Xilinx XC4005XL 1-Wire Pseudo-Random Number Generator

General Description

The 1-Wire pseudo-random number generator allows a microcontroller or microprocessor system to generate random numbers using the Dallas 1-Wire multi-drop bus. The device produces a random number using a linear feedback shift register that is clocked such that a number would be difficult to predict based on past readings. It contains a 64 bit unique serial number and implements all ROM codes of the DS1820 temperature sensor, including "SEARCH ROM."

The random number generator uses a flexible architecture that allows for easy addition of new commands or increasing the linear feedback shift register size. It can be implemented using only 55 XC4005XL CLBs, thus giving the possibility of synthesizing multiple devices onto one FPGA.

Applications

Dallas 1-Wire Testbed

Embedded Games

Simple Cryptography Demonstration

Features

- **64 bit unique serial number**
- **Implements all ROM codes of DS1820 temperature sensor**
- **Returns 64 random bits using arbitrarily sized linear feedback shift register**
- **Allows multiple devices to operate on 1-wire bus**
- **Easily expandable/scalable**

Contact Information

University of Missouri-Rolla
Department of Electrical and Computer Engr.
1870 Miner Circle
Rolla, MO 65409-1060

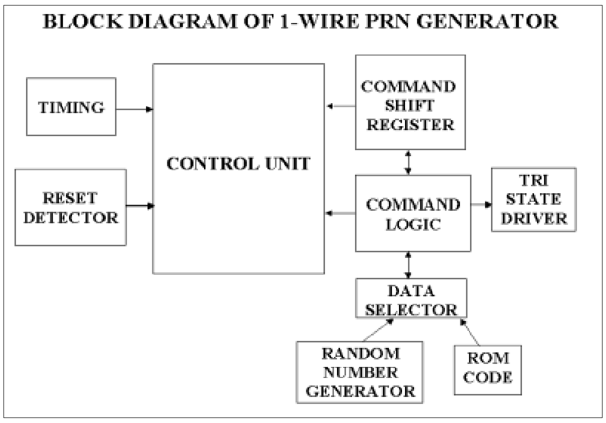
http://www.ece.umr.edu/courses/ee214

Acknowledgements

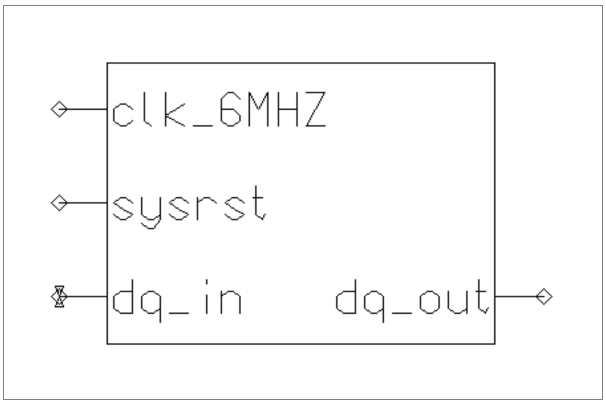
Most of the timing diagrams shown here were taken from the book of iButton standards which can be found at:

http://www.ibutton.com/ibuttons/standard.pdf

Functional Diagram



Pin Configurations



Rev 1: 1/10/2002-jjp

Xilinx XC4005XL 1-Wire Pseudo-Random Number Generator

Pin Descriptions

PIN Name	FUNCTION
dq_in	1-Wire input pin, should be connected so as to always copy the external dq line.
dq_out	1-Wire output pin, should be connected so that whenever this output is high, the external dq line is pulled low.
sysrst	Active high reset. During simulation this should be pulsed at the beginning of simulation, when actually in hardware it can be tied low with no problems.
clk_6MHz	6MHz externally applied clock. All 1-Wire timing is derived from this clock.

Detailed Description

Device Operation

The Xilinx XC4005XL Pseudo-Random Number Generator uses a 6MHz clock to drive a linear feedback shift register to produce a sequence of pseudo-random numbers. Numbers in the sequence can be read using the Dallas 1-Wire protocol over the dq pin.

Microprocessor Interface

1-Wire Protocol

The device is controlled via the Dallas 1-Wire interface. A detailed description of the protocol is given at:

http://www.ibutton.com/ibuttons/standard.pdf

A brief description of the waveforms necessary to control the device will be given here.

Bus Configuration

All devices on the 1-Wire bus are connected by a single wire, dq. They are configured in a wired AND network with an external pull up resistor. Any device may pull the bus low, or multiple devices may do so. One device on the bus is denoted the master and all other devices are slaves. All communication on the bus is initiated by the master.

Reset and Presence Pulse

Before any sequence of commands may begin the master must send a reset pulse to all connected devices. To do this the master holds the bus low for a minimum of 480us and a maximum of 960us. This will reset all connected devices into a known state and cause them to send a presence pulse.

This presence pulse is sent after waiting approximately 30us after the rising edge of the reset pulse. All connected devices then pull the dq line low for a nominal 120us.

Writing

When writing a bit onto the bus the master pulls the bus low a minimum of 1us to signal the start of a bit. To write a '0' it must hold the bus low for at least 60us, while to write a '1' it would let the bus go high immediately after the 1us was over. A nominal device will sample the bus 30us after the falling edge, so there is a wide timing tolerance.

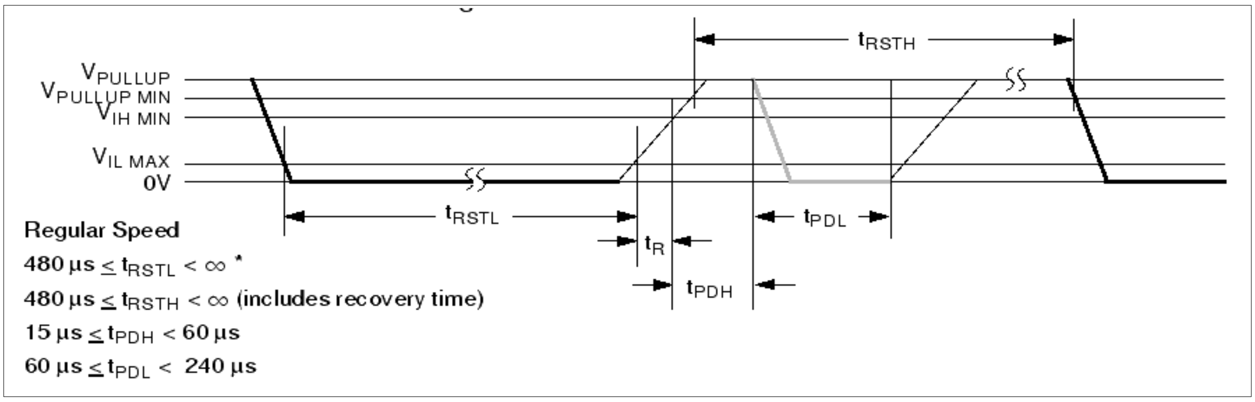
Reading

Reading from the bus is similar to writing. The master initiates a read cycle by pulling the bus low for a minimum of 1us, then releases it. The master should then sample the bus approximately 30us later. A slave that is writing a '0' will hold the bus low while a slave that is writing a '1' will let the bus go high.

Timing Diagrams

Timing diagrams for the reset pulse, presence pulse, write slot, and read slot can be found in Figures 1-4.

Xilinx XC4005XL 1-Wire Pseudo-Random Number Generator



t_{RSTH}
V_{PULLUP}
$V_{PULLUP\ MIN}$
$V_{IH\ MIN}$
$V_{IL\ MAX}$
0V
t_{RSTL}
t_R
t_{PDH}
t_{PDL}
Regular Speed
$480\ \mu s \le t_{RSTL} < \infty$ *
$480\ \mu s \le t_{RSTH} < \infty$ (includes recovery time)
$15\ \mu s \le t_{PDH} < 60\ \mu s$
$60\ \mu s \le t_{PDL} < 240\ \mu s$

