Lab 3 "Programmable Integrated Circuits“

Three Dice at Lattice ispMACH4256ZE

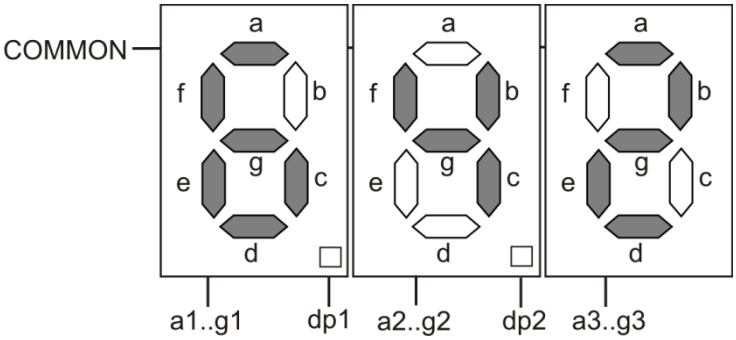
We want to implement 3 dice in an ispMACH400ZE-CPLD with 256 logic cells. The occurrence of the numbers 1 to 6 on every die shall be equally distributed and the results on the three dice shall be statistically independent.

We have a board (ispMACH4000ZE Pico Development Kit) with a 3-digit LCD-display, a push button (LOW-output when pressed, else HIGH) to use as RUN-button (roll the dice).

The Board uses the ispMACH4256ZE-5MN144C.The board has a clock generator (about 50 MHz). The clock may be divides by 1:1, 1k:1, 1M:1.

The 3-digit LCD-display shall show the three die values, when the RUN-button is released. When pressed the dice are running. The LCD display must be driven by an alternating voltage. The frequency should be 60.. 80 Hz.

There is one COMMON-input for all segments and separated inputs for the individual segments. To activate a segment (display a bar) there must be applied an alternating voltage between COMMON and SEGMENT inputs. An easy way to do so is to apply a clock CLK to COMMON and a signal CLK xor SEGMENT_VALUE to the SEGMENT input. For inactive segments (no bar) CLK will be applied both to COMMON and SEGMENT inputs.



Common Pin PM2					
Links (1)		Mitte (2)		Rechts(3)	
Segment	Pin	Segment	Pin	Segment	Pin
a1	PB1	a2	PE2	a3	PB6
b1	PC3	b2	PF2	b3	PA6
c1	PC2	c2	PM1	c3	PC6
d1	PC1	d2	PF1	d3	PB5
e1	PD1	e2	PF3	e3	PA5
f1	PD2	f2	PG1	f3	PA4
g1	PD3	g2	PE3	g3	PB4
dp1	PB2	dp2	PE1		

The synthesis program expects the Letter "P" in front of the pin name from the data sheet.

The Button is located at pin PH11. For the button input the CPLDs internal pulldown resistor must become deactivated. It is activated by default!
The pulldown in the CPLD pulls about 30µA. The external pullup at the button has a value of 100 kΩ. So there will never be a valid logic high level at the button output as long as the pulldown in the CPLD is active. Pulldown can be deactivated using the Constraint-Editor. When we generate the JEDEC-File we must select “Merge” for the constraint import.

The oscillator is included as a library module in the verilog description.
Declaration:

```
module osctimer(DYNOSCDIS, TIMERRES, OSCOUT, TIMEROUT);

parameter TIMER_DIV = “128”;
input DYNOSCDIS;
input TIMERRES;
output OSCOUT;
output TIMEROUT;
endmodule
```

Instantiation:

```
defparam I1.TIMER_DIV = “1024”,

osctimer I1 (.DYNOSCDIS(1'b0), .TIMERRES(resettimer),
             .OSCOUT(CLK_5M), .TIMEROUT(CLK_5K));
```

"resettimer“ is an H-active reset signal.

CLK_5M und CLK_5K should be replaced by the names used for the 5 MHz- and 5 KHz- clocks respectively. I1 is the user-defined instance name of the oscillator.

In the Lab we use ispLever Classic (Development) and ispVM(Configuration over USB) as software.Details will be discussed in the class.

Information sources:

Software: http://www.latticesemi.com/dynamic/view_document.cfm?document_id=47330

Demoboard:
http://www.latticesemi.com/products/developmenthardware/developmentkits/ispmach4000zepicodevkit.cfm

CPLD: http://www.latticesemi.com/dynamic/view_document.cfm?document_id=29095

HInts for the "Dice running"-Funktion:

To implement statistically independent run of the dice is challenge. We do not have a real random process here (noise). As a backdoor we can use a large pseudo random number (PRN). If we have a good algorithm for a large PRN we can consider parts of the large PRN as independent short PRNs.

So we can use parts (slices) of 6 Bits from a 32 Bit PRN. Unfortunately we can gather only numbers in ranges of 2^n and nothing that is divisible by 6 to get 6 equally distributed ranges.

If we have numbers between 0 and 63 (using 6 Bits) it would be possible to choose values from 0 to 59 (6*10) or from 0 to 65 (6*11). If we do so once for 6*10 and twice for 6*11 in three count turns than we have equally distribution for 1st to 6th interval.

You need an additional little counter from 0 to 2 to implement this scheme.

Better idea?

Possible polynomial for 32-Bit-PRN:
x31 + x28 + x27 + x26 + x20 + x18 + x17 + x15 + x11 + x10 + x9 + x8 + x5 + x4 + 1

You can generate the PRN from hardware using this polynomial by a shift register with feedbacks: