Prozedurale Zuweisungen

Register

Wires verbinden Gatter. Eine Leitung kann aber nichts speichern. Dafür verwendet man Register. Verilog stellt den Typ **reg** zur Verfügung, der vom Synthesizer in ein Register umgesetzt werden kann (aber nicht muss).

```
wire [15:0]num;
reg  bit;
reg  [31:0]counter;
```

Die Bits von **counter** können einzeln verwendet werden, oder ähnlich einer Variablen. Allerdings macht eine statische Zuweisung keinen Sinn. Warum?

```
assign num     = 12350;  // ok
assign bit     = 1;      // ERROR und nicht sinnvoll
assign counter = 123456; // ERROR und nicht sinnvoll
```

Speicher verändern macht nur auf ein Ereignis hin Sinn. Dieses definiert man mit **always@(...)** z.B.:

```
always@(posedge clk)
begin
  counter = counter + 1;
end
```



Prozedurale Zuweisungen

Beispiel Zähler

```
module counter_test(
    input clk,              // Wechsel 1/0 mit 100 Mhz
    output [15:0]HW_led
    );

    reg [31:0]counter;

    initial
    begin
      counter = 0;
    end

    always @(posedge clk)
    begin
        counter = counter + 1;
    end
endmodule
```

counter kann wie ein integer verwendet werden. Außerdem kann man leicht auf ein bel. Bit n zugreifen: **counter[n]**

BASYS3 Takt: 100 MHz = 100 Mio. Taktzyklen/Sekunde d.h.: pro Zyklus 1x steigende Flanke (posedge clk) u. 1x fallende Flanke => counter = counter+1;

```
assign HW_led[0] = counter[0]; // „blinkt“ mit 50 Mhz
assign HW_led[1] = counter[1]; // „blinkt“ mit 25 Mhz
assign HW_led[2] = counter[2]; // „blinkt“ mit  ...
```


© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Praktikum
Takt und Frequenzteiler

1) Innerhalb von **`always@( ... )`** wird **`counter`** mit 100 MHz hochgezählt (100 Mio. mal/Sekunde). Überlegen Sie, mit welcher Häufigkeit die 32 Bits von counter pro Sekunde blinken (gemeint ist ein Zyklus ein/aus). Verbinden Sie eine LED mit einem **`counter[?]`** der minimal schneller als mit 1 Hz blinkt und eine LED mit einem **`counter[?]`** der minimal langsamer blinkt als 1 Hz.
2) Verbinden Sie nun alle Bits von **`HW_led`** mit counter-Bits. Die Zuordnung sollte dabei so erfolgen, dass möglichst alle LEDs so blinken, dass ein Beobachter den Wechsel noch bzw. gerade noch wahrnehmen kann.
3) In einer vorherigen Übungen zeigten wir einen Integer auf 4 Stellen an. Dabei musste die anzuzeigende Stelle noch mit 2 Schaltern adressiert werden. Jetzt haben wir alle Mittel die vier 7-Segmentanzeigen durch die Bits unseres counters automatisch der Reihe nach durchzuschalten. Im Dokument „BASYS3 technische Informationen“ empfiehlt der Hersteller für die Refresh-Rate das Displays mindestens 60Hz und max. 1kHz zu wählen (die Reaktionszeiten der LEDs sind nicht bel. kurz).

© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Prozedurale Zuweisungen

Zusammenfassung

wire (Draht)

- steht 1x im Code und verdrahtet zwei Einheiten
- rein deklarativ – wird 1x verbaut. Existiert immer und zu jeder Zeit, ändert sich nicht mehr.
- Schlüsselwort: **`assign w = a | b;`** verdrahte a und b mit OR Gatter und nenne Ausgang w.
- Busse auch gut geeignet zur Deklaration von Konstanten.

reg (ein Speicher/Register)

- kann Wert anhängig von Ereignis ändern
- kann einem wire/output port zugewiesen werden. Beispiel: **`assign HW_led[0] = bit;`** kein Problem, LED zeigt Zustand von Register bit an. Register darf überall gelesen werden.
- Keine Zuweisung (setzen von reg) außerhalb von **`always@`**. Wenn ein reg dauerhaft den Zustand eines wires annehmen würde, dann bräuchte man keinen Speicher.
- Initialisierung geht innerhalb eines initial begin-end Blocks.

```
initial
begin
  bit = 0;
end
```


© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Prozedurale Zuweisungen

always@()

Typen von Triggern bei always@

Beispiel:

```
module my_bitmem_selfmade(
    input input,
    input enable,
    output bit
    );

    reg bit;

    always @(*)
    begin
        if(enable == true) bit = input;
    end

endmodule
```

trigger

Für **always@(*)** setzt verilog automatisch alle Netze (wire und reg) ein, die innerhalb von begin/end gelesen werden.

Hinweis: **if(...)** darf auch einem **begin/end** Block haben.

© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Prozedurale Zuweisungen

always@()

```
always @(„sensitivity list“)
begin
   …
end
```

Bei Eintreten eines Ereignisses in „sensitivity list“ wird die Schaltung zwischen begin und end aktiv. Typischerweise

```
always @(x)                     // wenn sich x ändert
always @(posedge x)             // bei steigender Flanke
                                // also x geht von 0 auf 1
always @(negedge x)             // bei fallender Flanke
                                // also x geht von 1 auf 0
always @(posedge clk or x)      // eines von beiden
always @(*)                     // immer
```


© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Prozedurale Zuweisungen

always@()

Folgendes ist möglich/nicht möglich:



```
reg [31:0]counter;

always @(posedge clk)
begin
  counter = counter + 1;
end

always @(posedge clk)
begin
  // bel. viele (parallele gleiche) alway@ möglich
  if(counter > 100) counter = 0;
end
```

counter kann nicht in mehreren **always@** Triggern zugewiesen werden. Warum? Weil eine Schaltung gebaut wird, die **counter** erzeugt. **counter** kann aber nicht an unterschiedlichen Stellen gleichzeitig gebaut werden.

© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Prozedurale Zuweisungen

always@()

Viele always@ sind üblich:

```
always @(posedge clk)
begin
  counter = counter + 1;
end

always @(posedge counter[9])
begin
  // nur jeder 2^9 Takt
end

always @(posedge btnC)
begin
  // center Button gedrückt
end
```


© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Prozedurale Zuweisungen

always@()

```
always @(posedge HW_switch[0])
begin
  // reagiere auf switch
end

reg last_switch;

always @(posedge clk)
begin
  if(last_switch == 0 &&
     HW_switch[0] == 1)
  begin
    // reagiere auf switch
  end
  last_switch = HW_switch[0];
end
```

Buttons und Switches könnten theoretisch auch Trigger sein – leider nicht auf unserem Board.

Was immer geht:

Auf eine steigende Flanke (posedge) der clk warten und dann einen Button/ Switch mit **`if`** abfragen.

© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

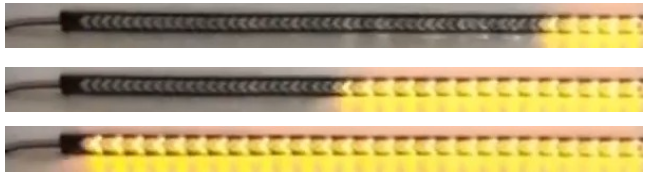
Praktikum
Wanduhr

1) Finden Sie nun eine Möglichkeit eine LED mit genau einem 1 Hz blinken zu lassen.
2) Vielleicht haben Sie das Problem innerhalb von always@ gelöst. Versuchen das Blinken der LED außerhalb von always@ zu realisieren. Der counter darf natürlich nach wie vor innerhalb von always@ inkrementiert werden.
3) Mit Ihrem Wissen können Sie jetzt leicht eine Digitaluhr implementieren. Diese sollte die Zeit im Format HHMM anzeigen.

   Für Fortgeschrittene: Zeigen Sie im Wechsel, z.B. in den geraden Sekunden die Zeit HHMM an, in den ungeraden die Zeit MMSS.

© Thomas Haenselmann, Prof. Dr. rer. nat. habil.

Praktikum
Blinker



Ihr Chef hat Sie beauftragt mit den 16 LEDs einen laufenden Blinker zu schalten so, wie er heute bei neueren Autos zu sehen ist.

1) Implementieren Sie ein solches Lauflicht in geeigneter Geschwindigkeit. Idealerweise nutzen Sie **`HW_switch`** für die Einstellung der Geschwindigkeit durch den Benutzer.
2) Vielleicht fällt Ihnen eine einfache Lösung ein, wie man die 16 LEDs für einen Links- und Rechtsblinker verwenden könnte, also die Laufrichtung umkehren.