Übung zur Vorlesung Einführung in Computer Microsystems

Prof. Dr. A. Koch
Thorsten Wink

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 11
Übungsblatt 3 - Lösungsvorschlag

Aufgabe 3.1 Logik, Latch, Register

Geben Sie für alle folgenden `reg`-Variablen an, ob sie bei der Synthese in Latches, Flip-Flops oder kombinatorische Logik übersetzt werden. Begründen Sie Ihre Antworten mit den Kriterien für potenzielle Register.

Aufgabe 3.1.1 a)

```
module a (input CLOCK, I, output reg O);
  reg T;
  always @(CLOCK, I)
    if (CLOCK) begin
      T = I;
      O = T;
    end
endmodule
```

`O` wird zu einem Latch, da nicht zeitlich lokal und nicht vollständig. `T` wird in kombinatorische Logik übersetzt, da zeitlich lokal.

Aufgabe 3.1.2 b)

```
module b (
  input wire       A,
  input wire [2:0] I,
  output reg [7:0] OCTAL);

always @(posedge A)
  case (I)
    3'h0: OCTAL = 8'b00000001;
    3'h1: OCTAL = 8'b00000010;
    3'h2: OCTAL = 8'b00000100;
    3'h3: OCTAL = 8'b00001000;
    3'h4: OCTAL = 8'b00010000;
    3'h5: OCTAL = 8'b00100000;
    3'h6: OCTAL = 8'b01000000;
    3'h7: OCTAL = 8'b10000000;
  endcase
endmodule
```

`OCTAL` wird Register/FF da *posedge* in der Aktivierungsliste und `OCTAL` nicht zeitlich lokal. Vollständigkeit ist irrelevant.

Aufgabe 3.1.3 c)

```
module c (
```

```
  input wire [3:0] I,
  output reg [7:0] OCTAL);


always @(I)
  case (I)
    4'h0: OCTAL = 8'b00000001;
    4'h1: OCTAL = 8'b00000010;
    4'h2: OCTAL = 8'b00000100;
    4'h3: OCTAL = 8'b00001000;
    4'h4: OCTAL = 8'b00010000;
    4'h5: OCTAL = 8'b00100000;
    4'h6: OCTAL = 8'b01000000;
    4'h7: OCTAL = 8'b10000000;
  endcase
endmodule
```

OCTAL wird Latch, da nicht zeitlich lokal und unvollständig.

Aufgabe 3.2 BCD nach Binär Konverter

Implementieren Sie den folgenden Pseudo-Code für einen BCD nach Binär Konverter als Verhaltensbeschreibung in Verilog HDL. Die Länge der BCD-Zahl sei 24 Bit. Achten Sie darauf, dass bei der Synthese keine Latches entstehen. Testen Sie die Funktion ihres Moduls mit ISE.

Pseudo-Code für BCD nach Binär Konverter:

a) `bcd` ← `bcd_data_input`

b) `bin` ← 0 (gleiche Bitbreite wie `bcd`)

c) Für `count` ← 1 bis Bitbreite von `bcd` iteriere:

   3.1. `{bcd, bin}` ← `{bcd, bin}` >> 1

   3.2. Für jede 4-Bit Folge (3...0, 7...4, ...) in `bcd` iteriere
   Wenn die 4-Bit Folge größer 7, dann ziehe 3 von dieser Folge ab

d) `bin` enthält die konvertierte Zahl

Zum besseren Verständnis hilft ggf. ein Trace des Algorithmus. Die beiden BCD-Zahlen 98 sollen in eine Binärzahl umgewandelt werden.
Folgende Schritte sind dabei durchzuführen:

Initial	1001 1000	0000 0000
1	0100 1100	0000 0000
-	0100 1001	0000 0000
2	0010 0100	1000 0000
3	0001 0010	0100 0000
4	0000 1001	0010 0000
-	0000 0110	0010 0000
5	0000 0011	0001 0000
6	0000 0001	1000 1000
7	0000 0000	1100 0100
8	0000 0000	0110 0010

1. Möglichkeit: `for`-Schleife über `count` und fester Bitbreite.

```
module bcd_to_bin(
  input [23:0] BCD,
  output [23:0] BIN
);

  reg [2*24-1:0] bcd_concat_bin;
  integer count;

  assign BIN = bcd_concat_bin[23:0];

  always @(BCD) begin
    bcd_concat_bin = {BCD, 24'b0};
    // Schieben und subtrahieren
for (count = 1; count <= 24; count = count + 1) begin
      bcd_concat_bin = bcd_concat_bin >> 1;
      // 4-Bit Folgen
      if(bcd_concat_bin[3+24] == 1) // grösser als 7 (MSB = 1)
        bcd_concat_bin[3+24:0+24] = bcd_concat_bin[3+24:0+24] - 3;
      if(bcd_concat_bin[7+24] == 1) // grösser als 7 (MSB = 1)
        bcd_concat_bin[7+24:4+24] = bcd_concat_bin[7+24:4+24] - 3;
      if(bcd_concat_bin[11+24] == 1) // grösser als 7 (MSB = 1)
        bcd_concat_bin[11+24:8+24] = bcd_concat_bin[11+24:8+24] - 3;
      if(bcd_concat_bin[15+24] == 1) // grösser als 7 (MSB = 1)
        bcd_concat_bin[15+24:12+24] = bcd_concat_bin[15+24:12+24] - 3;
      if(bcd_concat_bin[19+24] == 1) // grösser als 7 (MSB = 1)
        bcd_concat_bin[19+24:16+24] = bcd_concat_bin[19+24:16+24] - 3;
      if(bcd_concat_bin[23+24] == 1) // grösser als 7 (MSB = 1)
        bcd_concat_bin[23+24:20+24] = bcd_concat_bin[23+24:20+24] - 3;
    end
  end
endmodule
```

2. Möglichkeit: Mit definierbarer Bitbreite (`parameter`) und inneren `for`-Schleifen:

```
module bcd_to_bin #(
  parameter LENGTH = 24
  )(
  input  wire [LENGTH-1:0] BCD,
  output wire [LENGTH-1:0] BIN
  );

  reg [2*LENGTH-1:0] bcd_concat_bin;
  reg [3:0] temp;
  integer count, j, k;

  assign BIN = bcd_concat_bin[LENGTH-1:0];

  always @(BCD) begin
    bcd_concat_bin = {BCD, {LENGTH{1'b0}} };
    // Schieben und subtrahieren
    for (count = 1; count <= LENGTH; count = count + 1) begin
      bcd_concat_bin = bcd_concat_bin >> 1;
      // 4-Bit Folgen
      for (j = 0; j < LENGTH/4; j = j + 1) begin
        // Extrahiere 4 Einzelbits
        for (k = 0; k < 4; k = k + 1)
```

```
          temp[k] = bcd_concat_bin[LENGTH + j*4 + k];
        if (temp[3] == 1) // grösser als 7 (MSB = 1)
          temp = temp - 3;
        // Und wieder bitweise zurück
        for (k = 0; k < 4; k = k + 1)
          bcd_concat_bin[LENGTH + j*4 + k] = temp[k];
      end
    end
  end
endmodule
```

Aufgabe 3.3 Verkaufsautomat Multicoin

Der aus der Vorlesung (Foliensatz 4 ab Folie 86) bekannte Eis-Verkaufsautomat soll zusätzlich 10- und 20-Cent Münzen sowie 2-Euro Münzen akzeptieren. Dazu erhält er ein auf drei Bit verbreitertes `COIN`-Signal, welches wie bisher symbolisch dekodiert werden soll. Das Timing der Münzeingabe sowie die übrigen Spezifikationen bleiben gleich, die Eisausgabe darf einen Takt später als bisher erfolgen. Implementieren Sie das Verilog-Modell des erweiterten Mealy-Zustandsautomaten mit zugehöriger Testbench, welche zusätzlich die neuen Münzeingaben testen soll. Die Testausgabe erfolgt wie bisher in der bekannten Tabellenform. Achten Sie darauf, dass bei der Sythese keine Latches enstehen.

Tipp: Bei der nun vorhandenen Zahl von Kombinationsmöglichkeiten des Münzeinwurfs ist es nicht sinnvoll, den eingeworfenen Geldbetrag als symbolische Zustände zu kodieren. Verwenden Sie stattdessen ein internes Summenregister.

Mealy-Verkaufsautomat in der Multicoin-Version:

```
// Verkaufsautomat
`timescale 1 ns / 1 ps

module iglu_multicoin (
  input wire       CLOCK, // Takt
                   RESET, // Reset
  input wire [2:0] COIN,  // eingeworfene Muenze
  output reg       ICE);  // Warenausgabe

  // Zustandscodierung
  parameter INSERT_COINS = 1'b0, // Warten auf Muenzeinwurf
            ICECREAM = 1'b1;     // Eisausgabe

  // Eingabecodierung
  parameter X0   = 3'b000, // kein Einwurf
            X10  = 3'b001, // Zehner
            X20  = 3'b010, // Zwanziger
            X50  = 3'b011, // Fuenfziger
            X100 = 3'b100, // Euro
            X200 = 3'b101; // Zwei Euro

  // Ausgabecodierung
  parameter Yes = 1'b1,    // Warenausgabe
            No = 1'b0;     // keine Ausgabe

  // Preis fuer ein Eis
  parameter Price = 15;    // Warenpreis

  // interne Variablen
  reg       PRESENT,       // jetziger Zustand
            NEXT;          // naechster Zustand

  reg [5:0] SUM;           // eingezahlter Geldbetrag
                           // Einheit 10 Cent
  reg [5:0] NEXT_SUM;      // neue Summe
                           // Einheit 10 Cent

  always @(PRESENT, COIN, SUM)      // <-- flankenfrei!
    case (PRESENT)
      INSERT_COINS: begin           // Muenzeinwurf
```

```
        case (COIN)
          X0:                       // kein Einwurf
            NEXT_SUM = SUM + 0;
          X10:                      // 10 Cent
            NEXT_SUM = SUM + 1;
          X20:                      // 20 Cent
            NEXT_SUM = SUM + 2;
          X50:                      // 50 Cent
            NEXT_SUM = SUM + 5;
          X100:                     // 1 Euro
            NEXT_SUM = SUM + 10;
          X200:                     // 2 Euro
            NEXT_SUM = SUM + 20;
          default:                  // Don't care, Latch vermeiden
            NEXT_SUM = 'bx;
        endcase
        if (SUM >= Price)           // Ueberzahlung wird ignoriert
          {NEXT,ICE} = {ICECREAM, No};
        else
          {NEXT,ICE} = {INSERT_COINS, No};
      end
      ICECREAM: begin               // genug bezahlt
        NEXT_SUM = 0;               // eingezahlter Betrag wieder auf 0
        {NEXT,ICE} =                // Warenausgabe
          {INSERT_COINS, Yes};
      end
    endcase

  // naechsten Zustand mit jedem Takt endgueltig zum jetzigen machen;
  // synchrones Reset
  //
  always @(posedge CLOCK)             // <-- flankengesteuert!
    if (RESET == 1'b1) begin
      PRESENT <= INSERT_COINS;        // Anfangszustand
      SUM <= 0;
    end
    else begin
      PRESENT <= NEXT;
      SUM <= NEXT_SUM;
    end

endmodule // iglu_multicoin
```

Testbench:

```
`timescale 1 ns / 1 ps
module tb_iglu_multicoin;
  reg CLOCK, RESET;
  reg [2:0] COIN;
  wire ICE;

  // Eingabecodierung
  parameter X0   = 3'b000, // kein Einwurf
            X10  = 3'b001, // Zehner
            X20  = 3'b010, // Zwanziger
            X50  = 3'b011, // Fuenfziger
            X100 = 3'b100, // Euro
            X200 = 3'b101; // Zwei Euro

  // Instanz des Verkaufsautomaten
  iglu_multicoin UUT (.CLOCK(CLOCK), .RESET(RESET),
                      .COIN(COIN), .ICE(ICE));

  // Takt
  always
    #20 CLOCK = ~CLOCK;
```

```
  // Ergebnis drucken
  initial begin
    $display ("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Zeit␣Reset␣Eisausgabe\n");
    $monitor ("%d␣␣␣%d␣␣␣␣␣%d", $time, RESET, ICE);
  end

  // Stimuli anlegen: Muenzen eingeben
  initial begin
    CLOCK = 0; COIN = X0; RESET = 1'b1;
    #50 RESET = 1'b0;
    @(negedge CLOCK);
    // drei Fuenfziger
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    // einen Fuenfziger, dann einen Euro
    #160 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    // zwei Euro (keine Rueckgabe!)
    #160 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    #80 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    // einen Euro, dann einen Fuenfziger
    #160 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    // zwei Euro
    #160 $display ("Einwurf␣200␣Ct"); COIN = X200; #40 COIN = X0;
    // 6 Zehner, 2 Zwanziger, ein Fuenziger
    #160 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣20␣Ct"); COIN = X20; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣20␣Ct"); COIN = X20; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $stop;
  end

endmodule
```

