Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática
Maestría en Diseño Electrónico



Silicon Prototyping of Intel's Fully Integrated Cryo-CMOS SoC for State Manipulation, Readout and High-Speed Gate Pulsing of Spin Qubits.

TESIS que para obtener el **GRADO** de
MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: **FÉLIX III BRANDO PÉREZ ESPARZA**

Director: **ESDRAS JUAREZ HERNANDEZ**

Tlaquepaque, Jalisco. 4 de agosto de 2026

Quisiera agradecer, antes que nadie, a Dios por el tiempo y la vida que me dio para alcanzar este logro, y confiar en que me seguirá cuidando y prestando vida para alcanzar más metas. Agradezco a mis padres el esfuerzo que pusieron en mi educación y por ser mi ejemplo de éxitos a seguir siempre. Gracias a mi esposa y a mis hijos, que son el motor que me impulsó a terminar este trabajo después de años de abandono, espero ser un ejemplo para ustedes, así como mis padres lo son para mí. Agradezco también a mis compañeros de trabajo, sin los cuales no habría sido posible realizar este proyecto tan importante en especial al Phd. Stefano Pellerano por su apoyo para poder lograr la autorización de la empresa para poder publicar este documento.

Summary

The purpose of this work is to document the most common problems encountered while implementing the layout of digital logic for Intel's Horse Ridge II Cryo-CMOS SoC for Qubit control using Automatic Place and Route (APR) techniques and the solutions implemented for this design in the hope that it will serve as a guide to the novice SoC design engineer while implementing layout using APR tools and methodologies. The designs and techniques presented in this document are the result of the work experience acquired while working for Intel Labs' Silicon Prototyping Team and represent the most notable problems found while implementing the Fully Integrated Cryo-CMOS SoC for state manipulation, readout, and high-speed gate pulsing of spin qubits Horse Ridge II), for which an IEEE publication is available from 2021 in the IEEE Journal of Solid-State Circuits. Although the IEEE paper describes the design implementation and the results from testing the fully integrated design, it does not describe the digital layout implementation or detail the challenges and solutions the team had to address to finalize the design. This document attempts to preserve the hard work of Intel's silicon prototyping team in implementing the digital logic that interfaces with the analog portion of the design, as well as the full chip assembly flow for an Intel process. The problems and solutions documented here are presented in a way that will be helpful to novice SoC designers who encounter similar issues. Chapter 1 provides an overview of the project and details the implementation requirements we began with for the digital design. In Chapter 2, we cover the preparation of the collaterals required to implement the design, the full chip floor planning, partitioning, and partition implementation results, as well as the full chip assembly methodology. Chapter 3 details the methodology for timing, layout rules convergence, layout-versus-schematic (LVS) checks, and formal verification of the full chip against the final RTL model received from the RTL team.

Content

Summary **vii**

Content **vii**

Introduction **1**

1. The Fully Integrated Cryo-CMOS SoC for state manipulation, readout, and high-speed gate pulsing of spin qubits (Horse Ridge II) **3**

1.1. HIGH-LEVEL PROJECT DESCRIPTION 4
1.2. DIGITAL DESIGN LAYOUT REQUIREMENTS 4

2. Horse Ridge II Implementation **8**

2.1. COLLATERAL PREPARATION 8
2.2. FULL CHIP FLOORPLANNING 10
2.3. PARTITION IMPLEMENTATION 17
2.3.1 hr2_host_core_ss 23
2.3.2 hr2_smem_wrapper 25
2.3.3 hr2_host_core 28
2.3.4 QbitReadoutDetector_wrapper 29
2.3.5 QbitDrive_wrapper 31
2.3.6 QbitReadoutStimulus_wrapper 33
2.3.7 QbitDAC_Control_wrapper 34
2.3.8 hr2_soc_top 36
2.4. FULL CHIP ASSEMBLY 38

3. Design Verification **42**

3.1. TIMING CONVERGENCE 42
3.2. DRC & LVS 43
3.2.1 Stitching the Design Together 43
3.2.2 Verification runs for Large Designs 45
3.3. FORMAL EQUIVALENCE VERIFICATION 47

Conclusions **51**

Appendix **53**

A. QTM SCRIPTS USED TO GENERATE THE TIMING INFORMATION FOR THE ANALOG PORTION OF THE DESIGN **55**
B. TOP LEVEL NETLIST FOR FULL CHIP ASSEMBLY HR2_FCTOP.V **60**
C. FULL CHIP TIMING CONSTRAINTS **63**
D. Graph-based vs. Path-based TIMING ANALYSIS **72**

Bibliography **75**

Introduction

Quantum Computing presents a new computing paradigm that leverages quantum physics to solve computational problems exponentially faster than classical computers and promises to have a significant impact across industries and to solve critical problems such as climate change, among others. It is foreseen that Quantum computing systems won't replace conventional computing but rather augment supercomputers and high-performance computing. To achieve commercial relevance, quantum systems need to overcome multiple challenges, such as information fragility, software programmability, and scaling to millions of qubits. [5]

Of the many types of physical qubits, we decided to target silicon spin qubits for the Horse Ridge II chip, since these are well-suited to be manufactured with transistor fabrication processes, and Intel is an expert at fabricating transistors. As quantum computers start to require more qubits, silicon spin qubits are the right choice for Intel to build at large scale and, in the near future, achieve a useful quantum computer.[6]

Among the many challenges quantum computers face in becoming useful is that the qubits must be operated at temperatures very close to absolute zero, which requires them to be contained within cryogenic refrigerators. The electron spins of these qubits inside the fridge are manipulated by electronics that connect to the qubits via coaxial wires, which must be brought into the fridge and constitute a large amount of wiring in and out of the system. To simplify qubit control, Intel has developed the Horse Ridge II cryogenic control chip, based on Intel 22FFL technology, which operates at 4° K (degrees Kelvin) inside the cryogenic fridge to control the qubits.[5] The present document focuses on the physical implementation challenges of this control chip.

With respect to more recent developments, in June 2023, Intel announced the release of its newest quantum chip, "Tunnel Falls," making it available to the quantum research community to advance quantum computing research and accelerate its development for future generations. [7]

1. The Fully Integrated Cryo-CMOS SoC for state manipulation, readout, and high-speed gate pulsing of spin qubits (Horse Ridge II)

The current state-of-the-art quantum computing system consists of one or more qubit chips, control electronics, and a software stack running on a classical computer (Fig. 1a). The solid-state qubit chips are placed inside a dilution refrigerator with a typical base temperature of 10mK. They are controlled by analog/radio frequency (RF) signals generated by control instruments located outside the refrigerator (room-temperature electronics). Coaxial cables are needed to deliver control signals to the qubit inside the dilution refrigerator, and typically more than two cables are required per qubit. Due to this issue, the current quantum computing systems can't scale to a very large number of qubits due to form-factor limitations and thermal loading introduced by the interconnect to the room-temperature control electronics [1]

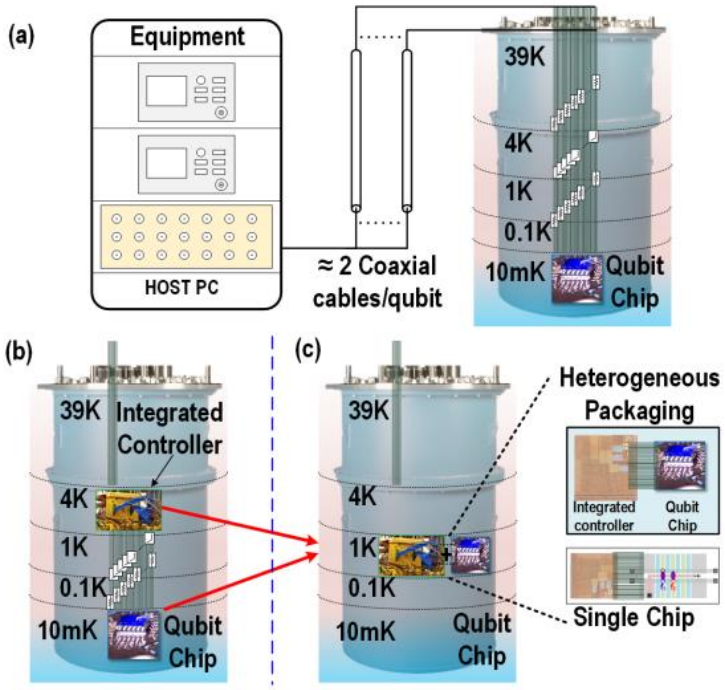


Figure 1: *(a) Current quantum computer setup, (b) intermediate step with an integrated controller at 4K (**this work**), and (c) co-integration scheme to enable scaling to many qubits.*

1. THE FULLY INTEGRATED CRYO-CMOS SOC FOR STATE MANIPULATION, READOUT, AND HIGH-SPEED GATE PULSING OF SPIN QUBITS

1.1. High-Level Project Description

The described project further advances the state of the art by integrating circuits to read the qubit state and generate high-speed voltage pulses required for the drive, readout two-qubit gate operations, and qubit characterization. The System on Chip (SoC) can manipulate up to 16 spin qubits by frequency division multiplexing (FDM) over a single RF line, read the qubit state of up to 6 qubits simultaneously, and control up to 22 qubit gate potentials. The SoC also integrates a microcontroller, providing additional flexibility in implementing quantum instruction sets and on-chip signal processing. The proposed cryogenic controller can replace all the high-speed room-temperature control electronics, paving the way toward scalable quantum computers. Please refer to Figure 1 b. For a detailed description of the analog circuitry, please refer to [1]. During the rest of this document, the design will be referred to as Cryo-CMOS SoC or Horse Ridge II.

1.2. Digital Design Layout Requirements

The SoC block diagram is shown in Figure 2. It is composed of three analog/RF front-end circuits, three digital back-end blocks, a microcontroller, on-chip memory, four temperature sensors, and digital I/Os. The SoC is mounted on the 4K plate of a dilution refrigerator and integrates spin-qubit state manipulation, readout, and qubit-gate pulsing, thereby replacing all room-temperature high-speed control electronics [1]. Due to the fact that this document will focus on the difficulties in implementing the digital layout for the design, only a very brief and high-level description of the main analog/digital circuits will be given, and the focus will be centered on the complexities of the digital layout implementation. The reader is welcome to look at the full circuit description found in the IEEE document referenced by [1] and [2] and the Nature magazine document [3], which describe in detail two different stepping's for this project.

1. The Fully Integrated Cryo-CMOS SoC for state manipulation, readout, and high-speed gate pulsing of spin qubits

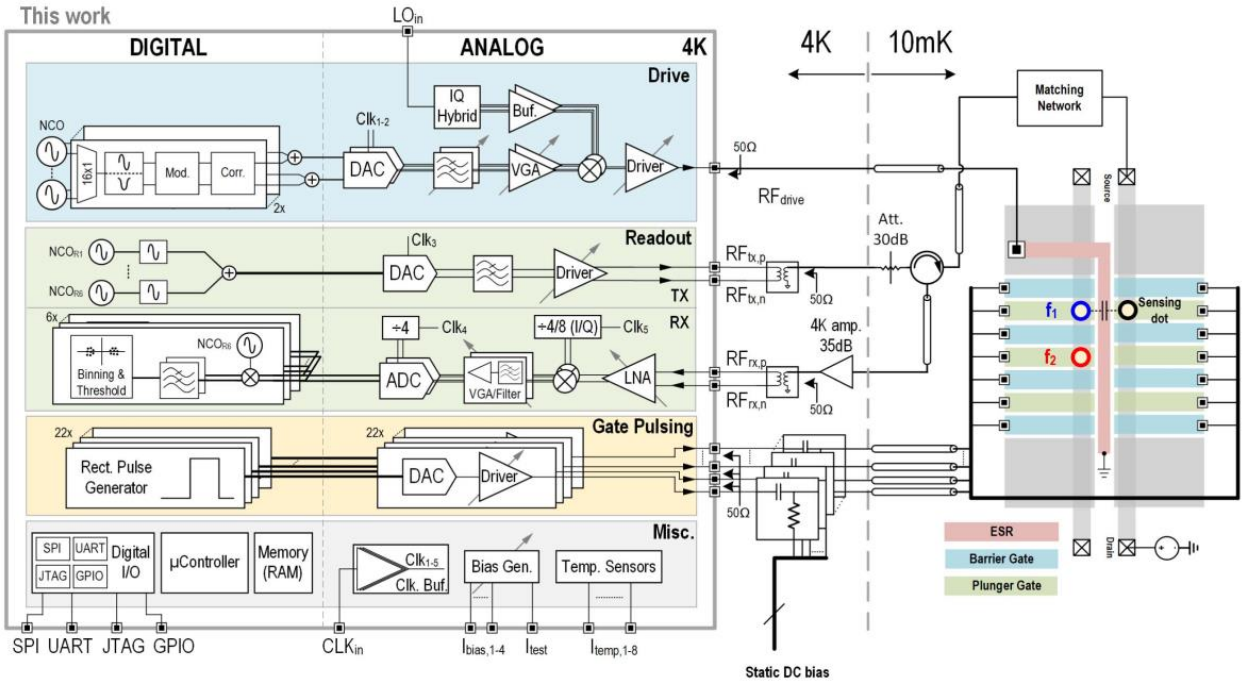


Figure 2: *Block diagram for the Horse Ridge II SoC with intended connection to spin-qubit at 10mK. The right portion of the design details the Analog circuit, and the left describes the digital logic [1].*

Qubit state manipulation is achieved by a single-sideband (SSB) pulse modulator front end and a digital back end for qubit phase tracking. 16 numerically controlled oscillators (NCOs) in the digital block can drive up to 16 qubits. The SoC is built to support a seven-qubit chip, but 16 NCOs are implemented to re-use the same design that was implemented on a previous version of the project, as described in [2].

Qubit state readout is achieved by implementing a multi-tone signal generator (Tx) and a receiver (Rx) with an on-chip digital qubit state demodulator. The Tx signal is sent to the matching network through a coupler, and the reflected signal from the matching network is received by the Rx through the same coupler. The digital demodulator has six parallel channels to detect up to six qubit states simultaneously with an RF bandwidth of 500MHz and a tone spacing of 100MHz. (For further details on the Rx and Tx, please refer to [1].

High-speed qubit gate pulsing is implemented using 22 DACs to support the linear array of seven qubits. The 22 DACs can be pulsed simultaneously, each at a different amplitude to suppress capacitive crosstalk among qubits.

1. The Fully Integrated Cryo-CMOS SoC for state manipulation, readout, and high-speed gate pulsing of spin qubits

The SoC is controlled by a host PC and FPGA at room temperature through multiple digital I/Os (SPI, UART, JTAG, GPIO).

The analog portion of the design is given to the digital layout design team as a hard IP (Intellectual Property) block having a rectilinear boundary. The given files include .oas (layout), .lef (layout exchange format containing interface shapes), and .sp (spice netlist). Refer to Figure 3, which shows a snapshot of the analog block .oas file.

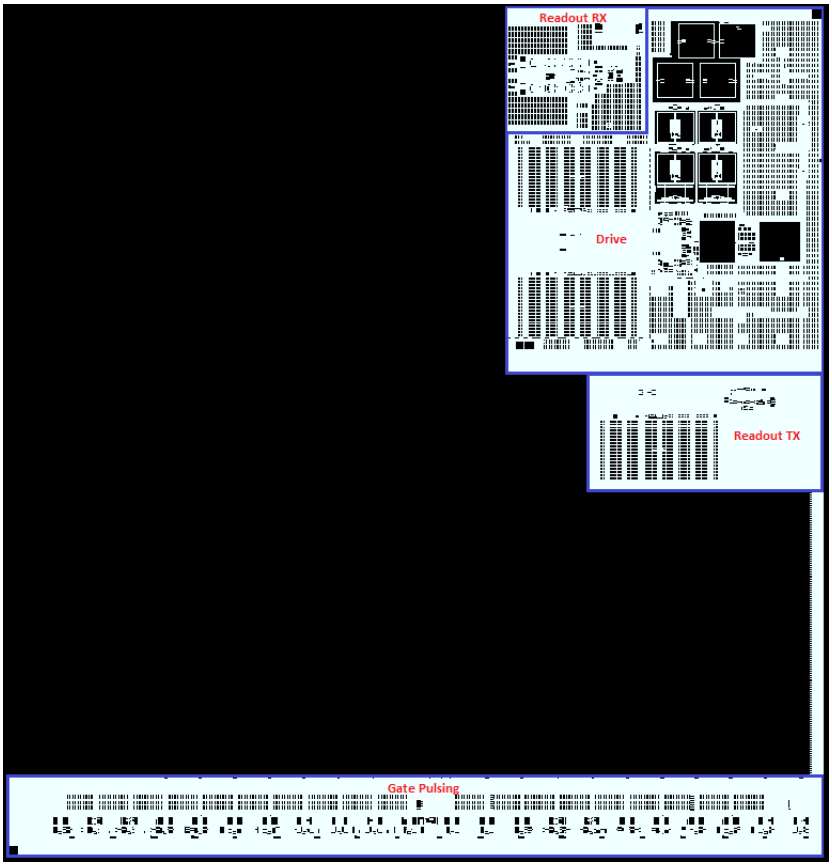


Figure3: *Rectilinear analog hard IP detailing the location of the Drive, Readout RX, Readout TX, and Gate Pulsing analog circuits.*

The design layout had the following requirements:

- The layout for the design was to be implemented using Intel's 22nm low-power Fin Fet Technology (Intel 22FFL)
- The design should target fast devices for which a model was built to estimate circuit operation at 4°K.

- The timing constraints had to be defined in such a way that the optimization tools would be able to optimize the design for a temperature window from 25C to -269.15C (4ºK). Given the fact that there were no standard cell libraries characterized to operate at 4ºK, the best model that we could find was -40C and we had to build extra margins for the clock network to over-fix hold timing to avoid functional issues in the design.

- Operating frequency of the functional blocks in the design is 625MHz for analog interface blocks, 416.66MHz for on-chip memory and interconnect fabric, and 25MHz for chip interface communication protocols. All clocks are derived from a 2.5GHz clock generated by the analog block, which drives the central clocking unit (CCU). The CCU contains the digital clock dividers, which in turn generate the remaining clocks, which are then distributed to all the functional blocks. Clock specifications are shown in Table 1.

Clock Name	Frequency
Clk2p5G	2.5GHz
Ext_Clk	250MHz
mclk	2.5GHz/250MHz
SPI Core Clk	25/2.5MHz
I2C Core Clk	25/2.5MHz
UART Core Clk	25/2.5MHz
GPIO Core Clk	25/2.5MHz
SPI (S) Core Clk	25/2.5MHz
Timer clk	25/2.5MHz
SPI XIP Core Clk	25/2.5MHz
accel_0_clk	625/62.5MHz
accel_90_clk	625/62.5MHz
accel_180_clk	625/62.5MHz
accel_270_clk	625/62.5MHz
Host_core_clk	416.66/41.66MHz
Host_core_aclk	416.66/41.66MHz
Smem_clk	416.66/41.66MHz
Ahb_fabric_clk	416.66/41.66MHz
Dma_hclk	416.66/41.66MHz
Apb_fabric_pclk	416.66/41.66MHz
base_clk	416.66/41.66MHz
core_clk	25/2.5MHz
Clk_25	25 MHz
JTAG_Clk	20MHz

Table1: *Clock frequency specification*

- The Layout physical area for the entire design is set at ~16mm^2 defined by a 4 mm x 4 mm die as defined by Intel's Multi Product Shuttle Team.

2. Horse Ridge II Implementation

The following chapter describes the problems encountered and the solutions implemented during the execution of the Cryo-CMOS SoC layout design implementation.

2.1. Collateral Preparation

Since digital logic synthesis and layout tools use timing as the optimization function, whenever you integrate an analog block with digital logic, you will need to create a timing model for the analog portion that is good enough for the tools to optimize the logic signals that interface to the analog circuits. The timing information for the design tools is read using Liberty format files (.lib). A full liberty model consists of delay, transition time, tristate, input capacitance, dynamic power, leakage power, setup time, hold time, recovery time, removal time, etc.

To build a liberty model of the analog block to help us optimize the digital design, we did not need to create a full timing model but rather a model describing the drive and load of the analog interface. Using static timing tools, we created a Quick Timing Model (QTM) to model the analog block. The full script used to create the model is provided in Appendix A.

The main parts of the script needed to create a QTM model are:

create_qtm_model : Begin the definition of a QTM description

set_qtm_technology : set QTM technology variables such as max cap, max tran.

set_qtm_global_parameter : Sets global parameters for a QTM such as setup, hold, recovery, removal or clk_to_output times

create_qtm_drive_type : create a drive type definition for QTM output ports

create_qtm_load_type : create a load type definition for a QTM input ports

create_qtm_port : create a port on a QTM model

set_qtm_port_drive : sets the drive on the QTM output port

set_qtm_port_load : sets the input load on the QTM input port.

save_qtm_model : saves the QTM and indicates the end of the definition.

The specification for which cells are to be used to model the output drive and input load was given by the custom layout team, and these were used in the creation of the QTM model. The output of the QTM creation script is a compiled (binary version) of the .lib file for the analog cell, which is then used along with the lef (layout exchange format) file, which contains the physical pin locations and size/shape of the cell, to create a simplified view of the cell called NDM to be used in the automatic place and route tool.

Since the QTM model generated for the analog block did not give different specifications for the different PVT (Process, Voltage, and Temperature) operating windows, the same model was used for the different operation scenarios during optimization and verification tasks. This indeed is not optimal, but given the limited inputs we had from the analog team, it was good enough to implement the design.

The different memory models (SRAM and RF) were obtained from Intel's SoC Memory group, and the released collateral did not include the NDM model for the blocks. To generate the NDM model for the different memory IPs, we used Intel's proprietary scripts and procedures, which use all the .lib files and an enclosed .lef file for each memory to create a frame view and annotate it with timing information for use in the different optimization tasks.

2.2. Full Chip Floorplanning

To accelerate the implementation of the design, the different hierarchies within the logic design were used to define the physical implementation of the design into sub-blocks in such a way that specific functions were kept together as much as possible and that layout execution on these sub-blocks could happen in parallel. The physical hierarchical implementation is shown in Table2. hr2_soc_top is the name of the cell which contains all sub-cells within the design and constitutes the top level of the Horse Ridge II SoC logic. This cell resides on hierarchy level 2. Level 1 of hierarchy is called hr2_fctop, this cell was created by the physical design team and is not part of the logic design delivered by the RTL team and its used to instantiate all the required bump cells used to interface the die to the package for the flip chip design as well as different

structures required by Intel's foundry: TIC cells (Tape In Collaterals) such as the Etch Ring (ER), Edge Damage Monitor (EDM) and Pattern Recognition System (PRS) cells in addition of the functional design hr2_soc_top. Intel requires that the topmost hierarchy should have a specific name given by the Intel Multi-product Shuttle (MPS) team to facilitate the tracking of the device through the fabrication process, in addition to having the full chip rotated 90° to make all poly gates horizontal (designs are drawn with the poly vertical) before fabrication. Level 0 of the hierarchy constitutes the rotated version of the Horse Ridge II chip. On hierarchy level 3, we have all the sub-blocks or functions that were synthesized and optimized in parallel to expedite the implementation of the SoC, in addition to the analog block.

Hier Level	Block Name
0	*<Intel Product Name>*
1	*hr2_fctop*
2	*Bumps*
2	*ER-EDM-PRS*
2	*hr2_soc_top*
3	*hr2_analog_top*
3	*hr2_smem_wrapper*
3	*hr2_host_core*
3	*hr2_host_core_ss*
3	*QbitReadoutDetector_wrapper*
3	*QbitDrive_wrapper*
3	*QbitReadoutStimulus_wrapper*
3	*QbitDAC_Control_wrapper*
3	*diamond_io_gen_top*

Table2: *Physical hierarchy planning for the Horse Ridge II design*

Floorplanning for these types of designs is one of the most important tasks since, depending on the selected locations for each unit and the type of floorplan (channel-based, abutted designs, mixed), this will impact how easy or difficult it will be for timing closure and layout cleanup to take place. In addition to deciding the location of cells, decisions for the size of the die should also be made, as the silicon area required for a chip directly impacts the cost that will be paid to the foundry for the design. For this design in particular, we chose to use Intel's 22nm FinFET technology and required a 4 mm × 4 mm die. When we use Intel as a foundry, we need to also take into consideration that the foundry requires our design to include an Etch ring (ER) around the die

edge, which protects the design from cracks when the laser saw cuts out the individual dies. An edge damage monitor (EDM) structure is also required by the foundry, which is nothing but a two-pin structure that is placed around the die immediately after the Etch Ring and helps detect if any cracks traversed the ER structure after the individual dies are cut from the wafer by allowing the measurement of signal continuity between the two pins. If signal continuity can't be measured, this means that a crack has traversed the ER and thus the design integrity could be compromised. These two structures are provided by the foundry but need to be instanced by the layout team at the top hierarchy of the design, and the layout team must make sure that no other blocks are placed on top of the EREDM (ER + EDM). Two other important structures required by the foundry are the Pattern Recognition System (PRS) markers and the DIC (Drop In Cells), which help the foundry align the different masks and the package during fabrication. PRS marker cells are placed at opposite corners of the design, each with specific patterns, which the foundry communicates to the design team. DIC cells need to be placed in the core area of the design at specific distances. After planning all the foundry-required structures, the remaining area can be used to implement the logic design.

For Horse Ridge II, we decided to implement it as a channel-based floorplan for the digital logic and abut the blocks that interact with the custom block to it for simplicity of connection. Considering that the interface for the custom block was given, the rest of the partitions were placed as shown in Figure 4. Also shown in Figure 4 are the planned locations for the DIC and PRS markers.

2. HORSE RIDGE II IMPLEMENTATION

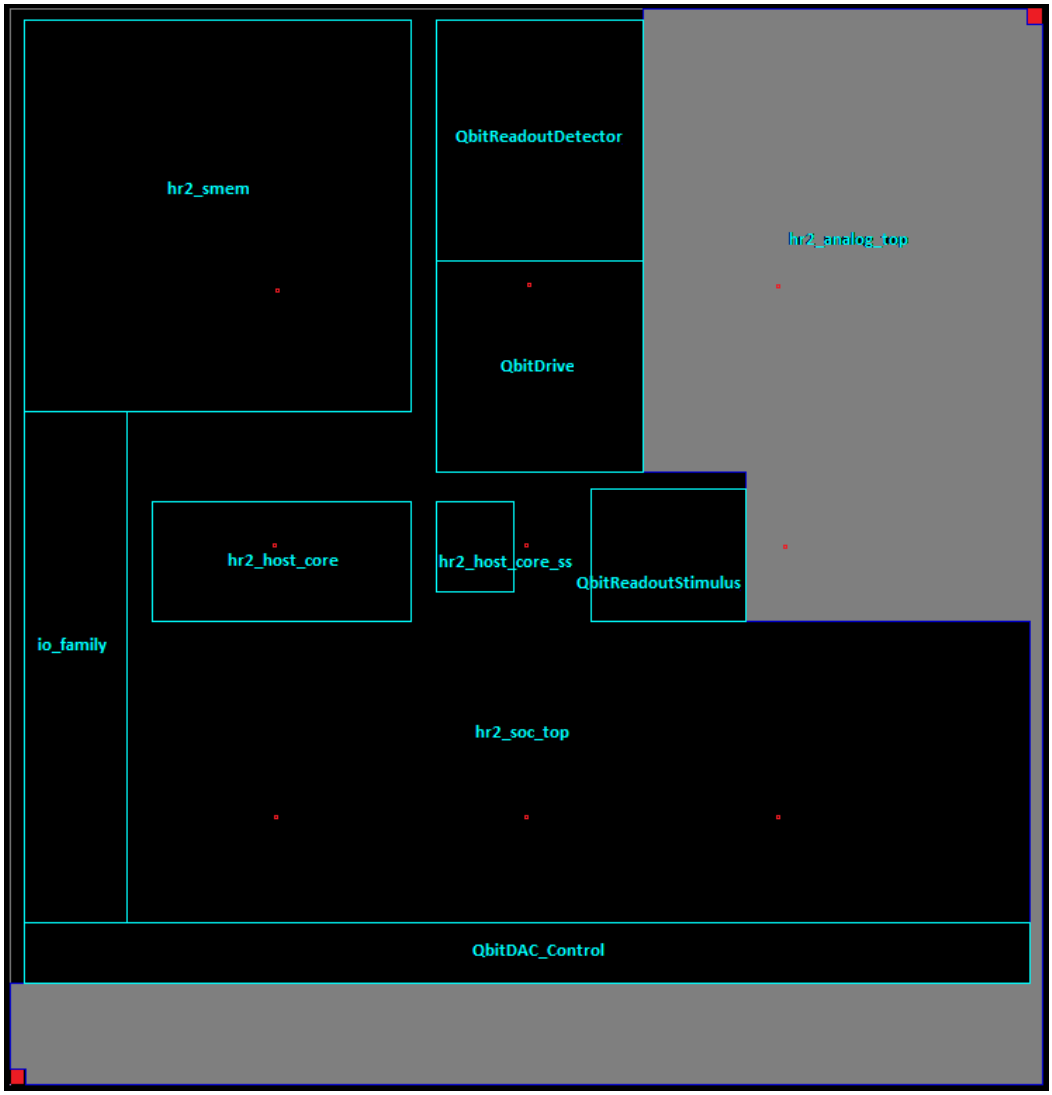


Figure4: *hr2_soc_top floorplan showing the analog partition in gray and the location for the rest of the core partitions in the channel-based floorplan. In red, locations for DIC cells (within the core area) and notches for the PRS markers required by the foundry.*

The analog partition's logic is placed roughly as shown in Figure 5.

2. HORSE RIDGE II IMPLEMENTATION

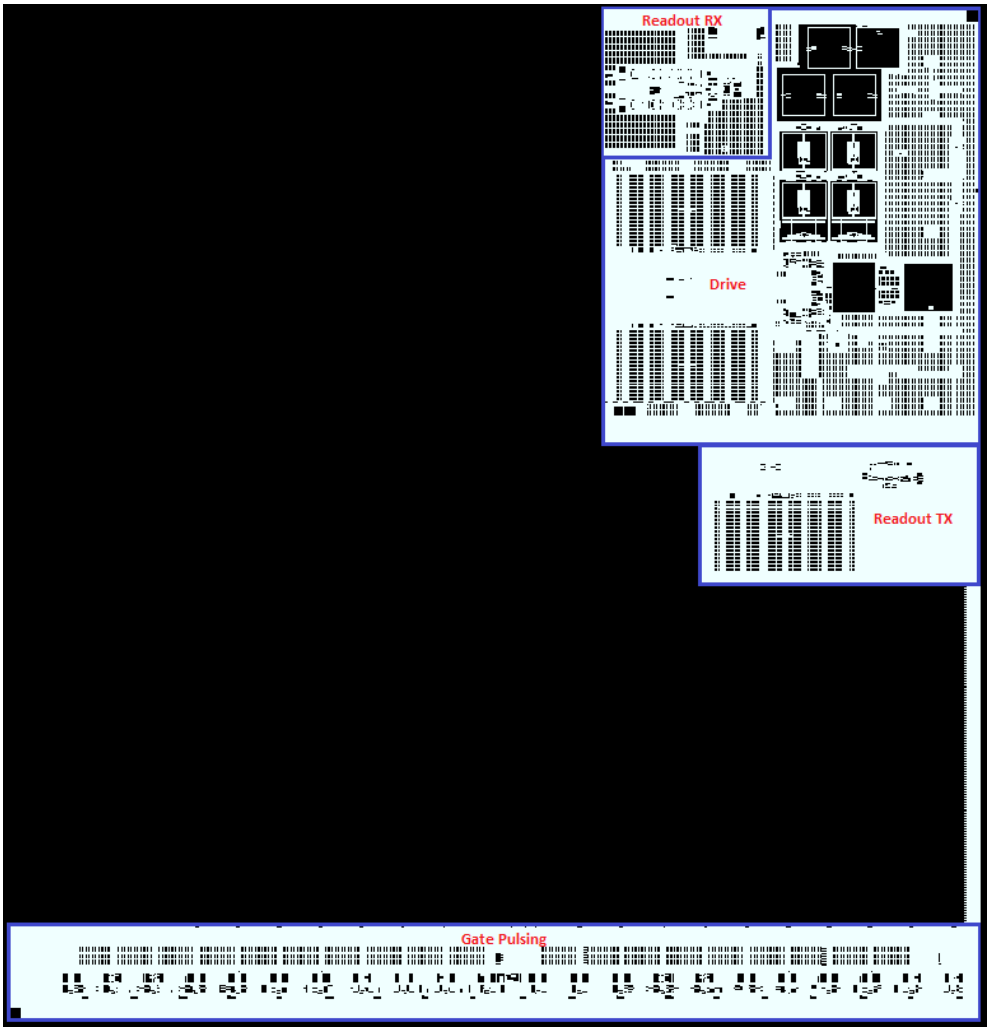


Figure5: *Rough location of the custom logic circuitry inside the custom analog block*

IO family partition was placed and designed so that the flip-chip bumps above this partition would be driven by the IO buffers with the shortest routing possible. The constraint driving the location of the IO family was the bump design, which was co-designed by the APR team and the custom layout teams. Bump locations can be seen in Figure 6.

2. HORSE RIDGE II IMPLEMENTATION

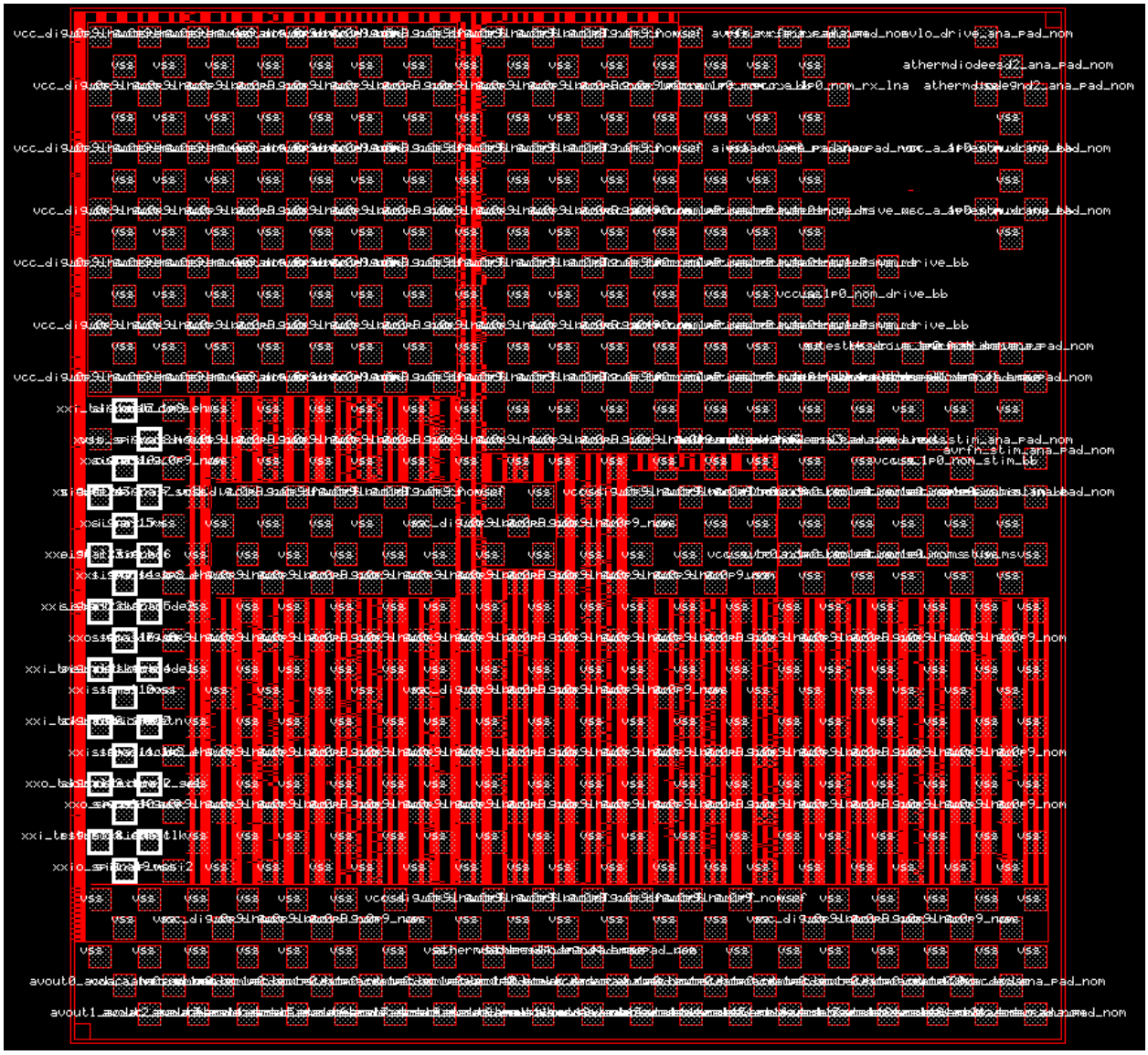


Figure 6: *Flip-Chip bump cell placement over the Horse Ridge II SoC floorplan. The digital IO locations are highlighted.*

After deciding on the size and shape of the logic sub-blocks in the design, a connectivity analysis was executed, and pins were created for each of the sub-blocks so that physical implementation runs could take place. Pin placement analysis and final pin locations for the sub-blocks can be seen in Figure 7.

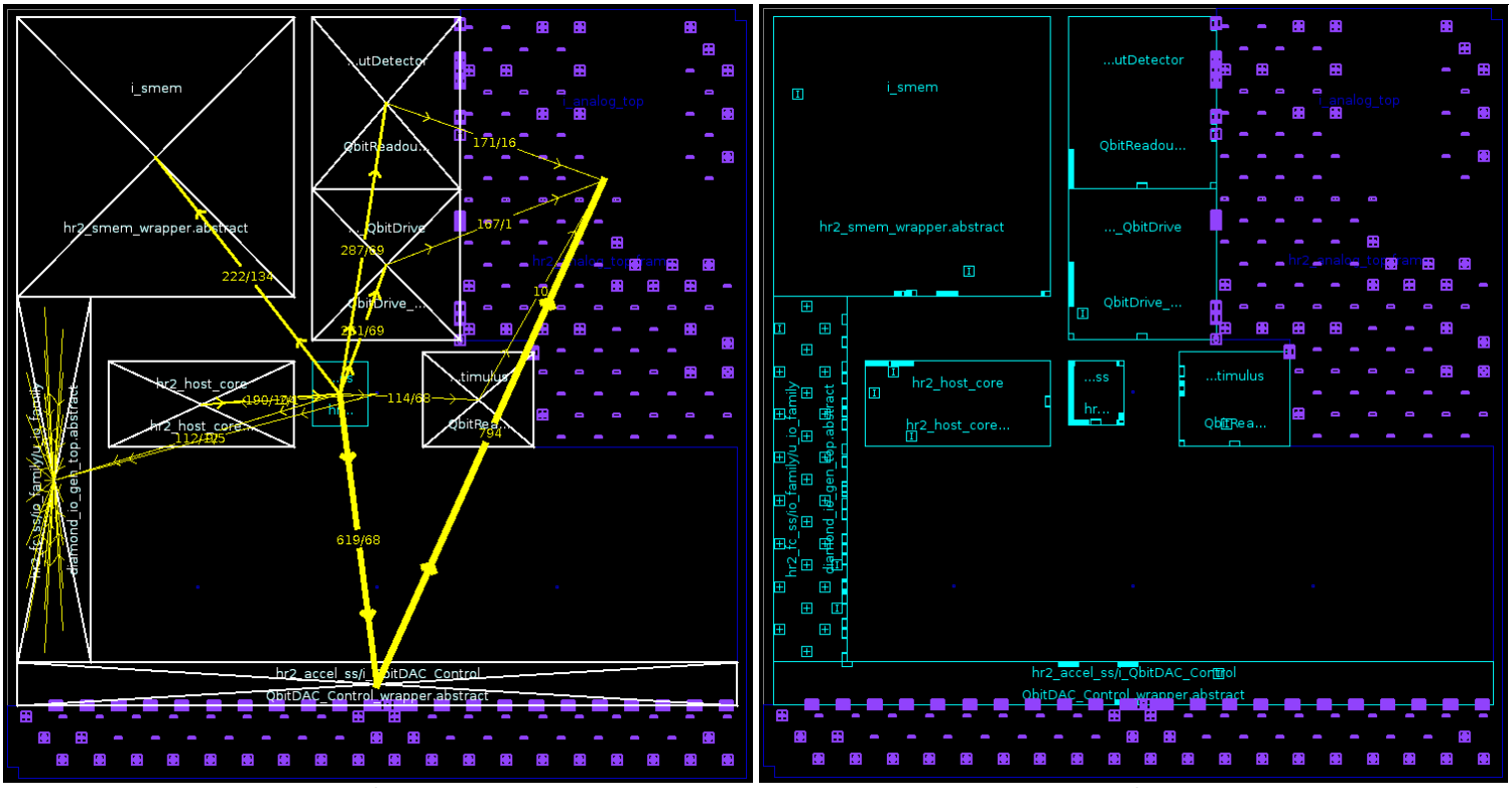


Figure 7: *A) Sub-block connectivity analysis: fly lines represent the number of connections for each block. B) final pin placement for sub-blocks (colored light blue). The analog block's pins are colored purple.*

After deriving block sizes and shape constraints, size and shape for each sub-block, the sub-block owners take these inputs, as well as pin location scripts and timing constraints, to implement each of the sub-blocks. In an ideal scenario, where the design has been completely finalized and timing constraints have been developed fully before floorplanning, the top-level owner pushes down the timing from the top level into the partitions and creates timing constraints for each of the partitions. The layout implementation of the hr2_soc_top, however, was done in parallel with the implementation of the logic design, and timing constraints had to be derived as the project progressed; thus, full timing constraints could not be given to the partition owners from the top-level perspective and had to be updated as the design progressed.

Since individual partitions could not get timing constraints from the top-level floorplan, it was decided to constrain the inputs and outputs using 2/3 of the period of the clock for input/output delay constraints, as described a bit later in section 2.3. Liberty was given to the partition owner to derive the best location for macro IPs within the partition to mitigate any routing congestion

issues and alleviate timing bottlenecks within the partition based on the pin placement and timing requirements of each block.

2.3 Partition Implementation

Because both logical synthesis and physical synthesis tools are timing-based and the main optimization function is the minimization of timing slack, timing constraints play a major role in how a design is optimized. In synchronous designs, clock signals constitute the "heart" of the design; they establish the pace at which data moves through the logic, launch and capture data, and establish how inputs/outputs move data in/out of the design. It is important to note that when defining timing constraints and reading reports, the timing units are defined specifically for each technology node; for Intel's 22FFT technology in particular, these units are time (ps), capacitive load (ff), voltage (1V), current (1mA), resistance (1 ohm).

From Table 1, the clock definitions show that we need to define a series of clocks; however, four of these clocks are coming into the design either from outside logic or the analog partition (Clk2p5G, Ext_Clk, Clk_25, JTAG_CLK), and the rest of these clocks are generated based on the first two (Clk2p5G, Ext_Clk), each running at 2.5GHz and 250MHz respectively. Now the clock definitions for the four external clocks are fairly straightforward as follows:

```
create_clock [get_pins i_analog_top/clk_2p5] -name Clk2p5G -period 400 -waveform {0 200}
create_clock [get_pins {hr2_fc_ss/io_family/u_io_family/OUTI[0]}]  -name Ext_Clk  -period 4000  -
waveform {0 2000}
create_clock [get_pins {hr2_fc_ss/io_family/u_io_family/OUTI[19]}] -name JTAG_Clk -period 50000 -
waveform {0 25000}
create_clock [get_pins i_analog_top/clk_0p5] -name Clk_25 -period 1600 -waveform {0 800}
```

These definitions handle clocks entering the design via top-level ports and the high-speed clock generated inside the analog custom block. Please note that even though the clocks Ext_Clk and JTAG_Clk come from outside, the clock definition was set on the output pin in the IO family hierarchy, not on the top-level port. This was due to an issue in which the model used for the IO buffers was not propagating the signal through them. Ideally, the definitions of these clock signals should be set on the top-level input pins so that timing tools account for input buffer delays and transition times when calculating clock latency to the registers. Since this SoC was a test chip only

and the external clock is for testing under controlled lab conditions, it was decided that having the clock definition at the output of the IO buffer was not an issue.

Since the tool's timing engine is static and no actual dynamic simulations are taking place when analyzing timing, internal clocks, which are a result of divisions of the main clocks, must be specified using timing constraints that tell the tool the clock relations of these derived clocks. For simplicity in this document and given that many of the generated clocks have the same frequency, not all generated clock definitions will be shown; only an example from each frequency group will be shown. An exception will be made with the generated clocks, which need to describe a frequency rotation, so that inexperienced designers can see how to create such constraints for a design. Since we need to describe clock transformations which have 90°, 180°, 270° rotations, please refer to figure 8, which depicts the high-frequency clock signal (2.5GHz) and the generated 625MHz clocks with rotations. The picture will help in understanding the timing constraints for the generated clocks.

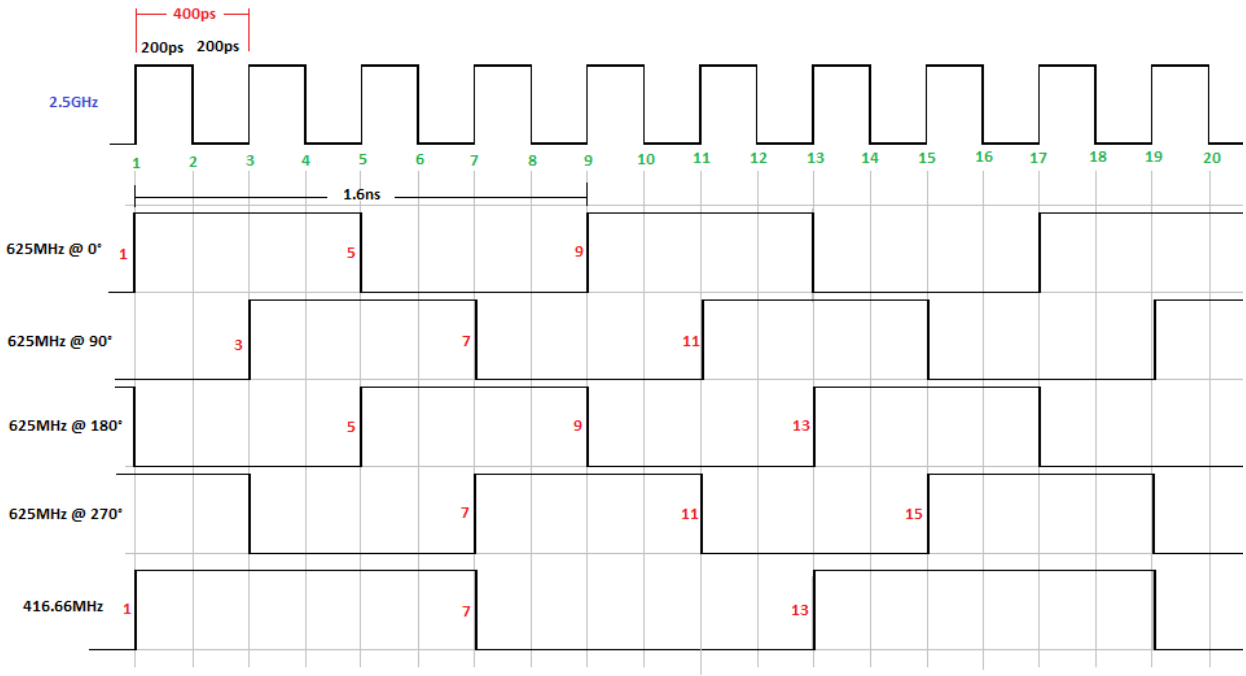


Figure 8: *2.5GHz clock waveform with edge numbers in green. In red is the edge number used in the definition of the rotated-generated clocks and 416.66 MHz clock definitions.*

To describe the first clock which has frequency of 625MHz and a 0° rotation, as a generated clock with respect to the 2.5GHz clock, we write the timing constraint as follows. The actual hierarchy for the pins has been removed (replaced with <hier>) to simplify the constraint notation:

```
create_generated_clock [get_pins <hier>/clk_0_out_div_reg/o] -name accel_0_clk_2p5G -source
[get_pins <hier>/clk_2p5]  -edges {1 5 9}
```

Please note that the create_generated clock command uses the **-edges** switch and that the edges of the 2.5GHz clock, which describe the 0° rotated clock, are edges 1 (rising), 5 (falling), and 9 (rising) from Figure 8. In this same way, we can describe all the generated clocks that have rotations as follows:

```
create_generated_clock [get_pins <hier>/clk_90_out_div_reg/o] -name accel_90_clk_2p5G -source
[get_pins <hier>/clk_2p5] -edges {3 7 11}

create_generated_clock [get_pins <hier>/clk_180_out_div_reg/o] -name accel_180_clk_2p5G -source
[get_pins <hier>/clk_2p5] -edges {5 9 13}

create_generated_clock [get_pins <hier>/clk_270_out_div_reg/o] -name accel_270_clk_2p5G -source
[get_pins <hier>/clk_2p5] -edges {7 11 15}
```

Clocks with a frequency of 416.66 MHz are generated from the 2.5 GHz clock by dividing it using edges 1, 7, and 13 in Figure 8.

```
create_generated_clock [get_pins <hier>/host_core_clk_i0_dcszo/clkout] -name Host_core_clk_2p5G -
source [get_pins <hier>/clk_2p5] -edges {1 7 13}
```

To generate clocks which have a frequency of 25MHz using the 2.5GHz clock as a base, we need to divide the clock frequency by 100 so the generated clock definition is:

```
create_generated_clock [get_pins <hier>/spi_s_clk_i0_dcszo/clkout] -name SPI_S_Core_clk_2p5G -
source [get_pins <hier>/clk_2p5] -divide_by 100
```

Here, we describe how to create the main clocks and the generated clocks based on the 2.5GHz main clock. The same type of definitions must be done based on the Ext_Clk, and the same edges and clock division factors will apply in the same way for that clock.

Since we decided to partition the full design as shown in Table 2, we need to set timing constraints for the input and output delays of the partitions, to the best of our knowledge, since we didn't have a constraints-budgeting flow in place. When we have no idea of how to budget input and output delay times for partitions, the best way is to use the 1/3 of the clock period rule of thumb, knowing that we might get timing violations at the interface at the partition level and that the full timing picture for the partitions will be seen once the full chip is assembled and timed as one. If you take a look at Figure 9, you will see that we can assume that partition A, which has a launching flop and some combinational logic, has a timing path going to the output of the partition. We can assume as an initial guess that the time it takes from the arrival of the launching clock edge at the clock pin of the flop, the clk to Q time plus the propagation time of the data signal through the combinational logic up to the output pin of partition A, is 1/3 of the period of the clock clk. In the same way, we can consider that the propagation time of the data signal through a top-level routing channel is 1/3 of the clock period. The propagation time from the input pin of partition B through the combinational logic cloud of partition B to pin D of the input flop can be allocated as the remaining 1/3 of the clock period. With these considerations, the input and output delay timing constraints for the partitions are 2/3 of the clock period and can be coded as follows:

```
set_input_delay -clock clk [expr (2.0/3.0)*$CLK_PERIOD] [get_ports {input_port}]
set_output_delay -clock clk [expr (2.0/3.0)*$CLK_PERIOD] [get_ports {output_port}]
```

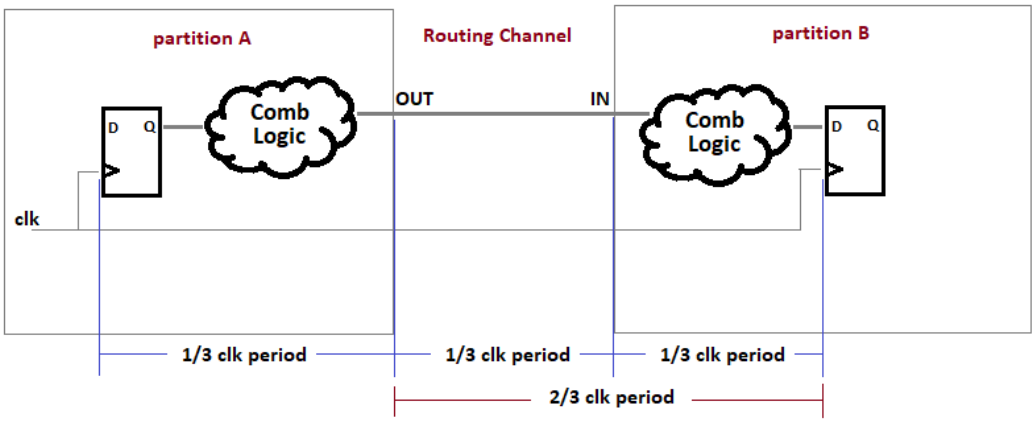


Figure 9: *Picture showing the 1/3 clk period rule of thumb to budget for input and output delays for partitions.*

The 1/3 rule of thumb works very well for designs where we have registerd inputs/ouptuts, but it doesn't work all that well when we have feedthrough paths through a partition. For feedthrough paths, we need to back off the 2/3 input and output delay constraints (since this would cause a 1/3 clk-period negative slack by definition) and set a 1/3 input and output delay instead (allowing 1/3 clk-period for signal propagation through the partition), as shown in Figure 10.

