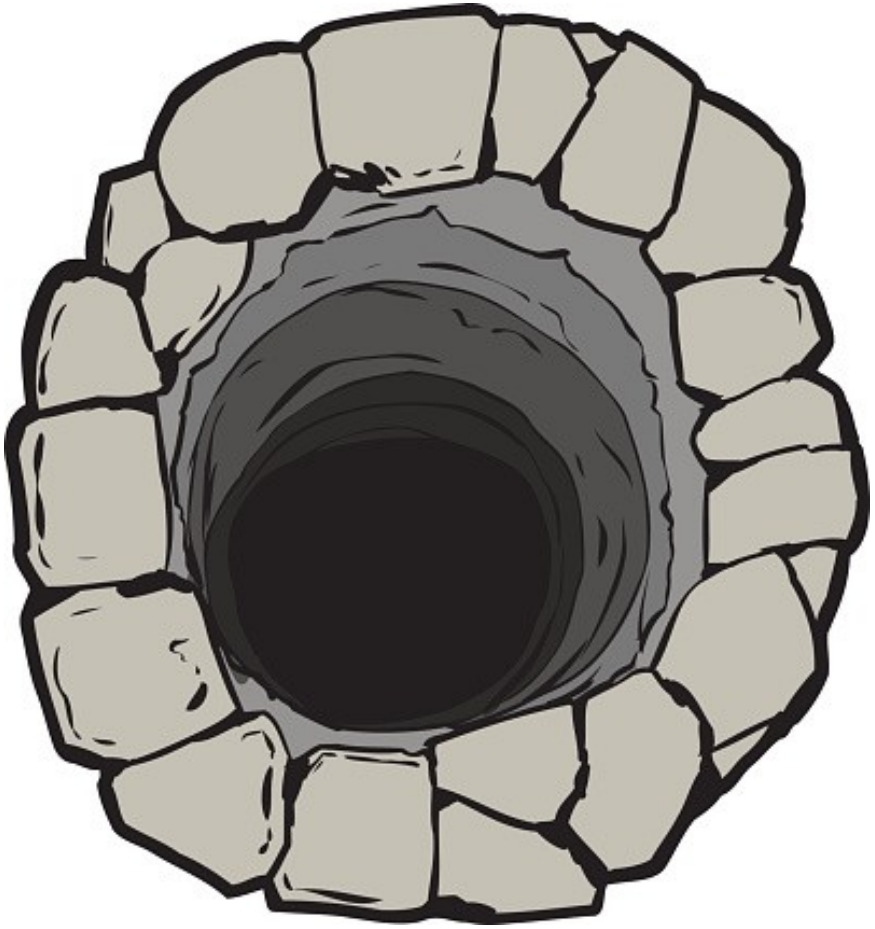
New Design: Top Down Approach



by Iain Waugh

Overview

- You've got a new dev board... and...
- You want to start a design from scratch
- The FPGA pins are connected to things
- The aim is to program the FPGA with a blank design and not have half-lit LEDs or loud buzzers going off
- i.e. outputs are driven to "safe" levels

Sample Code

- All sample code is here:

  https://github.com/iain-waugh/fpga_dev_board

- You are free to clone and use it.
- Note the "hyphen"

Start with Pin Constraints

- Your dev board will have a constraints file mapping FPGA pins to signal names
- Find it and take a look
  - Xilinx ISE : ".ucf"
  - Xilinx Vivado : ".xdc"
  - Intel Quartus : ".qsf"
  - Lattice Diamond : ".lpf"
- (hereafter, just called the ".ucf" or ".xdc")

Start with Pin Constraints

- Copy that file into your project and commit it
- I like to put it under:
  `syn/<project name>/<fpga part number>.ucf`
- Tidy it:
  - Most constraints files will be arranged in groups of lines per interface
    (RAMs, LEDs, peripherals, etc.)
  - If they aren't, then get the board schematic and spend some time grouping them

Add Prefixes and Suffixes

(a personal preference)

- This is a simple convention that helps A LOT
- Prefixes:
  - "`i_`" = input
  - "`o_`" = output
  - ** "`io_`" = bi-directional
- ** = only exists at the chip's top level

Add Prefixes and Suffixes

- Suffixes:
  - ** "_p" = part of diff pair: positive
  - ** "_n" = part of diff pair: negative
  - "_n" = an active low signal
- ** = usually only exists at the chip's top level