Testausgabe:

```
                Zeit Reset Eisausgabe
                   0   1     x
                  20   1     0
                  50   0     0
Einwurf 50 Ct
Einwurf 50 Ct
Einwurf 50 Ct
                 460   0     1
                 500   0     0
Einwurf 50 Ct
Einwurf 100 Ct
                 780   0     1
                 820   0     0
Einwurf 100 Ct
Einwurf 100 Ct
                1100   0     1
                1140   0     0
Einwurf 100 Ct
Einwurf 50 Ct
                1420   0     1
```

```
                1460   0     0
Einwurf 200 Ct
                1620   0     1
                1660   0     0
Einwurf 10 Ct
Einwurf 10 Ct
Einwurf 20 Ct
Einwurf 10 Ct
Einwurf 20 Ct
Einwurf 10 Ct
Einwurf 10 Ct
Einwurf 50 Ct
Einwurf 10 Ct
                2780   0     1
                2820   0     0
```

Aufgabe 3.4 Wechselgeld

Aufgrund der Unzufriedenheit vieler Kunden mit der fehlenden Wechselgeldausgabe ist der Umsatz des Eis-Verkaufsautomaten aus Aufgabe 1 eingebrochen. Zu allem Überfluss verkauft ein Eisstand gegenüber das Konkurrenzprodukt für 1,40 €. Als Chefdesigner des Eisautomaten erhalten Sie den Auftrag, folgende Maßnahmen aufbauend auf dem „Multicoin"-Automaten umzusetzen:

a) Wechselgeldausgabe

Ein zusätzliches Ausgangsignal `OUTCOIN` zeigt in derselben symbolischen Kodierung wie `COIN` an, dass eine Münze des entsprechenden Wertes an den Kunden ausgegeben werden soll. Das Timing dieses Signals soll identisch wie bei `COIN` sein, um die Münzausgabeeinheit korrekt anzusteuern. Achten Sie auf die Pausen von einem Takt zwischen den eigentlichen Münzwerten (`X10`, `X20`, ...), während denen `OUTCOIN` den Wert `X0` annehmen soll. Gehen Sie davon aus, dass immer eine unbegrenzte Menge an Münzen jeden Wertes als Wechselgeld vorhanden ist. Tipp: Wegen der 1-2-5 Münzstückelung bietet sich ein Greedy-Algorithmus für die Bestimmung der Wechselmünzen an.

b) Abbruch des Kaufs durch den Benutzer

Eine „1" auf dem zusätzlichen Eingang `CANCEL` zeigt an, dass der Kunde nun doch kein Eis will und sein eventuell schon eingeworfenes Geld wiederhaben möchte. Diese Funktion ist nur solange möglich, bis mindestens der Kaufpreis für ein Eis eingeworfen wurde. In diesem Fall wird das Eis sofort ausgegeben und das Geld abzüglich des Wechselgeldes einbehalten, das Wechselgeld anschließend ausgegeben. Nutzen Sie die Wechselgeldrückgabe aus a) für die Implementierung der Abbruchfunktion.

c) Preissenkung des Eises auf 1,30 €

Kodieren Sie hierzu den Preis für ein Eis als Parameter, um ihn später leichter anpassen zu können.

Implementieren Sie das Verilog-Modell des Mealy-Zustandsautomaten aufbauend auf Aufgabe 3. Die Mealy-Eigenschaft muss trotz der erweiterten Funktionalität erhalten bleiben. Erweitern Sie die Testbench um den Test der Wechselgeld- und Abbruchfunktionen. Die tabellarische Testausgabe soll zusätzlich das Signal `OUTCOIN` enthalten. Passen Sie die Wartezeiten nach einer Münzeinwurfsfolge den bedingt durch die Geldrückgabe verlängerten Reaktionszeiten des Automaten an. Achten Sie auch bei dieser Aufgabe darauf, dass bei der Synthese keine Latches entstehen.

