal mean):



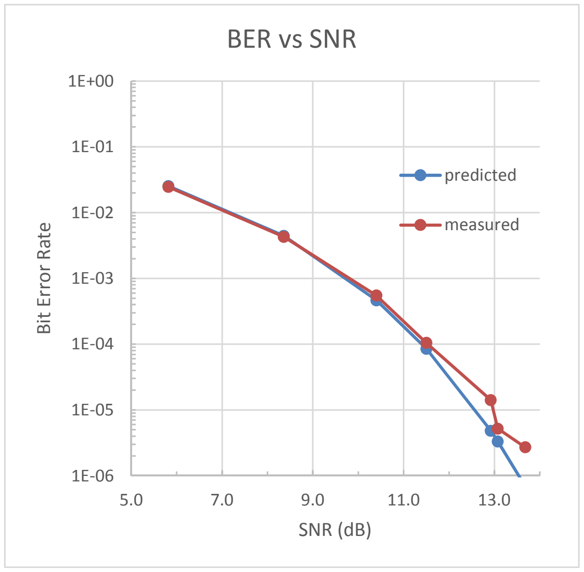
Avg (1) Cur:-8.52mV Avg:1.36mV Min:-39.1mV Max:51.7mV
RMS (1) Cur:131mV Avg:114mV Min:89.6mV Max:136mV
Avg (2) Cur:588mV Avg:509mV Min:310mV Max:617mV
RMS (2) Cur:770mV Avg:714mV Min:554mV Max:791mV

- a table from your spreadsheet showing the calculation of the predicted and measured BER. The BER values should range from about 10^{-1} to about 10^{-6}. For example:



	A	B	C	D	E	F	G	H
1	threshold voltage	RMS noise voltage	SNR (dB)	errors	bits	predicted BER	measured BER	notes
2	0.500	0.256	5.8	27995	1138847	2.5E-02	2.5E-02	
3	0.500	0.191	8.4	3525	825890	4.4E-03	4.3E-03	

- a plot showing both predicted and measured BER formatted as follows:


BER vs SNR
1E+00
1E-01
1E-02
1E-03
1E-04
1E-05
1E-06
Bit Error Rate
SNR (dB)
predicted
measured