Busse

Definition von Bussen

Bisher haben wir unsere Wires und Ports einzeln definiert.

```
module hello_world(
     input HW_switch_0, HW_switch_1, HW_switch_2,
     output HW_led_0, HW_led_1, HW_led_2
     );

     wire w2, w1, w0;
endmodule
```

Auf die Dauer macht die Indexschreibweise mehr Spaß. Sie funktioniert auch für die Ports im Kopf des Moduls. Dafür verwenden Sie bitte die neue advanced constraints Datei von der Webseite:

```
module hello_world(
     input [15:0]HW_switch,
     output [15:0]HW_led
     );

     wire [2:0]w;

endmodule
```


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

Definition von Bussen

Zuweisungen sind einzeln möglich ...

```
wire [2:0]w;

assign w[0] = 1;
assign w[1] = 0;
assign w[2] = 1;
```

... oder so wie Konstanten:

```
wire [7:0]N;

assign N = 127;
assign HW_led = 1023;
```

Ihre Schalterreihe ist eine praktische 16-Bit Konstante, mit der Sie einen von außen veränderbaren Integer in Ihre Schaltung einbauen können.

```
assign HW_led = HW_switch;
```

Datenwörter/Bitmuster

Literale*

Um dem Entwickler die Arbeit mit Literalen (direkten Konstanten wie Zahlen, Strings, true/false) leichter zu machen, sind die folgenden Konstrukte möglich:

```
0/1 : klar definiert als 0 oder 1
X   : Bit is egal
Z   : high impedance / hoher Widerstand = Leitung unverbunden
```

Schreibweisen von Konstanten in Verilog

```
#bits ' codierung{bodh} digit digit … digit
        (b=binary / o=octal / d=decimal / h=hex)
```

Beispiel:

```
8'b00110011  (binär)
8'b0011_0011 (underscore für bessere Lesbarkeit erlaubt, wird ignoriert)
8'b00XX_0011 (X = egal)
8'd0         (dezimal)
8'd1
8'd255
8'hFF        (hexadezimal/Bez. auch sedezimal)
```

*direkte Zeichen- oder Symbolfolge im Quelltext (im Gegensatz zur benannten Konstante, die einmal definiert und vielfach verwendet werden kann).

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

Verkettungen

Beispiele

```
wire [2:0] x;
wire y;
output [15:0]HW_led; // LSBit ganz rechts

assign HW_led = {4'b0101}     // Ausgabe 0101
assign HW_led = {3'b101,y}    // Ausgabe 101y
assign HW_led = {x,y}         // Ausgabe x[2-0]y
assign HW_led = {x,3'b101,y} // Ausgabe x[2-0]101y
```

// Wiederholung

```
assign HW_led = {x,3{y}} // Ausgabe x[2-0]yyy
```

// Beschränkung der Eingabe (Z = nicht def.)

```
assign HW_led[3:1] = {x,y} // Ausgabe Z..Zx[1-0]yZ
```

// Selektion der Eingabe

```
assign HW_led = {1'b1,y,x[2:1]} // Ausgabe Z..Z1yx[2-1]
```


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

Datenwörter/Bitmuster

Fallunterscheidungen ohne if (Multiplexer):

Generell kann man durch den „Trick“

```
wire [7:0] irgendwas;
wire [7:0] Erg1;
 ...

assign irgendwas = („bool Ausdruck 1“)*Erg1 +
                   („bool Ausdruck 2“)*Erg2 +
                   ...
                   („bool Ausdruck n“)*ErgN;
```

Unterschiedliche Ergebnisse durchschalten ohne dafür eine **`if`** Bedingung zu brauchen (der boolsche Ausdruck wird zu 0 oder 1 umgewandelt). Allerdings muss sichergestellt sein, dass nur eine Bedingung gleichzeitig wahr werden kann. Der Ausdruck implementiert einen Multiplexer, viele Eingänge auf einen Ausgang.

Beispiel für boolschen Ausdruck: **`(HW_switch[2:0] == 7)`**

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

Deklaration

Deklarationen von Bussen/Datenwörtern

```
wire [15:0] mein_bus        // LSB* rechts, MSB* links
wire [0:15] mein_bus        // umgekehrt
reg  [0:15] mein_bus        // auch mit reg

wire [15:0] mein_bus[3:0] // vier Busse à 16 Bit
```

das sind 16 Bit

das sind 4 Stück, durch-nummeriert von 0-3

*LSB/MSB : Least/Most significant bit

Beispiel:

```
wire [7:0] data[3:0];       // 4x ein Byte

assign data[0] = 127;       // 1. Byte = 127
assign data[1][7] = 1;      // setzt letztes Bit des 2. Bytes
assign data[HW_switch[1:0]] = HW_switch[9:2];
```


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

Generierung

Nützliche Schleifen zur Generierung von Code

Wiederholte Ausdrücke ...

```
wire [7:0] data [3:0]; // 4x ein Byte

assign data[0] = 0*10;
assign data[1] = 1*10;
assign data[2] = 2*10;
assign data[3] = 3*10;
```

können im Code mit einer genvar Variablen, generate und einer Schleife abgekürzt werden.

```
genvar i;
generate

   for(i = 0; i y ⩽ 3; i = i + 1) // i++ geht leider nicht
   begin
     assign data[i] = i*10;
   end
endgenerate
```

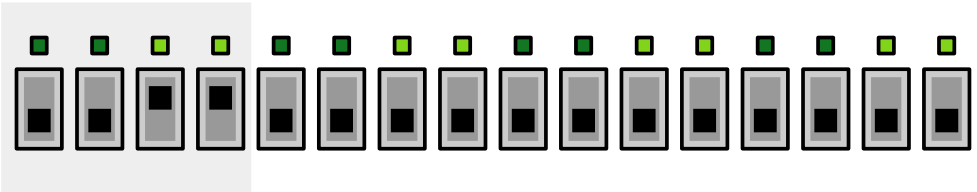
generate in Verbindung mit einer Schleife erzeugt keine Interation zur Laufzeit. Es erspart dem Programmierer lediglich Schreibarbeit.

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

D-Latch

1) Nutzen Sie die neuen Möglichkeiten, um die geraden LEDs 0, 2, 4, … 14 ein-, die ungeraten 1, 3, …, 15 auszuschalten. Definiert Sie das Muster z.B. in der Binärdarstellung wie drei Seiten zuvor beschrieben.

2) Kopieren Sie die Schalterzustände der 4 rechten Schalter 15 bis 12 4x in die LED Reihe. Das Bitmuster der vier Schalter sollte 4x hintereinander auf den LEDs sichtbar sein. Versuchen Sie dies möglichst in einer einzelnen Assign-Anweisung. Nutzen Sie dabei die Möglichkeit der „Selektion der Eingabe“ und der „Wiederholung“ auf der Folie 1.4 „Verkettungen“. Beispiel:



3) Kopieren Sie die 16 Zustände der Schalter in die LEDs, jedoch in umgekehrter Reihenfolge. Unterschiedliche Lösungen sind möglich.

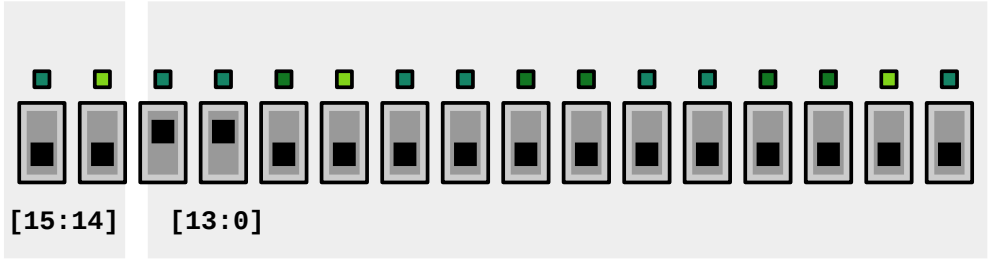
1) Verändern Sie das Beispiel (Datei **display_int.v**) dahingehend, dass man die vordefinierten Ziffern 9, 6, 4 und 8 mit den letzten beiden Switches HW_switch[15:14] wählen kann.

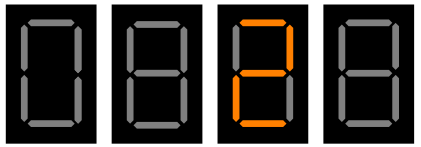
```
module display_int(
    output[15:0] HW_led,
    input[15:0] HW_switch,
    input btnC,
    output SSD_CA, SSD_CB, SSD_CC,SSD_CD, SSD_CE, SSD_CF, SSD_CG,
    output SSD_AN0, SSD_AN1, SSD_AN2, SSD_AN3
    );

    assign SSD_AN0 = 0;
    assign SSD_AN1 = 1;
    assign SSD_AN2 = 1;
    assign SSD_AN3 = 1;

    wire [3:0]digit[3:0]; // [3:0] vor digit bedeutet wir wollen 4-Bit Bus
                          // digit [3:0] (Klammer danach) bedeutet, 4 mal digit
    assign digit[0] = 9;
    assign digit[1] = 6;
    assign digit[2] = 4;
    assign digit[3] = 8;

    wire [3:0]final;
    assign final = digit[0]; // zeigt im Moment immer die 9 an

    seg7(final, SSD_CA, SSD_CB, SSD_CC, SSD_CD, SSD_CE, SSD_CF, SSD_CG);
endmodule
```

Praktikum
Datenwörter/Bitmuster

2) Nun soll je nach Schalterstellung auch die Anzeige angewählt sein, die zur jeweiligen Ziffer passt. Also die 9 ganze rechts, die 6 links daneben etc.

3) Jetzt interpretieren wir die ersten 14 Schalter von rechts HW_switch[13:0] als 14-Bit Integer und gehen davon aus, dass der Benutzer mit den Switches keine Zahlen > 9999 kodiert.





Erweitern Sie die Schaltung so, dass man die ersten vier Ziffern der 14-Bit Zahl ablesen kann.

Hinweis: Der Modulo Operator **%** ist Ihr Freund: **`(i % 10)`** liefert die 1. (least significant) Stelle im Dezimalsystem. Bsp.: **`12345 % 10 = 5`**