Mealy-Verkaufsautomat „Multicoin-Change" mit Wechselgeld- und Abbruchfunktion:

```
// Verkaufsautomat mit Wechselgeldrueckgabe und Abbruchfunktion
`timescale 1 ns / 1 ps
module iglu_multicoin_change (
  input wire       CLOCK,    // Takt
                   RESET,    // Reset
                   CANCEL,   // Abbruch mit Geldrueckgabe
  input wire [2:0] COIN,     // eingeworfene Muenze
  output reg       ICE,      // Warenausgabe
  output reg [2:0] OUTCOIN); // herauszugebene Muenze
```

```
// Zustandscodierung
parameter INSERT_COINS = 2'b00, // Warten auf Muenzeinwurf
          ICECREAM = 2'b01,     // Eisausgabe
          CHANGE = 2'b10,       // Wechselmuenze ausgeben
          WAIT = 2'b11;         // Warten zwischen Muenzausgaben

// Eingabecodierung
parameter X0   = 3'b000, // kein Einwurf
          X10  = 3'b001, // Zehner
          X20  = 3'b010, // Zwanziger
          X50  = 3'b011, // Fuenfziger
          X100 = 3'b100, // Euro
          X200 = 3'b101; // Zwei Euro

// Ausgabecodierung
parameter Yes = 1'b1,   // Warenausgabe
          No = 1'b0;    // keine Ausgabe

// Preis fuer ein Eis
parameter Price = 13;   // Warenpreis

// interne Variablen
reg [1:0] PRESENT,      // jetziger Zustand
          NEXT;         // naechster Zustand

reg [5:0] SUM;          // eingezahlter Geldbetrag
                        // Einheit 10 Cent
reg [5:0] NEXT_SUM;     // neue Summe
                        // Einheit 10 Cent

always @(PRESENT, COIN, SUM, CANCEL)// <-- flankenfrei!
  case (PRESENT)
    INSERT_COINS: begin             // Muenzeinwurf
      case (COIN)
        X0:                         // kein Einwurf
          NEXT_SUM = SUM + 0;
        X10:                        // 10 Cent
          NEXT_SUM = SUM + 1;
        X20:                        // 20 Cent
          NEXT_SUM = SUM + 2;
        X50:                        // 50 Cent
          NEXT_SUM = SUM + 5;
        X100:                       // 1 Euro
          NEXT_SUM = SUM + 10;
        X200:                       // 2 Euro
          NEXT_SUM = SUM + 20;
        default:                    // Don't care, Latch vermeiden
          NEXT_SUM = 'bx;
      endcase
      if (SUM >= Price)             // Eis ausgeben mit Wechselgeld
        {NEXT,ICE,OUTCOIN} = {ICECREAM,No,X0};
      else if (CANCEL)              // Abbruch durch Kunden, Geldrückgabe
        {NEXT,ICE,OUTCOIN} = {CHANGE,No,X0};
      else
        {NEXT,ICE,OUTCOIN} = {INSERT_COINS,No,X0};
    end
    ICECREAM: begin                 // genug bezahlt
      NEXT_SUM = SUM - Price;       // Restgeldbetrag
      {NEXT,ICE,OUTCOIN} =          // Warenausgabe, Wechselgeld
        {CHANGE,Yes,X0};
    end
    CHANGE: begin                   // Wechselgeld ausgeben, Greedy
      if (SUM >= 10) begin          // ein Euro ausgeben
        NEXT_SUM = SUM - 10;
        {NEXT,ICE,OUTCOIN} = {WAIT,No,X100};
```

```
        end
        else if (SUM >= 5) begin       // 50 Cent
          NEXT_SUM = SUM - 5;
          {NEXT,ICE,OUTCOIN} = {WAIT,No,X50};
        end
        else if (SUM >= 2) begin       // 20 Cent
          NEXT_SUM = SUM - 2;
          {NEXT,ICE,OUTCOIN} = {WAIT,No,X20};
        end
        else if (SUM == 1) begin       // 10 Cent
          NEXT_SUM = SUM - 1;
          {NEXT,ICE,OUTCOIN} = {WAIT,No,X10};
        end
        else begin                     // Alles Wechselgeld ausgegeben
          NEXT_SUM = SUM;
          {NEXT,ICE,OUTCOIN} = {INSERT_COINS,No,X0};
        end
      end
      WAIT: begin                      // Pause zwischen Muenzausgaben
        NEXT_SUM = SUM;
        {NEXT,ICE,OUTCOIN} = {CHANGE,No,X0};
      end
    endcase

  // naechsten Zustand mit jedem Takt endgueltig zum jetzigen machen;
  // synchrones Reset
  //
  always @(posedge CLOCK)              // <-- flankengesteuert!
    if (RESET == 1'b1) begin
      PRESENT <= INSERT_COINS;         // Anfangszustand
      SUM <= 0;
    end
    else begin
      PRESENT <= NEXT;
      SUM <= NEXT_SUM;
    end

endmodule // iglu_multicoin_change
```