Figure 1: Reset Timing

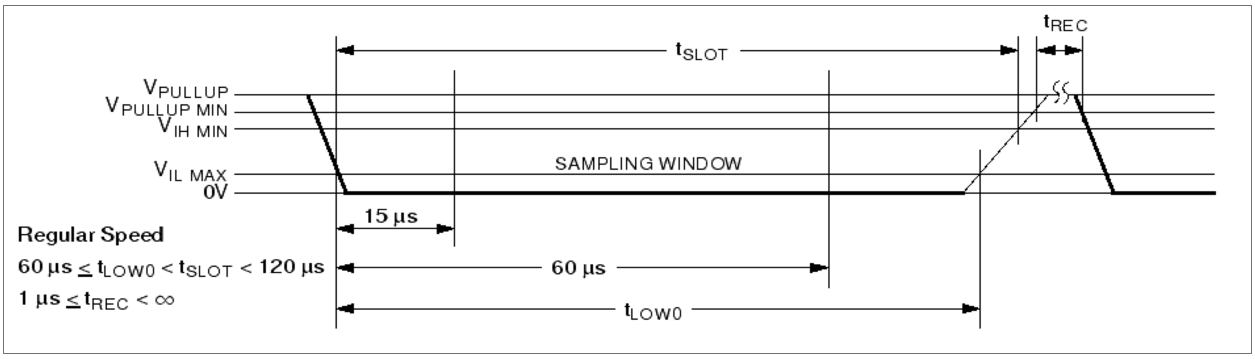


Figure 2: Write '0' Timing

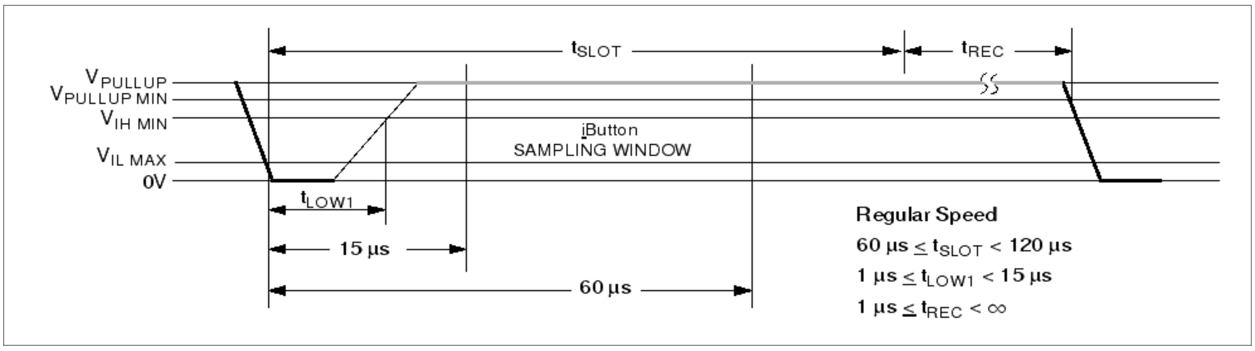


Figure 3: Write '1' Timing

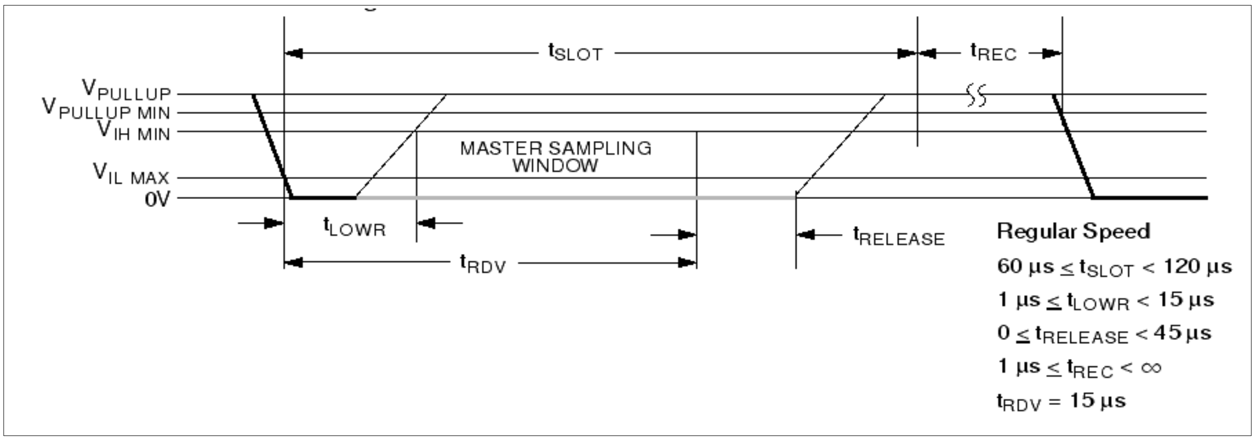


Figure 4: Read Timing

Command Sequence

After sending a reset pulse and receiving a presence pulse the devices will expect to receive an 8 bit command. The master should send this command by writing the 8 bits LSB first. All data sent over the 1-Wire bus is sent LSB first. A list of possible commands is shown below:

Name	Val	Description
READ ROM	0x33	Read the 64 bit unique device identifier.
READ RAND	0x43	Read a 64 bit random number.
SKIP ROM	0xCC	When used with a single device bus, selects the only device.
MATCH ROM	0x55	Selects a device for future commands based on it's device identifier.
SEARCH ROM	0xF0	Enumerate the device identifiers of all devices on the bus.

Depending upon the command sent, the master should proceed to either send another command, read/write a 64 bit value from the device, or use the SEARCH ROM device enumeration algorithm.

Command Descriptions

READ ROM 0x33

After sending this command the master should read 64 bits from the device. The writing slave will shift out it's device identifier one bit at a time starting with the LSB. The device will be prepared to receive another command after the completion of a READ ROM.

READ RAND 0x43

After sending this command the master should read 64 bits from the device. The writing slave will shift out a random number starting with the LSB. After this command the random number generator will be deselected and will not respond to any bus activity until another reset pulse is sent.

SKIP ROM 0xCC

This command is intended to be used to select a device if it is the only one on a bus. Other families of 1-Wire devices need to be selected before their functionalities are available. This function is redundant and is only included with the random number generator for compatibility with legacy drivers.

MATCH ROM 0x55

After sending this command the master should write a 64 bit device ID LSB first. If a slave's ID matches this one, it will be selected and available for further commands. All non matching devices will become deselected and ignore all bus activity until a reset pulse.

SEARCH ROM 0xF0

After sending this command the master should proceed with the following algorithm for determining the device IDs of all connected devices.

For each bit of the 64 device ID bits the master should:

1. Read one bit, during which all devices will write out that ROM bit.
2. Read another bit, during which all devices will write out the inverse of that ROM bit.
3. If the bus was pulled low during both reads the master now knows that at least two devices exist on the bus with that bit position differing.
4. The master must choose which set of devices to select by writing a '0' or '1'. This must still be done even if only one device responded.

This routine is repeated for every one of the 64 ROM bits. After once through the 64 bits, one device ID has been discovered. The routine would then be repeated with different choices made at critical points until all device IDs have been discovered.

Sample Access

Figure 5 shows a sample command cycle in which the READ ROM command is executed. The reset pulse is presumed to have been held for at least 480us.

Applications Information

dq Interfacing

To accommodate the Xilinx XC4000 series of FPGAs, which have no internal tri-state buffers, a separate dq_in and dq_out are provided. If only a single 1-Wire device is placed on the FPGA then dq_out can be wired to the enable of a tri-state buffer with it's input tied to ground. The tri-state buffer should drive an iopad. dq_in can then be wired directly to the output of this iopad.

If multiple devices are to be placed in the same FPGA a similar scheme can be used, except the dq_out lines of all devices should be OR'ed together before going to the enable of the tri-state buffer. See Figures 6 and 7 for examples.