Clarify Naming

- Some pin names and groups may not make sense
- The HDL code names must match these names
- Add comments for each group
- Add comments for any special signal functions
- Now is the best time to re-name things for HDL clarity (before revision control)

Create Top Level Entity

- Take every signal name from the ".ucf" or ".xdc" file and put it on a line of its own; you now have a file of just the pin names
- Convert everything to lower-case (if that's your HDL coding convention)
- Convert busses to std_logic_vector with a range
- ie. i_data0, i_data1 becomes

```
i_data : in std_logic_vector(1 downto 0)
```

Set Output Defaults

- For every "io_" signal on the entity, create:
  - A local "i_" version
  - A local "o_" version
  - A local "_out" version (for input/output control)
- In the architecture section, assign a default value to every "o_" signal and create the tristate like this:

```
o_cam_sclk  <= '0';
io_cam_sdat <= o_cam_sdat when cam_sdat_out = '1' else 'Z';
i_cam_sdat  <= io_cam_sdat;
```

Try Synthesis, P&R

- It sounds dumb to run on a blank design, but it shows you:
  - Syntax errors
  - Possible pin voltage level errors
    - Assigning +3.3v and +2.5v to the same bank, for example
- Lets you get an FPGA image made with minimal fuss.
- If you got the output signal levels correct, you should not have any LEDs lit.

Try Synthesis, P&R

- If things don't work, look for errors or warnings in the synthesis report:
- ISE: ".srp" file
- Vivado: ".log" file

Add Some Common Functions

- Utilities package
  (common types and handy functions)
- Clock sync/metastable hardening
- Delay logic
- Mechanical switch debounce circuit

Utilities Package

- Because VHDL is easier with some extras

```
-- Maths functions
function clog2(x : natural) return natural;

function maximum(x, y : std_logic_vector) return std_logic_vector;
function minimum(x, y : std_logic_vector) return std_logic_vector;
function maximum(x, y : integer) return integer;
function minimum(x, y : integer) return integer;

-- Bit ordering functions
function swap_bit(x  : std_logic_vector) return std_logic_vector;
function swap_byte(x : std_logic_vector) return std_logic_vector;

-- Misc functions
function all_ones(x  : natural) return std_logic_vector;
function all_ones(x  : unsigned) return unsigned;
function all_zeros(x : natural) return std_logic_vector;
function all_zeros(x : unsigned) return unsigned;

-- Type conversion functions
function to_integer(x   : std_logic) return integer;
function to_std_logic(x : boolean) return std_logic;

-- Logic functions
function and_reduce(x  : std_logic_vector) return std_logic;
function nand_reduce(x : std_logic_vector) return std_logic;
function or_reduce(x   : std_logic_vector) return std_logic;
function nor_reduce(x  : std_logic_vector) return std_logic;
function xor_reduce(x  : std_logic_vector) return std_logic;
function xnor_reduce(x : std_logic_vector) return std_logic;
```

Clock Sync/ Metastable Hardening

- So common:

  Do it right once

  Use it everywhere

- This version is Xilinx-specific (see the attribute)

```
library ieee;
use ieee.std_logic_1164.all;

entity sync_sl is
  port(
    clk : in std_logic;

    i_sig : in  std_logic;
    o_sig : out std_logic
    );
end sync_sl;

architecture sync_sl_rtl of sync_sl is

  signal sig_ss : std_logic := '0';     -- Semi-Synchronised (don't use this)
  signal sig    : std_logic := '0';

  -- Xilinx special attribute to pack 2x FFs right next to each other
  -- with minimal routing delay
  attribute ASYNC_REG : string;
  attribute ASYNC_REG of sig_ss: signal is "TRUE";

begin  -- debounce_rtl

  ---------------------------------------------------------------------------
  -- Make a metastable-hardened version of the input that's safe to use
  process (clk)
  begin
    if (rising_edge(clk)) then
      sig_ss <= i_sig;
      sig    <= sig_ss;
    end if;
  end process;
  o_sig <= sig;

end sync_sl_rtl;
```

Delay Logic

- What's special about a delay chain..?
- This implementation defines localised signals in a generate statement.
- This prevents synthesis warnings for unused signals when C_DELAY > 0.
- Can't get rid of the warning when C_DELAY = 0

Mechanical Switch Debounce

- Because push buttons are not simply on/off
- Output changes only after ~1ms of stability
- 40us/div, here

Questions?