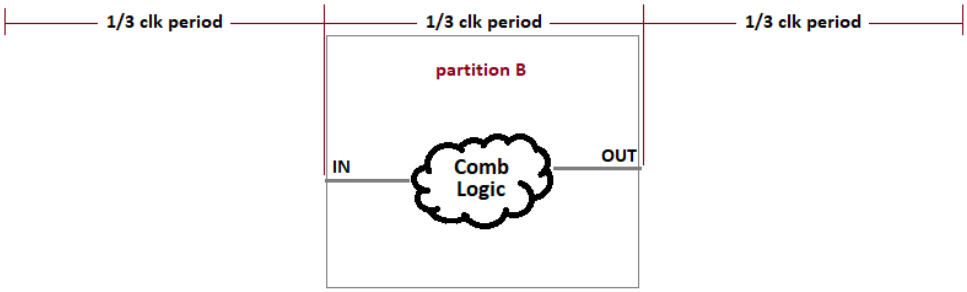


Figure 10: *The 2/3 clk period rule of thumb will over-constrain feedthrough paths; instead, we use 1/3 clk period input/output delay for these.*

When we run into this kind of situation, we need to create auxiliary external virtual clocks to ensure the timing constraints are correctly defined. The timing constraints would look as follows:

```
create_clock -name clk -period $CLK_PERIOD [get_ports clk]
create_clock -name clk_v -period $CLK_PERIOD
create_clock -name clk_fth_v -period $CLK_PERIOD

set_input_delay -clock clk_v [expr (2.0/3.0)*$CLK_PERIOD] [get_ports {input ports}]
set_input_delay -clock clk_fth_v [expr (1.0/3.0)*$CLK_PERIOD] [get_ports {input ports}] -add

set_output_delay -clock clk_v [expr (2.0/3.0)*$CLK_PERIOD] [get_ports {output ports}]
set_output_delay -clock clk_fth_v [expr (1.0/3.0)*$CLK_PERIOD] [get_ports {output ports}] -add
# timing exceptions
set_false_path -from [get_clocks clk_v] -to [get_clocks clk_v]
set_false_path -from [get_clocks clk_fth_v] -to [get_clocks clk_v]
set_false_path -from [get_clocks clk_v] -to [get_clocks clk_fth_v]
set_false_path -from [get_clocks clk_fth_v] -to [get_clocks clk]
set_false_path -from [get_clocks clk] -to [get_clocks clk_fth_v]
```

The last four false path timing exceptions can be simplified by means of a clock group as follows:

```
set_clock_group -async -name async1 -group [get_clocks {clk_v clk}] -group [get_clocks {clk_fth_v}]
```

So the timing exceptions would then be:

```
set_clock_group -async -name async1 -group [get_clocks {clk_v clk}] -group [get_clocks {clk_fth_v}]
set_false_path -from [get_clocks clk_v] -to [get_clocks clk_v]
```

With this, the valid and invalid timing checks are as shown in Figure 11. External Flip Flop E-FF1, which is timed by the virtual clk_v clock, only has valid timing paths to I-FF1, which is timed by the real clock clk. Likewise, I-FF2, clocked by clk, has valid time paths only to E-FF3, clocked by clk_v. In this way, inputs and outputs to/from internal registers are constrained by clk_v to clk and clk to clk_v timing relations. Invalid timing paths for clk_v would be feedthrough paths from clk_v to clk_v and clk_v to clk_fth_v. Now, to constrain the feedthrough paths, we have created clk_fth_v. Valid paths for this clock are: E-FF2, timed by clk_fth_v, going through the combinational logic in the design and out to E-FF4. Invalid paths would be E-FF2 timed by clk_fth_v to I-FF1, which is timed by clk, as well as I-FF2 timed by clk to E-FF4 timed by clk_fth_v.

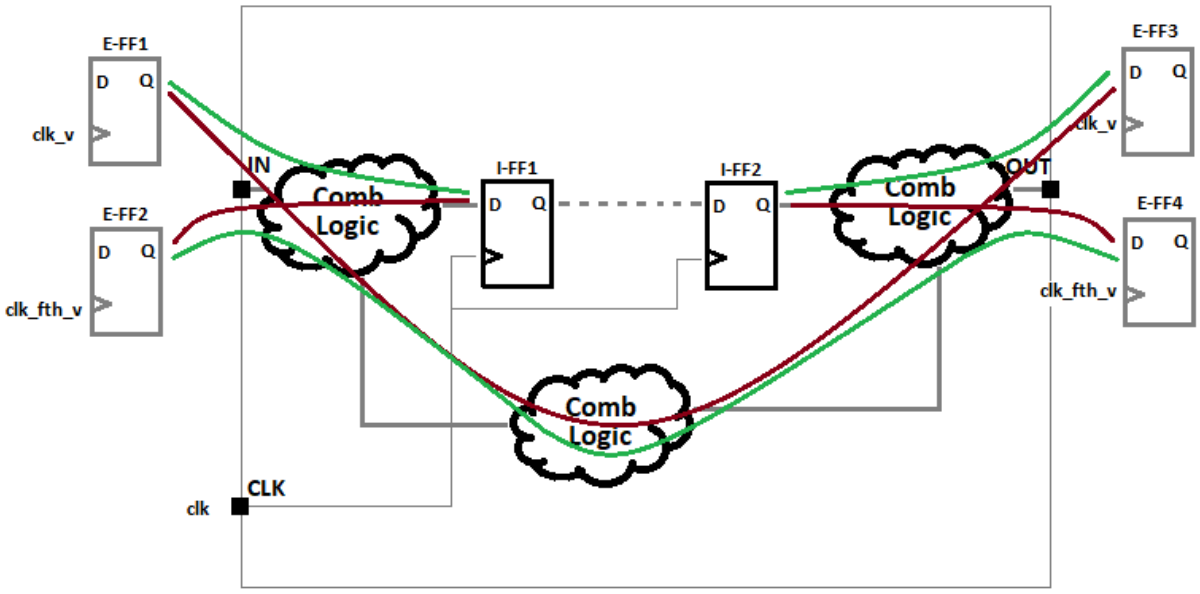


Figure 11: *Input/Output delay valid/invalid timing paths after applying the clock exceptions.*

Every partition was constrained in a similar way to the one shown above, although not all partitions had all the clock definitions listed as inputs.

2.3.1 Hr2_host_core_ss

This partition was created to hold the clock generation circuitry to be able to balance all the clocks inside this partition before distributing them to the rest of the design, the thought being that if clocks were balanced first, it would be easier to minimize clock skew between the rest of the partitions by distributing the clocks semi-automatically. It is worth saying that the planning for this partition was done mid-way through the implementation of the design since originally, we had intended for the clock circuitry to reside in the channel partition (hr2_soc_top) but this proved to be very difficult to control hence, the physical design team decided to implement the clock circuitry as an additional physical partition and requested the RTL team to create it in the code accordingly.

***NOTE**: When planning the full chip partitioning, try never to leave sequential logic at the channel level; all sequential logic should go in a partition, and if any, only combinational logic can reside at the channel level. Best results are obtained from not having any logic at the channel level.*

Input clocks to the hr2_host_core_ss partition are the 2.5GHz clock from the custom analog block (hr2_analog_top) and the external 250MHz clock. Output clocks are the rest of the clocks shown in Table 1, except for Clk_25 and JTAG_Clk, which come from the custom block and external pins, respectively.

Final synthesis results for this partition were as follows:

```
Total cell count: 64467
SEQ cell count: 16698
Cell area (um2): 48254.24016
Total macro cells count: 0
WNS_INT (setup): 0 ps
TNS_INT (setup):  0.0 ps
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): -1984.549 ps
TNS_IO (setup):  -19360.856 ps
Failing end points (IO - setup): 20.0
WNS_INT (hold): -31.563 ps
TNS_INT (hold):  -11911.382 ps
Failing end points (Reg2reg - hold): 2961.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  20
Max Cap Vios:  0
Worst Cap Slack (ff):  0
```

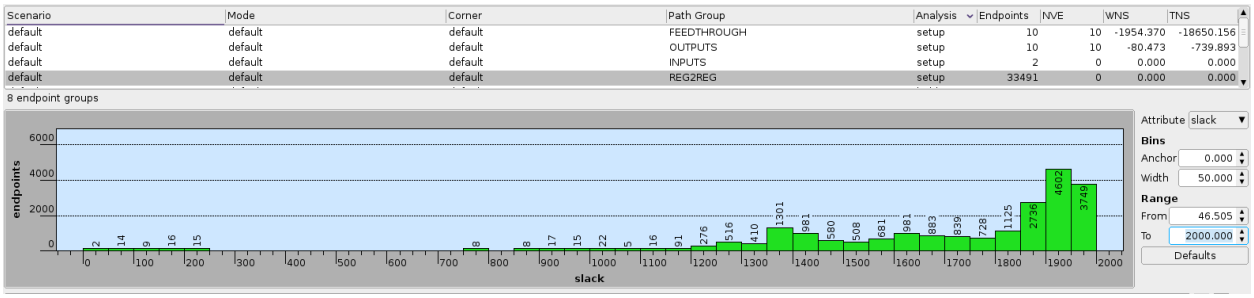
```
Max Slew Vios:  12
Worst Slew Slack:  -0.49734 ps
```

From the top-level floorplan, the size and shape of this block were: 291.60um x 340.20um, rectangular. This block only contains logic and no hard IPs, the final floorplan and implementation for this block look as shown in Figure 12.



Figure 12: *hr2_host_core_ss block implementation*

The final setup timing histogram from the static timing analysis tool for this block, after implementing the layout and extracting the parasitics, is shown below. Notice that all internal paths are passing timing at the block level.



Clock skews for the clocks generated in this partition are shown below for the slow corner at -40°C.

```
=====================================================
==== Summary Reporting for Corner psss_max_0p9_m40 ====
=====================================================

========================================== Summary Table for Corner psss_max_0p9_m40 ==========================================
Clock /                                Attrs     Sinks Levels    Clock     Clock     Clock      Max     Global  Trans DRC  Cap DRC
Skew Group                                                      Repeater  Repeater  Stdcell   Latency    Skew     Count     Count
                                                                  Count      Area      Area
---------------------------------------------------------------------------------------------------------------------------------
### Mode: func, Scenario: func_max_0p9_m40
Ahb_fabric_clk_2p5G                      M,D      6399     26       258    185.55    185.55    842.95    54.19         5         0
Host_core_aclk_2p5G                      M,D      1485     20        62     53.62     53.62    846.35    17.40         0         0
base_clk_2p5G                            M,D      1350     18        55     46.06     52.80    829.66    46.58         0         0
Apb_fabric_pclk_2p5G                     M,D      1267     28        58     51.51     51.51    899.68    24.36         0         0
SPI_Clk                                  M,D       125      4         5      3.27      3.67    111.03    11.73         0         0
Clk2p5G                                  M,D        31      5        50     31.64     68.65    100.00    95.65         0         1
 accel_0_clk_2p5G                          G         0      0         1      1.36      5.24        --        --         0         0
 accel_90_clk_2p5G                         G         0      0         1      1.36      4.83        --        --         0         0
 accel_180_clk_2p5G                        G         0      0         1      1.36      4.83        --        --         0         0
 accel_270_clk_2p5G                        G         0      0         1      1.36      4.83        --        --         0         0
 I2C_Core_clk_2p5G                         G         0      0         9      7.48      7.48        --        --         0         1
 UART_Core_clk_2p5G                        G         0      0         5      2.18      2.18        --        --         0         0
 GPIO_Core_clk_2p5G                        G         0      0         0      0.00      0.00        --        --         0         0
 SPI_S_Core_clk_2p5G                       G         0      0         0      0.00      0.00        --        --         0         0
 Timer_clk_2p5G                            G         0      0         0      0.00      0.00        --        --         0         0
 SPI_XIP_Core_clk_2p5G                     G         0      0        30     13.68     13.68        --        --         0         0
 core_clk_2p5G                             G         0      0         0      0.00     14.63        --        --         0         0
Dma_hclk_2p5G                            M,D       2     22        28     31.23     31.23    846.75      0.13         0         0
Host_core_clk_2p5G                       M,D       0      0         0      0.00      0.00        --        --         0         0
Smem_clk_2p5G                            M,D       0      0         0      0.00      0.00        --        --         0         0
---------------------------------------------------------------------------------------------------------------------------------
All Clocks                                       10659     28       516    402.86    447.02    899.68    95.65         5         1
```

NOTE: *IO timing groups usually have violations at the end of the APR implementation flows since interface clocks are ideal and internal clocks to the partition are propagated; hence, they have a large clock skew, which is unrealistic. To get more realistic timing, you need the implementation tool to compute estimated clock latencies for the interface clocks.*

Realistic interface timing for the partition will be obtained once the full design has been assembled and the full timing path can be reported from the top hierarchy. For block implementation, we decided to simply ignore interface timing violations, which we check at the top level.

2.3.2 Hr2_smem_wrapper

This partition holds the shared memory for the entire design and consists of 983,040 B distributed across 48 memory cells. The input clock for this partition is the 416 MHz Smem_clk. Synthesis results for this partition are as follows:

```
Total cell count: 49213
SEQ cell count: 4165
SEQ cell area: 6488.9748
Cell area (um2): 22173.215398
Total macro cells count: 48
WNS_INT (setup): 0 ps
```

```
TNS_INT (setup):  0.0 ps
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): -407.96 ps
TNS_IO (setup):  -129068.004 ps
Failing end points (IO - setup): 1176.0
WNS_INT (hold): -722.687 ps
TNS_INT (hold):  -160773.766 ps
Failing end points (Reg2reg - hold): 3072.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  1176
Max Cap Vios:  0
Worst Cap Slack (ff):  0
Max Slew Vios:  0
Worst Slew Slack:  0 ps
```

Dimensions for this block were given as 1,458.0 um x 1,474.2 um. This block is different from the previous one because top-level cells are pushed down into it. If you remember from Figure 4, this block is overlapped by a DIC cell, which is a foundry-required cell, and thus the block macros and logic must be placed and routed around the DIC cell. After several rounds of full-chip assembly and timing analysis, macros needed to be moved to meet the block's timing. The final macro placement can be seen in Figure 13.

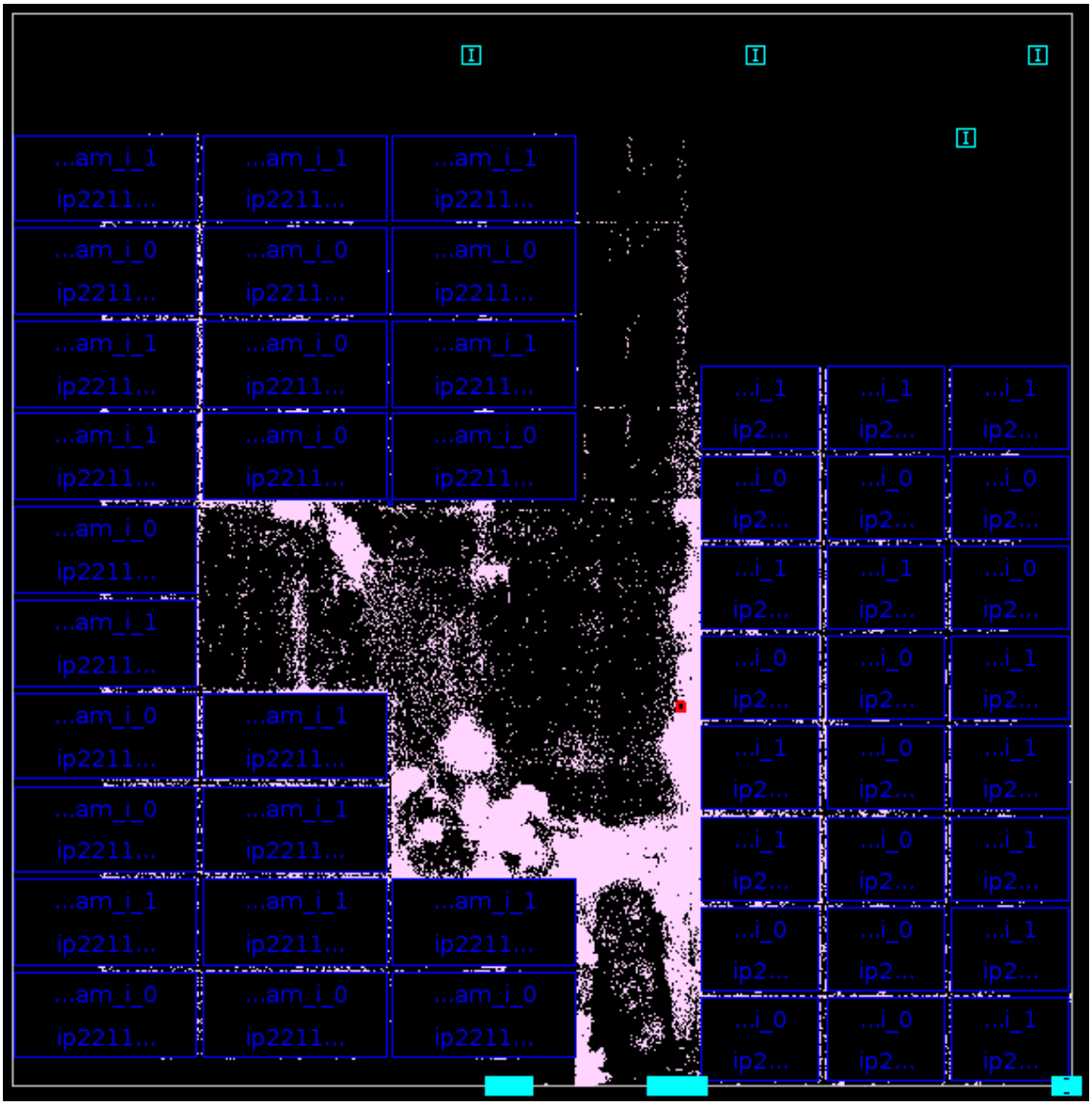
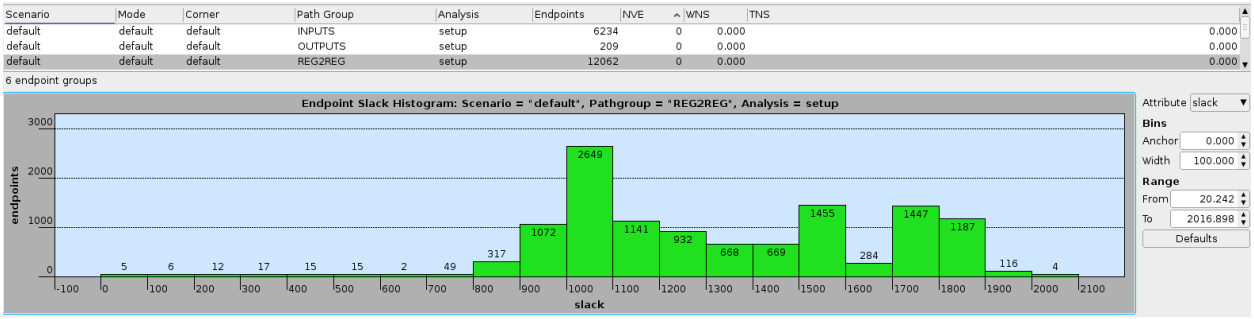


Figure13: *hr2_smem_wrapper block including push-down DIC cell (colored in red).*

The final setup timing histogram for the func_max_0p9_m40 scenario is shown below and indicates that all internal paths for this block meet timing.



2.3.3 Hr2_host_core

The host core partition houses the microcontroller unit responsible for handling external communication and moving data in and out of the design. The clocks fed into this partition are Host_core_clk, a 416MHz clock that drives the microcontroller logic; Host_core_aclk, a 416MHz clock intended to drive AHB/AXI bus communications; and the 20MHz external JTAG_CLK. This partition has 68,096 B of internal memory distributed in 16 memory macro modules. Final synthesis results for this partition are:

```
Total cell count: 197302
SEQ cell count: 28914
SEQ cell area: 44704.25316
Cell area (um2): 114603.92244
Total macro cells count: 16
WNS_INT (setup): 0 ps
TNS_INT (setup):  0.0 ps
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): 0 ps
TNS_IO (setup):  0.0 ps
Failing end points (IO - setup): 0.0
WNS_INT (hold): -164.026 ps
TNS_INT (hold):  -763326.768 ps
Failing end points (Reg2reg - hold): 19195.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  0
Max Cap Vios:  0
Worst Cap Slack (ff):  0
Max Slew Vios:  84
Worst Slew Slack:  -6.85033 ps
```

The dimensions of this block were given as 972.0 um x 453.6 um, and this block also contains a push-down DIC cell, as shown in red in Figure 14.

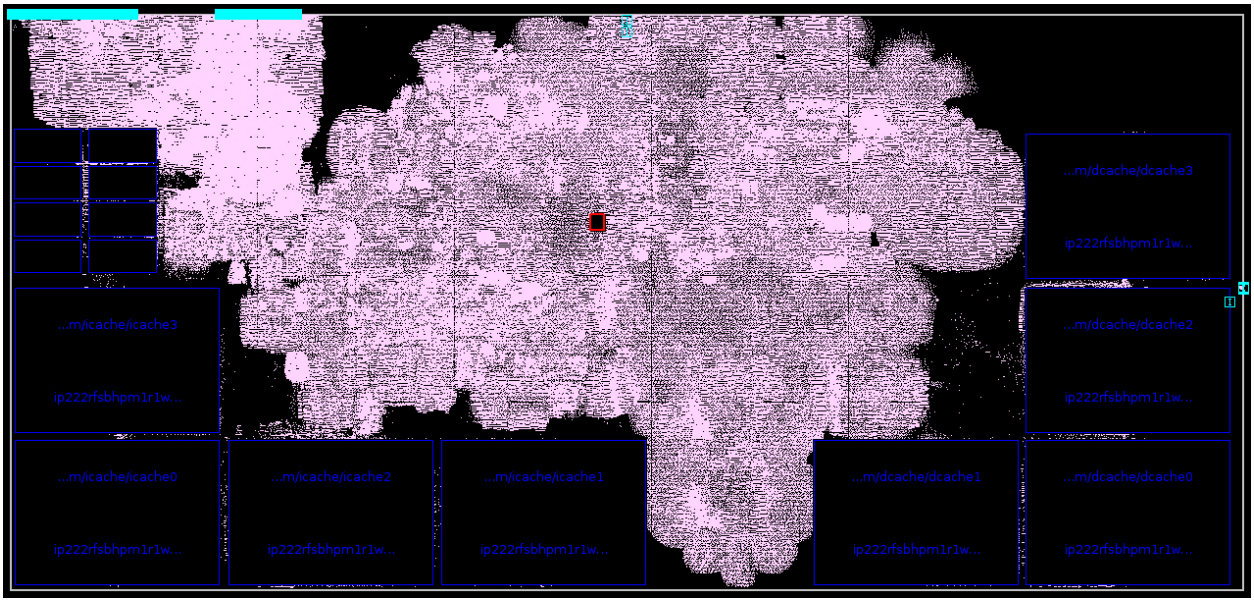
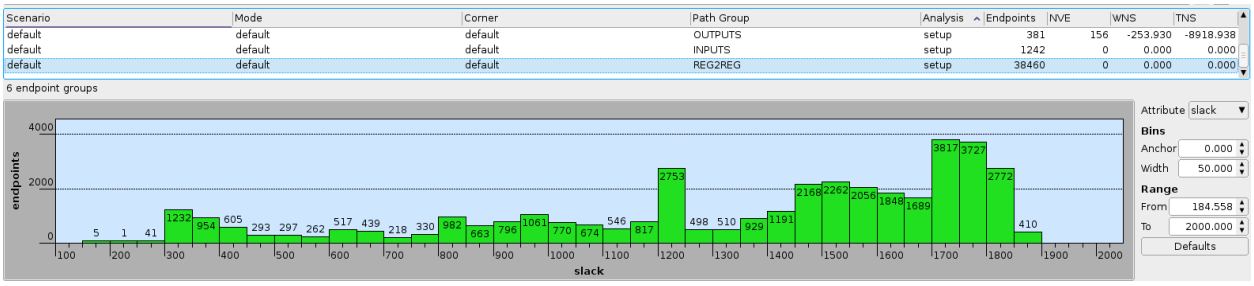


Figure14: *hr2_host_core unit, which contains the microcontroller for the design (DIC cell colored in Red)*

The final setup timing histogram from the post-layout and extraction static timing analysis tool for the scenario func_max_0p9_m40 is shown below, indicating that all internal paths meet timing requirements.



2.3.4 QbitReadoutDetector_wrapper

The circuit in this partition is responsible for the readout of up to six qubit states simultaneously, with an RF bandwidth of 500 MHz.

