

Inhaltsverzeichnis

Anzeigesteuerung	2
Sensorsteuerung	4
Motorsteuerung	4
- Motorsteuerung Root-Funktion	6
- Feinabstimmung der Motorbewegung	6
Systemsteuerkreis	8

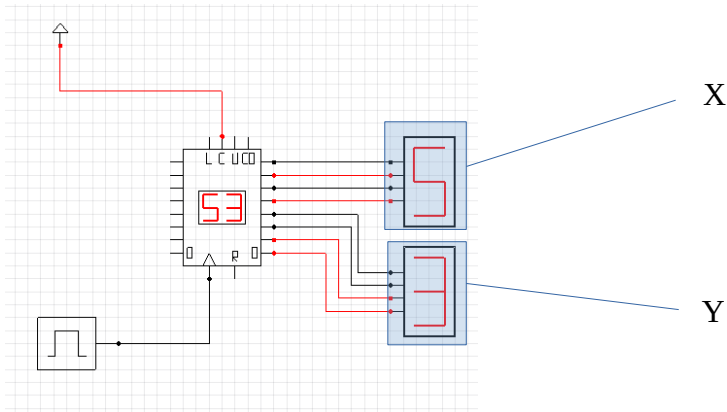
Hinweis:

Dieses Projekt wurde in Icarus Verilog erstellt, Funktionen wie `always_ff` / `comb` und andere Standard-Simulationsfunktionen sind nicht erforderlich, Die verwendete Plattform war Linux Debian für Simulation und Synthese. Ich habe auch versucht, die Eingaben der Module an die Moodle-Testbenches anzupassen, aber die alte Arbeitsweise von Icarus macht es schwierig, also habe ich einige von meinen beigefügt, nur als Beweis dafür, dass die Module wie beabsichtigt funktionieren

Anzeigesteuerung

Wie im Test erwähnt, haben wir zwei Anzeigemodule mit jeweils 16zehn Segmenten, also müssen wir uns nur um ein intelligentes Kabelmanagement und eine Kodierung kümmern. Um das Verständnis zu erleichtern, werde ich den Code mit einigen Grafiken subventionieren

Einfach ausgedrückt, alles, was wir tun müssen, ist, ein Display anzuschließen, um die erste Zahl der vierstelligen Zahl (X) anzuzeigen, und das andere, um die andere Ziffer (Y) anzuzeigen.



Das Übersetzen in Verilog gibt uns eine gewisse Flexibilität, da wir definieren können, welche Segmente bei welcher Nummer eingeschaltet werden, was bedeutet, dass wir die Zahlen für jedes Segment von 0 bis 9 fest codieren können, ausgenommen "Zahlen" von A bis F.

```
1 module display_if(input [3:0] bcd,enable_segment, output reg [15:0] seg);
2
3
4 //the block is executed only when any change in the input is detected
5
6 always @(bcd,enable_segment) begin
7
8
9 //the module will execute only when the enable is on
10
11 if(enable_segment) begin
12
13
14 //bcd is our number we want to display
15
16 case(bcd)
17
18 /*
19  each number has its own corresponding segment
20  from 1 to 9, the default simply turns the segments off untill further notice
21  */
22
23
24 0: seg = 16'b111111110000110;
25 1: seg = 16'b011000000000000;
26 2: seg = 16'b110111011100000;
27 3: seg = 16'b111110011000000;
28 4: seg = 16'b011000101100000;
29 5: seg = 16'b101110111100000;
30 6: seg = 16'b101111111100000;
31 7: seg = 16'b111000100000000;
32 8: seg = 16'b111111111100000;
33 9: seg = 16'b111110111100000;
34
35 default : seg = 16'b000000000000000;
36 endcase
37 end
38 end
39
40 endmodule
```

