How to generate a .xdc file (Updated and expanded)

Created by: David Rubio G.

Blog post: https://soceame.wordpress.com/2025/03/09/how-to-generate-a-xdc-file-updated-and-expanded/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Last modification date: 09/03/25

OS33, LVCMOS18, etc) and another in which the pin of the board was assigned. Well, there is another more **practical** method of doing both things in a single line, which makes it easier to create .xdc files in which y*ou only have to change the name of the signal that comes by default*.

The system is very simple, instead of writing:

```
set_property IOSTANDAR [get_ports ];
set_property PACKAGE_PIN [get_ports ];
```

You have to write:

```
set_property -dict { PACKAGE_PIN IOSTANDARD } [get_ports ];
```

Example

To better understand how to make this new .xdc, we are going to optimize the example from the other time.

In the previous example, there were two ports, dos_bits[0] and dos_bits[1], and the following four lines were used for this.

```
set_property IOSTANDAR LVCMOS33 [get_ports {dos_bits[0]}]; #pin 7
set_property IOSTANDAR LVCMOS33 [get_ports {dos_bits[1]}]; #pin 8

set_property PACKAGE_PIN K18 [get_ports {dos_bits[0]}]; #pin 7
set_property PACKAGE_PIN K17 [get_ports {dos_bits[1]}]; #pin 8
```

With the new XDC format it would be as follows:

```
set_property -dict { PACKAGE_PIN K18 IOSTANDARD LVCMOS33 } [get_ports
{dos_bits[0]}];
set_property -dict { PACKAGE_PIN K19 IOSTANDARD LVCMOS33 } [get_ports
{dos_bits[1]}];
```

With this format you save one line per port, so any pin modification is easier, and it also allows you to generate generic XDC files that allow you to have a single XDC to avoid having to go to the datasheet continuously to find out which pins you have to touch.

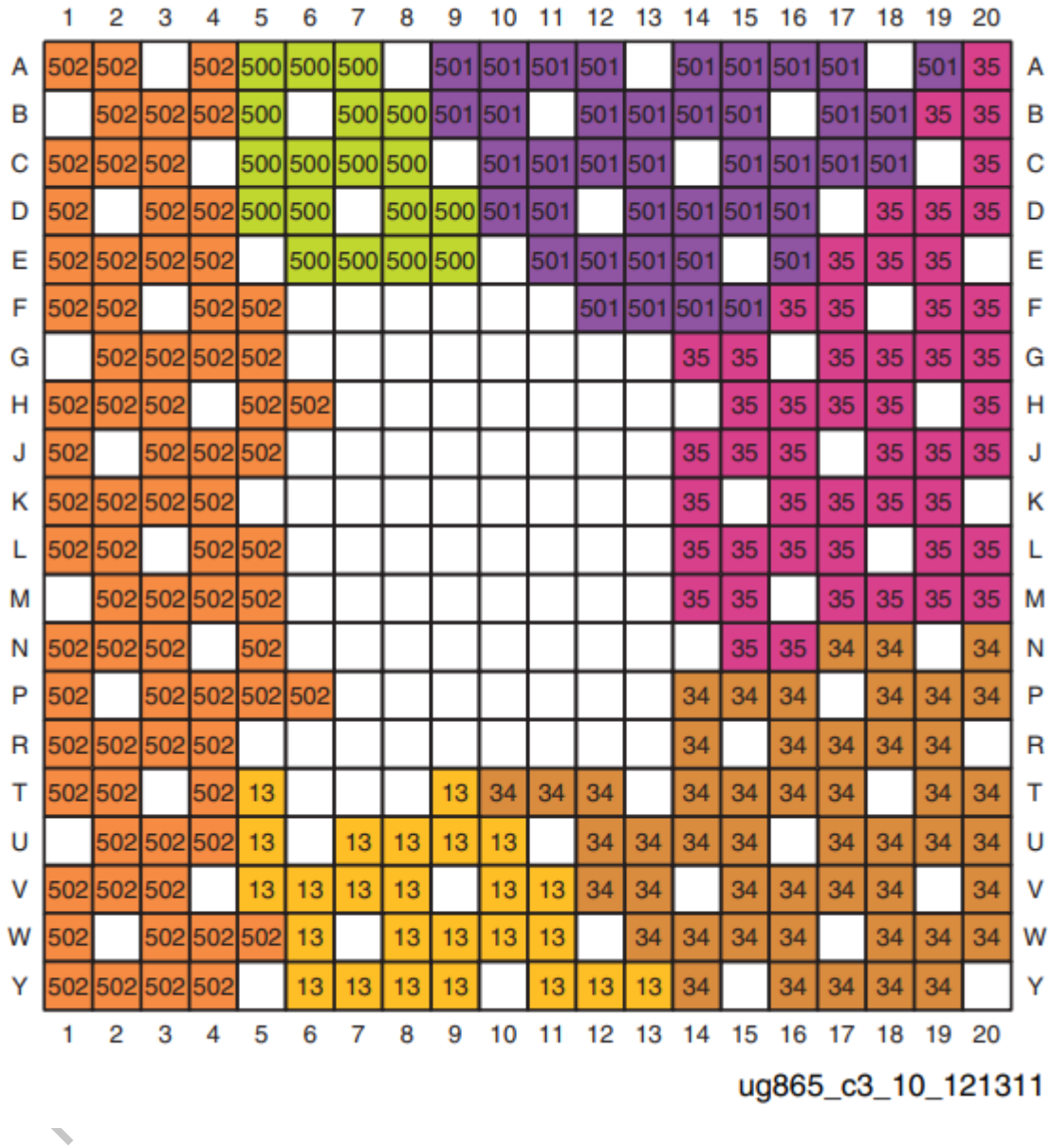
Here are the two XDC formats for a QMTECH board so that the differences can be seen more clearly.

- Old version
- New version (this version is extremely generic, so it is easier to change the pins)

Another version of making an XDC. Putting the entire bank of pins to the same voltage

Another way of making XDCs is to put the entire bank of pins to the same power bank. To understand this you have to know that the chip pins are grouped in banks, for example, in a

Zynq-7000 it has all the RAM pins in the same bank and the output pins are divided into two different banks. *[Attached image of an XC7Z010-CLG400]*



In the image you can see all the pin groups **Note: the missing pins are non-connected pins (NC), power supplies, grounds, etc. **

Well, these groups can be put to the same voltage, to do this you resort to what was explained in the previous version, but instead of putting each pin to a voltage you put the whole bank to the same voltage. Everything follows the following scheme

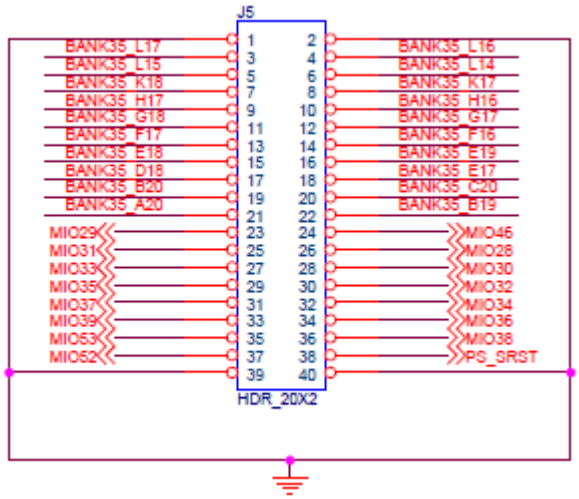
```
set_property PACKAGE_PIN <pin name>[get_ports <port name>];

set_property IOSTANDAR  <power bank>[get_ports -of_objects [get_iobanks <pin bank number>]];
```

The first part is the same way of putting an output pin to a port. The second part is the one that puts the whole bank to the same power bank.

Example

The example is the same one that has been used previously, but rescuing the image of the datasheet, why? because that is where the bank of pins to which the output pins belong appears.



As you can see in the image, pins K17 and K18 belong to bank 35, so the .xdc would be as follows.

```
set_property PACKAGE_PIN K18 [get_ports {dos_bits[0]}]; #pin 7
set_property PACKAGE_PIN K17 [get_ports {dos_bits[1]}]; #pin 8

set_property IOSTANDAR LVCMOS33 [get_ports -of_objects [get_iobanks 35]
]; #all bank pins at 3.3V
```

How to generate a clock on an xdc for an FPGA

To generate a clock you need several things, the first is to know which pin the oscillator on the board is connected to, for example, this pin on a Nexys 4 DDR is a MRCC type pin with the name E9. [*Nexys 4 DDR Datasheet*]. Another thing you need to know is the clock period, the period is the inverse of the frequency ($1/f_{clk}$). With all this, the clock can now be generated on the XDC.

The clock on the XDC has the following structure:

```
set_property -dict { PACKAGE_PIN <clock pin>  IOSTANDARD  <power bank>}
[get_ports <clock input port>];

create_clock -period <clock period in ns> [get_ports <clock input port>];
```

The first part is the declaration of the clock input pin. (Here it is used to declare the version explained at the top of this entry). The second part is when the clock is created. In it, the clock period is configured (by default it has a 50% cycle) and the name of the clock port.

Example

In this example we are going to create the clock for a Nexys 4 DDR with a 10 ns period, with the clock input port called «CLK»

To start, we must first locate which pin the oscillator on the board goes to
Nexys 4 DDR oscillator

As you can see, it goes to the E3 pin.

Once we know the name of the pin, the port and the period, we create the .xdc. The LVCMOS33 is used as the power supply.

```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports CLK];

create_clock -period 10.00 [get_ports CLK];
```

With this, we could generate the input clock.

Other configurations

To create a clock you can also configure the duty cycle of the clock signal and give it a name. For example, if in the previous example you wanted a clock with a duty cycle of 70% and a name like "clock_signal" you would only have to add the following parameters:

```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports CLK];

create_clock -add -name clock_signal -period 10.00 -waveform {0 7}[get_ports
CLK];
```

The duty cycle can be configured in other ways: {1 8}, {2 9}, etc. Always remember that the eligible duty cycle is in *ns* **(NOT in percentage)**, or what is the same in the example «{1 8}» is a ‘0’ up to 1ns, ‘1’ from 1ns to 8ns and then ‘0’ up to 10ns.

For more information on XDCs, see *the official Xilinx documentation.*