Inhaltsverzeichnis

1. Protokoll

1.1. Aufgabenstellung

In diesem Versuch wird die Kanalcodierung von Nutzinformationen mittels zyklischer Codes untersucht. Dabei wird jeweils eine Schieberegisterschaltung für die Generierung und die Prüfung eingesetzt, die einerseits eine Polynom-Multiplikation und bei der Prüfung eine Polynom-Division durchführen.

Im zweiten Teil wird das linear Rückgekoppelte Schieberegister von der Prüfung als Pseudo-Zufallsgenerator verwendet und dessen Periodizität gemessen.

Alle Schaltungen werden dabei auf einem CPLD realisiert.

1.2. Einrichtungen und Geräte

Für den Versuch standen folgende Geräte und Einrichtungen zur Verfügung:

- PC mit Xilinx ISE Synthese Tool
- ModSys-System mit:
  - XILINX Coolrunner CPLD XC2C256-PQ208-7
  - 2 Stk. ModSys IOM Board

1.3. Versuchsvorbereitung

1.3.1. Schaltungen zur Generierung und Prüfung von CRC-Werten

Es wurde je eine Schaltung zur Generierung des zyklischen Codes und eine zur Prüfung erstellt. Das verwendete Generator-Polynom lautet: $\mathbf{x^6+x^5+x^3+x^2+1}$.

CRC-Generation:

Die Schaltung (Abb.1) besteht aus einem Rückgekoppelten Schieberegister, das eine Polynom-Multiplikation mit dem Generator-Polynom vornimmt. Der VHDL-Code ist in Anhang 1 zu finden.

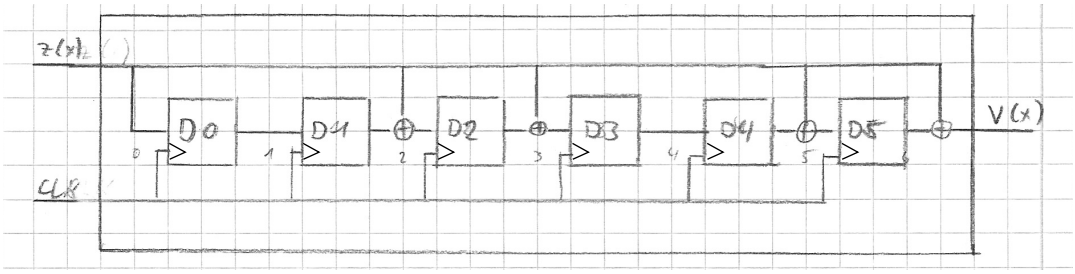

Abb.1: Bockschaltbild Generator

CRC-Prüfung:

Die Schaltung (Abb.2) besteht aus einem linear Rückgekoppelten Schieberegister, das eine Polynom-Division durch das Generator-Polynom vornimmt. Der VHDL-Code ist in Anhang 2 zu finden.

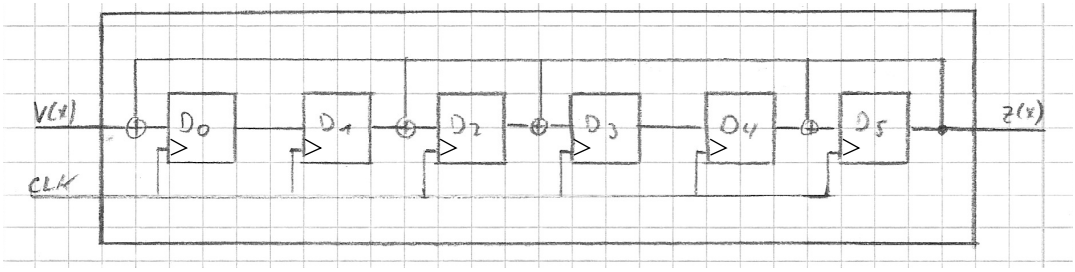

Abb.2: Bockschaltbild Prüfung

Testmuster/ Fehlermuster:

Zur Verifikation der Schaltungen wurden folgende Testmuster verwendet:

a. Aus der Nutzinformation ergibt sich durch Polynom-Multiplikation der CRC-Wert. Wenn die folgenden Nutzinformation seriell (MSB-first, taktsynchron) in den CRC-Generator eingegeben wird, soll der folgende CRC-Wert seriell (MSB-first, taktsynchron) ausgegeben werden.

Nutzinformation:	$x^7+x^5+x^3+x^2+1$	1010.1101
CRC:	$x^{13}+x^{12}+x^{11}+x^8+x^7+x^4+1$	11.1001.1001.0001

b. Die CRC-Prüfung wird getestet, indem das folgende CRC-Polynom seriell (MSB-first, taktsynchron) eingegeben wird. Dann soll die folgende Nutzinformation seriell (MSB-first, taktsynchron) ausgegeben werden und das Schieberegister mit Nullen belegt sein.

CRC:	$x^{13}+x^{12}+x^{11}+x^8+x^7+x^4+1$	11.1001.1001.0001
Nutzinformation:	$x^7+x^5+x^3+x^2+1$	1010.1101

c. Für das Testen einer fehlerhaften Übertragung wurde das folgende CRC-Polynom gewählt. Die Fehler sind fett markiert. Diesmal wird erwartet, dass ein Rest im Schieberegister verbleibt.

CRC:	$x^{13}+x^{12}+x^{10}+x^8+x^7+x^4+x^2+x^1+1$	11.**01**01.1001.0**11**1

1.3.2. Modifikation für ein CRC-16-Polynom

Um ein CRC-16-Polynom zu benutzen, müsste der VHDL-Code um weitere FlipFlops und XOR-Gatter ergänzt werden, sodass der Code dem entsprechenden Blockschaltbild entspricht. Außerdem müssten natürlich die internen Signale, die für die Ein- und Ausgänge der FlipFlops verwendet werden, auf eine Länge von 16 Bit erweitert werden.

Diese Schaltung könnte dann Bündelfehler von maximal 16 Bit erkennen.

1.3.3. Pseudo-Zufallsgenerator

Der Pseudo-Zufallsgenerator besteht erst einmal aus der gleichen Schaltung, wie die Schaltung zur CRC-Prüfung. Diese Mal werden jedoch die Inhalte (Eingänge) der Schieberegister benutzt, indem diese eine parallel gespeicherte binäre Zahl darstellen. Der Ausgang der Schieberegister-Schaltung wird nicht mehr benötigt.

Zu Beginn wird ein Mal eine 1 an den Eingang angelegt, das die Schaltung anregt. Danach wird mit jedem Takt eine neue Zahl quasi zufällig erzeugt, wobei sich die Zahlen nach einer gewissen Periodendauer wiederholen.

Die Periodizität wird mit Hilfe eines binären Zählers ermittelt. Dieser zählt mit jedem Takt eine Zahl weiter hoch und wird beim Auftreten einer bestimmten Zahl im Schieberegister zurückgesetzt. Vor jedem Zurücksetzen wir jedoch der Inhalt des Zählers auf einem Ausgang ausgegeben, sodass immer der maximale Wert des Zählers ausgegeben wird.

Die Schaltung (Abb.3) besteht aus einem linear Rückgekoppelten Schieberegister und einem binären Zähler von 7 Bit Breite. Außerdem gibt es ein Modul, dass das Schieberegister ausließt und bei einem gegebenen Wert den Zähler zurücksetzt, sowie den Zählerstand zu diesem Zeitpunkt speichert und ausgibt. Der VHDL-Code ist in Anhang 3 zu finden.

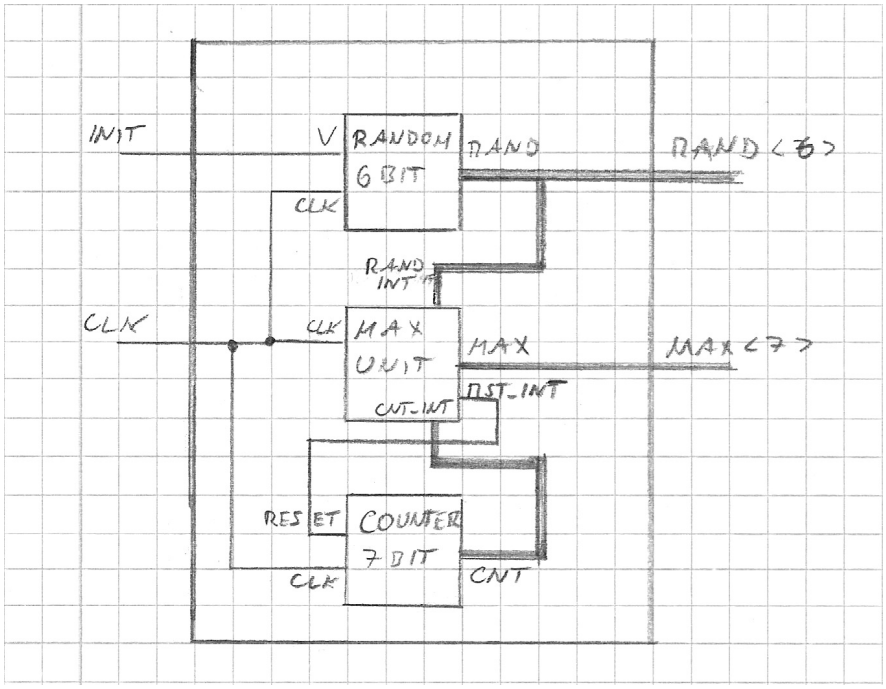

Abb.3: Bockschaltbild Pseudo-Zufall

Die maximale Periodizität beträgt 2^6-1 = 63 (oder auch 0x3F)

1.4. Versuchsdurchführung

1.4.1. Simulation der CRC-Generierung und -Prüfung

Es wurden die Testmuster aus 1.3.1 angewendet.

Test A:

Die Schaltung wurde mir der Nutzinformation über den Eingang Z gespeist, der korrekte CRC-Wert wurde über den Ausgang V ausgegeben, wie in Abb.4 zu sehen ist. Test erfolgreich.

Nutzinformation:	$x^7+x^5+x^3+x^2+1$	1010.1101
CRC:	$x^{13}+x^{12}+x^{11}+x^8+x^7+x^4+1$	11.1001.1001.0001

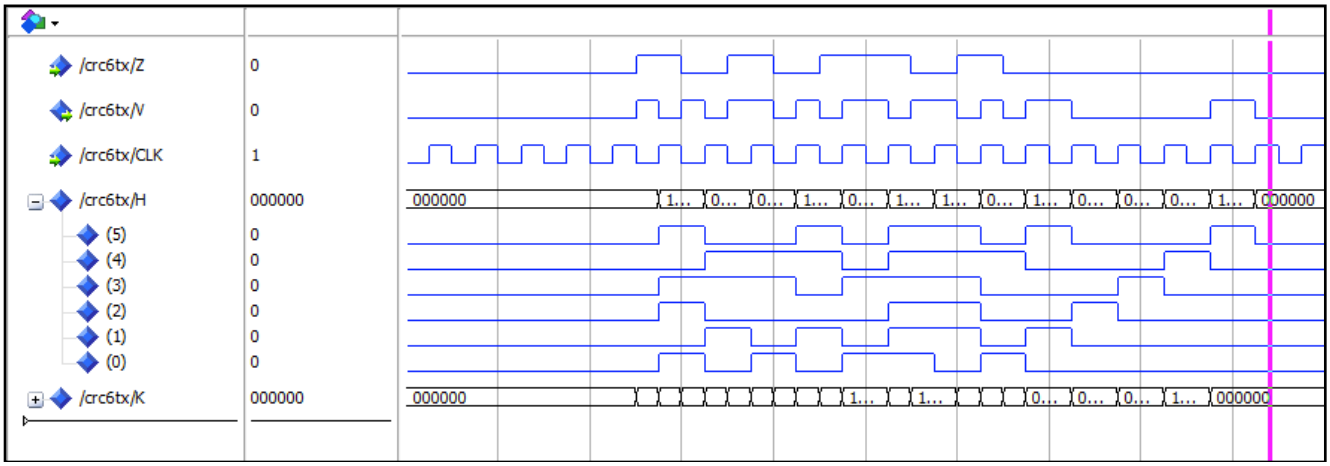

Abb.4: Simulation der CRC-Generierung

Test B:

Die Schaltung wurde mir dem CRC-Wert über den Eingang V gespeist, die korrekte Nutzinformation wurde über den Ausgang ZOUT ausgegeben, wie in Abb.5 zu sehen ist. Test erfolgreich.

CRC:	$x^{13}+x^{12}+x^{11}+x^8+x^7+x^4+1$	11.1001.1001.0001
Nutzinformation:	$x^7+x^5+x^3+x^2+1$	1010.1101

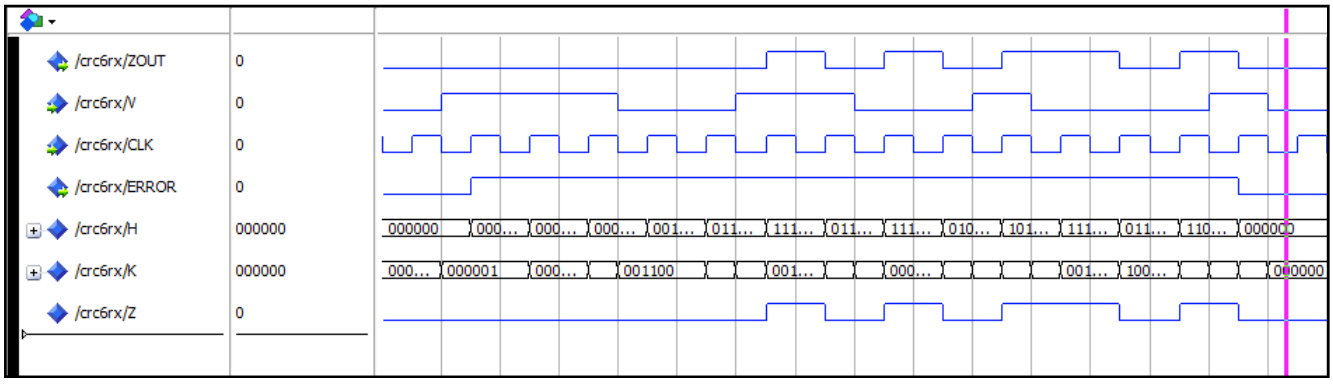

Abb.5: Simulation der CRC-Prüfung mit korrektem CRC-Wert

Test C:

Die Schaltung wurde mir dem CRC-Wert über den Eingang V gespeist, im Schieberegister (Signal H) verbleibt ein Rest, wie in Abb.6 zu sehen ist. Test erfolgreich.

CRC: $x^{13}+x^{12}+x^{10}+x^8+x^7+x^4+x^2+x^1+1$ 11.**01**01.1001.0**11**1

Rest: 01.0010

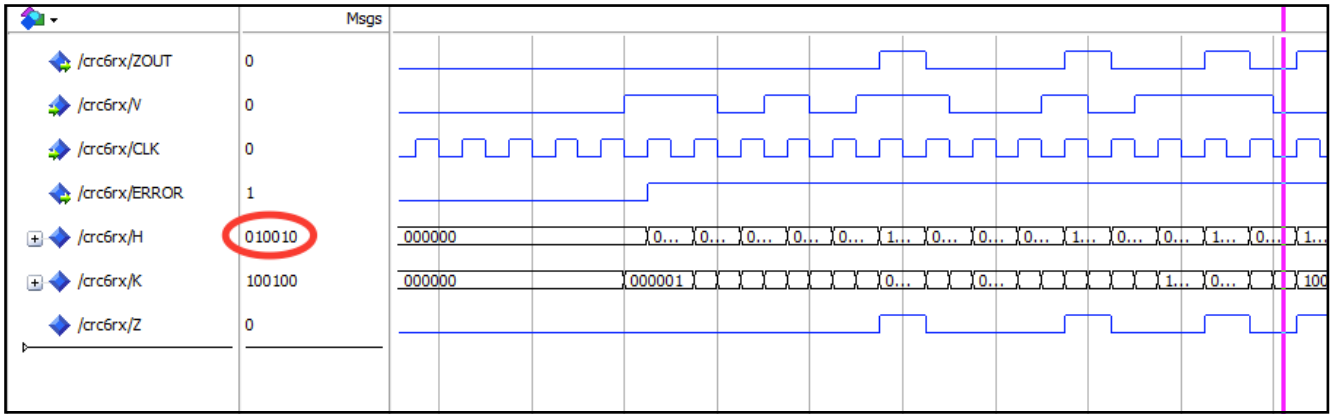

Abb.6: Simulation der CRC-Prüfung mit fehlerhaftem CRC-Wert

1.4.2. Pseudo-Zufallsgenerator

Simulation:

Der Pseudo-Zufallsgenerator wurde mit modelsim simuliert. Der Counter zählt jeden Takt eins hoch und wird vom Reset auf 1 zurückgesetzt.

Die Periodizität beträgt 63 (0x3F), wie in Abb.7 zu sehen ist. Der Pseudo-Zufallsgenerator funktioniert in der Simulation also wie vorgesehen.

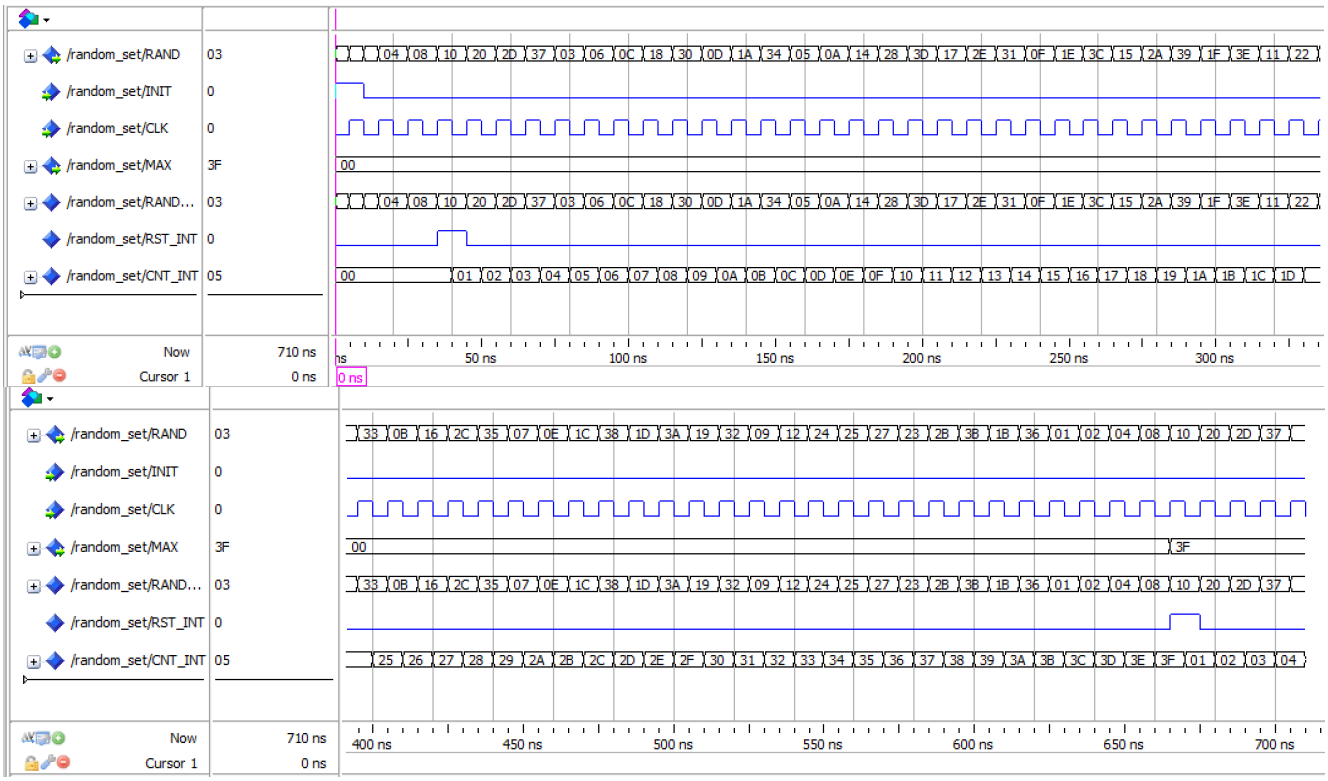

Abb.7: Simulation des Pseudo-Zufallsgenerators

Implementierung auf CPLD:

Der verwendete CPLD wurde auf über das Programm ISE programmiert, indem dieses den VHDL-Code synthetisiert hat und die Logik entsprechend auf den CPLD übertragen hat.
Für die Zuweisung der Anschlüsse des CPLD wurden entsprechende UCF-Dateien (siehe Anlagen) verwendet.

Auf der Versuchsplattform ModSys wurden, nach einem initialen Löschen des CPLD, je ein IOM-Board an die Erweiterungsanschlüsse 3 und 4 angesteckt, um die Zufallswerte, sowie den Maximalwert des Zählers auszugeben. Außerdem wurden die prellfreien Taster dafür genutzt den Eingang der Schaltung zu initialisieren, sowie den Takt einzugeben.

Auch der auf der Hardware implementierte Pseudo-Zufallsgenerator zeigte das gewünschte Verhalten. Die maximale Periodizität wurde auch hier als 63 ausgegeben.

2. Fazit

CRC-Generator und CRC-Prüfung:

Die simulierten Schaltungen für die CRC-Generierung und -Prüfung haben die Tests mit je einer zulässigen und einem unzulässigen Information erfolgreich bestanden.
Der vom Generator aus der Nutzinformation erzeugte Wert wird von der Schaltung zur CRC-Prüfung wieder in die ursprüngliche Nutzinformation übersetzt.
Wenn auf der Übertragungsstrecke Fehler auftreten und somit ein unzulässiges Codewort entsteht, verbleibt im Register ein Wert ungleich 0, was bedeutet, dass ein Fehler bei der Übertagung aufgetreten ist. Dieser Fehler wird erkannt, aber in dieser Schaltung nicht behoben.