Testbench:

```
`timescale 1 ns / 1 ps
module tb_iglu_multicoin_change;
  reg CLOCK, RESET, CANCEL;
  reg [2:0] COIN;
  wire ICE;
  wire [2:0] OUTCOIN;

  // Eingabecodierung
  parameter X0   = 3'b000, // kein Einwurf
            X10  = 3'b001, // Zehner
            X20  = 3'b010, // Zwanziger
            X50  = 3'b011, // Fuenfziger
            X100 = 3'b100, // Euro
            X200 = 3'b101; // Zwei Euro

  // Instanz des Verkaufsautomaten
  iglu_multicoin_change UUT
    (.CLOCK(CLOCK), .RESET(RESET), .CANCEL(CANCEL),
     .COIN(COIN), .ICE(ICE), .OUTCOIN(OUTCOIN));

  // Takt
  always
    #20 CLOCK = ~CLOCK;

  // Ergebnis drucken
  initial begin
```

```
    $display ("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Zeit␣Reset␣Eisausgabe␣Muenzausgabe\n");
    $monitor ("%d␣␣␣%d␣␣␣␣␣␣␣%d␣␣␣␣␣␣␣␣␣␣␣%d", $time, RESET, ICE, OUTCOIN);
  end

  // Stimuli anlegen: Muenzen eingeben
  initial begin
    CLOCK = 0; COIN = X0; CANCEL = 1'b0; RESET = 1'b1;
    #50 RESET = 1'b0;
    @(negedge CLOCK);
    // drei Fuenfziger
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    // einen Fuenfziger, dann einen Euro
    #240 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    // zweimal ein Euro
    #240 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    #80 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    // einen Euro, dann einen Fuenfziger
    #320 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    // zwei Euro
    #240 $display ("Einwurf␣200␣Ct"); COIN = X200; #40 COIN = X0;
    // 1 Euro, 1 Zwanziger, Abbruch
    #280 $display ("Einwurf␣100␣Ct"); COIN = X100; #40 COIN = X0;
    #80 $display ("Einwurf␣20␣Ct"); COIN = X20; #40 COIN = X0;
    #80 $display ("Abbruch"); CANCEL = 1'b1; #40 CANCEL = 1'b0;
    // einen Fuenziger, drei Zehner, Abbruch
    #320 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Abbruch"); CANCEL = 1'b1; #40 CANCEL = 1'b0;
    // 4 Zehner, 2 Zwanziger, ein Fuenziger (keine Rueckgabe!)
    #400 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣20␣Ct"); COIN = X20; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣20␣Ct"); COIN = X20; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #80 $display ("Einwurf␣50␣Ct"); COIN = X50; #40 COIN = X0;
    #80 $display ("Einwurf␣10␣Ct"); COIN = X10; #40 COIN = X0;
    #160 $stop;
  end

endmodule
```

