1- Hi!

2- In this last presentation we will look into the functional requirements prepared to guide the development of our second coursework assignment: an e-dice based on a finite-state machine.

3- After watching this presentation I hope that you will be able to explain the functional requirements of the e-dice using your own words, and to plan the tasks needed to implement a solution using Vivado.

4- Accordingly this presentation comprises the two parts shown in this slide.

5- Let's start by the requirements.

6- The new functional requirements expand what we considered in the previous assignment, since we now want to consider three cheating modes instead of just one. The overview representation shows that we now need two input switches to specify the operating mode, which is no longer restricted to triple probability or normal.

7- The three cheating modes comprise the previous triple probability mode, plus a predefined result mode, and a forbidden result mode. The 3-bit result selection input doesn't matter if the operating mode is normal, in which case all results are equally probable. And the "run" input continues to work as before: press it to throw the dice, and release it to stop.

8- These requirements can be summarised by the function table shown in this slide. Notice that for reasons of space and simplicity it doesn't represent a complete specification, since you have the freedom to decide what should happen when the result selection inputs specify an illegal combination for cheating.

9- Notice also that the former state diagram considered in the case of the counter-based implementation can still be used for the case of triple probability cheating, but it lacks information for the other two cheating modes. You can however use it as a starting point, and figure out how to expand it…

10- So what tasks do you have ahead?

11- You should start by realising that the state diagram presented earlier was predefined by the counter function, and we had limited options to customise it. That's not the case anymore, since the distinctive feature of finite state machines is precisely that you are entirely free to design the state diagram determining the circuit operation.

12- You may however start from the counter-based diagram shown earlier, and work to expand it by adding the two additional cheating modes.

13- Once you are confident that your state diagram works, you may create your VHDL descriptions. This is a fairly simple task, since you may just edit a previous finite state machine project and adapt the code. It's also simpler than the counter-based implementation in the sense that it is a flat description, instead of a hierarchical description comprising a top-level and sub-levels. Once you synthesise your code without errors, you'll be ready to continue to simulation.

14- Design verification calls for a simulation source file, but again you can create it just be editing another simulation file. Make sure that you set up a reasonable variety of operating conditions, comprising the normal mode and all three cheating modes, and likewise different result selection combinations for various cases. This may take a while and involve various simulation sessions, but it is not complex at all, just... slow. What if you wanted an exhaustive design verification procedure?

15- Once you're happy with what you see in the waveforms, move on to implementation. Edit the constraints file of the counter-based e-dice to take into account the additional inputs, and generate the bitstream to program the FPGA. You're now ready to try it out in the Basys-3 board, and that should be fun!

16- And this is the end of the finite state machine module, I hope that you may have enjoyed it. Thanks for your attention!