EE287 Project design
The data unconfuser engine

The data unconfused engine is designed to handle streaming data which occurs for long durations of time. The goal of the confuser is to make it difficult for others to easily determine the contents of the data. The confuser is a simple single key data processing engine intended to be placed in various communications products where some low level of data protection is desired, but the costs of true public/private key encryption are not justified.

The data unconfuser is based on simple CRC type registers. These are implemented by using shift registers and exclusive or gates. A CRC register may be used as a pseudo random number generator, or as a multiplier of data with the PRN sequence in a ECC syndrome generator. The test bench will generate confused data based on a ~700 bit key, and your data unconfuser engine will decode the confused data providing clean unconfused data on the output interface.

The data unconfuser is a stream oriented single pad system. The confusion/unconfusion blocks work on 4 byte blocks. The test bench will always send the data in groups of 4 bytes. The pads keep moving until reloaded through the configuration interface.

Your design should have a FIFO (Already built as a homework assignment) on the input and output interfaces. The design has a one flag input model, and a two flag output model. (To make the student work harder). Your design is assumed to have two 64k byte FIFOs, and the test bench can put data in the design much faster than you can unconfuse the data.

The block interface is shown below:

Name	# Bits	Direction	Comments
Din	8	IN	Confused data to be unconfused by your design
PushIn	1	IN	Push signal used with Din
Clk	1	IN	System clock (Assumed to be 180Mhz)
Reset	1	IN	System Reset (Initializes state machines and FIFO counters)
Dout	8	OUT	Unconfused data from your design
PushOut	1	OUT	Push flag from your design
StopIn	1	IN	Stop flag used with PushOut (Comes from test bench)
Caddr	7	IN	Configuration register address (See below)
Cdata	8	IN	Data to load in a configuration register
Cpush	1	IN	Push signal matching Caddr and Cdata. (Configuration is not changed while data is being unconfused. You do not need to pipeline the configuration registers)

Table 1 Interface signals

The data unconfuser has a large number of byte oriented configuration registers. Each of these registers is Little Endian (The first byte is the lowest byte of information, bit 0 is the low order bit).

Address	What	Bytes	Bits used
0	PAD0 initial value	21	160
21	PAD1 initial value	21	161
42	PAD2 initial value	21	162
63	PAD3 initial value	21	163
84	Injector initial value	4	32
88	Pad Sel initial value	5	40

Table 2 Table of registers on the initialization interface

The following table shows the polynomials used in the data confuser/unconfuser. The key is simply the initial values of all the CRC registers listed below. See the text below on how to implement a bi-directional polynomial register.

What	Polynomial	Notes
PAD0	$X^{160}+X^{139}+X^{119}+X^{98}+X^{79}+X^{60}+X^{40}+X^{20}+1$	
PAD1	$X^{161}+X^{140}+X^{121}+X^{100}+X^{80}+X^{60}+X^{40}+X^{20}+1$	
PAD2	$X^{162}+X^{141}+X^{121}+X^{100}+X^{80}+X^{60}+X^{40}+X^{20}+1$	
PAD3	$X^{163}+X^{142}+X^{122}+X^{102}+X^{82}+X^{61}+X^{41}+X^{20}+1$	
Injector	$X^{32}+X^{27}+X^{21}+X^{16}+X^{10}+X^{5}+1$	# pad bits used
Pad Sel	$X^{40}+X^{34}+X^{27}+X^{19}+X^{12}+X^{6}+1$	Which pad
CDATA	$X^{32}+X^{31}+X^{29}+X^{28}+X^{26}+X^{25}+X^{24}+X^{22}+X^{21}+X^{13}+X^{11}+X^{9}+X^{8}+X^{5}+1$	Data coding

Table 3 Internal function polynomials (All are primitive over GF(2))

Assuming the configuration registers are loaded, the unconfusion process is described below.

- Load 4 bytes of information into the CDATA register. The data may not be present in the FIFO, and the state machine must wait until the input FIFO is not empty. The first byte is placed in bits 31:24, and the third byte is placed in bits 7:0.
- The pad amount is formed by taking bits from the Injector CRC register.
- { 1'b1, I[30],I[5],I[9],I[2],I[27] } I is the injector register
- Step the Injector CRC forward 39 times
- Step the following forward by the amount determined as the pad amount
    - PAD0
    - PAD1
    - PAD2
    - PAD3
    - Pad Sel
- Step the following backwards pad amount times, undoing the CDATA register to get the original bytes of the message
    - PAD0, PAD1, PAD2, PAD3, Pad Sel, CDATA
    - The data from PAD0,1,2,3 are selected by Pad Sel as shown below in Table 4
- Extract the unconfused data from the CDATA register. The first byte is in bits 31:24, and the third byte is in bits 7:0. The data is placed in the output FIFO.
- Step the following forward by the pad amount to prepare for the next 4 byte group (or save and restore from the last forward stepping)
    - PAD0, PAD1,PAD2,PAD3,PAD4,Pad Sel
- Repeat all steps until reset

The Pad Sel CRC is used to select which bit of which PAD register will be used to unconfuse the data. A 5 bit binary index is created by concatenating:

{ PS[31],PS[3],PS[5],PS[19],PS[8] }

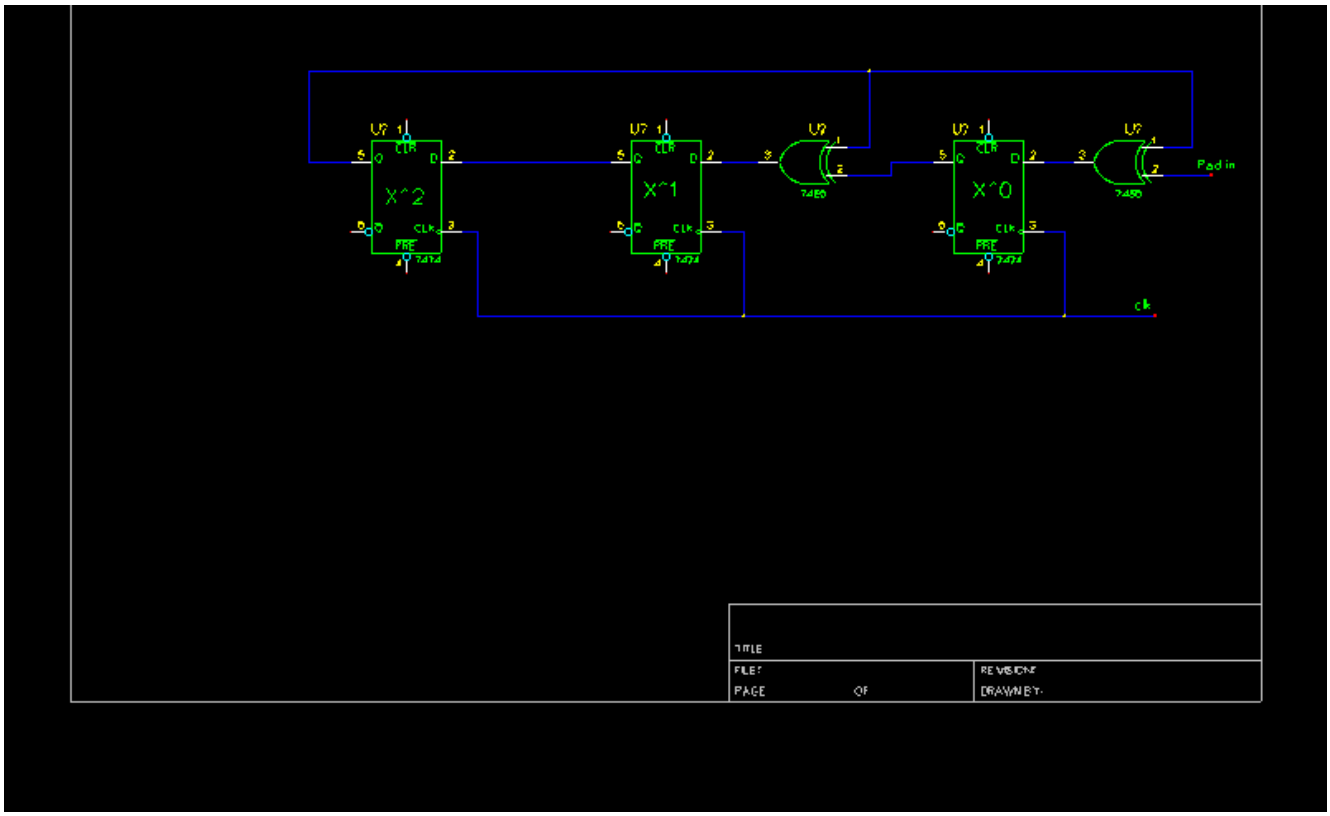
The PAD bits are selected for application to the CDATA register according to the following table:

Index	Selected Pad

0	PAD1[15]
1	PAD0[37]
2	PAD2[73]
3	PAD3[99]
4	PAD0[121]
5	PAD1[130]
6	PAD3[15]
7	PAD2[9]
8	PAD3[97]
9	PAD2[140]
10	PAD1[4]
11	PAD0[88]
12	PAD0[33]
13	PAD1[75]
14	PAD2[35]
15	PAD3[155]
16	PAD2[28]
17	PAD1[150]
18	PAD3[29]
19	PAD0[144]
20	PAD0[127]
21	PAD1[125]
22	PAD2[0]
23	PAD3[5]
24	PAD0[110]
25	PAD3[87]
26	PAD1[19]
27	PAD2[82]
28	PAD0[48]
29	PAD1[47]
30	PAD2[46]
31	PAD3[51]

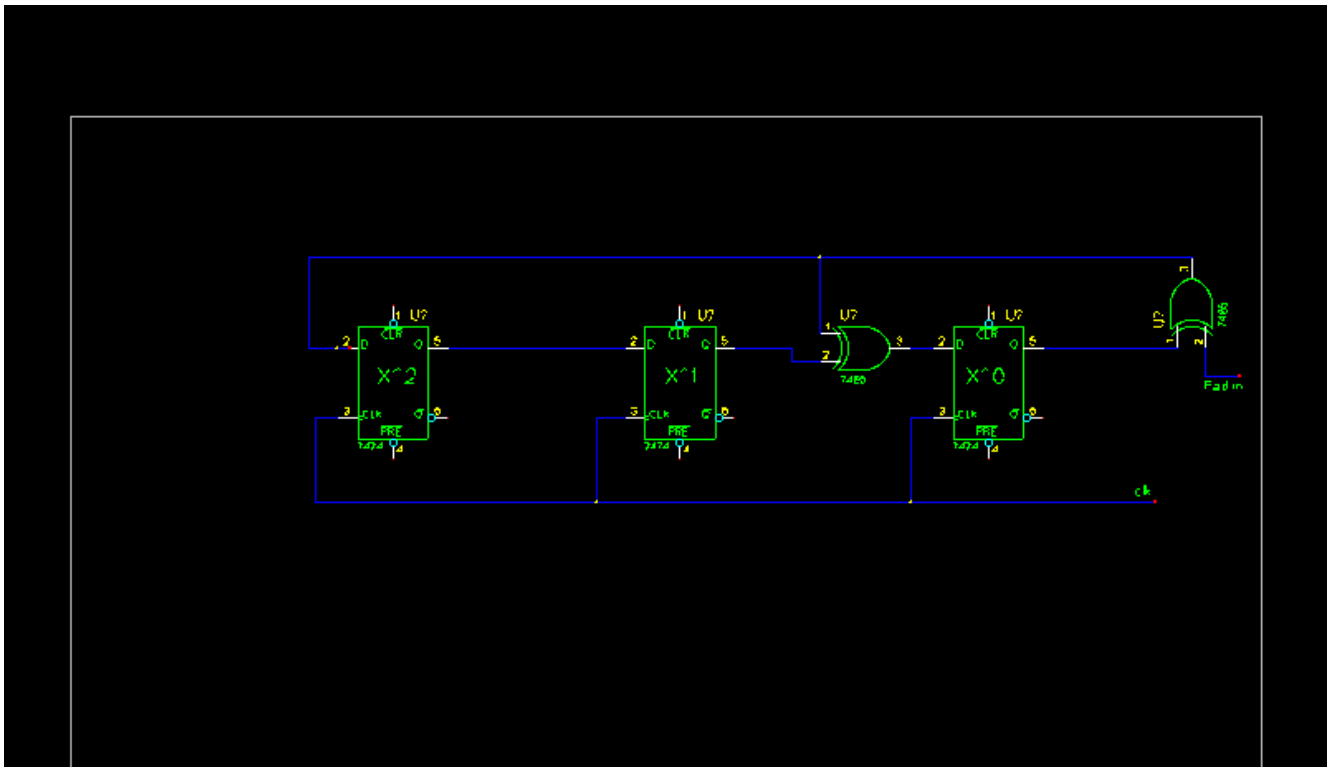
Table 4 PAD Index table to PAD bit mapping

A CRC register is formed of flip flops and exclusive or gates according to the generation polynomials. The generator polynomials are listed above in Table 3. The following examples will use a simple polynomial of X^3+X+1.



The high order bit of the polynomial is not implemented in a flip flop. (It is the feedback term). In each location where there is a 1 in the polynomial (Where the power of X is indicated) an exclusive or gate is placed in the D input.

To step a CRC in reverse, the XOR directions are reversed, and the X^0 bit is xored with the Pad bits to form the bit to the high order bit position.



This illustrated in the following table with some pad examples. Notice, that going backwards after going forwards results in the same data in the X2-X0 flip flops.

X2	X1	X0	Pad	
0	1	1	1	Forward CRC
1	1	1	0	
1	0	1	1	
0	0	0	1	
0	0	1	1	
0	1	1	0	
1	1	0	1	
1	1	0	0	
1	1	1	1	
1	0	0		
1	0	0	1	Reverse CRC
1	1	1	0	
1	1	0	1	
1	1	0	0	
0	1	1	1	
0	0	1	1	
0	0	0	1	
1	0	1	0	
1	1	1	1	
0	1	1		

Each CRC register must be able to retain the current value, be loaded with new data, step forward, and step backwards. This may be implemented with a 4:1 mux type selection on the D input of each flip flop. Your state machine and loading logic should control what is happening to each byte. Of each register.

Implement the design. Run and pass the supplied test bench. Synthesize the design for operation at 180 MHz assuming a 5 ns memory is available. Submit the verilog code, simulation output file, and synthesis results.

Last updated 2/23/2009