Testausgabe:

```
                Zeit Reset Eisausgabe Muenzausgabe
                   0   1       x           x
                  20   1       0           0
                  50   0       0           0
Einwurf 50 Ct
Einwurf 50 Ct
Einwurf 50 Ct
                 460   0       1           0
                 500   0       0           2
                 540   0       0           0
Einwurf 50 Ct
Einwurf 100 Ct
                 860   0       1           0
                 900   0       0           2
                 940   0       0           0
```

```
Einwurf 100 Ct
Einwurf 100 Ct
                1260    0       1           0
                1300    0       0           3
                1340    0       0           0
                1380    0       0           2
                1420    0       0           0
Einwurf 100 Ct
Einwurf 50 Ct
                1740    0       1           0
                1780    0       0           2
                1820    0       0           0
Einwurf 200 Ct
                2020    0       1           0
                2060    0       0           3
                2100    0       0           0
                2140    0       0           2
                2180    0       0           0
Einwurf 100 Ct
Einwurf 20 Ct
Abbruch
                2540    0       0           4
                2580    0       0           0
                2620    0       0           2
                2660    0       0           0
Einwurf 50 Ct
Einwurf 10 Ct
Einwurf 10 Ct
Einwurf 10 Ct
Abbruch
                3380    0       0           3
                3420    0       0           0
                3460    0       0           2
                3500    0       0           0
                3540    0       0           1
                3580    0       0           0
Einwurf 10 Ct
Einwurf 20 Ct
Einwurf 10 Ct
Einwurf 20 Ct
Einwurf 10 Ct
Einwurf 50 Ct
Einwurf 10 Ct
                4580    0       1           0
                4620    0       0           0
```

Diese Hausaufgaben müssen bis 3.6.11, 18:00 über das Moodle-System abgegeben werden.

Hausaufgabe 3.1 Modulare Multiplikation (10 Punkte)

Bei vielen Kryptographieverfahren wird eine besondere Multiplikation benötigt: Die modulare Multiplikation. Hierbei wird das Ergebnis der Multiplikation modulo des Körperpolynoms gerechnet, damit das Ergebnis auch wieder im Körper liegt.
Wir verwenden für die Koeffizienten der Polynome den Körper $\mathbb{GF}(2)$. Dieser besteht aus den Elementen 0 und 1. Es gelten die folgenden Regeln:

a	b	+	·
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Die Polynome selbst sind Elemente des Körpers $\mathbb{GF}(2^{11})$. Dies sind Polynome vom Grad 10 mit Koeffizienten aus dem $\mathbb{GF}(2)$. Diese Polynome können als einfacher Bitstring dargestellt werden. Das Polynom $x^{10}+x^5+x^3+1$ entspricht etwa dem String 10000101001. Die Multiplikation zweier Polynome folgt den Regeln der normalen Polynommultiplikation mit anschließender Reduktion modulo des Körperpolynoms.
Beispiel:
$(x^{10}+x^5+x^3+1)\star(x^3+1)$
Darstellung als Bitstring:
$10000101001 \star 00000001001 \mod 100000000101$
$= 10010101100001 \mod 100000000101$
$= 10101110101$

Hausaufgabe 3.1.1 a)

Erstellen Sie ein Modul in Verilog, welches 2 Polynome aus dem Körper $\mathbb{GF}(2^{11})$ vom Grad 10 multipliziert und das Ergebnis modulo des Körperpolynoms vom Grad 11 ausgibt. Das Modul soll folgende Schnittstelle haben:

```
module modmul(
  input  wire         clk,       //Takt
  input  wire         reset,     //Reset
  input  wire [11:0]  polynom,   //Körperpolynom vom Grad 11
  input  wire [10:0]  datain_a,  //Eingang A
  input  wire [10:0]  datain_b,  //Eingang B
  input  wire         enable,    //= 1, wenn gültige Daten am Eingang liegen
  output wire [10:0]  dataout,   //Ergebnis
  output wire         valid      //= 1, falls dataout gültig
  )
```

Der Multiplikationsoperator $\star$ darf nicht verwendet werden.

Hausaufgabe 3.1.2 b)

Beschreiben Sie Ihre Lösungsidee in Worten (extra pdf-Datei). Gehen Sie auch auf mögliche Schwächen Ihrer gewählten Lösung ein. Achten Sie auf eine effiziente Implementierung. Das Modul muss synthetisierbar sein. Überprüfen Sie die korrekte Funktion durch eine Testbench.

Hausaufgabe 3.1.3 c)

In einer weiteren Variante soll das Körperpolynom fest vorgegeben sein. Der Eingang `polynom` entfällt. Als Körperpolynom soll das Polynom $x^{11}+x^2+1$ verwendet werden. Erstellen Sie ein Modul `modmul_fix`. Achten Sie auf eine möglichst geringe Latenz.

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism