

Digitaltechnik

Wintersemester 2025/2026

Vorlesung 11



TECHNISCHE
UNIVERSITÄT
DARMSTADT












ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

- 1 Organisatorisches
- 2 Moore vs. Mealy Automaten
- 3 Zerlegen von Zustandsautomaten
- 4 SystemVerilog für parametrisierte Module
- 5 SystemVerilog für Testumgebungen



Harris 2016
Kap. 5.4, 4.8 -
4.10

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen










1 Organisatorisches

2 Moore vs. Mealy Automaten

3 Zerlegen von Zustandsautomaten

4 SystemVerilog für parametrisierte Module

5 SystemVerilog für Testumgebungen


Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen

■ Ziel / Nutzen

- Fortlaufende Verbesserung der Veranstaltung
 - Bewertungsgrundlage für Vergabe vom „Preis für gute Lehre“ des FB 20
- ⇒ kommt Studierenden und Lehrenden zugute

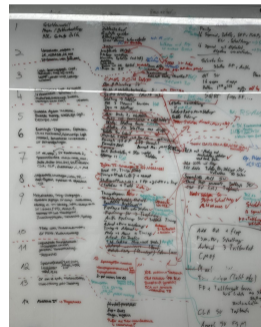
■ Ablauf

- Anonymisierte Online-Fragebögen bis spätestens **15.01.2026**:
<http://evaluation.tu-darmstadt.de/evasys/online.php>
 - Persönliche TANs in Moodle verfügbar
 - Vorlesung und Übung werden getrennt evaluiert
- ⇒ Online-Fragebögen mit zwei unterschiedlichen TANs öffnen

Bewertung	0,00 / 100,00
Bewertet am	Dienstag, 19. Dezember 2023, 11:42
Bewertet von	 Andreas Brüggemann

Feedback als Kommentar

Vorlesung: TAN1 Übung: TAN2












Umbau von DT für
das WiSe23/24



- Anmeldungen in TUCaN **bis 31.01.2026** nicht vergessen!
- Anmeldungen notwendig für
 - Fachprüfung
 - **und** Studienleistung (außer es wurde bereits eine Studienleistung in vorherigen Semestern erworben)
- Für den Erwerb der Studienleistung müssen
 - mindestens **60 Punkte** in den Hausaufgabenblättern erreicht werden
 - **und** mindestens **65 Punkte** aus den Moodle-Quizen erreicht werden
- Noch offene Abgaben: Quiz 9 (heute), 10, 11, 12 und 13 und Hausaufgaben E und F
- Bestenfalls aktuelle Punkte frühzeitig selber im Moodle abrufen und prüfen!

Abgabefrist für Hausaufgabe E zu
Vorlesungen 09 und 10 nächste Woche
Freitag 23:59!
Wöchentliches Moodle-Quiz nicht vergessen!

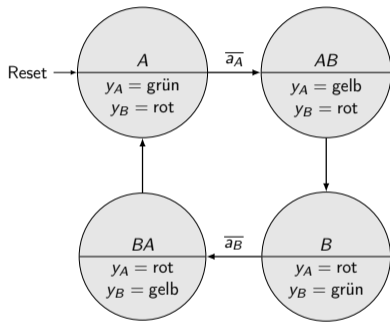
- 1 Organisatorisches
- 2 Moore vs. Mealy Automaten**
- 3 Zerlegen von Zustandsautomaten
- 4 SystemVerilog für parametrisierte Module
- 5 SystemVerilog für Testumgebungen

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen

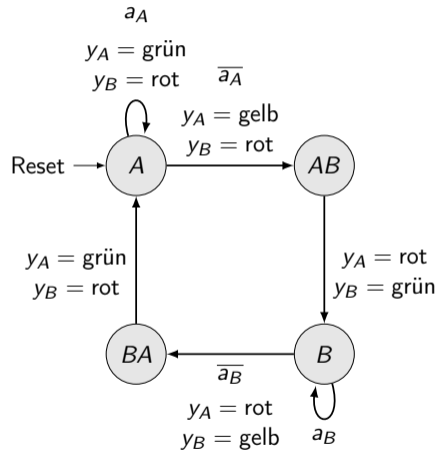
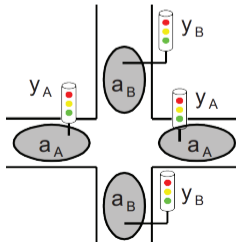
Wiederholung: Ampelsteuerung Moore-Automat besser geeignet



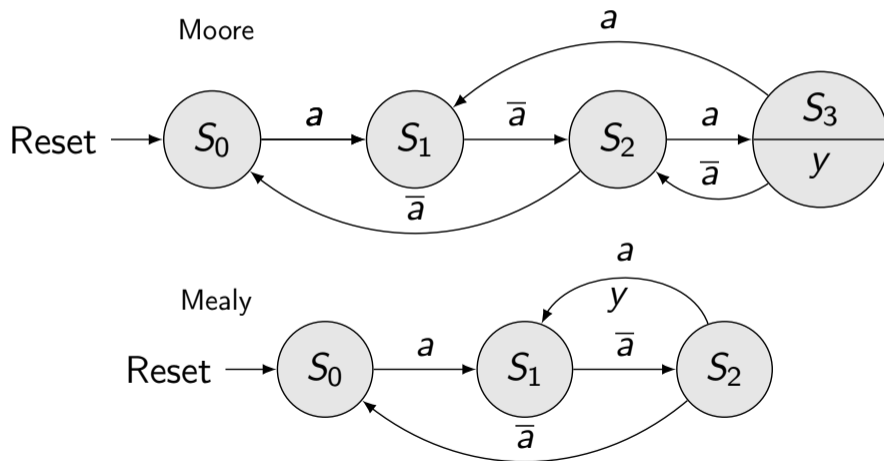
ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING



Moore

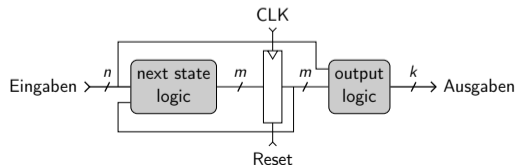
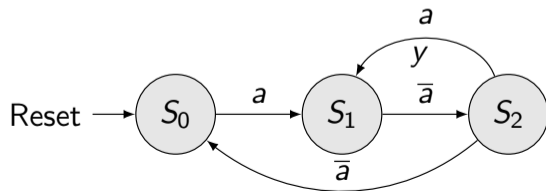
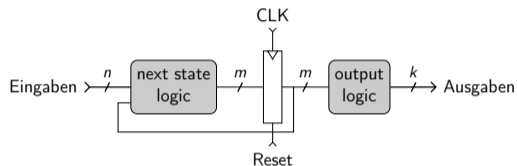
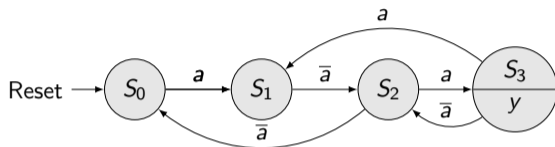


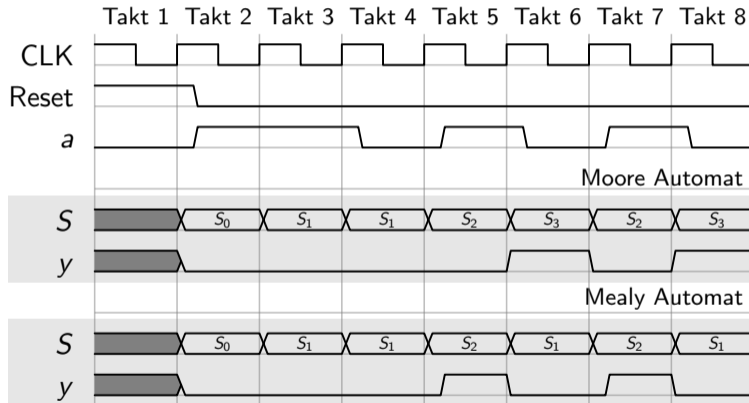
Mealy














- Ampelsteuerung: Moore-Automat besser geeignet, weniger explizite Transitionen notwendig, einfachere Ausgabelogik
 - 101 Mustererkennung: Mealy-Automat besser geeignet, weniger Zustände notwendig
- ⇒ Je nach Anwendungsfall kann eine der Optionen besser sein
- in der Regel:
 - Moore besser, wenn Ausgaben statisch (z.B. in aktueller Ampelphase)
 - Mealy besser, wenn Ausgaben kurzfristige Aktionen auslösen (z.B. 1 ausgeben, wenn die letzten 3 gelesenen Bits 101 waren)
 - außerdem: Mealy reagiert schneller auf Änderungen der Eingabe, siehe nächste Folien



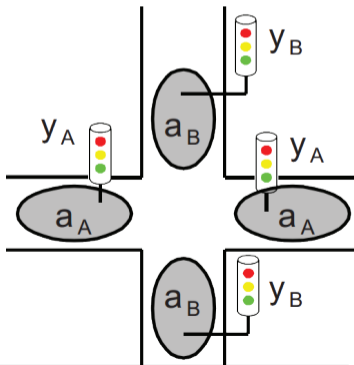
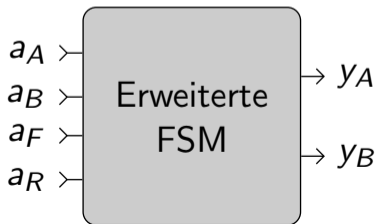


- Mealy-Automat erkennt Muster einen Takt früher

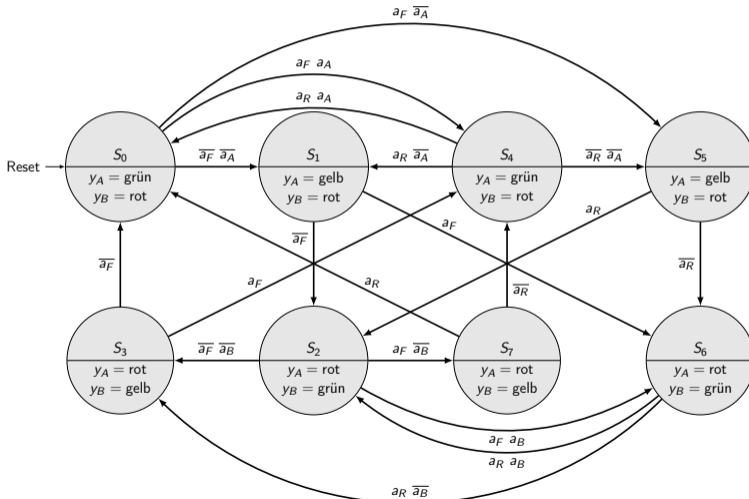
- 1 Organisatorisches
- 2 Moore vs. Mealy Automaten
- 3 Zerlegen von Zustandsautomaten**
- 4 SystemVerilog für parametrisierte Module
- 5 SystemVerilog für Testumgebungen

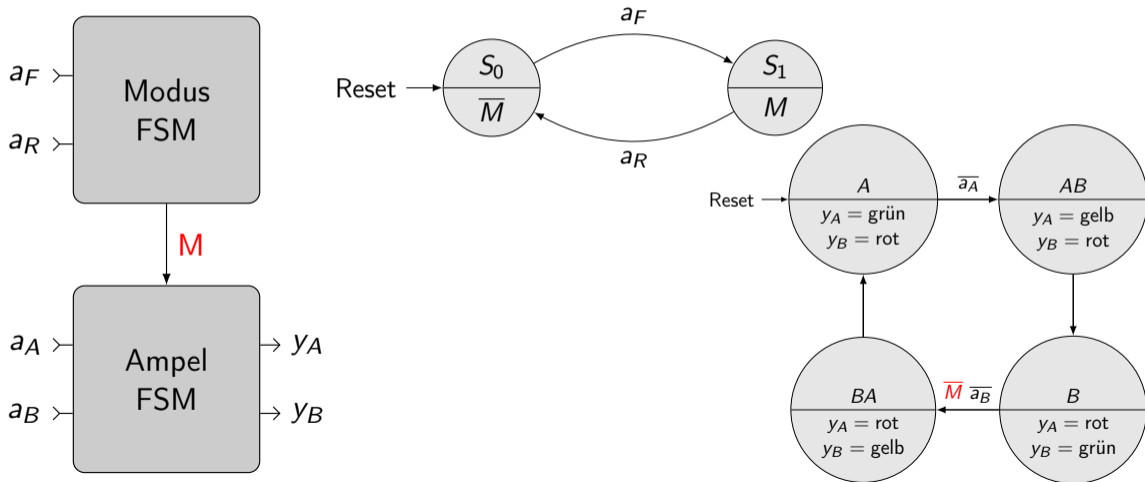
Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen

- Aufteilen komplexer FSMs in einfachere interagierende FSMs
- Beispiel: Ampelsteuerung mit Modus für Festumzüge (Ampel B bleibt permanent grün)
 - FSM bekommt zwei weitere Eingänge: a_F, a_R
 - $a_F = 1 \Rightarrow$ aktiviert Festumzugsmodus
 - $a_R = 1 \Rightarrow$ deaktiviert Festumzugsmodus












Unzerlegte FSM

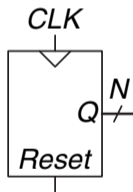




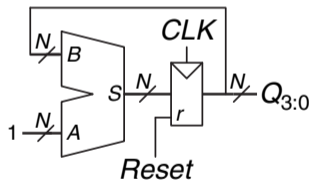
- 1 Organisatorisches
- 2 Moore vs. Mealy Automaten
- 3 Zerlegen von Zustandsautomaten
- 4 SystemVerilog für parametrisierte Module**
- 5 SystemVerilog für Testumgebungen

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen

- Erhöht sich bei jeder steigenden Taktflanke
- Dient zum Durchlaufen von Zahlen. Zum Beispiel:
 - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001...
- Verwendung (Beispiele):
 - Displays von Digitaluhren, Programmzähler in einer CPU, ...



Symbol



Implementierung

- Im Folgenden: Wie lassen sich Addierer und Register für beliebig viele Bits N in SystemVerilog umsetzen?



- neben Ein- und Ausgaben kann Modulschnittstelle auch parameter definieren
- parametrisierte Eigenschaften werden bei Instanziierung durch konkrete Werte ersetzt
 - Nach Synthese der Hardware und während Simulation nicht mehr änderbar
 - vergleichbar mit C-Präprozessor oder Java-Generics
- typische Parameter: Port-Breite, Speichertiefe, ...

adder.sv

```
1 module adder
2   # ( parameter N)
3
4   ( input  logic [N-1:0] A, B,
5     output logic [N-1:0] S);
6
7   assign S = A + B;
8
9 endmodule
```

counter.sv

```
1 //...
2 // adder mit N=4
3 // instantiieren
4 adder #(4) add(
5   .A(4'd1),
6   .B(current_value),
7   .S(next_value));
8 //...
```



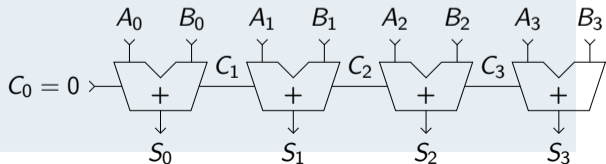
register.sv

```
1 module register
2   # ( parameter N = 8) // Hier mit Standardwert
3
4   ( input  logic          CLK, RST,
5     input  logic [N-1:0]  D, // Breite der Eingabe und Ausgabevektoren
6     output logic [N-1:0]  Q); // hängen von Parameter ab
7
8   always_ff @(posedge CLK) begin
9     if (RST) begin
10      Q <= 0;
11    end else begin
12      Q <= D;
13    end
14  end
15
16 endmodule
```

■ Anzahl von Submodulen hängt oft von Parameter ab

rca.sv

```
1  module adder
2      # ( parameter N)
3
4      ( input  logic [N-1:0] A, B,
5        output logic [N-1:0] S);
6
7      logic [N:0] C; // C[i] ist C_in für Bit i und C_out für Bit i-1
8
9      genvar i; // Schleifenvariable im generate-Block
10     generate
11         for (i = 0; i < N; i = i + 1) begin
12             full_adder fa(
13                 .A(      A[i]),
14                 .B(      B[i]),
15                 .C_in(    C[i]),
16                 .S(      S[i]),
17                 .C_out(    C[i + 1]));
18         end
19     endgenerate
20
21     assign C[0] = 0; // Carry-In ist 0
22
23 endmodule
```



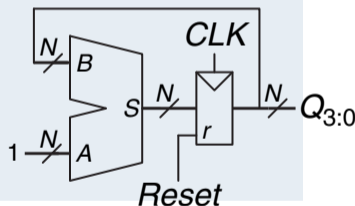
Implementierung eines 4-bit Zählers



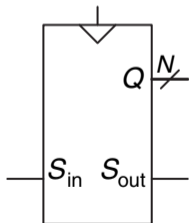
ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

counter.sv

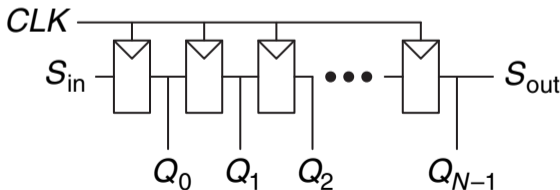
```
1 module counter // counter for 4 bit numbers
2   ( input logic      CLK, RST,
3     output logic [3:0] Q);
4
5   logic [3:0] current_value, next_value;
6
7   register #(.N(4)) regis( // Zuweisung an spezifischen Parameter-Namen
8     .CLK(CLK),
9     .RST(RST),
10    .D(next_value),
11    .Q(current_value));
12   adder #(4) add(
13     .A(4'd1),
14     .B(current_value),
15     .S(next_value));
16
17   assign Q = current_value;
18
19 endmodule
```



- Bei jeder steigenden Taktflanke wird der Speicherinhalt ein Flip-Flop weiter verschoben (FIFO-Prinzip: *First In – First Out*)
 - Neues Bit S_{in} wird eingelesen
 - Letztes Bit S_{out} wird nach außen verschoben/verworfen
- Seriell-Parallel-Wandler: Wandelt den seriellen Eingang (S_{in}) in den parallelen Ausgang ($Q_{0:N-1}$) um



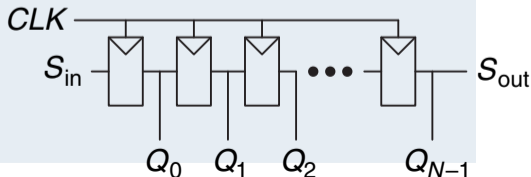
Symbol



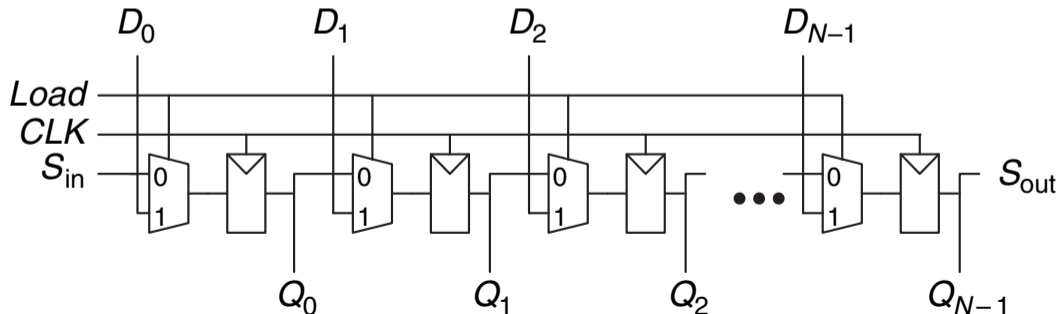
Implementierung

shift_reg.sv










```
1 module shift_reg
2   # ( parameter N = 32) // Anzahl der Register
3
4   ( input  logic          CLK, RST,
5     input  logic          S_in,
6     output logic [N-1:0]  Q
7     output logic          S_out);
8
9   logic [N:0] c; // c[i]: Eingabe für Register i
10  assign c[0]  = S_in;
11  assign Q     = c[N:1];
12  assign S_out = c[N];
13
14  genvar i; // Schleifenvariable im generate-Block
15  generate
16    for (i = 0; i < N; i = i + 1) begin
17      register #(1) regis (
18        .CLK( CLK),
19        .RST( RST),
20        .D(   c[i]),
21        .Q(   c[i+1]));
22    end
23  endgenerate
24
25 endmodule
```



- Für $Load = 1$: normales N -Bit Register
- Für $Load = 0$: Schieberegister
- Kann dadurch sowohl als Seriell-Parallel-Wandler (S_{in} zu $Q_{0:N-1}$, $Load = 0$) als auch als Parallel-Seriell-Wandler ($D_{0:N-1}$ zu S_{out} , $Load = 1$) fungieren



- 1 Organisatorisches
- 2 Moore vs. Mealy Automaten
- 3 Zerlegen von Zustandsautomaten
- 4 SystemVerilog für parametrisierte Module
- 5 SystemVerilog für Testumgebungen**

Anwendungs- software		Programme
Betriebs- systeme		Gerätetreiber
Architektur		Befehle Register
Mikro- architektur		Datenpfade Steuerung
Logik		Addierer Speicher
Digital- schaltungen		UND Gatter Inverter
Analog- schaltungen		Verstärker Filter
Bauteile		Transistoren Dioden
Physik		Elektronen

simple.sv

```
1 module simple(input logic A, B, C,  
2               output logic Y);  
3  
4   assign Y = ~B & ~C | A & ~B;  
5  
6 endmodule
```

Testbench

z.B. simple_tb.sv

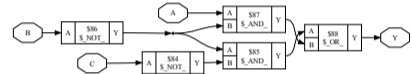
Synthese

z.B. ./synth.sh simple

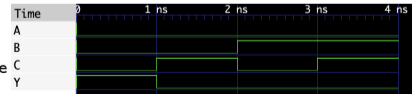
Simulation

z.B. ./sim.sh simple

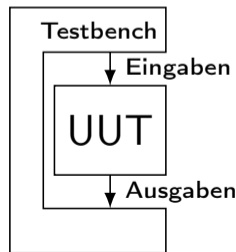
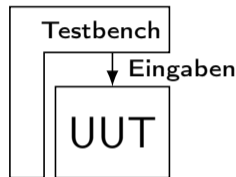
z.B. GTKWave



```
VCD info: dumpfile simple_selftest_tb.vcd opened for output.  
VCD warning: $dumpvars: Package ($unit) is not dumpable with VCD.  
000 ok.  
001 ok.  
010 ok.  
011 ok.  
FINISHED simple_selftest_tb  
simple_selftest_tb.sv:28: $finish called at 400 (10ps)
```



- HDL-Programm zum Testen eines anderen HDL-Moduls **vor** der Synthese
- getestetes Modul
 - Unit under test (UUT), manchmal Device under test (DUT)
- Testbench kann i.d.R. **nicht synthetisiert** werden
 - nur für **Simulation** genutzt
- Arten von Testbenches
 - einfach: Testdaten an uut anlegen und Ausgaben anzeigen
 - selbstprüfend: Ausgaben zusätzlich automatisch auf Korrektheit prüfen
 - selbstprüfend mit Testvektoren: variable Testdaten (bspw. aus Datei lesen),
s. Kapitel 4.9 in Harris 2013/2016





$$Y = \overline{B} \overline{C} + A \overline{B}$$

simple.sv

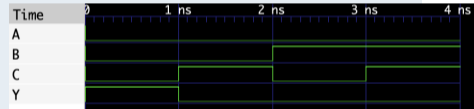
```
1 module simple(input  logic A, B, C,  
2               output logic Y);  
3  
4     assign Y = ~B & ~C | A & ~B;  
5  
6 endmodule
```

simple_tb.sv

```
1 `timescale 1 ns / 10 ps
2 module simple_tb; // Testbench ist auch ein Modul
3   logic A, B, C, Y;
4   // Instantiiere zu testendes Modul (unit under test / uut)
5   simple uut(.A(A), .B(B), .C(C), .Y(Y));
6
7   initial begin // Wird zu Beginn der Simulation ausgeführt
8     // Dump file für Werte aller Variablen anlegen:
9     $dumpfile("simple_tb.vcd");
10    $dumpvars;
11
12    // Eigentliche Simulation:
13    A = 0; B = 0; C = 0;
14    #1; // Verzögerung zwischen verschiedenen Eingaben notwendig,
15        // um pro Eingabe die entsprechende Ausgabe ablesen zu können
16    C = 1;
17    #1;
18    B = 1;
19    C = 0;
20    #1;
21    C = 1;
22    #1;
23    $display("FINISHED simple_tb"); // Einfache Textausgabe
24    $finish; // Beendet Simulation
25  end
26 endmodule
```

$$Y = \overline{B} \overline{C} + A \overline{B}$$

VCD info: dumpfile simple_tb.vcd opened for output.
VCD warning: \$dumpvars: Package (\$unit) is not dumpable with VCD.
FINISHED simple_tb
simple_tb.sv:24: \$finish called at 400 (10ps)



Code
synthetisiert
nicht!

simple_selftest_tb.sv

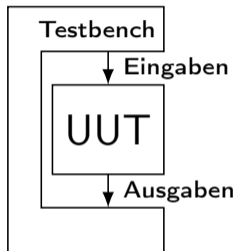
```
1 `timescale 1 ns / 10 ps
2 module simple_selftest_tb;
3     logic A, B, C, Y;
4     simple uut(.A(A), .B(B), .C(C), .Y(Y));
5     initial begin
6         $dumpfile("simple_selftest_tb.vcd");
7         $dumpvars;
8         // === in Testbenches als Gleichheitstest verwenden!
9         // Testet auf logische Gleichheit (0,1,X,Z)
10        A = 0; B = 0; C = 0;
11        #1;
12        if (Y === 1) $display("000 ok."); else $display("Error: 000 failed!");
13        C = 1;
14        #1;
15        if (Y === 0) $display("001 ok."); else $display("Error: 001 failed!");
16        B = 1;
17        C = 0;
18        #1;
19        if (Y === 0) $display("010 ok."); else $display("Error: 010 failed!");
20        C = 1;
21        #1;
22        if (Y === 0) $display("011 ok."); else $display("Error: 011 failed!");
23        $display("FINISHED simple_selftest_tb");
24        $finish;
25    end
26 endmodule
```

$$Y = \overline{B} \overline{C} + A \overline{B}$$

```
VCD info: dumpfile simple_selftest_tb.vcd opened for output.
VCD warning: $dumpvars: Package ($unit) is not dumpable with VCD.
000 ok.
001 ok.
010 ok.
011 ok.
FINISHED simple_selftest_tb
simple_selftest_tb.sv:28: $finish called at 400 (10ps)
```

Code
synthetisiert
nicht!

- Testbench ist Modul ohne eigene Ports (Eingänge und Ausgänge)
- Erzeugt Stimuli (Takt, Reset, Eingabedaten)
- Instantiiert „unit under test“
- Kann Ausgabedaten und Timing verifizieren
 - erschöpfend oder zufällig
 - Grenzfälle abdecken
- Wird nicht synthetisiert



- ⇒ **nur in Testbenches ist nicht synthetisierbarer Code erlaubt**
- ⇒ es ist möglich, Verzögerungen (z.B. #1;) und spezielle Anweisungen wie z.B. `$display(...)` zu benutzen
- ⇒ if-else, for-Schleifen etc. können hier wie bei Softwareprogrammierung genutzt werden
- ⇒ `initial begin ... end` für Code, der ab Beginn der Simulation einmalig ausgeführt werden soll



- **Achtung:** Zum Testen auf Gleichheit in Testbenches `===` für logische Gleichheit (vierwertig) nutzen!
- `===` testet, ob zwei Werte in vierwertiger Logik logisch gleich sind
- `==` gibt z.B. immer `x` aus, sobald ein Eingang `x` ist
 - ⇒ $1'b1 == 1'bx \Rightarrow x, 1'bx == 1'bx \Rightarrow x$
 - ⇒ $1'b1 === 1'bx \Rightarrow 0, 1'bx === 1'bx \Rightarrow 1$



- `$display(<format>, <values>*)`;
- ähnlich `printf` in C und Java
- wichtige Platzhalter:
 - `%d` `%b` `%h` für dezimale, binäre oder hexadezimale Darstellung
 - `%t` für Zeit (mit Einheit)
- `$time` ist aktueller Zeitpunkt der Simulation
- bspw.:
`$display("%t: %h + %d = %b", $time, 4'b1111, 4'b0001, 4'b1111+4'b0001);`
erzeugt: 3.0 ns: f + 1 = 0000
- z.B. `$timeformat(-9, 1, " ns", 8)`; zum Einstellen des Zeitformats
 - Skalierung auf 10^{-9} (=nano)
 - eine Nachkommastelle
 - Einheiten-Suffix (ns)
 - Anzahl der insgesamt anzuzeigenden Zeichen

Weiteres Beispiel für eine selbstprüfende Testbench



ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

register_tb.sv

```
1 `timescale 1 ns / 10 ps
2 module register_tb;
3     logic CLK, RST; logic [3:0] D, Q;
4     register #(4) uut(.CLK(CLK), .RST(RST), .D(D), .Q(Q));
5     always begin // Simuliere clock: 1ns 0, 1ns 1, repeat
6         CLK = 0; #1; CLK = 1; #1;
7     end
8     initial begin
9         $dumpfile("register_tb.vcd"); $dumpvars;
10        $timeformat(-9, 1, " ns", 7);
11        // aktuell CLK == 0
12        RST = 1;
13        #2; // posedge CLK, danach wieder CLK == 0
14        RST = 0;
15        for (logic [4:0] i = 0; i < 16; i++) begin // for-Schleife ohne generate in Testbenches erlaubt
16            D = i[3:0]; // (i hat 5 bits, damit 16 dargestellt werden kann)
17            #2; // posedge CLK, danach wieder CLK == 0
18            if (Q == D) begin
19                $display("Success for D=%b (%t)", D, $time);
20            end else begin
21                $display("Error for D=%b, expected Q=%b but got Q=%b! (%t)", D, D, Q, $time);
22            end
23        end
24        $display("FINISHED register_tb"); $finish;
25    end
26 endmodule
```

Code
synthetisiert
nicht!

Weiteres Beispiel für eine selbstprüfende Testbench

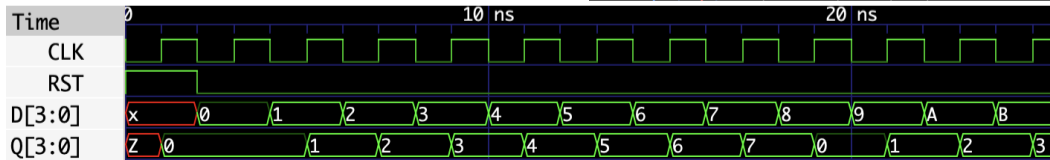


ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

register_tb.sv

```
1 `timescale 1 ns / 10 ps
2 module register_tb;
3 ...
4 always begin // Simuliere clock: 1ns 0, 1ns 1, repeat
5     CLK = 0; #1; CLK = 1; #1;
6 end
7 initial begin
8     ...
9     // aktuell CLK === 0
10    RST = 1;
11    #2; // posedge CLK, danach wieder CLK == 0
12    RST = 0;
13    for (logic [4:0] i = 0; i < 16; i++) begin
14        D = i[3:0];
15        #2; // posedge CLK, danach wieder CLK == 0
16        if (Q === D) begin
17            ...
```

```
VCD info: dumpfile register_tb.vcd opened for output.
VCD warning: $dumpvars: Package ($unit) is not dumpable with VCD.
Success for D=0000 ( 4.0 ns)
Success for D=0001 ( 6.0 ns)
Success for D=0010 ( 8.0 ns)
Success for D=0011 (10.0 ns)
Success for D=0100 (12.0 ns)
Success for D=0101 (14.0 ns)
Success for D=0110 (16.0 ns)
Success for D=0111 (18.0 ns)
Error for D=1000, expected Q=1000 but got Q=0000! (20.0 ns)
Error for D=1001, expected Q=1001 but got Q=0001! (22.0 ns)
Error for D=1010, expected Q=1010 but got Q=0010! (24.0 ns)
Error for D=1011, expected Q=1011 but got Q=0011! (26.0 ns)
Error for D=1100, expected Q=1100 but got Q=0100! (28.0 ns)
Error for D=1101, expected Q=1101 but got Q=0101! (30.0 ns)
Error for D=1110, expected Q=1110 but got Q=0110! (32.0 ns)
Error for D=1111, expected Q=1111 but got Q=0111! (34.0 ns)
FINISHED register_tb
register_tb.sv:24: $finish called at 3400 (10ps)
```



- Erstellen effizienter Testpläne ist nicht trivial
 - Abdeckung maximieren (gezielt vs. zufällig)
 - Wiederverwendbarkeit maximieren
 - Überlappung minimieren
- Multi-Domänen Cosimulation von Hardware und
 - Software
 - Event-basierten Kommunikationsprotokollen
 - kontinuierlichen physikalischen Prozessen
- Testgetriebene Entwicklung (test-driven development)
⇒ SystemVerilog bringt hier viele Verbesserungen
 - file IO
 - assertions, implications
 - (constrained) random
 - Klassen, Vererbung, Schnittstellen
 - Direct Programming Interface (DPI) zu C, C++, SystemC, etc.



Chris Spear:
SystemVerilog
for Verification
(Springer)












- 1 Organisatorisches
- 2 Moore vs. Mealy Automaten
- 3 Zerlegen von Zustandsautomaten
- 4 SystemVerilog für parametrisierte Module
- 5 SystemVerilog für Testumgebungen

nächste Vorlesung beinhaltet

- Speicherfelder: RAM, ROM, Lookup-Tabellen
- Logikfelder: PLA, FPGA

Hausaufgabe E zu Vorlesungen 09 und 10 muss bis nächste Woche Freitag 23:59 abgegeben werden.
Wöchentliches Moodle-Quiz nicht vergessen!

Anwendungssoftware 	Programme
Betriebssysteme 	Gerätetreiber
Architektur 	Befehle Register
Mikroarchitektur 	Datenpfade Steuerung
Logik 	Addierer Speicher
Digital-schaltungen 	UND Gatter Inverter
Analog-schaltungen 	Verstärker Filter
Bauteile 	Transistoren Dioden
Physik 	Elektronen