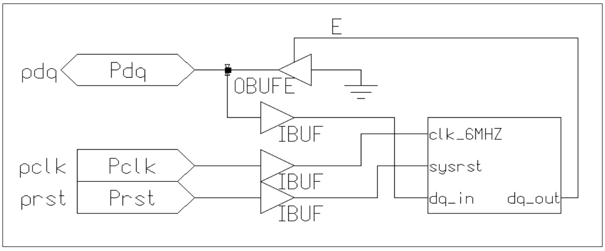


Figure 6: Sample Application 1

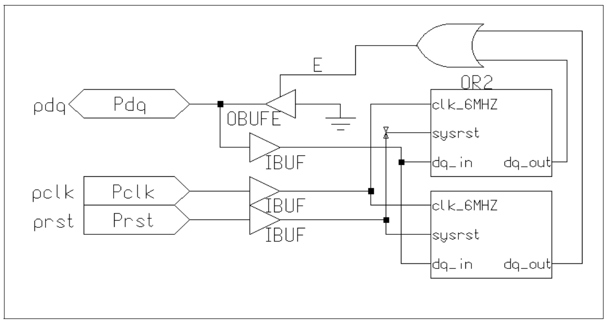


Figure 7: Multiple Device Bus

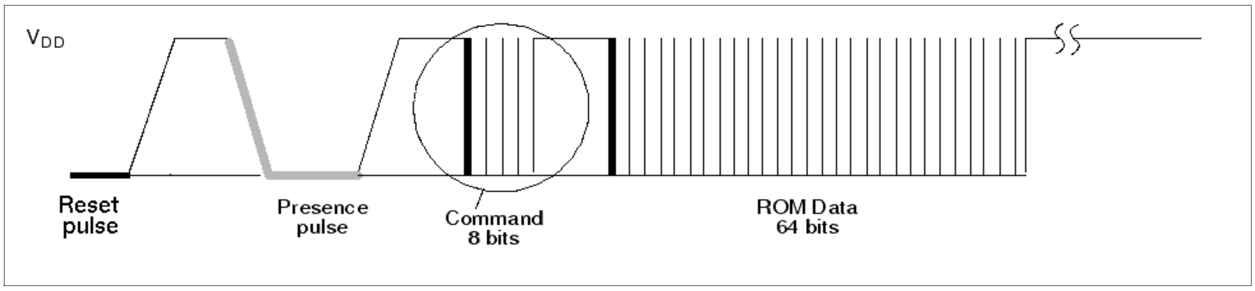


Figure 5: Sample READ ROM timing sequence

Xilinx XC4005XL 1-Wire Pseudo-Random Number Generator

Sample Application

Shown in Listing 1 is sample 8051 code to drive the 1-Wire Pseudo-Random Number Generator as part of the XS40 prototyping board. It assumes that the dq line has been properly wired to pin 4 of Port 1 and that the seven segment display has been mapped into memory location 0xAA55.

Porting Concerns

The Pseudo-Random number generator was designed to run in a Xilinx XC4005XL FPGA. It is implemented completely in VHDL with a Mentor Graphics symbol on top. To move it to another device the VHDL code must be resynthesized for the new architecture. This can be done with the following procedure:

1. Modify `src/syn.scr` to contain the correct architecture
2. Run the `build_owprn` script

If multiple random number generators are desired on a single FPGA, separate EDIF netlists will have to be made for each device and associated with different top level symbols. Then it is possible to change the device ID listed in `src/owprn.vhd`, re-synthesize, and move the resulting EDIF file to an alternate name. The following procedure could be used:

1. Move the original EDIF file, `hdl/owprn.edf` to a new name, for example "`mv hdl/owprn.edf hdl/owprn_orig.edf`"
2. Modify the file `src/owprn.vhd` to contain a different device ID
3. Re-synthesize the design using the `build_owprn` script
4. Instantiate a new top-level one wire symbol, and modify it's `file` property to point to the new EDIF file. In this case `../../hdl/owprn_orig.edf`

When creating a .BIT file for this design, ngdbuild must be passed the `-sd` option with the directory the synthesized netlists. If using the included `xmake` script this is done already.