Indem wir nur einstellige Zahlen definieren, können wir die Buchstaben des Zählers effektiv skinnen, bis der Zähler später im Prüfstand wieder von 0 zurückgesetzt wird

der Prüfstand gibt an, welche Nummer angezeigt werden soll, Sie können sie gerne ändern. Wichtig zu beachten ist, dass das Register mit der anzuzeigenden Nummer auf beide Displays aufgeteilt ist, und angesichts der Tatsache, dass die Displays nur Zahlen darstellen können, ermöglicht uns dies effektiv, nur bis zu Nummer 99 anzuzeigen, im schlimmsten Fall ist das Display standardmäßig ausgeschaltet

```

1  `timescale 1ns/10ps
2
3  module display_if_tb;
4
5      reg [7:0] bcd = 0;
6      reg [3:0] enable_segment;
7      wire [15:0] seg;
8      wire [15:0] segtwo;
9
10
11
12      initial begin
13
14
15          $dumpfile("display_if_tb.vcd");
16          $dumpvars;
17
18          enable_segment = 1;
19
20          //number to be displayed
21          bcd = 2'd55;
22
23          #50 $stop;
24
25
26      end
27
28      //the bcd register array splits its connections between display 1 and 2
29
30      display_if display1(bcd[3:0],enable_segment,seg);
31      display_if display2(bcd[7:4],enable_segment,segtwo);
32
33      endmodule
34

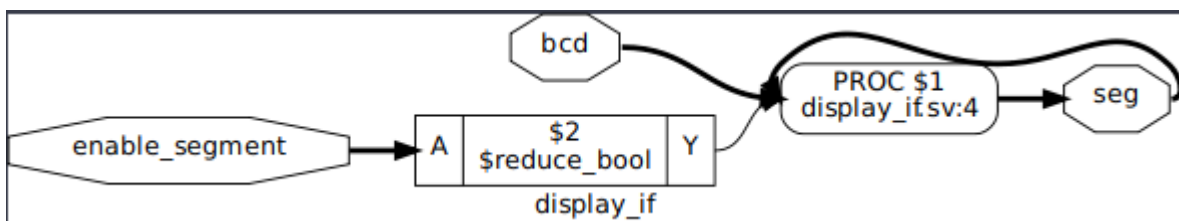
```

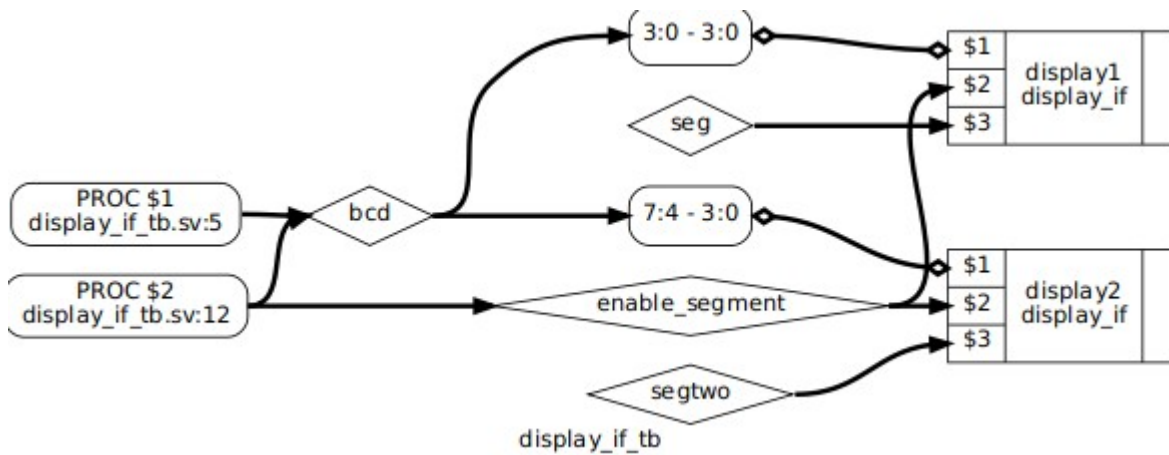
Hinweis: Um yoSyS für den TB zu verwenden, muss diese Codezeile kommentiert oder gelöscht werden, yoSyS problem

gtkwave



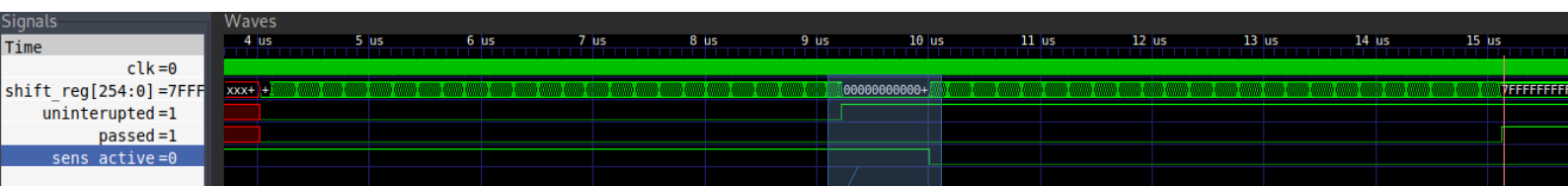
yoSyS





Sensorsteuerung

Durch Ausnutzen einer Schieberegister-Fortpflanzungsverzögerung können wir eine Zeit erzeugen, die eine bestimmte Anzahl von Zyklen wartet, Durch die Verwendung eines internen Schalters können wir feststellen, ob das Signal unterbrochen wurde oder nicht, und dann mit dem nächsten Ausführungsblock fortfahren. Der ununterbrochene Schalter wartet einfach darauf, dass der Sensor unterbrochen wird, bevor er fortfährt, was die lange Pause in der Wellenform erklärt, bevor der TB-Sensor deaktiviert wurde

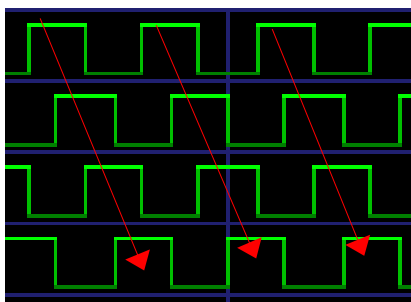


der unterbrechungsfreie Schalter pausiert die Ausführung bis auf weiteres (Sensordeaktivierung)

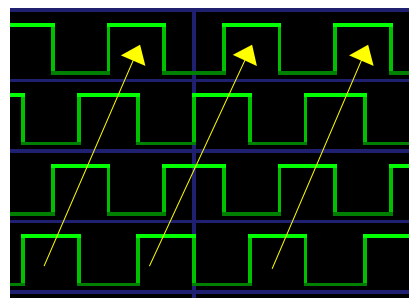
Motorsteuerung

Es gibt ein Muster für den Betrieb eines Schrittmotors. Um den Programmiervorgang zu vereinfachen, können wir uns eine Treppe oder eine Linie vorstellen, die je nach Richtung nach unten oder oben führt.

Forward



Backwards



Mit diesem Praktikanten können wir einfach eine Treppe aus fallenden oder steigenden Bits programmieren, die durch die Spulen geworfen werden und Viertelkreise erzeugen, und Viertelkreise zu erstellen, indem die Bits an der Position nach rechts oder links verschoben werden

```

if(enable && dir) begin

    case(p)

        0 : {A, B, C, D} = {1'd1,1'd0,1'd0,1'd1};
        1 : {A, B, C, D} = {1'd1,1'd1,1'd0,1'd0};
        2 : {A, B, C, D} = {1'd0,1'd1,1'd1,1'd0};
        3 : {A, B, C, D} = {1'd0,1'd0,1'd1,1'd1};

        default: {A, B, C, D} = {1'd0,1'd0,1'd0,1'd0};
    endcase
end

if(enable && ~dir) begin

    case(p)

        0 : {D, C, B, A} = {1'd1,1'd0,1'd0,1'd1};
        1 : {D, C, B, A} = {1'd1,1'd1,1'd0,1'd0};
        2 : {D, C, B, A} = {1'd0,1'd1,1'd1,1'd0};
        3 : {D, C, B, A} = {1'd0,1'd0,1'd1,1'd1};

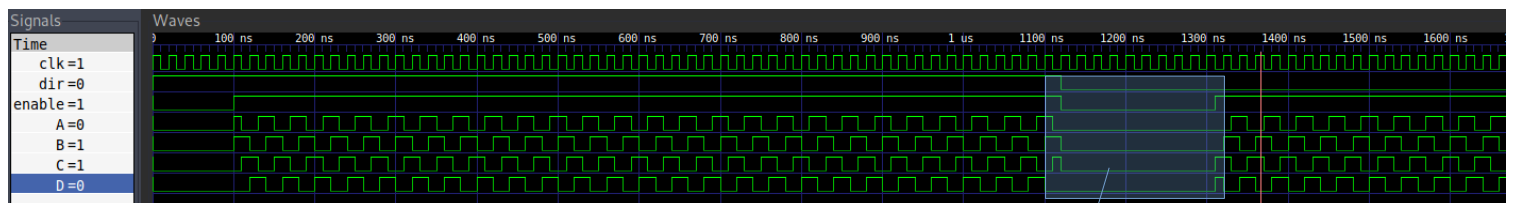
        default: {A, B, C, D} = {1'd0,1'd0,1'd0,1'd0};
    endcase
end

```

Forward

Backwards

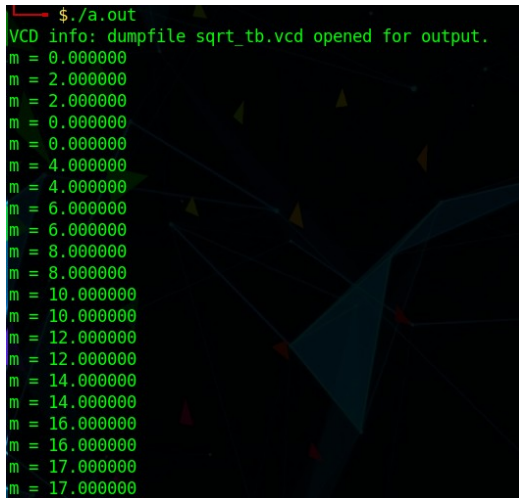
Wir ersetzen lediglich die Spulen von A bis D umgekehrt von D nach A und erhalten das gleiche Ergebnis



Umschalten von Vorwärtsfahrt auf Rückwärtsfahrt

Motorsteuerung Root-Funktion

Wie am Anfang dieses Artikels erwähnt, hat Icarus Verilog ein seltsames Verhalten und weigert sich, Gleitkommazahlen zu berechnen. Es war zu spät, jetzt etwas zu ändern, obwohl derselbe Code auf anderen Plattformen funktioniert.



```
./a.out
VCD info: dumpfile sqrt_tb.vcd opened for output.
m = 0.000000
m = 2.000000
m = 2.000000
m = 0.000000
m = 0.000000
m = 4.000000
m = 4.000000
m = 6.000000
m = 6.000000
m = 8.000000
m = 8.000000
m = 10.000000
m = 10.000000
m = 12.000000
m = 12.000000
m = 14.000000
m = 14.000000
m = 16.000000
m = 16.000000
m = 17.000000
m = 17.000000
```

Feinabstimmung der Motorbewegung

Um den Beschleunigungs- und Verzögerungsprozess zu vereinfachen, können wir eine Pulse width modulation (PWM) und das Konzept der Arbeitszyklen verwenden, um die Geschwindigkeit unserer Motoren zu manipulieren.

Indem wir manipulieren, mit welcher Frequenz der Motor läuft, können wir seine Geschwindigkeit direkt manipulieren, hier müssen wir den Abstand von mm zu Schritten berechnen, damit wir das Beschleunigungs- und Verzögerungsmuster leichter berechnen können

$$a = 1.8^\circ$$

$$a * 200 = 360 \quad \text{Wir wissen also, dass wir für eine ganze Umdrehung 200 Schritte benötigen}$$

Unter der Annahme, dass dieser Motor auf einem Standard-2-mm-Riemen mit Standard-Zähnezahl an der Polley verwendet wird, können wir etwas rechnen, um zu berechnen, wie viele Schritte wir für 1 mm benötigen.

Denken Sie daran, dass ich diese Berechnungen auf der Grundlage minimaler Standardkonfigurationen von Schrittmotoren durchführe, die fast überall verfügbar sind. Diese Methode wird auch zum Kalibrieren von 3D-Druckern und CNC-Maschinen verwendet.

Serv → Schritte pro Umdrehung

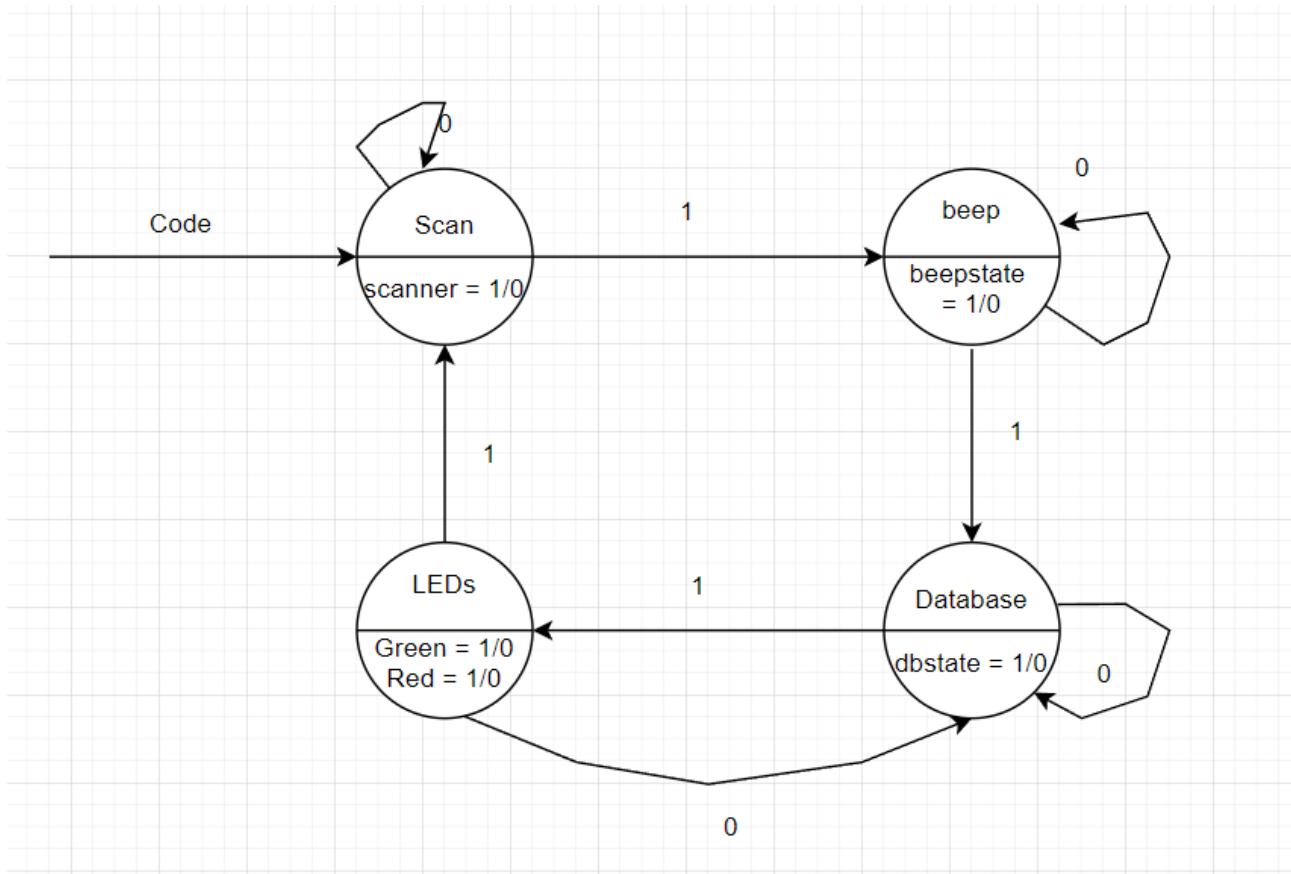
Fm → Mikroschrittfaktor *da wir nur an Geschwindigkeit interessiert sind, spielt Microstepping keine allzu große Rolle*

P → pitch

Nt → Anzahl der Zähne einer an der Motorwelle befestigten Riemenscheibe

Microstepping ist nichts anderes als eine asynchrone Art, Spulen zu erregen, so dass die Rotation nicht von einer Münze zur anderen geht, sondern dazwischen, da wir 4 Spulen und zwei Magnete haben (N rot / S grün) Der potenzielle Mikroschrittfaktor beträgt 8 ($4 * 2 = 8$).

Systemsteuerkreis



Die FSM ist in 4 Module unterteilt, 5, wenn wir den Decoder mitzählen, jeder Zustand hat seinen eigenen Übergang. beeper wartet 50 Zyklen, wenn der Scanner einen gültigen Scan hat, standardmäßig ist die Zustandsmaschine so eingestellt, dass sie bis auf weiteres in den Scan-Zustand zurückgesetzt wird. Der Übergang zwischen der Datenbank und der LED erfolgt in einer Schleife, die LED schaltet eines ihrer Lichter basierend auf dem vorherigen Zustand ein, bevor sie entweder zur Datenbank zurückkehrt oder zurückgesetzt und in den Scan-Zustand zurückkehrt, je nachdem, ob die Datenbank ob der Code gefunden wurde oder nicht, die Schleife wiederholt sich einfach

die LED blinkt solange rot, bis das FSM wieder den LED-Zustand erreicht hat, auch wenn das System neu gestartet wurde

jedes Modul entspricht dem Freigabesignal auf der rechten Seite