Pseudo-Zufallsgenerator

Der Pseudo-Zufallsgenerator hat eine maximale Periodizität von 63, welche auch der theoretisch erreichbaren Periodizität entspricht. Dies bedeutet, dass das verwendete Generatorpolynom irreduzibel ist.

3. Anhang

- Anlage 1: CRC-Generation: VHDL-Code
- Anlage 2: CRC-Prüfung: VHDL-Code
- Anlage 3: Pseudo-Zufallsgenerator: VHDL-Code
- Anlage 4: Pseudo-Zufallsgenerator: UCF-Datei

4. Bildquellen:

- Abb.1: Blockschaltbild Generator
  Jannik Beyerstedt
- Abb.2: Blockschaltbild Prüfung
  Jannik Beyerstedt
- Abb.3: Blockschaltbild Pseudo-Zufall
  Jannik Beyerstedt
- Abb.4: Simulation der CRC-Generierung
  Jannik Beyerstedt
- Abb.5: Simulation der CRC-Prüfung mit korrektem CRC-Wert
  Jannik Beyerstedt
- Abb.6: Simulation der CRC-Prüfung mit fehlerhaftem CRC-Wert
  Jannik Beyerstedt
- Abb.7: Simulation des Pseudo-Zufallsgenerators
  Jannik Beyerstedt

```
------------------------
--     D-FlipFlip     --
------------------------
entity FF is
  port(D:    in bit;
       Q:   out bit;
       CLK:  in bit);
end FF;

architecture behv of FF is
begin
  process(CLK)
  begin
    if CLK = '1' then
       Q <= D;
    end if;
  end process;
end behv;

------------------------
--     CRC encoder    --
------------------------
entity CRC6TX is
  port(Z:    in bit;
       V:   out bit;
       CLK:  in bit);
end CRC6TX;

architecture behv of CRC6TX is

component FF
  port(D:    in bit;
       Q:   out bit;
       CLK:  in bit);
end component;

signal H: bit_vector(5 downto 0);
signal K: bit_vector(5 downto 0);

begin

D0:FF port map (Z,    H(0), CLK);
D1:FF port map (H(0), H(1), CLK);
K(2) <= Z XOR H(1);
D2:FF port map (K(2), H(2), CLK);
K(3) <= Z XOR H(2);
D3:FF port map (K(3), H(3), CLK);
D4:FF port map (H(3), H(4), CLK);
K(5) <= Z XOR H(4);
D5:FF port map (K(5), H(5), CLK);

V <= Z XOR H(5);

end behv;
```

```
1  ------------------------
2  --     D-FlipFlip      --
3  ------------------------
4  entity FF is
5    port(D:    in bit;
6         Q:   out bit;
7         CLK:  in bit);
8  end FF;
9
10 architecture behv of FF is
11 begin
12   process(CLK)
13   begin
14     if CLK = '1' then
15        Q <= D;
16     end if;
17   end process;
18 end behv;
19
20 ------------------------
21 --     CRC decoder     --
22 ------------------------
23 entity CRC6RX is
24   port(ZOUT: out bit;
25        V:    in bit;
26        CLK:  in bit;
27        ERROR: out bit);
28 end CRC6RX;
29
30 architecture behv of CRC6RX is
31
32 component FF
33   port(D:    in bit;
34        Q:   out bit;
35        CLK:  in bit);
36 end component;
37
38 signal H: bit_vector(5 downto 0);
39 signal K: bit_vector(5 downto 0);
40 signal Z: bit;
41
42 begin
43
44 K(0) <= Z XOR V;
45 D0:FF port map (K(0), H(0), CLK);
46 D1:FF port map (H(0), H(1), CLK);
47 K(2) <= Z XOR H(1);
48 D2:FF port map (K(2), H(2), CLK);
49 K(3) <= Z XOR H(2);
50 D3:FF port map (K(3), H(3), CLK);
51 D4:FF port map (H(3), H(4), CLK);
52 K(5) <= Z XOR H(4);
53 D5:FF port map (K(5), H(5), CLK);
54
55 Z <= H(5);
56 ZOUT <= Z;
57 ERROR <= H(0) OR H(1) OR H(2) OR H(3) OR H(4) OR H(5);
58
59 end behv;
```

```
1  ------------------------
2  --     D-FlipFlip      --
3  ------------------------
4  entity FF is
5    port(D:    in bit;
6         Q:   out bit;
7         CLK:  in bit);
8  end FF;
9
10 architecture behv of FF is
11 begin
12   process(CLK)
13   begin
14     if CLK'event and CLK = '1' then
15        Q <= D;
16     end if;
17   end process;
18 end behv;
19
20 ------------------------
21 --     CRC decoder     --
22 ------------------------
23 entity RANDOM_6BIT is
24   port(V:    in  bit;
25        CLK:  in  bit;
26        RAND: out bit_vector(5 downto 0));
27 end RANDOM_6BIT;
28
29 architecture behv of RANDOM_6BIT is
30
31 component FF
32   port(D:    in bit;
33        Q:   out bit;
34        CLK:  in bit);
35 end component;
36
37 signal D: bit_vector(5 downto 0);
38 signal Q: bit_vector(5 downto 0);
39 signal Z: bit;
40
41 begin
42 D(0) <= Z XOR V;
43 D0:FF port map (D(0), Q(0), CLK);
44 d(1) <= Q(0);
45 D1:FF port map (D(1), Q(1), CLK);
46 D(2) <= Z XOR Q(1);
47 D2:FF port map (D(2), Q(2), CLK);
48 D(3) <= Z XOR Q(2);
49 D3:FF port map (D(3), Q(3), CLK);
50 D(4) <= Q(3);
51 D4:FF port map (D(4), Q(4), CLK);
52 D(5) <= Z XOR Q(4);
53 D5:FF port map (D(5), Q(5), CLK);
54 Z <= Q(5);
55
56 RAND <= D;
57 end behv;
58
59 ------------------------
60 --    7-Bit counter    --
61 ------------------------
62
63 library ieee;
64 use ieee.std_logic_1164.all; -- Beschreibung der std_logic Datentypen
65 use ieee.std_logic_unsigned.all; -- Konvertierungsfunktionen std_logic
66
67 entity RANDOM_CT is
68   port(RESET: in  bit;
69        CLK:   in  bit;
```

```
       CNT:  out  bit_vector (6 downto 0));
end RANDOM_CT;

architecture counter of RANDOM_CT is

signal CNT_INT: std_logic_vector (6 downto 0);

begin
  process(CLK, RESET)
  begin
    if (CLK'event and CLK='0') then
      if (RESET = '1') then
        CNT_INT <= "0000001";
      else
        CNT_INT <= CNT_INT + 1;
      end if;
    end if;
  end process;
CNT <= to_bitvector(CNT_INT);
end counter;


-------------------------
--  SET FOR EVERYTHING --
-------------------------

entity RANDOM_SET is
  port(RAND: out bit_vector(5 downto 0);
       INIT: in  bit;
       CLK:  in  bit;
       MAX:  out bit_vector(6 downto 0));
end RANDOM_SET;

architecture behv of RANDOM_SET is

component RANDOM_6BIT is
  port(V:    in  bit;
       CLK:  in  bit;
       RAND: out bit_vector(5 downto 0));
end component;
component RANDOM_CT is
  port(RESET: in  bit;
       CLK:   in  bit;
       CNT:  out  bit_vector (6 downto 0));
end component;

signal RAND_INT: bit_vector(5 downto 0);
signal RST_INT:  bit;
signal CNT_INT:  bit_vector (6 downto 0);

begin
RND0:RANDOM_6BIT port map(INIT, CLK, RAND_INT);
CNT0:RANDOM_CT   port map(RST_INT, CLK, CNT_INT);

process (RAND_INT, CLK)
begin
  if (CLK'event and CLK='1') then
    RST_INT <= '0';
    if (RAND_INT = "001000") then
      RST_INT <= '1';
      MAX <= CNT_INT;
    end if;
  end if;
end process;

RAND <= RAND_INT;

end behv;
```

```
# User Constraint File fuer Modulares System
# Autor: Friedrich, Daniel und Beyerstedt, Jannik

# Datum der Erstellung: Mittwoch, der 14. Januar 2015
# Xilinx Coolrunner
# Baustein XC2C256-PQG208-7C

# Komponenten, die an das Mainboard angeschlossen sind:
# Connector3:  IOM-Board
# Connector4:  IOM-Board

# NET SYSCLK LOC = P55  | IOSTANDARD=LVCMOS33;
# Diesen Pin vorzugsweise fuer Clock verwenden.

# NET NRESET LOC = P206 | IOSTANDARD=LVCMOS33;
# Dieser Pin liegt auf der RESET-Taste des Mainboards, Low-aktiv

#-------------------------------------------------------------#
# Input-Output Messboard (IOM) an Connector 3
#
########################### INPUTS ##############################
#NET  IOM_IN_SW<0>  LOC = P76 | IOSTANDARD=LVCMOS33; # IN_0
#NET  IOM_IN_SW<1>  LOC = P75 | IOSTANDARD=LVCMOS33; # IN_1
#NET  IOM_IN_SW<2>  LOC = P74 | IOSTANDARD=LVCMOS33; # IN_2
#NET  IOM_IN_SW<3>  LOC = P73 | IOSTANDARD=LVCMOS33; # IN_3
#NET  IOM_IN_SW<4>  LOC = P72 | IOSTANDARD=LVCMOS33; # IN_4
#NET  IOM_IN_SW<5>  LOC = P71 | IOSTANDARD=LVCMOS33; # IN_5
#NET  IOM_IN_SW<6>  LOC = P70 | IOSTANDARD=LVCMOS33; # IN_6
#NET  IOM_IN_SW<7>  LOC = P69 | IOSTANDARD=LVCMOS33; # IN_7
#
NET   CLK   LOC = P66 | IOSTANDARD=LVCMOS33; # IN_T0
NET   INIT   LOC = P65 | IOSTANDARD=LVCMOS33; # IN_T1
#
########################### OUTPUTS #############################
NET  MAX<0>    LOC = P80 | IOSTANDARD=LVCMOS33; # OUT_0
NET  MAX<1>    LOC = P82 | IOSTANDARD=LVCMOS33; # OUT_1
NET  MAX<2>    LOC = P83 | IOSTANDARD=LVCMOS33; # OUT_2
NET  MAX<3>    LOC = P84 | IOSTANDARD=LVCMOS33; # OUT_3
NET  MAX<4>    LOC = P85 | IOSTANDARD=LVCMOS33; # OUT_4
NET  MAX<5>    LOC = P86 | IOSTANDARD=LVCMOS33; # OUT_5
NET  MAX<6>    LOC = P87 | IOSTANDARD=LVCMOS33; # OUT_6
#NET  IOM_OUT<7>    LOC = P88 | IOSTANDARD=LVCMOS33; # OUT_7
#
#-------------------------------------------------------------#
# Input-Output Messboard (IOM) an Connector 4
#
########################### INPUTS ##############################
#NET  IOM_IN_SW<0>  LOC = P123 | IOSTANDARD=LVCMOS33; # IN_0
#NET  IOM_IN_SW<1>  LOC = P122 | IOSTANDARD=LVCMOS33; # IN_1
#NET  IOM_IN_SW<2>  LOC = P121 | IOSTANDARD=LVCMOS33; # IN_2
#NET  IOM_IN_SW<3>  LOC = P120 | IOSTANDARD=LVCMOS33; # IN_3
#NET  IOM_IN_SW<4>  LOC = P119 | IOSTANDARD=LVCMOS33; # IN_4
#NET  IOM_IN_SW<5>  LOC = P118 | IOSTANDARD=LVCMOS33; # IN_5
#NET  IOM_IN_SW<6>  LOC = P117 | IOSTANDARD=LVCMOS33; # IN_6
#NET  IOM_IN_SW<7>  LOC = P116 | IOSTANDARD=LVCMOS33; # IN_7
#
#NET  IOM_IN_T<0>   LOC = P114 | IOSTANDARD=LVCMOS33; # IN_T0
#NET  IOM_IN_T<1>   LOC = P113 | IOSTANDARD=LVCMOS33; # IN_T1
#
########################### OUTPUTS #############################
NET  RAND<0>    LOC = P127 | IOSTANDARD=LVCMOS33; # OUT_0
NET  RAND<1>    LOC = P128 | IOSTANDARD=LVCMOS33; # OUT_1
NET  RAND<2>    LOC = P131 | IOSTANDARD=LVCMOS33; # OUT_2
NET  RAND<3>    LOC = P134 | IOSTANDARD=LVCMOS33; # OUT_3
NET  RAND<4>    LOC = P135 | IOSTANDARD=LVCMOS33; # OUT_4
NET  RAND<5>    LOC = P136 | IOSTANDARD=LVCMOS33; # OUT_5
#NET  IOM_OUT<6>    LOC = P137 | IOSTANDARD=LVCMOS33; # OUT_6
#NET  IOM_OUT<7>    LOC = P138 | IOSTANDARD=LVCMOS33; # OUT_7
```