Xilinx XC4005XL 1-Wire Pseudo-Random Number Generator

Listing 1: Sample 8051 Source

```
#include<reg51.h>
/* ROM COMANDS */
#define READ_ROM            0x33
#define READ_RAND                       0x43
#define MATCH_ROM           0x55
#define SKIP_ROM            0xCC
#define SEARCH_ROM          0xF0

void displayArray(unsigned int length, char *d);
char command(char comm, char second);
void c_write(char out);
char c_read(void);
char reset();
void msec(int Delay);

/* global variables */
sbit IO = P1^3;
sbit RESET = P1^2;

char ROM[8];
char SCRATCHPAD[8];

xdata char seven_seg _at_ 0xAA55;

char seven_seg_table[16]= {
        0x7e, 0x30,     0x6d, 0x79,
        0x33, 0x5b, 0x5f, 0x70,
        0x7f, 0x7b, 0x77, 0x1f,
        0x4e, 0x3d, 0x4f, 0x47
        };

#define DELAY_VAL 500

void main(void){
    char i = 0;

    RESET=1; // make sure the device is not in reset

    while(1){
        /* read ROM and random number, then display */
        i = command( READ_ROM, i);
                displayArray(8,ROM);

                i = command ( READ_RAND, i);
                displayArray(8,SCRATCHPAD);
                msec(100);
    }
}

// wait a specified amount of time
void msec(int Delay)
{
    int i, j;
    for(j=0; j<Delay; j++)
    {
        for (i=0; i<500; i++);
    }
}
```

Listing 1: Sample 8051 Source (continued)

```
// display an array of numbers on the seven segment display
void displayArray(unsigned int length, char *d)
{
        char t,i;

        for (i=0;i<length;i++) {
                // write out high nibble to seven segment display
                seven_seg=0; msec(DELAY_VAL/2);
                t=d[i]>>4; t=t&0x0F;
                seven_seg=seven_seg_table[t]; msec(DELAY_VAL);

                // write low nibble to seven segment display
                seven_seg=0; msec(DELAY_VAL/2);
                t=(d[i]&0x0F);
                seven_seg=seven_seg_table[t]; msec(DELAY_VAL);
        }
}

// process a command on the 1-Wire bus
char command(char comm, char second){
     char retvalue=0;//OK
     char i;

     switch(comm){
          case READ_ROM:
               while(reset());
               c_write(READ_ROM);
               for(i=7; i >= 0; i--) ROM[i] = c_read();
               break;
                  case READ_RAND:
                           c_write(READ_RAND);
                           for (i=7; i >= 0; i--) SCRATCHPAD[i]=c_read();
                           break;
     }
     return retvalue;
}

// write one byte to the 1-Wire bus
void c_write(char out){
     char i,j;

     for (j=0; j<=7;j++){
          IO = 0;             //set low
          if(out &  0x01){      /* write '1' */
               IO=0; i=1; while(i--);  // start write slot
               IO=1; i=12; while(i--); // wait for end of slot
          }
          else {
               I=0; i=12; while(i--); // start write slot and write 0
               IO=1; i=1; while(i--); // wait for end of slot
          }
          out = out >> 1;
     }
}
```

Listing 1: Sample 8051 Source (continued)

```
     // read in one byte from the 1-Wire bus
     char c_read(void){
          char i,j;
          char t = 0;

          for(i=0; i<=7; i++) {
               IO = 0; j=1; while(j--);   // set output 0 for instant
               IO = 1; j=3; while(j--);   // set to input for approx 10us
               t>>=1;          // shift in next bit

               if(IO == 1)  t |= 0x80;
                  else t &= 0x7f;
               j=12; while(j--);  // 60us delay
          }
          return t;
     }

     // send a reset pulse and wait for a presence pulse
     char reset(){
          unsigned char i;
          unsigned char status;

          IO=0; i=255; while(i--);   // hold reset pulse
          IO=1; i=20; while(i--);    // wait for presence

           /* read presence pulse */
          if (IO) status = 1;
             else status = 0;

          i=255; while(i--);         // wait for idle
          return status;
     }
```