Dear All,

This document, which is centered on **HDL optimization** with a focus on power, area, timing, etc., has been authored by **Ali Haji Ebrahim Zargar** and will be finalized over time. If you have any suggestions or are aware of alternative methods to enhance this list, please feel free to share them with me.

Thank you,

PHONE:
+989108846298

LINKEDIN:
https://www.linkedin.com/in/ali-h-zargar1993/

EMAIL:
alizargar1372@gmail.com

GitHub:
https://github.com/Ali-Hajiebrahim-Zargar

#	Des.
Power Optimization	
1	Reducing the operating frequency of modules to the minimum possible state is essential. For example, efforts should always be made to minimize the frequency of PLL outputs so that the overall performance of our circuit is not disrupted.
2	Preferably, blocks and resources should be placed next to each other in floor planning. This leads to a reduction in Place and Route time. In pipeline design, due to its architectural nature, it is better for modules to be placed next to each other in an FPGA. This not only shortens the signal path but also reduces Place & Route time.
3	Eliminating input pins as much as possible through time multiplexing is advisable. For example, if there are inputs with a specified time difference for data arrival at the FPGA, it is better to consider common pins and modify the code in a way that, with a control circuit, inputs can be distinguished from each other.
4	Using clock-gating flip-flops in some technologies. Some technologies have designed this capability for each flip-flop so that we can have the ability to reduce power consumption. This allows us to disable the clock when the FPGA is not actively working or in an idle state, thereby conserving power.
5	Eliminating glitches using the mentioned techniques (AND GATE, PIPELINING, Using Enable). However, when utilizing Handshaking in a Pipeline architecture, issues arising from glitches are no longer created.
6	A binary counter consumes less power than a Gray counter. However, there is no obligation to use a Gray counter.
7	Among multiplier blocks, using Non-Booth Encoding Wallace consumes less power compared to others.
8	Among adder blocks, Brent & Kung adder consumes less power compared to others.
9	In a scenario where a frequency lower than the external clock of the FPGA is required, it is better to use a frequency divider circuit instead of a PLL to reduce the frequency.
10	In space designs, it is recommended not to use SRAM-based FPGAs. Preferably, FPGA devices with Flash-based technology should be utilized.
11	Using Flash-freezing technology in the design (available as an IP core in Actel products).
12	Using Power-driven Layout in Place and Route. (Libero)
13	Using I/O technologies with lower voltage and current if possible. This affects the efficiency and speed of the circuit. For example, in our designs, we can opt for modules that operate at lower voltages.
14	For Single_ended pins, use LVCMOS technology.
15	For differential pins, use LVDS or LVPECL technologies.
Area Optimization	
1	Using controllers to utilize one module instead of multiple modules. This is effective when the control circuit occupies less space compared to the modules. For example, if you have used the same computation module multiple times in a code, it is better to use only one module and, if possible, select inputs in a control circuit and feed them to the module.
2	In our modules, we should look for blocks that are common with other modules. Then, isolate these blocks and bring them into the TOP module so that we can use them in multiple functions and other modules.
3	For blocks that are inherently part of IP cores, it is necessary to use them as provided by default, rather than coding them ourselves. This leads to increased speed and reduced occupancy space.

4	Not unrolling modules such as multipliers. While unrolling increases the efficiency and throughput of a system, it also leads to increased occupancy space.
5	Using Resource Sharing in the Synplify software or employing appropriate attributes in each module for the purpose of resource sharing or not. If the desired arithmetic block is critical in the timing path, that block needs to be disabled for sharing using syn_sharing.
6	Determining the type of Reset and/or Set in modules based on the type of resources. Whenever possible, use Reset based on the type of block in each FPGA. It is also necessary to specify the Reset logic based on the type of block in the FPGA. In general, efforts should be made to make the Reset and/or Set logic in IP core modules consistent with what exists in the FPGA.
Timing optimization	
1	Designing for high throughput by increasing the data transfer rate (transitioning from 8-bit architecture to higher architectures). For example, when using a 64-bit architecture, the processing volume per clock cycle increases, leading to higher computational speeds. However, this approach comes with an increase in occupancy space and power consumption.
2	If timing issues arise, consider dividing the module into smaller modules if possible. However, this might increase latency, as each module, when written in a standard way, is registered at the beginning and end of its path. The estimated time in modules with fewer Core Cells will be lower.
3	If the critical path is within a multiple path structure, it is possible to reduce the critical path by changing priorities. For example, based on the hardware structure for If...else conditions, placing the critical signal in higher-level conditions within the If statement.
4	To increase throughput in a circuit, it is recommended to use Unrolled loops. Unrolled loops refer to repeated loops that replicate a block along with optimizations.
5	In situations where the desired latency of the circuit is not met, aggregating modules can be used if possible. However, this increases the Register to Register delay. In other words, it is possible to make the Pipeline design more compressed. In this case, gate delays become predominant, and care should be taken not to face issues with the estimated frequency.
6	As much as possible, avoid combining critical signals with other signals. In other words, try to keep critical signals separate from other signals and do not pass them through a logical gate together. This greatly helps in improving the timing of the circuit.
7	If you want to improve the timing of the code, you can enable the retiming option in Synplify. Additionally, you can use appropriate attributes to specify whether or not to perform retiming for modules (e.g., syn_allow_retiming).
8	As much as possible, avoid using prioritized conditions in conditional statements. Increasing the priority conditions will result in the addition of serial multiplexing gates. In other words, if conditions have equal priority, use a Case statement instead of an If statement.
9	In the modularization of blocks, all inputs and outputs of the module that are pipeline communication interfaces should be registered. By registering these interfaces, the distance between modules in floorplanning will no longer pose a timing issue.
Avoid CCD & Metastability	
1	Using DLL (Delay-Locked Loop) to obtain feedback from the first clock when one of the clocks has a different clock frequency multiplier. It should be noted that this block is not available in all FPGAs; therefore, its usage depends on the specific FPGA you are working with.
2	Using FIFO (First-In-First-Out) when you need to transfer multiple bits of a signal between non-synchronous clocks is recommended.

3	Registers used for synchronizing the circuit need to be defined outside the modules. In other words, flip-flops used in Double flopping should be defined outside the modules to ensure their creation and be used as ready-made macros.
4	Using Handshaking in the presence of multiple clock domains.
Floor Planning	
1	When the design is implemented in a pipeline fashion, selecting an appropriate partition for the implementation of blocks can not only reduce delays but also decrease synthesis time.
2	Modules that function to control other modules or serve as glue logic, especially those without a specific data path, may not be suitable for floorplanning. On the contrary, modules with clear data paths are suitable. Note that modules with high fanout are also suitable for floorplanning.
3	In floorplanning design, it is necessary to consider the placement of fixed cores such as RAMs or counters and design accordingly to accommodate them.