Einführung in

Computer Microsystems

Sommersemester 2010

Wolfgang Heenes

TECHNISCHE UNIVERSITÄT DARMSTADT

6. Aufgabenblatt mit Lösungsvorschlag

26.05.2010

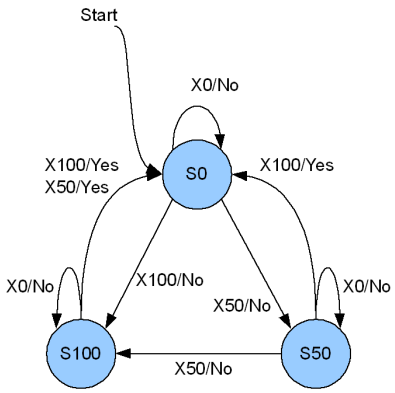
Aufgabe 1: Entwurf der Steuerung eines Verkaufsautomaten

Folgende Spezifikation für den Entwurf einer Steuerung für einen Verkaufsautomaten ist gegeben.

- *Eingabe: Münzen im Wert von 1 Euro und 50 Cent*
- *Ausgabe: Ein Eis (Wert 1,50 Euro)*
- *Überzahlung möglich, aber kein Wechselgeld*
- *Reihenfolge des Münzeinwurfs beliebig.*
- *Es kann angenommen werden, dass eine kurze Pause zwischen zwei Münzeinwürfen vorhanden ist.*

a) *Geben Sie den Zustandsgraphen zur Steuerung des Verkaufsautomaten an. Beim Zustandsautomat soll es sich um einen Mealy-Automaten handeln. Es sollten nicht mehr als drei Zustände verwendet werden.*

Lösungsvorschlag:



b) *Implementieren Sie den Mealy-Automaten in Verilog HDL. Weisen Sie die korrekte Funktionsweise des Automaten durch Simulation nach. Wieviele Flip-Flops werden zur Realisierung Ihres Schaltwerks benötigt? Der Automat soll einen asynchronen Reset haben, der auf positive Flanken reagiert.*

Lösungsvorschlag:

Die Implementierung des Automaten benutzt symbolische Eingaben, die als Parameter definiert werden.

```
module automat_eis(clk, reset, geld, eis);
// Zustandskodierung
parameter
S_0 = 2'b00,
S_50 = 2'b01,
S_100 = 2'b10;
```

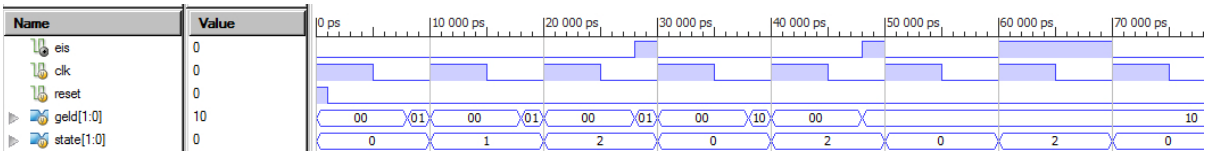
```
// Eingabecodierung
parameter
X0 = 2'b00,
X50 = 2'b01,
X100 = 2'b10;

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

// Eingänge
input clk, reset;
input [1:0] geld;
output eis;

reg [1:0] state, next_state;
reg eis;

// Realisierung der Transitionen und Ausgaben
always@(state, geld)
begin
case (state)
S_0: if (geld == X0)
     begin
      next_state = S_0;
      eis = No;
         end
     else if (geld == X50)
        begin
         next_state = S_50;
         eis = No;
        end
        else
        begin
         next_state = S_100;
         eis = No;
        end
S_50: if (geld == X0)
      begin
           next_state = S_50;
           eis = No;
          end
        else if (geld == X50)
        begin
         next_state = S_100;
         eis = No;
        end
        else
        begin
         next_state = S_0;
         eis = Yes;
        end
S_100: if (geld == X0)
     begin
          next_state = S_100;
          eis = No;
         end
        else
        begin
         next_state = S_0;
         eis = Yes;
        end
default: begin next_state = S_0; eis = No; end
endcase
end

always@(posedge clk or posedge reset)
begin
 if (reset == 1) state <= S_0;
```

```
 else
 begin
  state <= next_state;
 end
end

endmodule
```

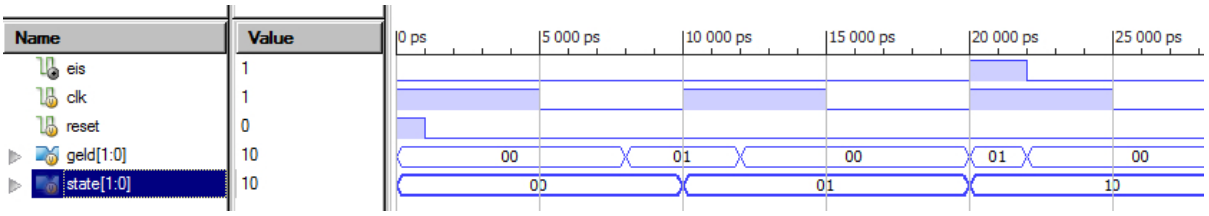
Das Stimuli zum Testen des Automaten ist im Folgenden zu sehen.

```
‘timescale 1ns / 1ps
module testbed;

parameter
X0 = 2’b00,
X50 = 2’b01,
X100 = 2’b10;

    // Inputs
    reg clk;
    reg reset;
    reg [1:0] geld;

    // Outputs
    wire eis;

    // Instantiate the Unit Under Test (UUT)
    automat_eis uut (
        .clk(clk),
        .reset(reset),
        .geld(geld),
        .eis(eis)
    );

    initial begin
        // Initialize Inputs
        reset = 1;
        geld = X0;
        #1;
        reset = 0;
        //#7;
        #8; geld = X50;
        #2; geld = X0;
        #8; geld = X50;
        #2; geld = X0;
        #8; geld = X50;
        #2; geld = X0;
        #8; geld = X100;
        #2; geld = X0;
        #8; geld = X100;
    end

   // Takterzeugung
   initial begin
        clk = 1’b1;
        forever #5 clk = !clk;
    end
endmodule
```

Die Verhaltenssimulation ergibt folgendes Bild.

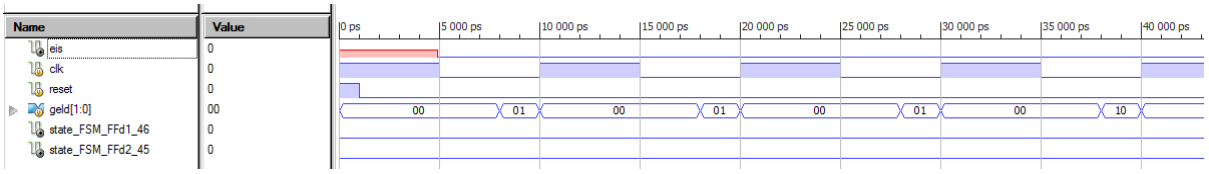


Bei der Betrachtung der Testbench fällt auf, dass das Eingabesignal (Geld) kurz vor der steigenden Flanke des Taktsignals wieder auf Null gesetzt wird. Wird das nicht gemacht, ergibt sich folgendes Bild.

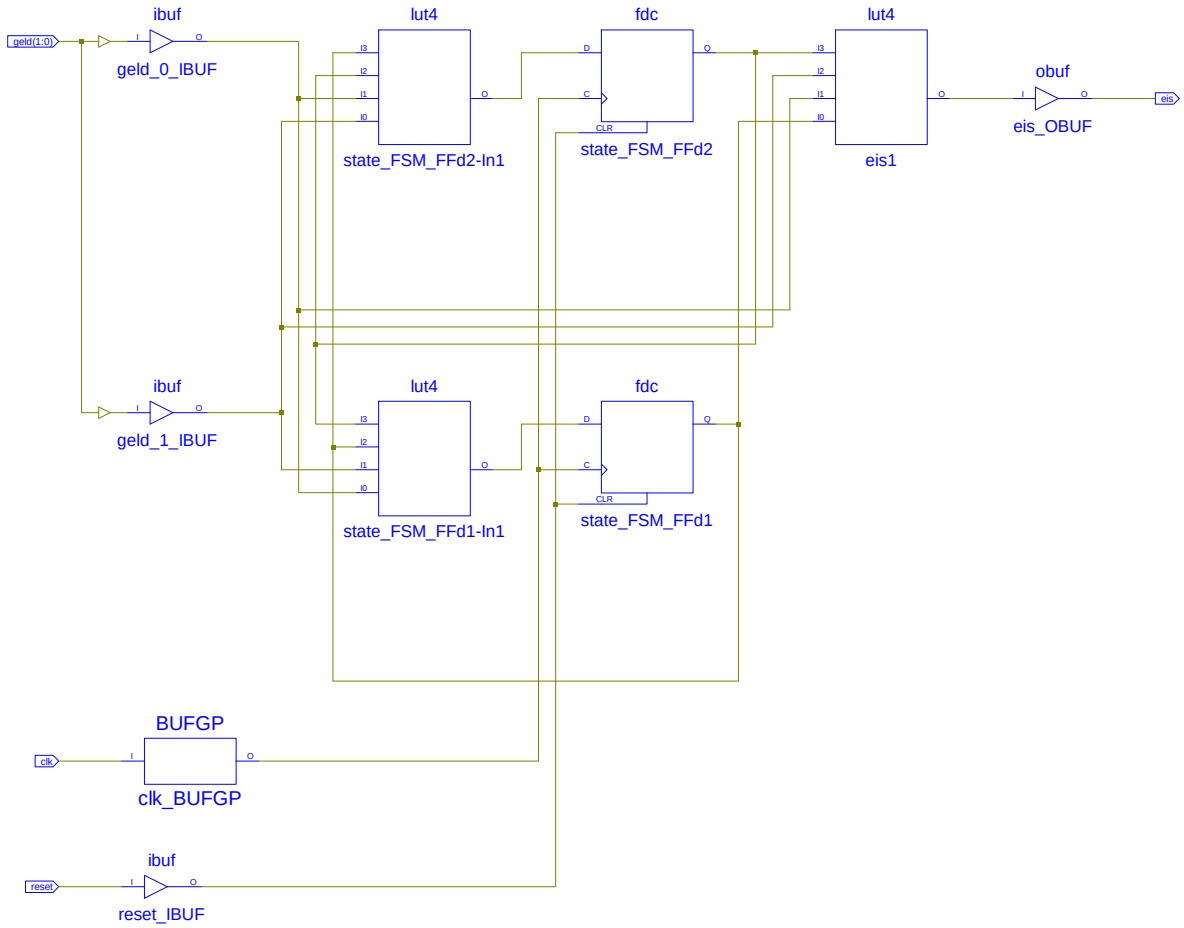


Der Automat gibt also schon nach dem Einwurf von zwei 50 Cent Stücken das Eis aus.

Wird die angegebene Testbench zur Post-Route Simulation verwendet, zeigt der Automat nicht mehr das gewünschte Verhalten. Da die Zustandsfortschaltung durch kombinatorische Logik realisiert wird und vor bzw. während der steigenden Flanken das Eingangssignal nicht lange genug anliegt, funktioniert die Schaltung nicht mehr korrekt (Verletzung der Setup- und Hold-Zeiten).



Die Synthese ergibt folgende Schaltung. Zur Zustandsspeicherung werden zwei Flip-Flops benötigt.



Aufgabe 2: Erweiterung und Modifikation der Steuerung des Verkaufsautomaten

Der Verkaufsautomat soll jetzt erweitert und modifiziert werden. So kann z. B. das asynchrone Ausgabeverhalten synchronisiert werden. Auch die Synchronisierung der Eingaben ist möglich.

a) *Zusätzlich sollen nun 10- und 20-Cent Münzen sowie 2-Euro Münzen akzeptiert werden. Erweitern Sie die Verilog HDL Spezifikation des Automaten entsprechend.*
Hinweis*: Bei den nun vorhandenen Möglichkeiten ist zu überlegen, anstatt weiterer Zustände ein Summenregister einzuführen.*

Lösungsvorschlag:

Der Automat wird nun als Moore-Automat implementiert. Da ein Summenregister vorhanden ist, dass das eingeworfene Geld asynchron zählt, werden nur noch zwei Zustände benötigt.

1. Noch nicht genug Geld eingeworfen. Es wird kein Eis ausgegeben.
2. Der Betrag ist erreicht, das Eis wird ausgegeben und alle Zähler zurückgesetzt.

Eine Implementierung zeigt das folgende Listing. Es gibt allerdings verschiedene andere Möglichkeiten die Aufgabe zu lösen.

```
module automat_eis_2(
    input clk,
    input reset,
    input [2:0] geld,
    output reg eis
    );

parameter
    // Zustandscodierung
    S_0 = 1'b0,
    S_1 = 1'b1,
    // Eingabecodierung
    X0   = 3'b000,
    X10  = 3'b001,
    X20  = 3'b010,
    X50  = 3'b011,
    X100 = 3'b100,
    X200 = 3'b101,
    // Preis eines Eises
    Preis = 6'd15,
    // Ausgabecodierung
    Yes = 1'b1,
    No  = 1'b0;

reg state, next_state;
reg [5:0] betrag;   // speichert den bereits eingeworfenen Betrag (0 - 34 * 10 Cent)

wire reset_betrag = eis | reset;    // Betrag bei Eisausgabe oder reset zurücksetzen

// Geld zählen
reg tmp_reset_betrag;   // Um negative von positiven Flanken an reset_betrag zu unterscheiden
always @( geld, reset_betrag ) begin
    if (reset_betrag)
        betrag <= 6'd0;
    else if (~tmp_reset_betrag) // nur wenn keine negative Flanke an reset_betrag
        case (geld)
            X10:    betrag <= betrag + 6'd1;
            X20:    betrag <= betrag + 6'd2;
            X50:    betrag <= betrag + 6'd5;
            X100:   betrag <= betrag + 6'd10;
            X200:   betrag <= betrag + 6'd20;
            default: betrag <= betrag;
        endcase
    tmp_reset_betrag <= reset_betrag;
end

// Realisierung der Transitionen und Ausgabe
always @( state, betrag ) begin
    case (state)
        S_0: begin
            if (betrag < Preis)
```

```
                next_state <= S_0;
            else
                next_state <= S_1;
            eis <= No; end
        S_1: begin
            next_state <= S_0;
            eis <= Yes; end
        default: begin
            next_state <= S_0;
            eis <= No; end
    endcase

end

// Zustandsfortschaltung
always @( posedge clk, posedge reset ) begin
    if (reset)
        state <= S_0;
    else
        state <= next_state;
end
endmodule
```

Der Automat zählt keine Münzen während er das Eis ausgibt. Da der Output `eis` aber nur einen Takt lang gesetzt ist, sollte das kein Problem darstellen (besonders unter Berücksichtigung der Taktrate, siehe Aufgabenteil b)).

b) *Analysieren Sie die Implementierung[1]. Latches sollten bei Ihrer Realisierung nicht auftreten. Andernfalls korrigieren Sie Ihre Spezifikation entsprechend. Wie hoch ist die maximale Taktfrequenz, mit der Ihr Automat betrieben werden kann.*

Lösungsvorschlag:

Es sollten keine Latches verwendet werden bzw. auftreten. Der vorliegende Lösungsvorschlag enthält aber trotzdem sechs Latches. So werden die sechs Register, welche den Betrag speichern zu Latches. Obwohl die Implementierung Latches enthält, kann eine maximale Taktfrequenz angegeben werden, da zur Zustandsspeicherung Register verwendet werden. Die maximale Taktfrequenz, mit der der Automaten betrieben werden kann, beträgt etwa 203 MHz (FPGA: Spartan 3E).

c) *Erweitern Sie Ihren Automaten um die Funktion der Wechselgeldausgabe. Sie können davon ausgehen, dass stets genug Wechselgeld vorhanden ist. Ergänzen Sie die Verilog HDL Spezifikation und weisen Sie die korrekte Funktionsweise durch Simulation nach.*

Lösungsvorschlag:

Für die Münzrückgabe wird ein weiterer Zustand, in dem das Wechselgeld ausgegeben wird, benötigt. Das folgende Listing zeigt die modifizierte Spezifikation.

```
module automat_eis_3(
    input clk,
    input reset,
    input [2:0] geld,
    output reg [2:0] wechselgeld,
    output reg eis
    );

parameter
    // Zustandscodierung
    S_0 = 2'b00,
    S_1 = 2'b01,
    S_2 = 2'b10,
    // Eingabecodierung
    X0   = 3'b000,
    X10  = 3'b001,
```

[1] Syntheseergebnis

```
    X20  = 3'b010,
    X50  = 3'b011,
    X100 = 3'b100,
    X200 = 3'b101,
    // Preis eines Eises
    Preis = 6'd15,
    // Ausgabecodierung
    Yes = 1'b1,
    No  = 1'b0;

reg [1:0] state, next_state;
reg [5:0] betrag;   // speichert den bereits eingeworfenen Betrag (0 - 34 * 10 Cent)
reg [5:0] restgeld, next_restgeld;
wire reset_betrag = eis | reset;    // Betrag bei Eisausgabe oder reset zurücksetzen

// Geld zählen
reg tmp_reset_betrag;   // Um negative von positiven Flanken an reset_betrag zu unterscheiden
always @( geld, reset_betrag ) begin
    if (reset_betrag)
        betrag <= 6'd0;
    else if (~tmp_reset_betrag) // nur wenn keine negative Flanke an reset_betrag
        case (geld)
            X10:    betrag <= betrag + 6'd1;
            X20:    betrag <= betrag + 6'd2;
            X50:    betrag <= betrag + 6'd5;
            X100:   betrag <= betrag + 6'd10;
            X200:   betrag <= betrag + 6'd20;
            default: betrag <= betrag;
        endcase
    tmp_reset_betrag <= reset_betrag;
end

// Realisierung der Transitionen und Ausgabe
always @( state, betrag, restgeld ) begin
    case (state)
        S_0: begin
            if (betrag < Preis)
                next_state <= S_0;
            else
                next_state <= S_1;
            eis <= No;
            wechselgeld <= X0;
            next_restgeld <= betrag - Preis; end
        S_1: begin
            next_state <= S_2;
            eis <= Yes;
            next_restgeld <= restgeld;
            wechselgeld <= X0; end
        S_2: begin  // Restgeld ausgeben
            if (restgeld >= 6'd10) begin
                next_restgeld <= restgeld - 6'd10;
                wechselgeld <= X100;
                next_state <= S_2;
            end else if (restgeld >= 6'd5) begin
                next_restgeld <= restgeld - 6'd5;
                wechselgeld <= X50;
                next_state <= S_2;
            end else if (restgeld >= 6'd2) begin
                next_restgeld <= restgeld - 6'd2;
                wechselgeld <= X20;
                next_state <= S_2;
            end else if (restgeld >= 6'd1) begin
                next_restgeld <= restgeld - 6'd1;
                wechselgeld <= X10;
                next_state <= S_2;
            end else begin
                next_restgeld <= 6'd0;
                wechselgeld <= X0;
                next_state <= S_0;
            end
```

```
            eis <= No;
        end
        default: begin
            next_state <= S_0;
            eis <= No;
            wechselgeld <= X0;
            next_restgeld <= 6'd0; end
    endcase
end

// Zustandsfortschaltung
always @( posedge clk, posedge reset ) begin
    if (reset)
        state <= S_0;
    else
        state <= next_state;
end

// Restgeldberechnung
always @( posedge clk)
    restgeld <= next_restgeld;
endmodule
```

Von der korrekten Funktionsweise kann man sich mit folgender Abbildung überzeugen.