Final synthesis results for this partition are:

```
Total cell count: 59795
SEQ cell count: 5732
SEQ cell area: 9440.27784
Cell area (um2): 34589.56284
Gate Count: 95320
Total macro cells count: 90
```

```
WNS_INT (setup): 0 ps
TNS_INT (setup):  0.0 ps
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): -2293.716 ps
TNS_IO (setup):  -2058338.969 ps
Failing end points (IO - setup): 1843.0
WNS_INT (hold): -124.449 ps
TNS_INT (hold):  -73833.018 ps
Failing end points (Reg2reg - hold): 1325.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  1843
Max Cap Vios:  0
Worst Cap Slack (ff):  0
Max Slew Vios:  0
Worst Slew Slack:  0 ps
```

The dimensions of this block are 777.6 um x 907.2 um. It contains 90 IP blocks, and the implemented layout is shown in Figure 15.

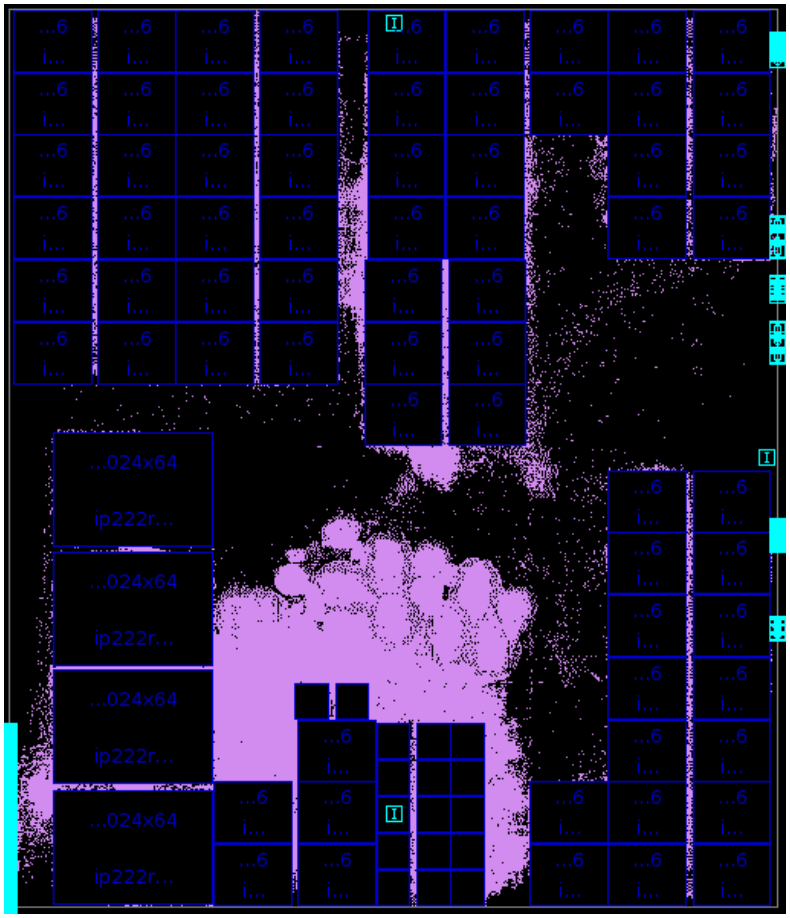
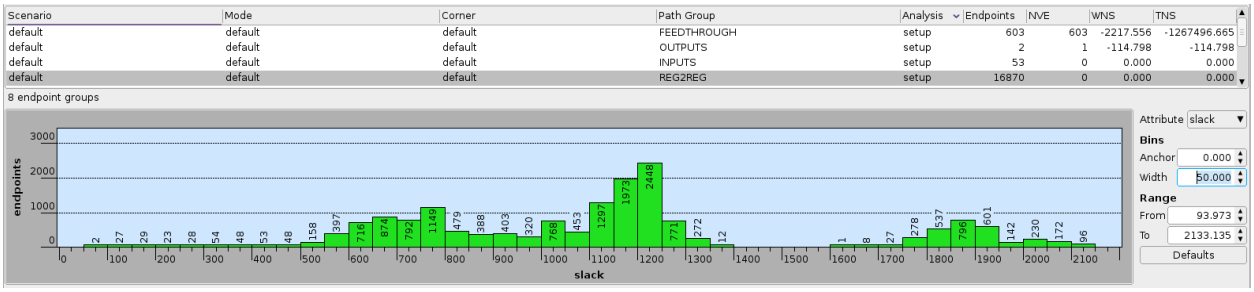


Figure15: *QbitReadoutDetector_wrapper layout*

The final setup timing histogram for this block, obtained from the static timing analysis tool after layout implementation and parasitics extraction for the scenario func_max_0p9_m40, is shown below.



2.3.5 QbitDrive_wrapper

This block contains the 16 NCOs (Numeric Controlled Oscillators) to track the phase of up to 16 qubits and the memory for the instructions and pulse envelope, as well as the LUTs (Look Up Tables) for sine and cosine waveform transformations, which feed the high-speed I/Q DACs that drive the qubits themselves.

Synthesis results for this partition are:

```
Total cell count: 116768
SEQ cell count: 4381
SEQ cell area: 6943.1418
Cell area (um2): 78173.40132
Gate Count: 215425
Total macro cells count: 52
WNS_INT (setup): 0 ps
TNS_INT (setup):  0.0 ps
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): -2639.073 ps
TNS_IO (setup):  -1543795.75 ps
Failing end points (IO - setup): 610.0
WNS_INT (hold): -98.756 ps
TNS_INT (hold):  -83010.651 ps
Failing end points (Reg2reg - hold): 1958.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  610
Max Cap Vios:  0
Worst Cap Slack (ff):  0
Max Slew Vios:  0
Worst Slew Slack:  0 ps
```

Block dimensions for this partition are 777.6 um x 793.8 um; the layout has 52 IP macro cells, and the implemented layout is shown in Figure 16.

2. HORSE RIDGE II IMPLEMENTATION

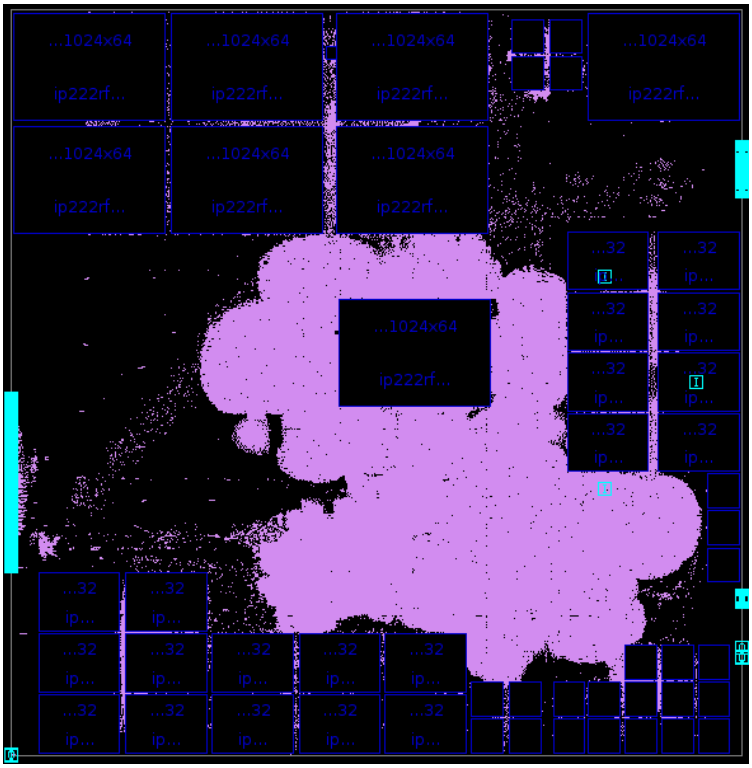
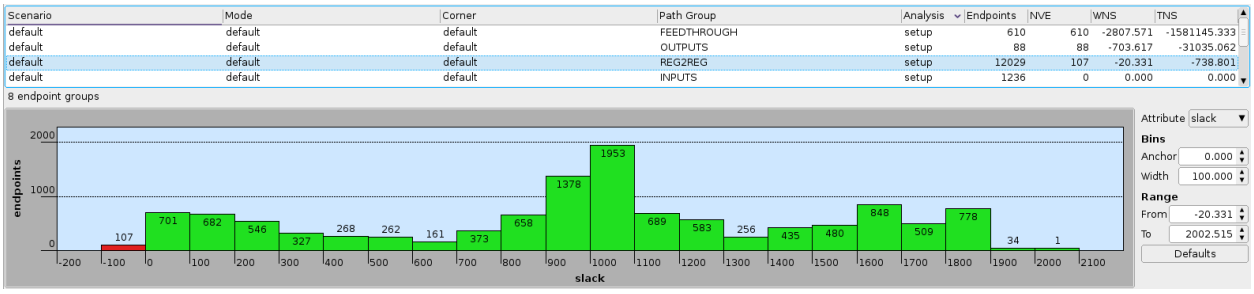


Figure16: *QbitDrive_wrapper implemented layout*

The setup timing histogram from the static timing analysis tool for this block, post-layout and after parasitic extraction, for the scenario func_max_0p9_m40 is shown below. This timing histogram shows timing violations for reg2reg paths. Since the tool uses graph-based analysis to generate this histogram, it is more pessimistic than path-based analysis, which is used to generate the timing reports for the block (an explanation of graph-based vs path-based timing can be found in Appendix D). Timing reports which use path-based analysis show that the timing for setup is clean for this partition, so regardless of the timing histogram that shows a few timing violations, the block is considered to be timing clean.



2.3.6 QbitReadoutStimulus_wrapper

This block is the multi-tone signal generator and is responsible for delivering the Tx signal to the matching network via the coupler, enabling qubit state readout in the QbitReadoutDetector_wrapper partition.

Synthesis Results for this block are:

```
Total cell count: 34525
SEQ cell count: 1621
SEQ cell area: 2532.04056
Cell area (um2): 14518.17108
Gate Count: 40008
Total macro cells count: 4
WNS_INT (setup): 0 ps
TNS_INT (setup):  0.0 ps
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): 0 ps
TNS_IO (setup):  0.0 ps
Failing end points (IO - setup): 0.0
WNS_INT (hold): -166.126 ps
TNS_INT (hold):  -47665.209 ps
Failing end points (Reg2reg - hold): 1138.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  0
Max Cap Vios:  0
Worst Cap Slack (ff):  0
Max Slew Vios:  0
Worst Slew Slack:  0 ps
```

The block dimensions for this partition are 583.2 um x 498.96 um. It contains only 4 IP macro cells, and the implemented layout is shown in Figure 17.

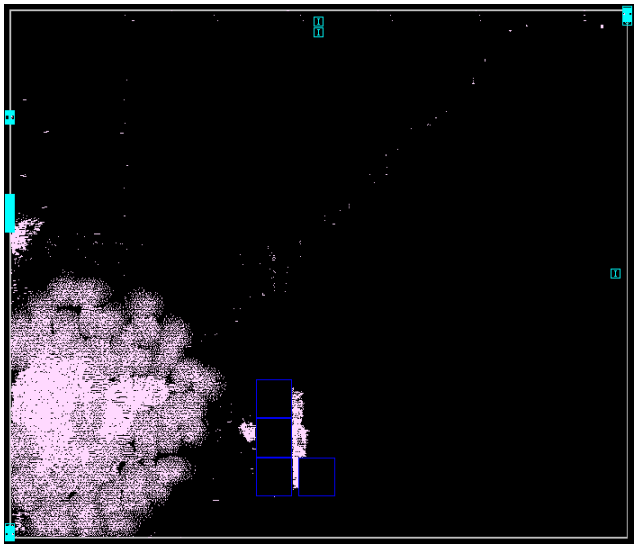
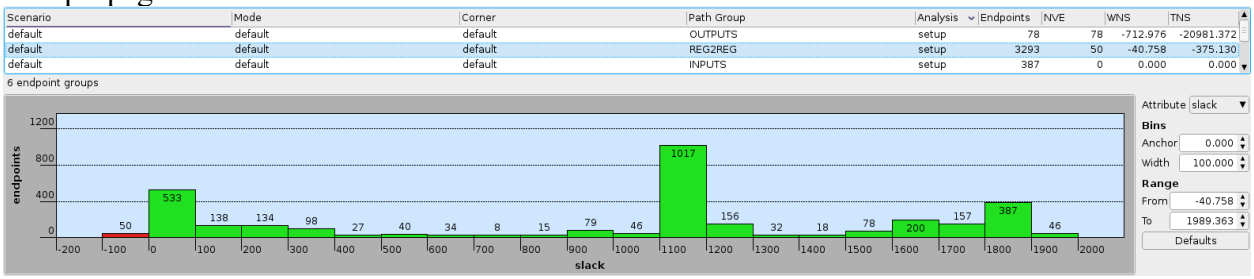


Figure17: *Qbit_readoutStimulus_wrapper implemented layout*

The setup timing histogram, as reported by the static timing analysis tool post-layout and parasitic extraction for the scenario func_max_0p9_m40, is shown below. The timing histogram shows negative slack for register-to-register paths of up to -40 ps because the tool uses graph-based timing analysis to generate it. Path-based timing reports at the block level showed timing violations of up to -11 ps for 5 cross-clock-domain paths. Since the inter-clock uncertainty is over-constrained at 160ps, it was decided to waive these violations and consider this block timing-clean. The violated paths are launching data using clock hclk (2,400ps period) at 2400ps and capturing data using accel_0_clk (1,600ps period) at 3200ps, resulting in an effective clock period of only 800ps for data propagation.



2.3.7 QbitDAC_Control_wrapper

This block contains the control logic for the 22 DACs that set the gate potentials of the qubit chip.

Synthesis Results for this block are:

```
Total cell count: 10061
  SEQ cell count: 1256
  SEQ cell area: 2019.69936
  Cell area (um2): 5604.38676
  Gate Count: 15444
  Total macro cells count: 44
  WNS_INT (setup): 0 ps
  TNS_INT (setup):  0.0 ps
```

```
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): -2187.61 ps
TNS_IO (setup):  -1849166.5 ps
Failing end points (IO - setup): 864.0
WNS_INT (hold): -93.077 ps
TNS_INT (hold):  -60690.502 ps
Failing end points (Reg2reg - hold): 1734.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  864
Max Cap Vios:  0
Worst Cap Slack (ff):  0
Max Slew Vios:  0
Worst Slew Slack:  0 ps
```

Block dimensions for this unit were given as 3790.8 um x 226.8 um, and extend the entire length of the analog partition, where the DACs are located at the bottom side of the full chip block. It contains 14 IP macro cells. The physical implementation of this block is shown in Figure 18a. To note from this partition, the output data from the registers to the analog DACs needed to be as well aligned as possible, and the propagation time had to be as short as possible. There is no timing constraint we can give the tool to force it to place and size the logic as required above, so we can't guarantee that the path delay for this output logic will be aligned. To comply with this requirement, the output flops were placed directly next to the output ports, so that the propagation time would be only the routing from the register to the port, and the route itself would be minimal. A zoom-in of one of the output port regions at the bottom of the block is shown in Figure 18b.

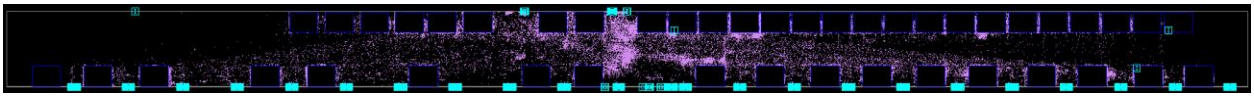


Figure18a: *QbitDAC_Control_wrapper partition layout*

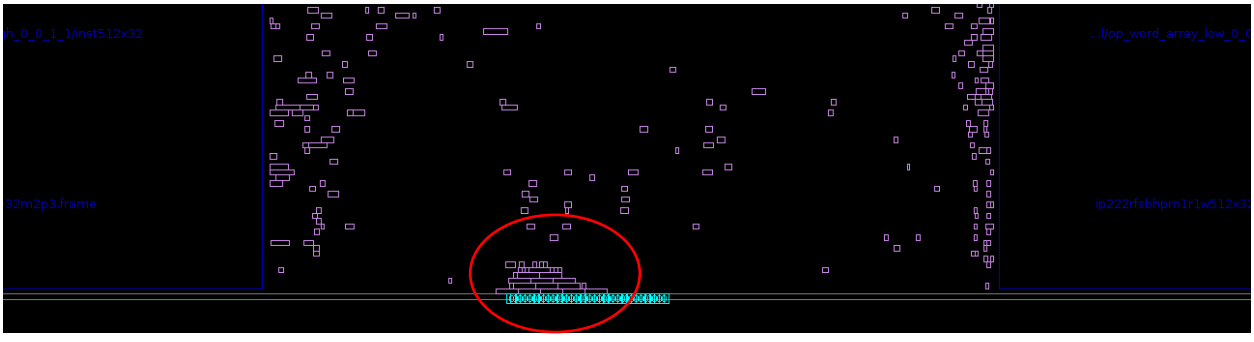
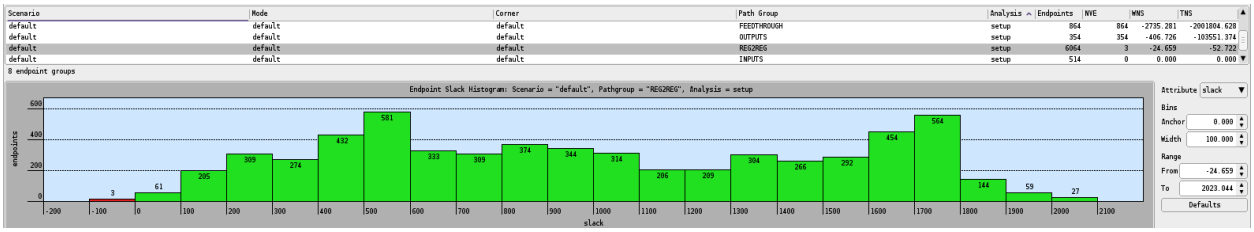


Figure18b: *QbitDAC_Control_wrapper, zoom to one of the output ports in the bottom showing the placement of the launching flops (in the red circle) next to the outputs to minimize path delay.*

The timing path histogram shown below shows a few timing violations for register-to-register paths due to the graph-based analysis done by the tool when creating the histogram. Path-based timing reports show the partition is timing clean for all internal timing paths



2.3.8 hr2_soc_top

The top level has to be synthesized because there is glue logic that interconnects the other partitions. Had this design been partitioned differently from the beginning, and all logic moved to other internal partitions, no top-level synthesis would be needed. It is important to note that the top-level synthesis shows there are no sequential elements at this level serving as glue logic. Synthesis results for this top module are:

```
Total cell count: 4967
RPT cell count: 4947
RPT cell area: 2742.48828
Total flop count (scanned and unscanned): 0
Total latch count (scanned and unscanned): 0
Total unclocked flop count (scanned and unscanned): 0
Total unclocked latch count (scanned and unscanned): 0
Cell area (um2): 2746.70676
Gate Count: 7569
Total macro cells count: 8
WNS_INT (setup): 0 ps
TNS_INT (setup):  0.0 ps
Failing end points (Reg2reg - setup): 0.0
WNS_IO (setup): -563.99 ps
TNS_IO (setup):  -1056.529 ps
Failing end points (IO - setup): 2.0
WNS_INT (hold): -441.486 ps
TNS_INT (hold):  -122539.062 ps
Failing end points (Reg2reg - hold): 334.0
WNS_IO (hold): 0.0 ps
TNS_IO (hold):  0.0 ps
Failing end points (IO - hold): 0.0
Failing Endpoints:  2
Max Cap Vios:  39
Worst Cap Slack (ff):  -2735.50024
Max Slew Vios:  421
```

```
Worst Slew Slack:  -1855.51465 ps
```

The layout of this unit is shown in Figure 19. Please note that, since this top unit hosts the other partitions, they have been loaded into the APR design as IP blocks. The pink cells shown in Figure 19 are the logic cells and buffers that need to be inserted into hr2_soc_top to take signals from one side to the other with an acceptable transition time and voltage level. The placement and sizing of buffers in this channel region are automatically optimized by the placer to comply with the timing and design rules for each cell. The full hierarchical view of the Horse Ridge chip is shown in Figure 20.

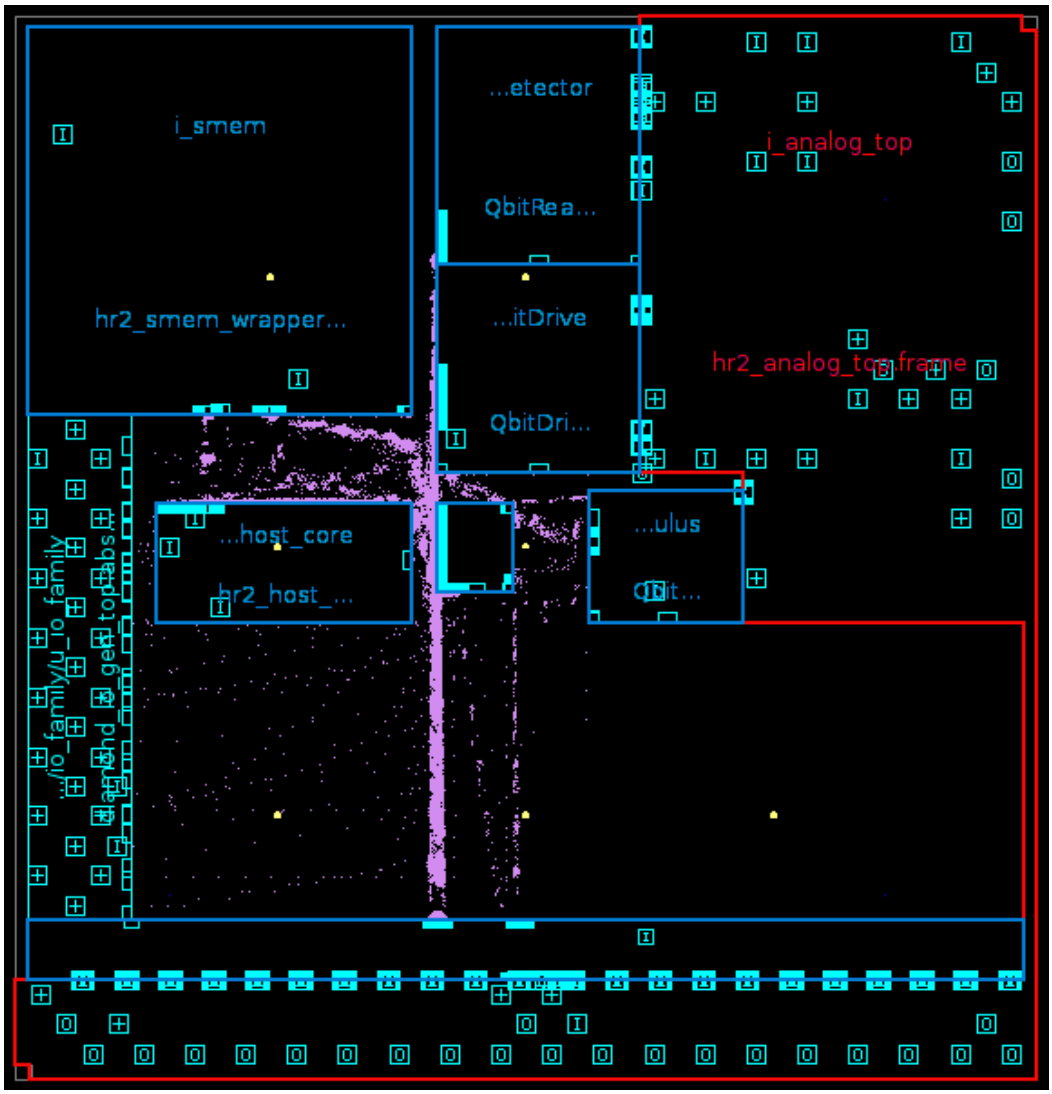


Figure19: *hr2_soc_top implementation showing loaded sub-blocks and signal buffering as well as DIC cell locations*

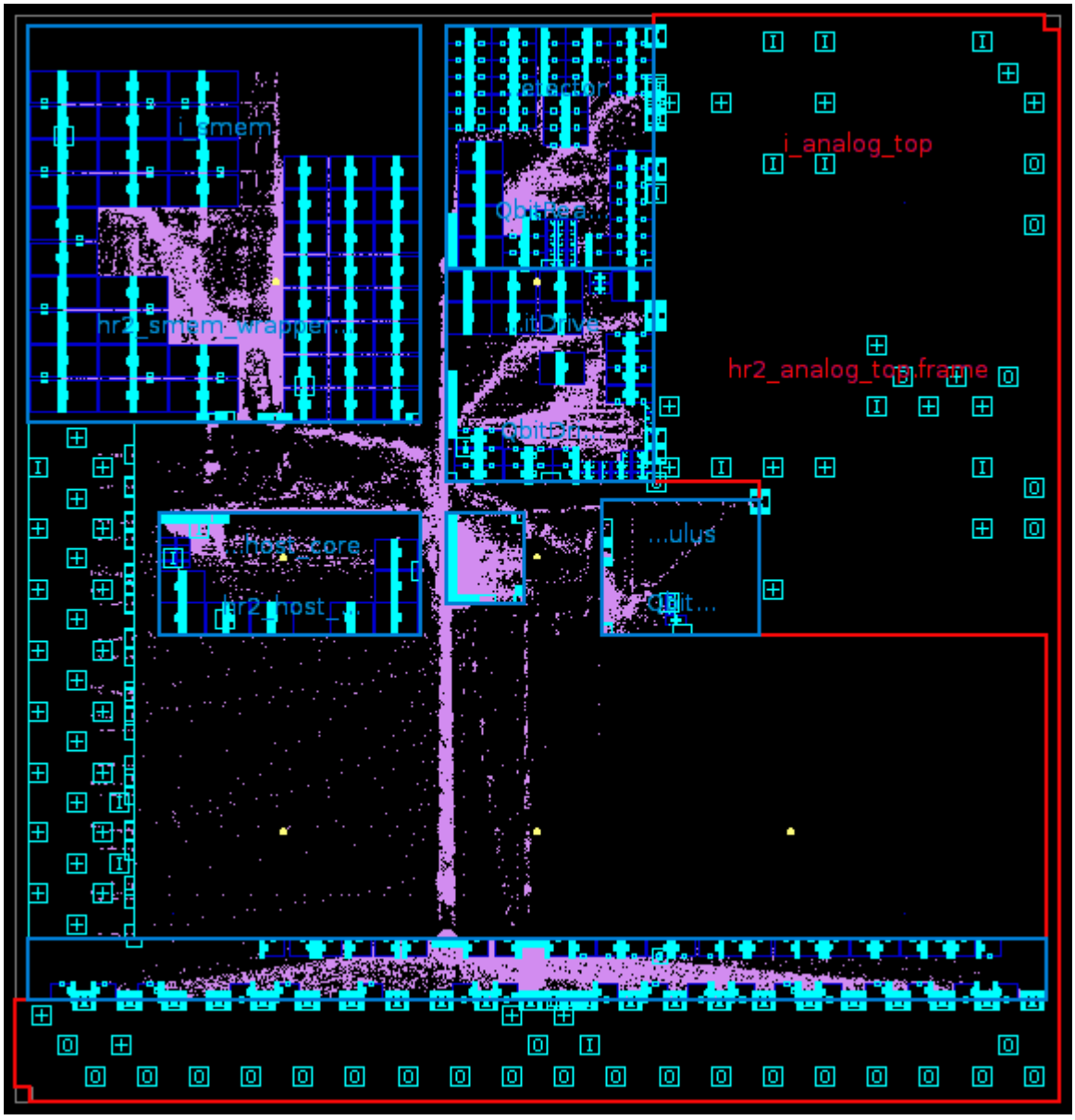


Figure20: *Full hierarchical view of the Horse Ridge design showing macro placement inside the different partitions.*

2.4 Full Chip Assembly

Please take the time to look at Figure 2 again and notice that, up until now, we have built the design up to hr2_soc_top, which resides at hierarchy level 2. The hr2_soc_top partition, which contains 100% of the design logic, must be assembled with the ER-EDM-PRS macro and

the bumps that will connect the internal signals in hr2_soc_top to the package. The ER-EDM-PRS macro is provided by Intel Foundry, so we don't have to do much with that IP other than instantiate it at the hr2_fctop level. We need to create an additional Verilog file that defines the hr2_fctop module and instantiates the ER-EDM-PRS cell, as well as hr2_soc_top. The created hr2_fctop module should have the same interface as the hr2_soc_top, and pins should be connected one-to-one to hr2_soc_top. The Verilog file, which was created for hr2_fctop, can be seen in Appendix B. The bumps for each signal are created in the IC Compiler II tool when implementing hr2_fctop, and the bump placement must meet design rules, including center-to-center spacing, bump density, etc. The Horse Ridge design contains an analog partition with RF coils, so we can't place bumps at the minimum pitch in these locations; as a result, we have bump-depopulated areas. The layout for hr2_fctop is shown in Figure 21, which highlights the bump de-populated areas.

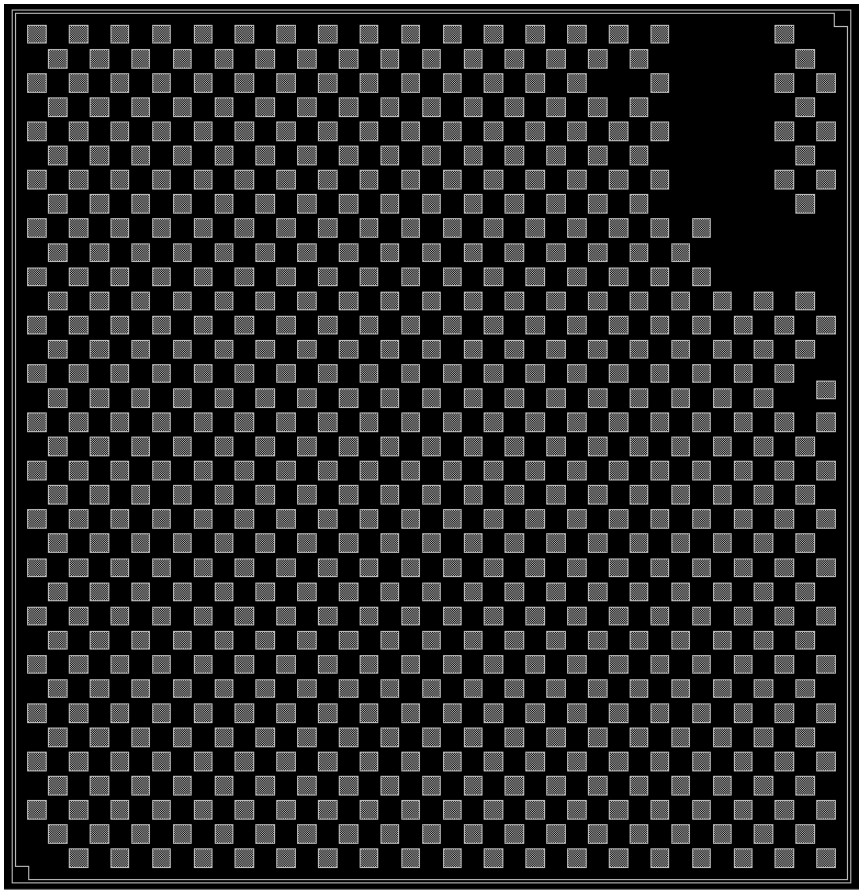


Figure21: *hr2_fctop showing the bump placement over the hr2_soc_top cell, note the bump de-populated area on the top right corner over the analog partition.*

2. HORSE RIDGE II IMPLEMENTATION

To implement a level-0 hierarchy for the foundry, we needed to obtain the product name for the Horse Ridge chip from the foundry. The top-level module name for the final database must be the product name, the Horse Ridge design must be rotated 90° clockwise (horizontal poly layer), and all DRC checks must be executed on the rotated database to meet TI quality. The rotated database is shown in Figure 22.

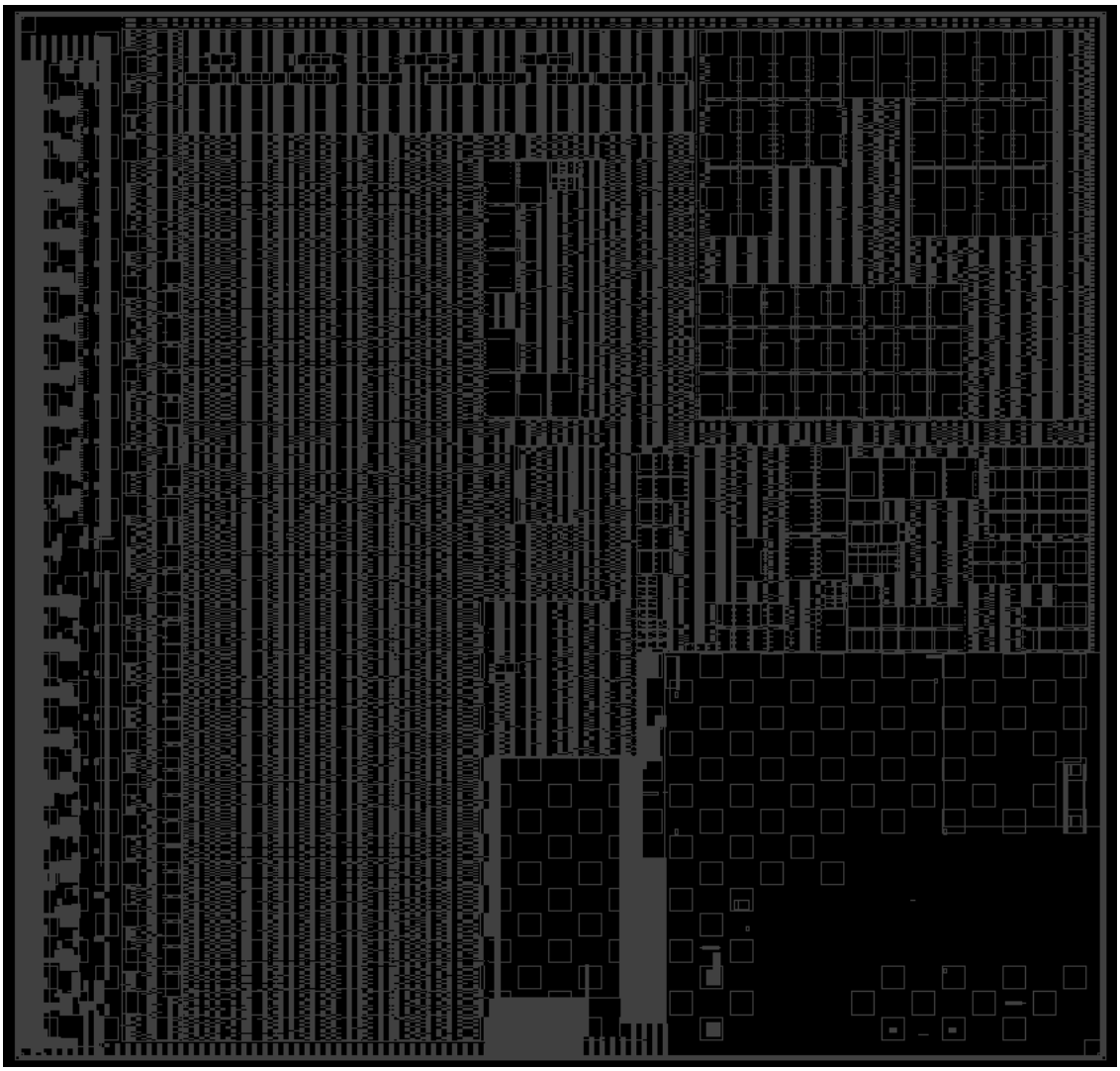


Figure22: *TI database showing 90-degree rotation of the Horse Ridge design.*

It is worth mentioning that for the delivery of this database, we had to split the tape-in database into two: the base layer database and the metal layer database, due to the fact that by the time we had to deliver the database for fabrication, hold timing convergence was still not completely finished, and more time was required to get this done.

When we converge the hold timing, the easiest way to do this is by inserting buffers, which in turn add delay to the signals and help meet the hold timing requirement. By splitting the database into two, we could no longer add buffers to fix hold timing, since doing so would alter the base layers, which were already taped in. The final hold timing fixes after splitting the tape in the database had to be done by adding capacitance to the signal nets by stretching the wires and/or adding more metal to the nets to effectively increase the capacitance and, in turn, add delay to help fix the violations.

***NOTE**: When fixing hold timing violations, if the timing paths are very sensitive to delay, it is best to address the violations by adding capacitance to the signals rather than adding buffers. Adding capacitance to the nets by metal stretching is a finer-grained way to fix timing than adding a buffer to the paths; however, it is much more time-consuming and is not recommended when the number of paths to fix is large.*

3. Design Verification

3.1 Timing Convergence

As mentioned earlier in this document, the Horse Ridge chip was intended to operate at 4 degrees Kelvin. The biggest challenge for a test chip to operate in these conditions is that no Intel product was rated to operate at such extreme temperatures, so standard cell libraries were not characterized for these temperatures. Characterizing standard cell libraries is a time-consuming task that requires dedicated library designers. Since Horse Ridge was only a "proof of concept" test chip for Intel and would not generate any revenue in the immediate future, the resources necessary for library characterization were out of reach. Since we didn't have the libraries characterized for the correct operating conditions, we needed to account for temperature effects in the semiconductor at such low temperatures such as reduced resistance, impacts to capacitance, etc which translates to faster operating digital circuits, which could cause hold timing violations and render the design unfunctional if proper mitigation measures were not implemented.

Using an analog transistor model with a specific threshold voltage, we found that the material resistance at 4 K was reduced by ~50%, while the capacitance remained unchanged. Usually, as a rule of thumb, the hold timing margin uncertainty (in addition to the hold timing requirement for the synchronous logic) is set to match the delay of two medium-sized clock buffers at the fastest operating condition, but since in this case the resistance is dropping by 50%, we decided to set the hold timing uncertainty to twice the recommended. The delay of four clock buffers operating at -40ºC was set as the hold timing margin uncertainty; this number was set to ~40ps. The use of this uncertainty can be seen in the timing constraints in Appendix C. The standard cell libraries used were limited to the same transistor Vt type used to model the operational effects at 4ºK.

To accelerate timing convergence for the design, as we did for the layout implementation, we first converged timing at the partition level for internal paths for both setup and hold, and then rolled up the converged blocks into a full-chip assembly to analyze interface timing between partitions. Interface timing is left for the full chip to analyze for two main reasons, first, the full chip will see the full interface paths in case the logic is split between two blocks and second, at partition level we had used the rule of thumb for input/output delays set for the interface paths

which, for some paths might be over constraining and for other it would be under constraining. The full chip timing accounts for the complete logic/clock path to each partition, so clock latencies for data entering and exiting the block are fully annotated, making the full-chip timing analysis for the partitions much more accurate. After analyzing the interface paths for the partitions at the full-chip level, the full-chip owner generated ECO (Engineering Change Order) scripts to be executed at the partition level to fix any hold timing violations, bad transition violations, etc., primarily for the interface paths. Partition owners take the ECO files generated by the full chip owner, execute the timing ECOs in each partition, and roll up the implemented timing fixes for full-chip analysis once again. This cycle continues until all setup/hold/drc violations have been fixed.

3.2 DRC & LVS Verification

3.2.1 Stitching the design together

After implementing the different layouts for each partition in parallel and assembling the full chip in the automatic place and route tool, how can we accelerate the layout DRC (Design Rule Checks) and LVS (Layout vs Schematic) cleanup and verification for the full chip? The answer is layout merging. In layout merging, we get one top layout (hr2_fctop) which contains empty instances for each of the lower level partitions; In addition, we also have the individual layout for each of the partitions which were executed in parallel so we can combine these layouts to create one single output oasis file which make up the full design by replacing the empty instances at the top level with the full partition layouts. By merging, we can run DRC checks on the full-chip merged layout database, then split the results by sub-block and work in parallel to clean the DRCs. With respect to LVS cleanup, each partition must submit a clean LVS database for full chip integration, and the top-level integrator simply verifies that LVS is clean at the top level. With this methodology, if there are any LVS issues shown at the top, they can be isolated either to a specific partition, the interface of the partition to the top level, or to the top level, which might be adding shapes over a partition in a non-allowed layer.

Once the sub-blocks have finished cleaning DRCs at the block level, the block owners deliver their blocks for full chip assembly, and we then merge the sub-blocks as well as all the collaterals

involved (memories, analog blocks, standard cells, etc) to check DRCs at the full chip level iteratively until all DRCs have been cleaned up. Figure23 shows a simplified diagram of the merging process for the partitions. It is important to note that when merging, the order in which the different collaterals are read is critical to avoid layout version conflicts, since the last-read layout is the one kept in the merge. In other words, we would merge as follows:

1. Full chip layout (with empty partition instances)
2. Partition level full layouts
3. Macro cells (memories, PLL, etc)
4. Standard Cells
5. TI required cells

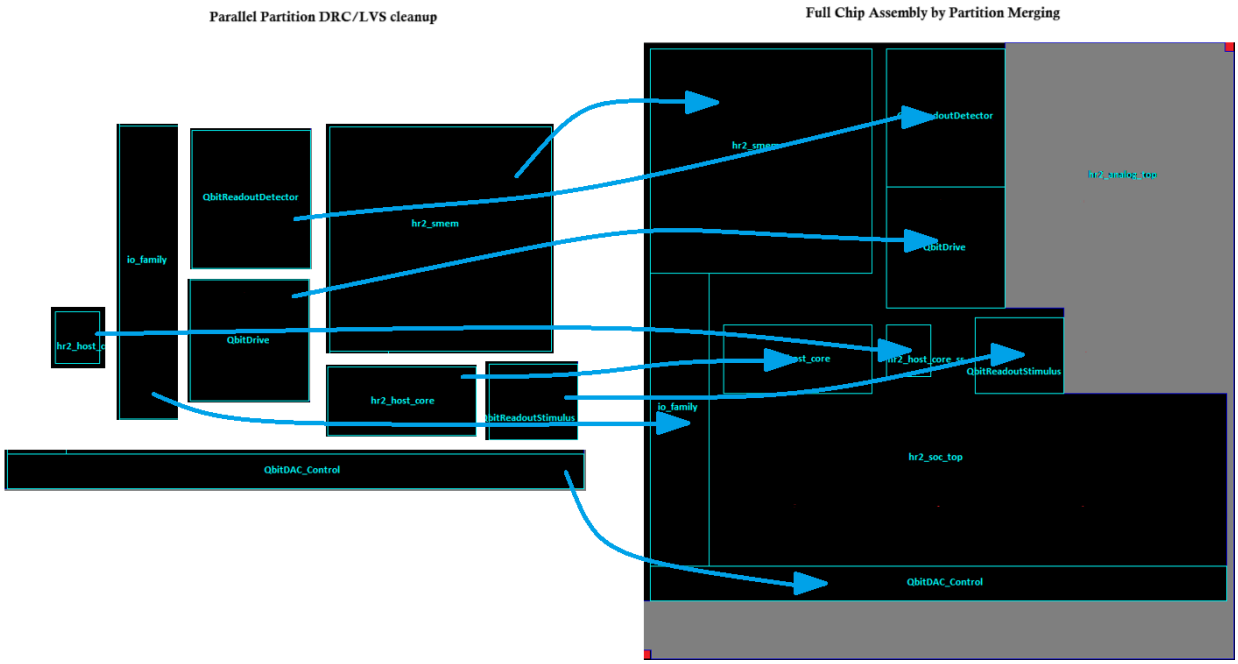


Figure23: *Layout Merging cartoon for DRC/LVS verification. On the right, we have the different partitions that make up the design and that can be worked on in parallel. On the left, the design partitions are assembled at the top level.*

A similar approach is taken with respect to the full chip netlist, which needs to be created to run LVS at the full chip level using the sub-block partitions. Each partition has its own netlist, which needs to be merged with the sub-design netlist files (memories, PLLs, standard cells, etc.). At the

full-chip level, the full-chip netlist must then be merged with the partition-level netlists to obtain a single top-level netlist. It is worth mentioning that netlist merging happens in the opposite order than layout merging since the first definition read is the definition that is kept, so in this sense, the merging of the netlist would have the following order:

1. TI required cells
2. Standard cells
3. Macro cells
4. Partition level full netlist
5. Full chip netlist

LVS needs to be checked and cleaned first at the partition level, then at the full-chip level, to help isolate any LVS issues that may occur. Partitions must be LVS-clean before it makes sense to check LVS at the full-chip level.

Common issues that can be seen at full chip and are hard to debug include shorts between power and ground. These are particularly difficult to fix since the tools flag the entire power grid as the source of the short, making it difficult to pinpoint these kinds of shorts from a full-chip perspective. If LVS has been cleaned at the partition level first and we still see shorts at the full chip level, then we can assume the LVS issue is caused by the integration of the sub-blocks, making it easier to identify the source and fix it.

3.2.2 Verification runs for Large Designs

The Horse Ridge design measures ~16 mm^2 (3924.72 um x 4084.92 um). By silicon area, this is not a very large design, but it is large enough that running DRC checks on a single server will take over a week to complete. To avoid long run times and high memory usage, we decided to run our checks on Intel's NetBatch pool of servers. In NetBatch, we can launch several DRC checks in parallel across different machines to accelerate overall runtime and results turnaround time. Horse Ridge had to launch 89 runsets (each consisting of several checks) whenever a full chip was built, and thanks to NetBatch, these runs could be executed overnight to get results the next day.

Once the full chip DRC/LVS runs have been executed, partition owners can load the results into their blocks and clean any violations that fall within their partitions. Now, if you remember from section 2.5, we said that the full chip needs to be rotated 90° before delivery to the foundry; hence, the DRC checks also need to run on the 90° rotated database. YES, the DRC results are also rotated by 90°, so in order to load the failure markers into any given partition, the partition owner needs to:

1) Un-rotate the coordinates for each of the violation markers
2) Select only those violations that correspond to his/her partition
3) Translate the full chip coordinates to partition coordinates
4) Create markers for each of the violations in the partition view.
5) Fix the violation

There is no easy way to do this. In addition, this needs to be done for any DRC check that caused a violation, and if you remember, for this project, we ran 89 runsets. To help with this task, we had to implement TCL scripts that helped with this task. The TCL scripts used to rotate the violations can't be shared due to proprietary software code restrictions.

A historical view of how DRC cleanup progressed in this design is shown in Figure 24. Please note how, at the beginning of the cleanup process, some DRCs increased while others decreased. Many DRCs could not be cleaned up until the final stages of implementation, and some were okay to be waived since this was only a test chip and not a product.

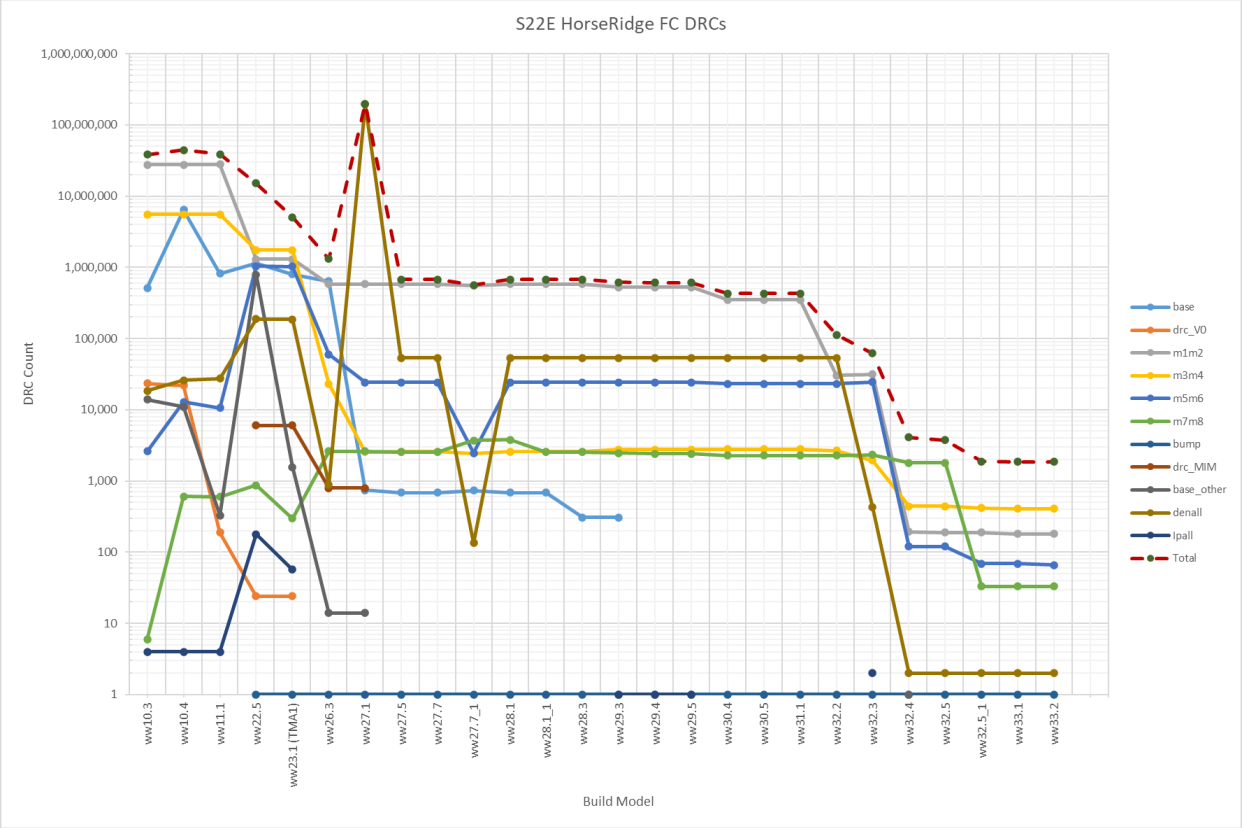


Figure24: *Logarithmic Scale chart showing DRC cleanup progress for the Horse Ridge II test chip.*

3.3 Formal Equivalence Verification

During the design implementation, our test chip underwent several transformations. It started as Verilog RTL code, then this code was transformed into Boolean logic equations, which were optimized. The design was then mapped to standard-cell logic for a specific technology and further optimized. After logical optimizations, the design had to undergo timing optimizations to meet the required setup/hold timing requirements through the insertion of buffers, splitting/cloning of timing paths, etc. Even if the design is DRC/LVS and timing clean, this does not guarantee that the function was preserved as it was originally intended. To verify that the function was preserved through the optimization and implementation process, we have two options: 1) circuit simulation using SPICE and the injection of simulation vectors, and 2) formal verification.

3. DESIGN VERIFICATION

In circuit simulation, we need to load the functional models for all standard cells and macros in addition to the full gate-level netlist, inject test vectors, and verify that the outputs are as expected. As designs get larger, it becomes increasingly difficult and time-consuming to run a full verification of the design through SPICE simulations. Usually, circuit simulation is only done for certain critical analog operations, and full verification is not an option.

In formal verification, we use mathematical/logical algorithms to compare the final design netlist against the input reference Verilog design. Since formal verification does not require test vectors and considers only logical functions during comparison, it is much faster than simulation and provides 100% comparison coverage when it runs without abort points.

During the execution of the Horse Ridge project, the RTL team decided to disable part of the functions implemented in the design but kept the logic code for them in the RTL, leaving the inputs/outputs dangling (not connected). During formal verification runs, the disabled logic caused the tool to report “Abort Points”. Abort points during formal verification are reported by the tool when the comparing algorithms can’t determine if the logic cones being analyzed have the same function as the coded RTL. A logic cone is the method used by the formal verification tool to compare functional paths. A timing endpoint is the tip of the cone and expands backward to all its start points; all combinational logic associated with a timing endpoint is considered part of the logic cone. Abort points are a common problem encountered with complex logic, such as multipliers and dividers. The tool vendor's recommendation when you encounter abort points is to increase the comparison effort in the tool by using more time-consuming algorithms to try to match the logic. When we tried increasing the comparison effort for the formal verification tool, we still got abort points, and nothing we did would indicate whether the logic was passing or failing.

In this particular case, where formal verification was not passing or failing for a very defined section of the code, the decision was made to conduct a gate-level simulation of the affected logic to make sure that the outputs behaved as expected and that having dangling logic in the RTL did not affect surrounding logic during synthesis optimizations.

3. DESIGN VERIFICATION

After the RTL team conducted gate-level simulations and analyzed the results, they concluded that the floating logic did not affect the surrounding logic in the design, and that it was OK to waive the abort points for the formal verification runs.

Conclusions

Throughout this document, I have attempted to preserve the hard work of Intel's silicon prototyping team in implementing the digital logic on Intel's Cryo CMOS SoC Horse Ridge II test chip. While much of the data presented in this document is intended to show the results obtained in the implementation, an effort has also been made to recall the hurdles encountered along the way and the solutions implemented to overcome technical challenges. Insights have been provided on how to overcome timing issues, such as not having complete timing constraints for a block during synthesis, where we can use the 2/3-of-a-clock-cycle rule of thumb for input/output delays. The solution for not having the correct characterization for the operating temperature for the standard cells was overcome by understanding the consequence of operating at 4ºK and at least having a model of 1 transistor operating at 4ºK temperature, as well as implementing additional timing margins for the hold timing checks, which would be the most critical. The methodology for full-chip layout implementation was described, and the merging of layouts for the full-chip assembly was introduced. Divide-and-conquer techniques for layout/timing verification were also discussed and described as best as possible. Finally, formal verification was introduced, along with the issues encountered during verification of the implementation, and the solutions to these roadblocks were also explained.

This document serves its purpose by preserving the hard work of Intel's Silicon Prototyping Team in implementing the Horse Ridge II Cryo CMOS SoC and detailing the issues encountered along the way, as well as the solutions implemented to overcome the roadblocks. This document should serve as a guide for novice physical designers to tackle many of the problems encountered when building a digital ASIC using automatic place-and-route tools.

Appendix

A. QTM SCRIPT USED TO GENERATE THE TIMING INFORMATION FOR THE ANALOG PORTION OF THE DESIGN.

```
#LOAD REFERENCE LIBRARY AND INITIALIZE QTM
read_db b15_hn_p1222_11x1r1v1_psss_0.900v_-40c_max_ccst_pdn.ldb
create_qtm_model hr2_analog_top
set_qtm_technology -library "b15_hn_p1222_11x1r1v1_psss_0.900v_-40c_max_ccst_pdn"
set_qtm_global_parameter -param setup -value 10.0
set_qtm_global_parameter -param hold  -value 10.0
set_qtm_global_parameter -param clk_to_output -value 50.0

#DEFINE CELLS DRIVING OUTPUTS
create_qtm_drive_type -lib_cell b15bfn000ah1n12x5 -output_pin o drive_clk_0p5
create_qtm_drive_type -lib_cell b15inv000ah1n20x5 -output_pin o1 drive_clk_2p5
create_qtm_drive_type -lib_cell b15bfn000ah1n04x5 -output_pin o drive_data_i_rx
create_qtm_drive_type -lib_cell b15bfn000ah1n04x5 -output_pin o drive_data_q_rx

#DEFINE CELLS USED AS CAPACITIVE LOADS FOR INPUTS
create_qtm_load_type -lib_cell b15fpn000ah1n02x5  -input_pin d load_b15fpn000ah1n02x5_d
create_qtm_load_type -lib_cell b15nor002ah1n02x5  -input_pin a load_b15nor002ah1n02x5_a
create_qtm_load_type -lib_cell b15nor002ah1n02x5  -input_pin b load_b15nor002ah1n02x5_b
create_qtm_load_type -lib_cell b15bfn000ah1n04x5  -input_pin a load_b15bfn000ah1n04x5_a
create_qtm_load_type -lib_cell b15inv000ah1n04x5  -input_pin a load_b15inv000ah1n04x5_a
create_qtm_load_type -lib_cell b15nand02ah1n04x5  -input_pin a load_b15nand02ah1n04x5_a
create_qtm_load_type -lib_cell b15bfn001ah1n08x5  -input_pin a load_b15bfn001ah1n08x5_a
create_qtm_load_type -lib_cell b15nand02ah1n02x5  -input_pin a load_b15nand02ah1n02x5_a
create_qtm_load_type -lib_cell b15nor002ah1n04x5  -input_pin a load_b15nor002ah1n04x5_a
create_qtm_load_type -lib_cell b15nand02ah1n02x5  -input_pin b load_b15nand02ah1n02x5_b

#CREATE OUTPUT PORTS
create_qtm_port -type output {aitestmux_ana_pad_nom}
create_qtm_port -type output {avrf_drive_ana_pad_nom}
create_qtm_port -type output {avrf_stim_ana_pad_nom}
create_qtm_port -type output {avtestbb_drive_ana_pad_nom}
create_qtm_port -type output {avtestmux_ana_pad_nom}
create_qtm_port -type output {avout0_acdac_ana_pad_nom}
create_qtm_port -type output {avout1_acdac_ana_pad_nom}
create_qtm_port -type output {avout2_acdac_ana_pad_nom}
create_qtm_port -type output {avout3_acdac_ana_pad_nom}
create_qtm_port -type output {avout4_acdac_ana_pad_nom}
create_qtm_port -type output {avout5_acdac_ana_pad_nom}
create_qtm_port -type output {avout6_acdac_ana_pad_nom}
create_qtm_port -type output {avout7_acdac_ana_pad_nom}
create_qtm_port -type output {avout8_acdac_ana_pad_nom}
create_qtm_port -type output {avout9_acdac_ana_pad_nom}
create_qtm_port -type output {avout10_acdac_ana_pad_nom}
create_qtm_port -type output {avout11_acdac_ana_pad_nom}
create_qtm_port -type output {avout12_acdac_ana_pad_nom}
create_qtm_port -type output {avout13_acdac_ana_pad_nom}
create_qtm_port -type output {avout14_acdac_ana_pad_nom}
create_qtm_port -type output {avout15_acdac_ana_pad_nom}
create_qtm_port -type output {avout16_acdac_ana_pad_nom}
create_qtm_port -type output {avout17_acdac_ana_pad_nom}
create_qtm_port -type output {avout18_acdac_ana_pad_nom}
create_qtm_port -type output {avout19_acdac_ana_pad_nom}
create_qtm_port -type output {avout20_acdac_ana_pad_nom}
create_qtm_port -type output {avout21_acdac_ana_pad_nom}
create_qtm_port -type output {clk_0p5}
create_qtm_port -type output {clk_2p5}
create_qtm_port -type output {data_q_rx[7:0]}
create_qtm_port -type output {data_i_rx[7:0]}

#CREATE INOUT PORTS
create_qtm_port -type inout {athermdiodebjt_ana_pad_nom}
create_qtm_port -type inout {athermdiodeesd_ana_pad_nom}
create_qtm_port -type inout {vcc_a_1p0_nom_acdac_driver}
```

```
create_qtm_port -type inout {vcc_a_1p0_nom_acdac_ms}
create_qtm_port -type inout {vcc_a_1p0_nom_drive_bb}
create_qtm_port -type inout {vcc_a_1p0_nom_drive_ms}
create_qtm_port -type inout {vcc_a_1p0_nom_drive_pa}
create_qtm_port -type inout {vcc_a_1p0_nom_rx_bb}
create_qtm_port -type inout {vcc_a_1p0_nom_rx_lna}
create_qtm_port -type inout {vcc_a_1p0_nom_rx_ms}
create_qtm_port -type inout {vcc_a_1p0_nom_stim_bb}
create_qtm_port -type inout {vcc_a_1p0_nom_stim_ms}
create_qtm_port -type inout {vcc_d_0p9_nom}
create_qtm_port -type inout {vssx}

#CREATE INPUT PORTS
create_qtm_port -type input {aclk_ana_pad_nom}
create_qtm_port -type input {avlo_drive_ana_pad_nom}
create_qtm_port -type input {avrfn_rx_ana_pad_nom}
create_qtm_port -type input {avrfp_rx_ana_pad_nom}
create_qtm_port -type input {clock_enable}
create_qtm_port -type input {test_mux_select[7:0]}
create_qtm_port -type input {backup_bits_toplevel[31:0]}
create_qtm_port -type input {reset_clkdiv_rx}
create_qtm_port -type input {reset_fsm_rx}
create_qtm_port -type input {bb_i_en_rx}
create_qtm_port -type input {bb_q_en_rx}
create_qtm_port -type input {bias_select_rx}
create_qtm_port -type input {lna_en_fb_rx}
create_qtm_port -type input {lna_en_main_rx}
create_qtm_port -type input {mx_clk_en_rx}
create_qtm_port -type input {mx_en_rx}
create_qtm_port -type input {sel_clk_dac_rx}
create_qtm_port -type input {select_divide_4or8_rx}
create_qtm_port -type input {mx_q_deg_r_tune_rx[3:0]}
create_qtm_port -type input {bias_tune_ibias_rx[3:0]}
create_qtm_port -type input {bb_i_gain_stg3_rx[3:0]}
create_qtm_port -type input {lna_r_tune_rx[3:0]}
create_qtm_port -type input {bb_q_gain_stg3_rx[3:0]}
create_qtm_port -type input {filter_q_tune_ibias_rx[3:0]}
create_qtm_port -type input {filter_q_tune_res_rx[2:0]}
create_qtm_port -type input {filter_i_tune_ca_rx[3:0]}
create_qtm_port -type input {adc_q_clk_div_rx[2:0]}
create_qtm_port -type input {bb_q_gain_stg1_rx[3:0]}
create_qtm_port -type input {adc_q_bias_rx[3:0]}
create_qtm_port -type input {mx_i_deg_r_tune_rx[3:0]}
create_qtm_port -type input {mx_i_r_tune_rx[3:0]}
create_qtm_port -type input {backup_bits_rx[31:0]}
create_qtm_port -type input {lna_bias_ctr_main_rx[3:0]}
create_qtm_port -type input {filter_i_tune_ibias_rx[3:0]}
create_qtm_port -type input {bb_i_gain_stg1_rx[3:0]}
create_qtm_port -type input {adc_q_smpdly_rx[4:0]}
create_qtm_port -type input {mx_q_r_tune_rx[3:0]}
create_qtm_port -type input {adc_i_smpdly_rx[4:0]}
create_qtm_port -type input {filter_q_tune_cb_rx[3:0]}
create_qtm_port -type input {filter_q_tune_ca_rx[3:0]}
create_qtm_port -type input {bb_i_gain_stg2_rx[3:0]}
create_qtm_port -type input {filter_i_tune_res_rx[2:0]}
create_qtm_port -type input {adc_i_bias_rx[3:0]}
create_qtm_port -type input {adc_i_clk_div_rx[2:0]}
create_qtm_port -type input {bb_q_gain_stg2_rx[3:0]}
create_qtm_port -type input {filter_i_tune_cb_rx[3:0]}
create_qtm_port -type input {mx_bias_ctr_rx[3:0]}
create_qtm_port -type input {lna_bias_ctr_fb_rx[3:0]}
create_qtm_port -type input {trig_dac_acdac}
create_qtm_port -type input {din_dac10_acdac[11:0]}
create_qtm_port -type input {din_dac0_acdac[11:0]}
create_qtm_port -type input {din_dac13_acdac[11:0]}
create_qtm_port -type input {din_dac15_acdac[11:0]}
create_qtm_port -type input {din_dac1_acdac[11:0]}
create_qtm_port -type input {din_dac5_acdac[11:0]}
create_qtm_port -type input {din_dac3_acdac[11:0]}
```

```
create_qtm_port -type input {din_dac4_acdac[11:0]}
create_qtm_port -type input {din_dac9_acdac[11:0]}
create_qtm_port -type input {din_dac7_acdac[11:0]}
create_qtm_port -type input {din_dac18_acdac[11:0]}
create_qtm_port -type input {din_dac11_acdac[11:0]}
create_qtm_port -type input {din_dac19_acdac[11:0]}
create_qtm_port -type input {din_dac16_acdac[11:0]}
create_qtm_port -type input {din_dac20_acdac[11:0]}
create_qtm_port -type input {din_dac17_acdac[11:0]}
create_qtm_port -type input {din_dac14_acdac[11:0]}
create_qtm_port -type input {din_dac12_acdac[11:0]}
create_qtm_port -type input {din_dac2_acdac[11:0]}
create_qtm_port -type input {din_dac8_acdac[11:0]}
create_qtm_port -type input {din_dac21_acdac[11:0]}
create_qtm_port -type input {din_dac6_acdac[11:0]}
create_qtm_port -type input {bias_select_acdac}
create_qtm_port -type input {bias_tune_ibias_acdac[3:0]}
create_qtm_port -type input {tune_gm_acdac[87:0]}
create_qtm_port -type input {backup_bits_acdac[127:0]}
create_qtm_port -type input {tune_cap_acdac[87:0]}
create_qtm_port -type input {enable_dac_acdac[21:0]}
create_qtm_port -type input {tune_op_acdac[87:0]}
create_qtm_port -type input {data_i_drive[9:0]}
create_qtm_port -type input {data_q_drive[9:0]}
create_qtm_port -type input {bb_out_enable_drive}
create_qtm_port -type input {bias_select_drive}
create_qtm_port -type input {en_class_a_drive}
create_qtm_port -type input {en_class_c_drive}
create_qtm_port -type input {lo_buffer_en_drive}
create_qtm_port -type input {mx_en_drive}
create_qtm_port -type input {off_sw_drive}
create_qtm_port -type input {sel_clk_dac_drive}
create_qtm_port -type input {filter_i_tune_cb_drive[4:0]}
create_qtm_port -type input {dac_i_tune_iunit_drive[3:0]}
create_qtm_port -type input {dac_i_tune_ibleed_drive[3:0]}
create_qtm_port -type input {filter_q_tune_cb_drive[4:0]}
create_qtm_port -type input {filter_i_tune_ibias_drive[4:0]}
create_qtm_port -type input {lo_i_buffer_res_tune_drive[2:0]}
create_qtm_port -type input {dac_q_tune_ibleed_drive[3:0]}
create_qtm_port -type input {filter_q_tune_ivga_drive[3:0]}
create_qtm_port -type input {vga_i_tune_test_drive[3:0]}
create_qtm_port -type input {vga_i_tune_gain_drive[3:0]}
create_qtm_port -type input {backup_bits_drive[31:0]}
create_qtm_port -type input {filter_i_tune_ca_drive[4:0]}
create_qtm_port -type input {cas_bias_res_ctr_drive[2:0]}
create_qtm_port -type input {mixer_mn_res_tune_drive[2:0]}
create_qtm_port -type input {class_a_gain_drive[7:0]}
create_qtm_port -type input {filter_q_tune_imix_drive[3:0]}
create_qtm_port -type input {filter_i_tune_imix_drive[3:0]}
create_qtm_port -type input {filter_q_tune_ca_drive[4:0]}
create_qtm_port -type input {bias_tune_ibias_drive[3:0]}
create_qtm_port -type input {filter_i_tune_ivga_drive[3:0]}
create_qtm_port -type input {lo_q_buffer_bias_ctr_drive[3:0]}
create_qtm_port -type input {class_c_gain_drive[7:0]}
create_qtm_port -type input {vga_q_tune_test_drive[3:0]}
create_qtm_port -type input {lo_i_buffer_bias_ctr_drive[3:0]}
create_qtm_port -type input {class_c_bias_ctr_drive[6:0]}
create_qtm_port -type input {vga_q_tune_gain_drive[3:0]}
create_qtm_port -type input {dac_q_tune_iunit_drive[3:0]}
create_qtm_port -type input {cas_bias_i_ctr_drive[3:0]}
create_qtm_port -type input {lo_q_buffer_res_tune_drive[2:0]}
create_qtm_port -type input {filter_q_tune_ibias_drive[4:0]}
create_qtm_port -type input {class_a_bias_ctr_drive[6:0]}
create_qtm_port -type input {data_stim[9:0]}
create_qtm_port -type input {bias_select_stim}
create_qtm_port -type input {buffer_en_stim}
create_qtm_port -type input {sel_clk_dac_stim}
create_qtm_port -type input {filter_tune_cb_stim[4:0]}
create_qtm_port -type input {backup_bits_stim[31:0]}
```

```
create_qtm_port -type input {filter_tune_ibias_stim[4:0]}
create_qtm_port -type input {dac_tune_ibleed_stim[3:0]}
create_qtm_port -type input {buffer_tune_gain_stim[4:0]}
create_qtm_port -type input {filter_tune_imix_stim[3:0]}
create_qtm_port -type input {filter_tune_ivga_stim[3:0]}
create_qtm_port -type input {vga_tune_gain_stim[3:0]}
create_qtm_port -type input {vga_tune_test_stim[3:0]}
create_qtm_port -type input {bias_tune_ibias_stim[3:0]}
create_qtm_port -type input {dac_tune_iunit_stim[3:0]}
create_qtm_port -type input {filter_tune_ca_stim[4:0]}

#ASSIGN DRIVE CELLS TO OUTPUT PINS
set_qtm_port_drive -type drive_clk_2p5 {clk_2p5}
set_qtm_port_drive -type drive_clk_0p5 {clk_0p5}
set_qtm_port_drive -type drive_data_i_rx {data_i_rx[7:0]}
set_qtm_port_drive -type drive_data_q_rx {data_q_rx[7:0]}
set_qtm_port_drive    -type    drive_data_q_rx    {aitestmux_ana_pad_nom    avrf_drive_ana_pad_nom
avrf_stim_ana_pad_nom  avtestbb_drive_ana_pad_nom  avtestmux_ana_pad_nom  avout0_acdac_ana_pad_nom
avout1_acdac_ana_pad_nom            avout2_acdac_ana_pad_nom            avout3_acdac_ana_pad_nom
avout4_acdac_ana_pad_nom            avout5_acdac_ana_pad_nom            avout6_acdac_ana_pad_nom
avout7_acdac_ana_pad_nom            avout8_acdac_ana_pad_nom            avout9_acdac_ana_pad_nom
avout10_acdac_ana_pad_nom          avout11_acdac_ana_pad_nom          avout12_acdac_ana_pad_nom
avout13_acdac_ana_pad_nom          avout14_acdac_ana_pad_nom          avout15_acdac_ana_pad_nom
avout16_acdac_ana_pad_nom          avout17_acdac_ana_pad_nom          avout18_acdac_ana_pad_nom
avout19_acdac_ana_pad_nom avout20_acdac_ana_pad_nom avout21_acdac_ana_pad_nom}

#ASSIGN LOAD CELLS TO INPUT PINS

set_qtm_port_load -type load_b15fpn000ah1n02x5_d -factor 1 {data_i_drive[9:0] data_q_drive[9:0]
data_stim[9:0]}
set_qtm_port_load -type load_b15bfn000ah1n04x5_a -factor 1 {reset_clkdiv_rx}
set_qtm_port_load -type load_b15inv000ah1n04x5_a -factor 1 {reset_fsm_rx}
set_qtm_port_load    -type    load_b15nand02ah1n04x5_a    -factor    1    {din_dac0_acdac[11:0]
din_dac1_acdac[11:0]      din_dac2_acdac[11:0]      din_dac3_acdac[11:0]      din_dac4_acdac[11:0]
din_dac5_acdac[11:0]      din_dac6_acdac[11:0]      din_dac7_acdac[11:0]      din_dac8_acdac[11:0]
din_dac9_acdac[11:0]     din_dac10_acdac[11:0]     din_dac11_acdac[11:0]     din_dac12_acdac[11:0]
din_dac13_acdac[11:0]     din_dac14_acdac[11:0]     din_dac15_acdac[11:0]     din_dac16_acdac[11:0]
din_dac17_acdac[11:0]     din_dac18_acdac[11:0]     din_dac19_acdac[11:0]     din_dac20_acdac[11:0]
din_dac21_acdac[11:0]}
set_qtm_port_load -type load_b15bfn001ah1n08x5_a -factor 1 {aclk_ana_pad_nom adc_i_bias_rx[3:0]
adc_i_clk_div_rx[2:0]      adc_i_smpdly_rx[4:0]      adc_q_bias_rx[3:0]      adc_q_clk_div_rx[2:0]
adc_q_smpdly_rx[4:0]     avlo_drive_ana_pad_nom     avrfn_rx_ana_pad_nom     avrfp_rx_ana_pad_nom
backup_bits_acdac[127:0]   backup_bits_drive[31:0]   backup_bits_rx[31:0]   backup_bits_stim[31:0]
backup_bits_toplevel[31:0]      bb_i_en_rx      bb_i_gain_stg1_rx[3:0]      bb_i_gain_stg2_rx[3:0]
bb_i_gain_stg3_rx[3:0]        bb_out_enable_drive        bb_q_en_rx        bb_q_gain_stg1_rx[3:0]
bb_q_gain_stg2_rx[3:0]  bb_q_gain_stg3_rx[3:0]  bias_select_acdac  bias_select_drive  bias_select_rx
bias_select_stim  bias_tune_ibias_acdac[3:0]  bias_tune_ibias_drive[3:0]  bias_tune_ibias_rx[3:0]
bias_tune_ibias_stim[3:0]  buffer_en_stim  buffer_tune_gain_stim[4:0]  cas_bias_i_ctr_drive[3:0]
cas_bias_res_ctr_drive[2:0]          class_a_bias_ctr_drive[6:0]          class_a_gain_drive[7:0]
class_c_bias_ctr_drive[6:0]  class_c_gain_drive[7:0]  clock_enable  dac_i_tune_ibleed_drive[3:0]
dac_i_tune_iunit_drive[3:0]       dac_q_tune_ibleed_drive[3:0]       dac_q_tune_iunit_drive[3:0]
dac_tune_ibleed_stim[3:0]     dac_tune_iunit_stim[3:0]     en_class_a_drive     en_class_c_drive
enable_dac_acdac[21:0]           filter_i_tune_ca_drive[4:0]           filter_i_tune_ca_rx[3:0]
filter_i_tune_cb_drive[4:0]       filter_i_tune_cb_rx[3:0]       filter_i_tune_ibias_drive[4:0]
filter_i_tune_ibias_rx[3:0]      filter_i_tune_imix_drive[3:0]      filter_i_tune_ivga_drive[3:0]
filter_i_tune_res_rx[2:0]          filter_q_tune_ca_drive[4:0]          filter_q_tune_ca_rx[3:0]
filter_q_tune_cb_drive[4:0]       filter_q_tune_cb_rx[3:0]       filter_q_tune_ibias_drive[4:0]
filter_q_tune_ibias_rx[3:0]      filter_q_tune_imix_drive[3:0]      filter_q_tune_ivga_drive[3:0]
filter_q_tune_res_rx[2:0]           filter_tune_ca_stim[4:0]           filter_tune_cb_stim[4:0]
filter_tune_ibias_stim[4:0]        filter_tune_imix_stim[3:0]        filter_tune_ivga_stim[3:0]
lna_bias_ctr_fb_rx[3:0]  lna_bias_ctr_main_rx[3:0]  lna_en_fb_rx  lna_en_main_rx  lna_r_tune_rx[3:0]
lo_buffer_en_drive        lo_i_buffer_bias_ctr_drive[3:0]        lo_i_buffer_res_tune_drive[2:0]
lo_q_buffer_bias_ctr_drive[3:0]    lo_q_buffer_res_tune_drive[2:0]    mixer_mn_res_tune_drive[2:0]
mx_bias_ctr_rx[3:0] mx_clk_en_rx mx_en_drive mx_en_rx mx_i_deg_r_tune_rx[3:0] mx_i_r_tune_rx[3:0]
mx_q_deg_r_tune_rx[3:0]   mx_q_r_tune_rx[3:0]   off_sw_drive   sel_clk_dac_drive   sel_clk_dac_rx
sel_clk_dac_stim  select_divide_4or8_rx  test_mux_select[7:0]  trig_dac_acdac  tune_cap_acdac[87:0]
tune_gm_acdac[87:0]  tune_op_acdac[87:0]  vga_i_tune_gain_drive[3:0]  vga_i_tune_test_drive[3:0]
vga_q_tune_gain_drive[3:0]          vga_q_tune_test_drive[3:0]          vga_tune_gain_stim[3:0]
vga_tune_test_stim[3:0]}
```

```
set_qtm_port_load       -value       10       {aclk_ana_pad_nom       athermdiodebjt_ana_pad_nom
athermdiodeesd_ana_pad_nom avlo_drive_ana_pad_nom avrfn_rx_ana_pad_nom avrfp_rx_ana_pad_nom}

#SAVE THE QTM MODEL IN LIBERTY FORMAT
save_qtm_model -format lib -output hr2_analog_top -library_cell

#Make sure to create the qtm directory before to write the qtm model (mkdir qtm), otherwise it will
fail
#write_qtm_model -out_dir qtm
```

B. TO LEVEL NETLIST FOR FULL CHIP ASSEMBLY HR2_FCTOP.V

```
module hr2_fctop  (
inout xxi_bootmode1,
inout xxi_bootmode2,
inout xxi_ext_clk,
inout xxi_ext_clk_sel,
inout xxi_resetn,
inout xxi_spi_s_mosi,
inout xxi_spi_s_sclk,
inout xxi_spi_s_ss,
inout xxi_start_in,
inout xxi_tck_host_core,
inout xxi_tdi_host_core,
inout xxi_tms_host_core,
inout xxi_trst_host_core,
inout xxi_uart_rx,
inout xxio_gpio3,
inout xxio_spi_xip_mosi1,
inout xxio_spi_xip_mosi2,
inout xxo_bof,
inout xxo_spi_s_miso,
inout xxo_spi_xip_sclk,
inout xxo_spi_xip_ss0,
inout xxo_start_out,
inout xxo_tdo_host_core,
inout xxo_uart_tx,
input aadcvref_rx_ana_pad_nom,
input aclk_ana_pad_nom,
input aiext_acdac_ana_pad_nom,
input aiext_drive_ana_pad_nom,
input aiext_rx_ana_pad_nom,
input aiext_stim_ana_pad_nom,
input avlo_drive_ana_pad_nom,
input avrfn_rx_ana_pad_nom,
input avrfp_rx_ana_pad_nom,
inout athermdiodeesd1_ana_pad_nom,
inout athermdiodeesd2_ana_pad_nom,
inout athermdiodeesd3_ana_pad_nom,
inout athermdiodeesd4_ana_pad_nom,
inout athermdiodegnd1_ana_pad_nom,
inout athermdiodegnd2_ana_pad_nom,
inout athermdiodegnd3_ana_pad_nom,
inout athermdiodegnd4_ana_pad_nom,
output aitestmux_ana_pad_nom,
output avout0_acdac_ana_pad_nom,
output avout10_acdac_ana_pad_nom,
output avout11_acdac_ana_pad_nom,
output avout12_acdac_ana_pad_nom,
output avout13_acdac_ana_pad_nom,
output avout14_acdac_ana_pad_nom,
output avout15_acdac_ana_pad_nom,
output avout16_acdac_ana_pad_nom,
output avout17_acdac_ana_pad_nom,
output avout18_acdac_ana_pad_nom,
output avout19_acdac_ana_pad_nom,
output avout1_acdac_ana_pad_nom,
output avout20_acdac_ana_pad_nom,
output avout21_acdac_ana_pad_nom,
output avout2_acdac_ana_pad_nom,
output avout3_acdac_ana_pad_nom,
output avout4_acdac_ana_pad_nom,
output avout5_acdac_ana_pad_nom,
output avout6_acdac_ana_pad_nom,
output avout7_acdac_ana_pad_nom,
```

```
output avout8_acdac_ana_pad_nom,
output avout9_acdac_ana_pad_nom,
output avrf_drive_ana_pad_nom,
output avrfn_stim_ana_pad_nom,
output avrfp_stim_ana_pad_nom,
output avtestbb_drive_ana_pad_nom,
output avtestmux_ana_pad_nom
);

hr2_soc_top hr2_soc_top (
.xxi_bootmode1 (xxi_bootmode1),
.xxi_bootmode2 (xxi_bootmode2),
.xxi_ext_clk (xxi_ext_clk),
.xxi_ext_clk_sel (xxi_ext_clk_sel),
.xxi_resetn (xxi_resetn),
.xxi_spi_s_mosi (xxi_spi_s_mosi),
.xxi_spi_s_sclk (xxi_spi_s_sclk),
.xxi_spi_s_ss (xxi_spi_s_ss),

////////////////////////////////////////////
/// Check this connection at hr2_fc_top level
.xxi_start_in (xxi_start_out),
////////////////////////////////////////////

.xxi_tck_host_core (xxi_tck_host_core),
.xxi_tdi_host_core (xxi_tdi_host_core),
.xxi_tms_host_core (xxi_tms_host_core),
.xxi_trst_host_core (xxi_trst_host_core),
.xxi_uart_rx (xxi_uart_rx),
.xxio_gpio3 (xxio_gpio3),
.xxio_spi_xip_mosi1 (xxio_spi_xip_mosi1),
.xxio_spi_xip_mosi2 (xxio_spi_xip_mosi2),
.xxo_bof (xxo_bof),
.xxo_spi_s_miso (xxo_spi_s_miso),
.xxo_spi_xip_sclk (xxo_spi_xip_sclk),
.xxo_spi_xip_ss0 (xxo_spi_xip_ss0),

/// Check this connection at hr2_soc_top level
.xxo_start_out (xxo_start_in),
////////////////////////////////////////////

.xxo_tdo_host_core (xxo_tdo_host_core),
.xxo_uart_tx (xxo_uart_tx),
.aadcvref_rx_ana_pad_nom (aadcvref_rx_ana_pad_nom),
.aclk_ana_pad_nom (aclk_ana_pad_nom),
.aiext_acdac_ana_pad_nom (aiext_acdac_ana_pad_nom),
.aiext_drive_ana_pad_nom (aiext_drive_ana_pad_nom),
.aiext_rx_ana_pad_nom (aiext_rx_ana_pad_nom),
.aiext_stim_ana_pad_nom (aiext_stim_ana_pad_nom),
.avlo_drive_ana_pad_nom (avlo_drive_ana_pad_nom),
.avrfn_rx_ana_pad_nom (avrfn_rx_ana_pad_nom),
.avrfp_rx_ana_pad_nom (avrfp_rx_ana_pad_nom),
.athermdiodeesd1_ana_pad_nom (athermdiodeesd1_ana_pad_nom),
.athermdiodeesd2_ana_pad_nom (athermdiodeesd2_ana_pad_nom),
.athermdiodeesd3_ana_pad_nom (athermdiodeesd3_ana_pad_nom),
.athermdiodeesd4_ana_pad_nom (athermdiodeesd4_ana_pad_nom),
.athermdiodegnd1_ana_pad_nom (athermdiodegnd1_ana_pad_nom),
.athermdiodegnd2_ana_pad_nom (athermdiodegnd2_ana_pad_nom),
.athermdiodegnd3_ana_pad_nom (athermdiodegnd3_ana_pad_nom),
.athermdiodegnd4_ana_pad_nom (athermdiodegnd4_ana_pad_nom),
.aitestmux_ana_pad_nom (aitestmux_ana_pad_nom),
.avout0_acdac_ana_pad_nom (avout0_acdac_ana_pad_nom),
.avout10_acdac_ana_pad_nom (avout10_acdac_ana_pad_nom),
.avout11_acdac_ana_pad_nom (avout11_acdac_ana_pad_nom),
.avout12_acdac_ana_pad_nom (avout12_acdac_ana_pad_nom),
.avout13_acdac_ana_pad_nom (avout13_acdac_ana_pad_nom),
.avout14_acdac_ana_pad_nom (avout14_acdac_ana_pad_nom),
.avout15_acdac_ana_pad_nom (avout15_acdac_ana_pad_nom),
.avout16_acdac_ana_pad_nom (avout16_acdac_ana_pad_nom),
```

```
.avout17_acdac_ana_pad_nom (avout17_acdac_ana_pad_nom),
.avout18_acdac_ana_pad_nom (avout18_acdac_ana_pad_nom),
.avout19_acdac_ana_pad_nom (avout19_acdac_ana_pad_nom),
.avout1_acdac_ana_pad_nom (avout1_acdac_ana_pad_nom),
.avout20_acdac_ana_pad_nom (avout20_acdac_ana_pad_nom),
.avout21_acdac_ana_pad_nom (avout21_acdac_ana_pad_nom),
.avout2_acdac_ana_pad_nom (avout2_acdac_ana_pad_nom),
.avout3_acdac_ana_pad_nom (avout3_acdac_ana_pad_nom),
.avout4_acdac_ana_pad_nom (avout4_acdac_ana_pad_nom),
.avout5_acdac_ana_pad_nom (avout5_acdac_ana_pad_nom),
.avout6_acdac_ana_pad_nom (avout6_acdac_ana_pad_nom),
.avout7_acdac_ana_pad_nom (avout7_acdac_ana_pad_nom),
.avout8_acdac_ana_pad_nom (avout8_acdac_ana_pad_nom),
.avout9_acdac_ana_pad_nom (avout9_acdac_ana_pad_nom),
.avrf_drive_ana_pad_nom (avrf_drive_ana_pad_nom),
.avrfn_stim_ana_pad_nom (avrfn_stim_ana_pad_nom),
.avrfp_stim_ana_pad_nom (avrfp_stim_ana_pad_nom),
.avtestbb_drive_ana_pad_nom (avtestbb_drive_ana_pad_nom),
.avtestmux_ana_pad_nom (avtestmux_ana_pad_nom)
);

fdk22tic2_s22_4x4_er_edm_prs hr2_frame(.edmres0( ),.edmres1( ));
endmodule
```

C. FULL CHIP TIMING CONSTRAINTS

```
set sdc_version 2.1

set_units -time ps -resistance kOhm -capacitance fF -voltage V -current mA
#set_operating_conditions typical_1.00 -library b15_hn_p1222_2x1r2_psss_0.900v_-40c_max_ccst_pdn
global G_OPCOND_PVT
set_operating_conditions typical_1.00 -library $G_OPCOND_PVT
set_max_transition 80 [current_design]
set_max_fanout 50 [current_design]
set_load -pin_load 10 [get_ports xxi_bootmode1]
set_load -pin_load 10 [get_ports xxi_bootmode2]
set_load -pin_load 10 [get_ports xxi_ext_clk]
set_load -pin_load 10 [get_ports xxi_ext_clk_sel]
set_load -pin_load 10 [get_ports xxi_resetn]
set_load -pin_load 10 [get_ports xxi_spi_s_mosi]
set_load -pin_load 10 [get_ports xxi_spi_s_sclk]
set_load -pin_load 10 [get_ports xxi_spi_s_ss]
set_load -pin_load 10 [get_ports xxi_start_in]
set_load -pin_load 10 [get_ports xxi_tck_host_core]
set_load -pin_load 10 [get_ports xxi_tdi_host_core]
set_load -pin_load 10 [get_ports xxi_tms_host_core]
set_load -pin_load 10 [get_ports xxi_trst_host_core]
set_load -pin_load 10 [get_ports xxi_uart_rx]
set_load -pin_load 10 [get_ports xxio_gpio3]
set_load -pin_load 10 [get_ports xxio_spi_xip_mosi1]
set_load -pin_load 10 [get_ports xxio_spi_xip_mosi2]
set_load -pin_load 10 [get_ports xxo_bof]
set_load -pin_load 10 [get_ports xxo_spi_s_miso]
set_load -pin_load 10 [get_ports xxo_spi_xip_sclk]
set_load -pin_load 10 [get_ports xxo_spi_xip_ss0]
set_load -pin_load 10 [get_ports xxo_start_out]
set_load -pin_load 10 [get_ports xxo_tdo_host_core]
set_load -pin_load 10 [get_ports xxo_uart_tx]
set_load -pin_load 10 [get_ports athermdiodeesd1_ana_pad_nom]
set_load -pin_load 10 [get_ports athermdiodeesd2_ana_pad_nom]
set_load -pin_load 10 [get_ports athermdiodeesd3_ana_pad_nom]
set_load -pin_load 10 [get_ports athermdiodeesd4_ana_pad_nom]
set_load -pin_load 10 [get_ports athermdiodegnd1_ana_pad_nom]
set_load -pin_load 10 [get_ports athermdiodegnd2_ana_pad_nom]
set_load -pin_load 10 [get_ports athermdiodegnd3_ana_pad_nom]
set_load -pin_load 10 [get_ports athermdiodegnd4_ana_pad_nom]
set_load -pin_load 10 [get_ports aitestmux_ana_pad_nom]
set_load -pin_load 10 [get_ports avout0_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout10_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout11_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout12_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout13_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout14_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout15_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout16_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout17_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout18_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout19_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout1_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout20_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout21_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout2_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout3_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout4_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout5_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout6_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout7_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout8_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avout9_acdac_ana_pad_nom]
set_load -pin_load 10 [get_ports avrf_drive_ana_pad_nom]
```

```
set_load -pin_load 10 [get_ports avrfn_stim_ana_pad_nom]
set_load -pin_load 10 [get_ports avrfp_stim_ana_pad_nom]
set_load -pin_load 10 [get_ports avtestbb_drive_ana_pad_nom]
set_load -pin_load 10 [get_ports avtestmux_ana_pad_nom]
create_clock [get_pins <hier>/clk_2p5]  -name Clk2p5G  -period 400  -waveform {0 200}
set_clock_uncertainty -setup 100  [get_clocks Clk2p5G]
set_clock_uncertainty -hold 40  [get_clocks Clk2p5G]
set_max_transition 40 -clock_path [get_clocks Clk2p5G]
create_clock [get_pins {<hier>/u_io_family/OUTI[0]}]  -name Ext_Clk  -period 4000  -waveform {0
2000}
set_clock_uncertainty -setup 200  [get_clocks Ext_Clk]
set_clock_uncertainty -hold 40  [get_clocks Ext_Clk]
set_max_transition 40 -clock_path [get_clocks Ext_Clk]
create_clock [get_ports xxi_spi_s_sclk]  -name SPI_Clk  -period 40000  -waveform {0 20000}
set_clock_uncertainty -setup 200  [get_clocks SPI_Clk]
set_clock_uncertainty -hold 40  [get_clocks SPI_Clk]
set_max_transition 40 -clock_path [get_clocks SPI_Clk]
create_clock [get_pins {<hier>/u_io_family/OUTI[19]}]  -name JTAG_Clk  -period 50000  -waveform
{0 25000}
set_clock_uncertainty -setup 200  [get_clocks JTAG_Clk]
set_clock_uncertainty -hold 40  [get_clocks JTAG_Clk]
set_max_transition 40 -clock_path [get_clocks JTAG_Clk]
create_clock [get_pins i_analog_top/clk_0p5]  -name Clk_25  -period 1600  -waveform {0 800}
set_clock_uncertainty -setup 200  [get_clocks Clk_25]
set_clock_uncertainty -hold 40  [get_clocks Clk_25]
set_max_transition 40 -clock_path [get_clocks Clk_25]
create_generated_clock [get_pins <hier>/clk_0_out_div_reg/o]  -name accel_0_clk_2p5G  -source
[get_pins <hier>/clk_2p5]  -edges {1 5 9}
set_clock_uncertainty -setup 200  [get_clocks accel_0_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks accel_0_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks accel_0_clk_2p5G]
create_generated_clock [get_pins <hier>/clk_0_out_div_reg/o]  -name accel_0_clk_Ext  -source
[get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {1 5 9}  -add
set_clock_uncertainty -setup 200  [get_clocks accel_0_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks accel_0_clk_Ext]
set_max_transition 40 -clock_path [get_clocks accel_0_clk_Ext]
create_generated_clock [get_pins <hier>/clk_90_out_div_reg/o]  -name accel_90_clk_2p5G  -source
[get_pins <hier>/clk_2p5]  -edges {3 7 11}
set_clock_uncertainty -setup 200  [get_clocks accel_90_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks accel_90_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks accel_90_clk_2p5G]
create_generated_clock [get_pins <hier>/clk_90_out_div_reg/o]  -name accel_90_clk_Ext  -source
[get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {3 7 11}  -add
set_clock_uncertainty -setup 200  [get_clocks accel_90_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks accel_90_clk_Ext]
set_max_transition 40 -clock_path [get_clocks accel_90_clk_Ext]
create_generated_clock [get_pins <hier>clk_180_out_div_reg/o]  -name accel_180_clk_2p5G  -source
[get_pins <hier>/clk_2p5]  -edges {5 9 13}
set_clock_uncertainty -setup 200  [get_clocks accel_180_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks accel_180_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks accel_180_clk_2p5G]
create_generated_clock [get_pins <hier>/clk_180_out_div_reg/o]  -name accel_180_clk_Ext  -source
[get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {5 9 13}  -add
set_clock_uncertainty -setup 200  [get_clocks accel_180_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks accel_180_clk_Ext]
set_max_transition 40 -clock_path [get_clocks accel_180_clk_Ext]
create_generated_clock [get_pins <hier>/clk_270_out_div_reg/o]  -name accel_270_clk_2p5G  -source
[get_pins <hier>/clk_2p5]  -edges {7 11 15}
set_clock_uncertainty -setup 200  [get_clocks accel_270_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks accel_270_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks accel_270_clk_2p5G]
create_generated_clock [get_pins <hier>/clk_270_out_div_reg/o]  -name accel_270_clk_Ext  -source
[get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {7 11 15}  -add
set_clock_uncertainty -setup 200  [get_clocks accel_270_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks accel_270_clk_Ext]
set_max_transition 40 -clock_path [get_clocks accel_270_clk_Ext]
create_generated_clock [get_pins <hier>/i2c_core_clk_i0_dcszo/clkout]  -name I2C_Core_clk_2p5G  -
source [get_pins <hier>/clk_2p5]  -divide_by 100
set_clock_uncertainty -setup 200  [get_clocks I2C_Core_clk_2p5G]
```

```
set_clock_uncertainty -hold 40  [get_clocks I2C_Core_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks I2C_Core_clk_2p5G]
create_generated_clock [get_pins <hier>/i2c_core_clk_i0_dcszo/clkout]  -name I2C_Core_clk_Ext  -
source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -divide_by 100  -add
set_clock_uncertainty -setup 200  [get_clocks I2C_Core_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks I2C_Core_clk_Ext]
set_max_transition 40 -clock_path [get_clocks I2C_Core_clk_Ext]
create_generated_clock [get_pins <hier>/uart_core_clk_i0_dcszo/clkout]  -name UART_Core_clk_2p5G
-source [get_pins <hier>/clk_2p5]  -divide_by 100
set_clock_uncertainty -setup 200  [get_clocks UART_Core_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks UART_Core_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks UART_Core_clk_2p5G]
create_generated_clock [get_pins <hier>/uart_core_clk_i0_dcszo/clkout]  -name UART_Core_clk_Ext
-source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -divide_by 100  -add
set_clock_uncertainty -setup 200  [get_clocks UART_Core_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks UART_Core_clk_Ext]
set_max_transition 40 -clock_path [get_clocks UART_Core_clk_Ext]
create_generated_clock [get_pins
hr2_fc_ss/hr2_host_core_ss/i_pmuccu/u_pmuccu/u_ccu/gpio_core_clk_i0_dcszo/clkout]  -name
GPIO_Core_clk_2p5G  -source [get_pins <hier>/clk_2p5]  -divide_by 100
set_clock_uncertainty -setup 200  [get_clocks GPIO_Core_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks GPIO_Core_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks GPIO_Core_clk_2p5G]
create_generated_clock [get_pins <hier>/gpio_core_clk_i0_dcszo/clkout]  -name GPIO_Core_clk_Ext
-source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -divide_by 100  -add
set_clock_uncertainty -setup 200  [get_clocks GPIO_Core_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks GPIO_Core_clk_Ext]
set_max_transition 40 -clock_path [get_clocks GPIO_Core_clk_Ext]
create_generated_clock [get_pins <hier>/spi_s_clk_i0_dcszo/clkout]  -name SPI_S_Core_clk_2p5G  -
source [get_pins <hier>/clk_2p5]  -divide_by 100
set_clock_uncertainty -setup 200  [get_clocks SPI_S_Core_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks SPI_S_Core_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks SPI_S_Core_clk_2p5G]
create_generated_clock [get_pins <hier>/spi_s_clk_i0_dcszo/clkout]  -name SPI_S_Core_clk_Ext  -
source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -divide_by 100  -add
set_clock_uncertainty -setup 200  [get_clocks SPI_S_Core_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks SPI_S_Core_clk_Ext]
set_max_transition 40 -clock_path [get_clocks SPI_S_Core_clk_Ext]
create_generated_clock [get_pins <hier>/timer_clk_i0_dcszo/clkout]  -name Timer_clk_2p5G  -source
[get_pins <hier>/clk_2p5]  -divide_by 100
set_clock_uncertainty -setup 200  [get_clocks Timer_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks Timer_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks Timer_clk_2p5G]
create_generated_clock [get_pins <hier>/timer_clk_i0_dcszo/clkout]  -name Timer_clk_Ext  -source
[get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -divide_by 100  -add
set_clock_uncertainty -setup 200  [get_clocks Timer_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks Timer_clk_Ext]
set_max_transition 40 -clock_path [get_clocks Timer_clk_Ext]
create_generated_clock [get_pins <hier>/spi_xip_core_clk_i0_dcszo/clkout]  -name
SPI_XIP_Core_clk_2p5G  -source [get_pins <hier>/clk_2p5]  -divide_by 100
set_clock_uncertainty -setup 200  [get_clocks SPI_XIP_Core_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks SPI_XIP_Core_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks SPI_XIP_Core_clk_2p5G]
create_generated_clock [get_pins <hier>/spi_xip_core_clk_i0_dcszo/clkout]  -name
SPI_XIP_Core_clk_Ext  -source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -
divide_by 100  -add
set_clock_uncertainty -setup 200  [get_clocks SPI_XIP_Core_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks SPI_XIP_Core_clk_Ext]
set_max_transition 40 -clock_path [get_clocks SPI_XIP_Core_clk_Ext]
create_generated_clock [get_pins <hier>/u_core_div/clk_o_i0_dcszo/clkout]  -name core_clk_2p5G  -
source [get_pins <hier>/clk_2p5]  -divide_by 100
set_clock_uncertainty -setup 200  [get_clocks core_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks core_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks core_clk_2p5G]
create_generated_clock [get_pins <hier>/u_core_div/clk_o_i0_dcszo/clkout]  -name core_clk_Ext  -
source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -divide_by 100  -add
set_clock_uncertainty -setup 200  [get_clocks core_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks core_clk_Ext]
set_max_transition 40 -clock_path [get_clocks core_clk_Ext]
```

```
create_generated_clock [get_pins <hier>/host_core_clk_i0_dcszo/clkout]  -name Host_core_clk_2p5G
-source [get_pins <hier>/clk_2p5]  -edges {1 7 13}
set_clock_uncertainty -setup 200  [get_clocks Host_core_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks Host_core_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks Host_core_clk_2p5G]
create_generated_clock [get_pins <hier>/host_core_clk_i0_dcszo/clkout]  -name Host_core_clk_Ext
-source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {1 7 13}  -add
set_clock_uncertainty -setup 200  [get_clocks Host_core_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks Host_core_clk_Ext]
set_max_transition 40 -clock_path [get_clocks Host_core_clk_Ext]
create_generated_clock [get_pins <hier>/host_core_aclk_i0_dcszo/clkout]  -name
Host_core_aclk_2p5G  -source [get_pins <hier>/clk_2p5]  -edges {1 7 13}
set_clock_uncertainty -setup 200  [get_clocks Host_core_aclk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks Host_core_aclk_2p5G]
set_max_transition 40 -clock_path [get_clocks Host_core_aclk_2p5G]
create_generated_clock [get_pins <hier>/host_core_aclk_i0_dcszo/clkout]  -name Host_core_aclk_Ext
-source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {1 7 13}  -add
set_clock_uncertainty -setup 200  [get_clocks Host_core_aclk_Ext]
set_clock_uncertainty -hold 40  [get_clocks Host_core_aclk_Ext]
set_max_transition 40 -clock_path [get_clocks Host_core_aclk_Ext]
create_generated_clock [get_pins <hier>/smem_clk_i0_dcszo/clkout]  -name Smem_clk_2p5G  -source
[get_pins <hier>/clk_2p5]  -edges {1 7 13}
set_clock_uncertainty -setup 200  [get_clocks Smem_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks Smem_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks Smem_clk_2p5G]
create_generated_clock [get_pins <hier>/smem_clk_i0_dcszo/clkout]  -name Smem_clk_Ext  -source
[get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {1 7 13}  -add
set_clock_uncertainty -setup 200  [get_clocks Smem_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks Smem_clk_Ext]
set_max_transition 40 -clock_path [get_clocks Smem_clk_Ext]
create_generated_clock [get_pins <hier>/ahb_fabric_hclk_i0_dcszo/clkout]  -name
Ahb_fabric_clk_2p5G  -source [get_pins <hier>/clk_2p5]  -edges {1 7 13}
set_clock_uncertainty -setup 200  [get_clocks Ahb_fabric_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks Ahb_fabric_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks Ahb_fabric_clk_2p5G]
create_generated_clock [get_pins <hier>/ahb_fabric_hclk_i0_dcszo/clkout]  -name
Ahb_fabric_clk_Ext  -source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -
edges {1 7 13}  -add
set_clock_uncertainty -setup 200  [get_clocks Ahb_fabric_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks Ahb_fabric_clk_Ext]
set_max_transition 40 -clock_path [get_clocks Ahb_fabric_clk_Ext]
create_generated_clock [get_pins <hier>/dma_hclk_i0_dcszo/clkout]  -name Dma_hclk_2p5G  -source
[get_pins <hier>/clk_2p5]  -edges {1 7 13}
set_clock_uncertainty -setup 200  [get_clocks Dma_hclk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks Dma_hclk_2p5G]
set_max_transition 40 -clock_path [get_clocks Dma_hclk_2p5G]
create_generated_clock [get_pins <hier>/dma_hclk_i0_dcszo/clkout]  -name Dma_hclk_Ext  -source
[get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {1 7 13}  -add
set_clock_uncertainty -setup 200  [get_clocks Dma_hclk_Ext]
set_clock_uncertainty -hold 40  [get_clocks Dma_hclk_Ext]
set_max_transition 40 -clock_path [get_clocks Dma_hclk_Ext]
create_generated_clock [get_pins <hier>/apb_fabric_pclk_i0_dcszo/clkout]  -name
Apb_fabric_pclk_2p5G  -source [get_pins <hier>/clk_2p5]  -edges {1 7 13}
set_clock_uncertainty -setup 200  [get_clocks Apb_fabric_pclk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks Apb_fabric_pclk_2p5G]
set_max_transition 40 -clock_path [get_clocks Apb_fabric_pclk_2p5G]
create_generated_clock [get_pins <hier>/apb_fabric_pclk_i0_dcszo/clkout]  -name
Apb_fabric_pclk_Ext  -source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -
edges {1 7 13}  -add
set_clock_uncertainty -setup 200  [get_clocks Apb_fabric_pclk_Ext]
set_clock_uncertainty -hold 40  [get_clocks Apb_fabric_pclk_Ext]
set_max_transition 40 -clock_path [get_clocks Apb_fabric_pclk_Ext]
create_generated_clock [get_pins <hier>/u_base_div/clk_o_i0_dcszo/clkout]  -name base_clk_2p5G  -
source [get_pins <hier>/clk_2p5]  -edges {1 7 13}
set_clock_uncertainty -setup 200  [get_clocks base_clk_2p5G]
set_clock_uncertainty -hold 40  [get_clocks base_clk_2p5G]
set_max_transition 40 -clock_path [get_clocks base_clk_2p5G]
create_generated_clock [get_pins <hier>/u_base_div/clk_o_i0_dcszo/clkout]  -name base_clk_Ext  -
source [get_pins {<hier>/u_io_family/OUTI[0]}]  -master_clock Ext_Clk  -edges {1 7 13}  -add
```

```
set_clock_uncertainty -setup 200  [get_clocks base_clk_Ext]
set_clock_uncertainty -hold 40  [get_clocks base_clk_Ext]
set_max_transition 40 -clock_path [get_clocks base_clk_Ext]
group_path -name FEED_THROUGH  -from [list [get_ports xxi_bootmode1] [get_ports xxi_bootmode2]
[get_ports xxi_ext_clk] [get_ports xxi_ext_clk_sel] [get_ports xxi_resetn] [get_ports
xxi_spi_s_mosi] [get_ports xxi_spi_s_ss] [get_ports xxi_start_in] [get_ports xxi_tck_host_core]
[get_ports xxi_tdi_host_core] [get_ports xxi_tms_host_core] [get_ports xxi_trst_host_core]
[get_ports xxi_uart_rx] [get_ports xxio_gpio3] [get_ports xxio_spi_xip_mosi1] [get_ports
xxio_spi_xip_mosi2] [get_ports xxo_bof] [get_ports xxo_spi_s_miso] [get_ports xxo_spi_xip_sclk]
[get_ports xxo_spi_xip_ss0] [get_ports xxo_start_out] [get_ports xxo_tdo_host_core] [get_ports
xxo_uart_tx] [get_ports aadcvref_rx_ana_pad_nom] [get_ports aclk_ana_pad_nom] [get_ports
aiext_acdac_ana_pad_nom] [get_ports aiext_drive_ana_pad_nom] [get_ports aiext_rx_ana_pad_nom]
[get_ports aiext_stim_ana_pad_nom] [get_ports avlo_drive_ana_pad_nom] [get_ports
avrfn_rx_ana_pad_nom] [get_ports avrfp_rx_ana_pad_nom] [get_ports athermdiodeesd1_ana_pad_nom]
[get_ports athermdiodeesd2_ana_pad_nom] [get_ports athermdiodeesd3_ana_pad_nom] [get_ports
athermdiodeesd4_ana_pad_nom] [get_ports athermdiodegnd1_ana_pad_nom] [get_ports
athermdiodegnd2_ana_pad_nom] [get_ports athermdiodegnd3_ana_pad_nom] [get_ports
athermdiodegnd4_ana_pad_nom]]  -to [list [get_ports xxi_bootmode1] [get_ports xxi_bootmode2]
[get_ports xxi_ext_clk] [get_ports xxi_ext_clk_sel] [get_ports xxi_resetn] [get_ports
xxi_spi_s_mosi] [get_ports xxi_spi_s_sclk] [get_ports xxi_spi_s_ss] [get_ports xxi_start_in]
[get_ports xxi_tck_host_core] [get_ports xxi_tdi_host_core] [get_ports xxi_tms_host_core]
[get_ports xxi_trst_host_core] [get_ports xxi_uart_rx] [get_ports xxio_gpio3] [get_ports
xxio_spi_xip_mosi1] [get_ports xxio_spi_xip_mosi2] [get_ports xxo_bof] [get_ports xxo_spi_s_miso]
[get_ports xxo_spi_xip_sclk] [get_ports xxo_spi_xip_ss0] [get_ports xxo_start_out] [get_ports
xxo_tdo_host_core] [get_ports xxo_uart_tx] [get_ports athermdiodeesd1_ana_pad_nom] [get_ports
athermdiodeesd2_ana_pad_nom] [get_ports athermdiodeesd3_ana_pad_nom] [get_ports
athermdiodeesd4_ana_pad_nom] [get_ports athermdiodegnd1_ana_pad_nom] [get_ports
athermdiodegnd2_ana_pad_nom] [get_ports athermdiodegnd3_ana_pad_nom] [get_ports
athermdiodegnd4_ana_pad_nom] [get_ports aitestmux_ana_pad_nom] [get_ports
avout0_acdac_ana_pad_nom] [get_ports avout10_acdac_ana_pad_nom] [get_ports
avout11_acdac_ana_pad_nom] [get_ports avout12_acdac_ana_pad_nom] [get_ports
avout13_acdac_ana_pad_nom] [get_ports avout14_acdac_ana_pad_nom] [get_ports
avout15_acdac_ana_pad_nom] [get_ports avout16_acdac_ana_pad_nom] [get_ports
avout17_acdac_ana_pad_nom] [get_ports avout18_acdac_ana_pad_nom] [get_ports
avout19_acdac_ana_pad_nom] [get_ports avout1_acdac_ana_pad_nom] [get_ports
avout20_acdac_ana_pad_nom] [get_ports avout21_acdac_ana_pad_nom] [get_ports
avout2_acdac_ana_pad_nom] [get_ports avout3_acdac_ana_pad_nom] [get_ports
avout4_acdac_ana_pad_nom] [get_ports avout5_acdac_ana_pad_nom] [get_ports
avout6_acdac_ana_pad_nom] [get_ports avout7_acdac_ana_pad_nom] [get_ports
avout8_acdac_ana_pad_nom] [get_ports avout9_acdac_ana_pad_nom] [get_ports avrf_drive_ana_pad_nom]
[get_ports avrfn_stim_ana_pad_nom] [get_ports avrfp_stim_ana_pad_nom] [get_ports
avtestbb_drive_ana_pad_nom] [get_ports avtestmux_ana_pad_nom]]
group_path -name INPUT_PATHS  -from [list [get_ports xxi_bootmode1] [get_ports xxi_bootmode2]
[get_ports xxi_ext_clk] [get_ports xxi_ext_clk_sel] [get_ports xxi_resetn] [get_ports
xxi_spi_s_mosi] [get_ports xxi_spi_s_ss] [get_ports xxi_start_in] [get_ports xxi_tck_host_core]
[get_ports xxi_tdi_host_core] [get_ports xxi_tms_host_core] [get_ports xxi_trst_host_core]
[get_ports xxi_uart_rx] [get_ports xxio_gpio3] [get_ports xxio_spi_xip_mosi1] [get_ports
xxio_spi_xip_mosi2] [get_ports xxo_bof] [get_ports xxo_spi_s_miso] [get_ports xxo_spi_xip_sclk]
[get_ports xxo_spi_xip_ss0] [get_ports xxo_start_out] [get_ports xxo_tdo_host_core] [get_ports
xxo_uart_tx] [get_ports aadcvref_rx_ana_pad_nom] [get_ports aclk_ana_pad_nom] [get_ports
aiext_acdac_ana_pad_nom] [get_ports aiext_drive_ana_pad_nom] [get_ports aiext_rx_ana_pad_nom]
[get_ports aiext_stim_ana_pad_nom] [get_ports avlo_drive_ana_pad_nom] [get_ports
avrfn_rx_ana_pad_nom] [get_ports avrfp_rx_ana_pad_nom] [get_ports athermdiodeesd1_ana_pad_nom]
[get_ports athermdiodeesd2_ana_pad_nom] [get_ports athermdiodeesd3_ana_pad_nom] [get_ports
athermdiodeesd4_ana_pad_nom] [get_ports athermdiodegnd1_ana_pad_nom] [get_ports
athermdiodegnd2_ana_pad_nom] [get_ports athermdiodegnd3_ana_pad_nom] [get_ports
athermdiodegnd4_ana_pad_nom]]
group_path -name OUTPUT_PATHS  -to [list [get_ports xxi_bootmode1] [get_ports xxi_bootmode2]
[get_ports xxi_ext_clk] [get_ports xxi_ext_clk_sel] [get_ports xxi_resetn] [get_ports
xxi_spi_s_mosi] [get_ports xxi_spi_s_sclk] [get_ports xxi_spi_s_ss] [get_ports xxi_start_in]
[get_ports xxi_tck_host_core] [get_ports xxi_tdi_host_core] [get_ports xxi_tms_host_core]
[get_ports xxi_trst_host_core] [get_ports xxi_uart_rx] [get_ports xxio_gpio3] [get_ports
xxio_spi_xip_mosi1] [get_ports xxio_spi_xip_mosi2] [get_ports xxo_bof] [get_ports xxo_spi_s_miso]
[get_ports xxo_spi_xip_sclk] [get_ports xxo_spi_xip_ss0] [get_ports xxo_start_out] [get_ports
xxo_tdo_host_core] [get_ports xxo_uart_tx] [get_ports athermdiodeesd1_ana_pad_nom] [get_ports
athermdiodeesd2_ana_pad_nom] [get_ports athermdiodeesd3_ana_pad_nom] [get_ports
athermdiodeesd4_ana_pad_nom] [get_ports athermdiodegnd1_ana_pad_nom] [get_ports
athermdiodegnd2_ana_pad_nom] [get_ports athermdiodegnd3_ana_pad_nom] [get_ports
athermdiodegnd4_ana_pad_nom] [get_ports aitestmux_ana_pad_nom] [get_ports
```

```
avout0_acdac_ana_pad_nom] [get_ports avout10_acdac_ana_pad_nom] [get_ports
avout11_acdac_ana_pad_nom] [get_ports avout12_acdac_ana_pad_nom] [get_ports
avout13_acdac_ana_pad_nom] [get_ports avout14_acdac_ana_pad_nom] [get_ports
avout15_acdac_ana_pad_nom] [get_ports avout16_acdac_ana_pad_nom] [get_ports
avout17_acdac_ana_pad_nom] [get_ports avout18_acdac_ana_pad_nom] [get_ports
avout19_acdac_ana_pad_nom] [get_ports avout1_acdac_ana_pad_nom] [get_ports
avout20_acdac_ana_pad_nom] [get_ports avout21_acdac_ana_pad_nom] [get_ports
avout2_acdac_ana_pad_nom] [get_ports avout3_acdac_ana_pad_nom] [get_ports
avout4_acdac_ana_pad_nom] [get_ports avout5_acdac_ana_pad_nom] [get_ports
avout6_acdac_ana_pad_nom] [get_ports avout7_acdac_ana_pad_nom] [get_ports
avout8_acdac_ana_pad_nom] [get_ports avout9_acdac_ana_pad_nom] [get_ports avrf_drive_ana_pad_nom]
[get_ports avrfn_stim_ana_pad_nom] [get_ports avrfp_stim_ana_pad_nom] [get_ports
avtestbb_drive_ana_pad_nom] [get_ports avtestmux_ana_pad_nom]]

set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_spi_s_sclk]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_bootmode1]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_bootmode2]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_ext_clk_sel]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_resetn]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_spi_s_mosi]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_spi_s_ss]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_start_in]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_tck_host_core]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_tdi_host_core]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_tms_host_core]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_trst_host_core]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxi_uart_rx]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxio_gpio3]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxio_spi_xip_mosi1]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxio_spi_xip_mosi2]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxo_bof]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxo_spi_s_miso]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxo_spi_xip_sclk]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxo_spi_xip_ss0]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxo_start_out]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxo_tdo_host_core]
set_input_delay -clock Ext_Clk  2666.67  [get_ports xxo_uart_tx]
set_input_delay -clock Ext_Clk  2666.67  [get_ports aadcvref_rx_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports aclk_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports aiext_acdac_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports aiext_drive_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports aiext_rx_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports aiext_stim_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports avlo_drive_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports avrfn_rx_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports avrfp_rx_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodeesd1_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodeesd2_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodeesd3_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodeesd4_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodegnd1_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodegnd2_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodegnd3_ana_pad_nom]
set_input_delay -clock Ext_Clk  2666.67  [get_ports athermdiodegnd4_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_spi_s_sclk]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_spi_s_sclk]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_spi_s_sclk]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_bootmode1]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_bootmode1]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_bootmode1]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_bootmode2]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_bootmode2]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_bootmode2]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_ext_clk]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_ext_clk]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_ext_clk]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_ext_clk_sel]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_ext_clk_sel]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_ext_clk_sel]
```

```
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_resetn]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_resetn]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_resetn]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_spi_s_mosi]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_spi_s_mosi]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_spi_s_mosi]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_spi_s_ss]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_spi_s_ss]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_spi_s_ss]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_start_in]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_start_in]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_start_in]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_tck_host_core]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_tck_host_core]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_tck_host_core]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_tdi_host_core]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_tdi_host_core]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_tdi_host_core]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_tms_host_core]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_tms_host_core]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_tms_host_core]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_trst_host_core]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_trst_host_core]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_trst_host_core]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxi_uart_rx]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxi_uart_rx]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxi_uart_rx]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxio_gpio3]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxio_gpio3]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxio_gpio3]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxio_spi_xip_mosi1]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxio_spi_xip_mosi1]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxio_spi_xip_mosi1]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxio_spi_xip_mosi2]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxio_spi_xip_mosi2]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxio_spi_xip_mosi2]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxo_bof]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxo_bof]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxo_bof]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxo_spi_s_miso]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxo_spi_s_miso]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxo_spi_s_miso]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxo_spi_xip_sclk]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxo_spi_xip_sclk]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxo_spi_xip_sclk]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxo_spi_xip_ss0]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxo_spi_xip_ss0]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxo_spi_xip_ss0]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxo_start_out]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxo_start_out]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxo_start_out]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxo_tdo_host_core]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxo_tdo_host_core]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxo_tdo_host_core]
set_output_delay -clock Clk2p5G  266.667  [get_ports xxo_uart_tx]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports xxo_uart_tx]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports xxo_uart_tx]
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodeesd1_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodeesd1_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodeesd1_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodeesd2_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodeesd2_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodeesd2_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodeesd3_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodeesd3_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodeesd3_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodeesd4_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodeesd4_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodeesd4_ana_pad_nom]
```

```
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodegnd1_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodegnd1_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodegnd1_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodegnd2_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodegnd2_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodegnd2_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodegnd3_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodegnd3_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodegnd3_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports athermdiodegnd4_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports athermdiodegnd4_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports athermdiodegnd4_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports aitestmux_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports aitestmux_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports aitestmux_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout0_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout0_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout0_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout10_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout10_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout10_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout11_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout11_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout11_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout12_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout12_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout12_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout13_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout13_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout13_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout14_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout14_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout14_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout15_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout15_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout15_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout16_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout16_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout16_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout17_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout17_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout17_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout18_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout18_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout18_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout19_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout19_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout19_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout1_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout1_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout1_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout20_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout20_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout20_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout21_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout21_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout21_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout2_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout2_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout2_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout3_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout3_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout3_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout4_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout4_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout4_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout5_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout5_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout5_acdac_ana_pad_nom]
```

```
set_output_delay -clock Clk2p5G  266.667  [get_ports avout6_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout6_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout6_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout7_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout7_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout7_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout8_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout8_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout8_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avout9_acdac_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avout9_acdac_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avout9_acdac_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avrf_drive_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avrf_drive_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avrf_drive_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avrfn_stim_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avrfn_stim_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avrfn_stim_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avrfp_stim_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avrfp_stim_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avrfp_stim_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avtestbb_drive_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avtestbb_drive_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avtestbb_drive_ana_pad_nom]
set_output_delay -clock Clk2p5G  266.667  [get_ports avtestmux_ana_pad_nom]
set_output_delay -clock Ext_Clk  2666.67  -add_delay  [get_ports avtestmux_ana_pad_nom]
set_output_delay -clock Clk_25  2666.67  -add_delay  [get_ports avtestmux_ana_pad_nom]

set_clock_groups -asynchronous -name JTAG_Clk_others_1 -group [get_clocks JTAG_Clk]
set_clock_groups -asynchronous -name Clk_25_others_1 -group [get_clocks Clk_25]


set_clock_groups -physically_exclusive -name clks_pexclusive -group [list [get_clocks Clk2p5G]
[get_clocks accel_0_clk_2p5G] [get_clocks accel_90_clk_2p5G] [get_clocks accel_180_clk_2p5G]
[get_clocks accel_270_clk_2p5G] [get_clocks I2C_Core_clk_2p5G] [get_clocks UART_Core_clk_2p5G]
[get_clocks GPIO_Core_clk_2p5G] [get_clocks SPI_S_Core_clk_2p5G] [get_clocks Timer_clk_2p5G]
[get_clocks SPI_XIP_Core_clk_2p5G] [get_clocks core_clk_2p5G] [get_clocks Host_core_clk_2p5G]
[get_clocks Host_core_aclk_2p5G] [get_clocks Smem_clk_2p5G] [get_clocks Ahb_fabric_clk_2p5G]
[get_clocks Dma_hclk_2p5G] [get_clocks Apb_fabric_pclk_2p5G] [get_clocks base_clk_2p5G]] -group
[list [get_clocks Ext_Clk] [get_clocks accel_0_clk_Ext] [get_clocks accel_90_clk_Ext] [get_clocks
accel_180_clk_Ext] [get_clocks accel_270_clk_Ext] [get_clocks I2C_Core_clk_Ext] [get_clocks
UART_Core_clk_Ext] [get_clocks GPIO_Core_clk_Ext] [get_clocks SPI_S_Core_clk_Ext] [get_clocks
Timer_clk_Ext] [get_clocks SPI_XIP_Core_clk_Ext] [get_clocks core_clk_Ext] [get_clocks
Host_core_clk_Ext] [get_clocks Host_core_aclk_Ext] [get_clocks Smem_clk_Ext] [get_clocks
Ahb_fabric_clk_Ext] [get_clocks Dma_hclk_Ext] [get_clocks Apb_fabric_pclk_Ext] [get_clocks
base_clk_Ext]]

set_clock_groups -asynchronous -name async1 -group [list [get_clocks core_clk_2p5G] [get_clocks
I2C_Core_clk_2p5G] [get_clocks UART_Core_clk_2p5G] [get_clocks GPIO_Core_clk_2p5G] [get_clocks
SPI_S_Core_clk_2p5G] [get_clocks SPI_XIP_Core_clk_2p5G]] -group [list [get_clocks base_clk_2p5G]
[get_clocks Host_core_clk_2p5G] [get_clocks Host_core_aclk_2p5G] [get_clocks Smem_clk_2p5G]
[get_clocks Ahb_fabric_clk_2p5G] [get_clocks Dma_hclk_2p5G] [get_clocks Apb_fabric_pclk_2p5G]] -
group [list [get_clocks accel_0_clk_2p5G] [get_clocks accel_90_clk_2p5G] [get_clocks
accel_180_clk_2p5G] [get_clocks accel_270_clk_2p5G]]

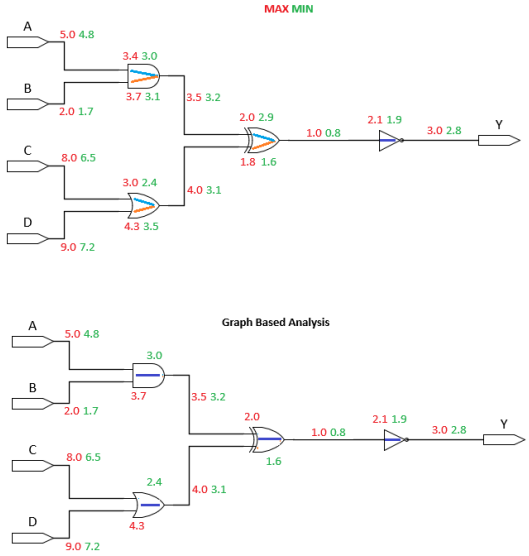
set_clock_groups -asynchronous -name async1a -group [list [get_clocks core_clk_Ext] [get_clocks
I2C_Core_clk_Ext] [get_clocks UART_Core_clk_Ext] [get_clocks GPIO_Core_clk_Ext] [get_clocks
SPI_S_Core_clk_Ext] [get_clocks SPI_XIP_Core_clk_Ext]] -group [list [get_clocks base_clk_Ext]
[get_clocks Host_core_clk_Ext] [get_clocks Host_core_aclk_Ext] [get_clocks Smem_clk_Ext]
[get_clocks Ahb_fabric_clk_Ext] [get_clocks Dma_hclk_Ext] [get_clocks Apb_fabric_pclk_Ext]] -
group [list [get_clocks accel_0_clk_Ext] [get_clocks accel_90_clk_Ext] [get_clocks
accel_180_clk_Ext] [get_clocks accel_270_clk_Ext]]

set_clock_groups -asynchronous -name async0 -group [get_clocks Clk2p5G]
set_clock_groups -asynchronous -name async0a -group [get_clocks Ext_Clk]
set_clock_groups -asynchronous -name async5 -group [get_clocks Timer_clk_2p5G]
set_clock_groups -asynchronous -name async6 -group [get_clocks Timer_clk_Ext]
set_clock_groups -asynchronous -name async_spi -group [get_clocks SPI_Clk]
```

D. GRAPH-BASED VS PATH-BASED TIMING ANALYSIS

Graph-based and path-based timing analysis are methods for calculating path delays in the static timing analysis engine used in synthesis and automatic place-and-route tools, as well as in sign-off static timing analysis tools. Graph-based timing analysis saves on compute resources and memory compared to path-based timing analysis, at the expense of being more pessimistic in terms of timing. Typically, design synthesis tools use graph-based timing analysis for path optimization, placement, and related tasks. Path-based timing analysis is reserved for the final sign-off to produce detailed timing reports for any remaining paths with path slack violations and to create directed ECO (Engineering Change Order) scripts to fix those violations.

Graph-based vs Path-based timing analysis can be roughly described as shown below.



If we take the circuit shown on the top, it shows the maximum and minimum delays through the wires and cells, respectively, for each path the signal can take. In graph-based timing analysis, the tool simplifies this by taking the worst-case timing path through the cells, independent of which input the signal passes through. Graph-based timing analysis is always more pessimistic than path-based timing analysis.

If we calculate the delay through the circuit on the right using both graph-based and path-based timing analyses, the results would be as follows.

Graph-based timing analysis:

```
Max delay PATH A-Y: 5.0 + 3.7 + 3.5 + 2.0 + 1.0 + 2.1 + 3.0 = 20.3
Min delay PATH A-Y:  4.8 + 3.0 + 3.2 + 1.6 + 0.8 + 1.9 + 2.8 = 18.1

Max delay PATH B-Y: 2.0 + 3.7 + 3.5 + 2.0 + 1.0 + 2.1 + 3.0 = 17.3
Min delay PATH B-Y:  1.7 + 3.0 + 3.2 + 1.6 + 0.8 + 1.9 + 2.8 = 15.0

Max delay PATH C-Y: 8.0 + 4.3 + 4.0 + 2.0 + 1.0 + 2.1 + 3.0 = 24.4
Min delay PATH C-Y: 6.5 + 2.4 + 3.1 + 1.6 + 0.8 + 1.9 + 2.8 = 19.1

Max delay PATH D-Y: 9.0 + 4.3 + 4.0 + 2.0 + 1.0 + 2.1 + 3.0 = 25.4
Min delay PATH D-Y:  7.2 + 2.4 + 3.1 + 1.6 + 0.8 + 1.9 + 2.8 = 19.8
```

Path-based timing analysis:

```
Max delay PATH A-Y: 5.0 + 3.4 + 3.5 + 2.0 + 1.0 + 2.1 + 3.0 = 20.0
Min delay PATH A-Y: 4.8 + 3.0 + 3.2 + 2.9 + 0.8 + 1.9 + 2.8 = 19.4

Max delay PATH B-Y: 2.0 + 3.7 + 3.5 + 2.0 + 1.0 + 2.1 + 3.0 = 17.3
Min delay PATH B-Y:  1.7 + 3.1 + 3.2 + 2.9 + 0.8 + 1.9 + 2.8 = 16.4
```

```
Max delay PATH C-Y: 8.0 + 3.0 + 4.0 + 1.8 + 1.0 +2.1 + 3.0 = 22.9
Min delay PATH C-Y: 6.5 + 2.4 + 3.1 + 1.6 + 0.8 + 1.9 + 2.8 = 19.1

Max delay PATH D-Y: 9.0 + 4.3 + 4.0 + 1.8 + 1.0 +2.1 + 3.0 = 25.2
Min delay PATH D-Y:  7.2 + 3.5 + 3.1 + 1.6 + 0.8 + 1.9 + 2.8 = 20.9
```

Based on the previous results, the maximum delay through the circuit, as determined by graph-based timing analysis, is always greater than or equal to the delay obtained with path-based timing analysis.

$Max_Delay_{GB} \geq Max_Delay_{PB}$

Likewise, when we consider the minimum path delay through the circuit, the delay obtained with graph-based timing analysis is always less than or equal to that obtained with path-based timing analysis.

$Min_Delay_{GB} \leq Min_Delay_{PB}$

By using graph-based timing analysis, automatic place-and-route tools save on memory and compute resources by analyzing only a minimal set of timing paths, which translate to the worst-case timing through the circuit.

Bibliography

[1] *A Fully Integrated Cryo-CMOS SoC for State Manipulation, Readout, and High-Speed Gate Pulsing of Spin Qubits* Jongseok Park;Sushil Subramanian;Lester Lampert;Todor Mladenov;Ilya Klotchkov;Dileep J. Kurian;Esdras Juarez-Hernandez;**Brando Perez Esparza**;Sirisha Rani Kale;Asma Beevi K. T.;Shavindra P. Premaratne;Thomas F. Watson;Satoshi Suzuki;Mustafijur Rahman;Jaykant B. Timbadiya;Saksham Soni;Stefano Pellerano. IEEE Journal of Solid-State Circuits Year: 2021 | Volume: 56, Issue: 11 | Journal Article | Publisher: IEEE

[2] *A Scalable Cryo-CMOS Controller for the Wideband Frequency-Multiplexed Control of Spin Qubits and Transmons* Jeroen Petrus Gerardus Van Dijk;Bishnu Patra;Sushil Subramanian;Xiao Xue;Nodar Samkharadze;Andrea Corna;Charles Jeon;Farhana Sheikh;Esdras Juarez-Hernandez;**Brando Perez Esparza**;Huzaifa Rampurawala;Brent R. Carlton;Surej Ravikumar;Carlos Nieva;Sungwon Kim;Hyung-Jin Lee;Amir Sammak;Giordano Scappucci;Menno Veldhorst;Lieven M. K. Vandersypen;Edoardo Charbon;Stefano Pellerano;Masoud Babaie;Fabio Sebastiano. IEEE Journal of Solid-State Circuits Year: 2020 | Volume: 55, Issue: 11 | Journal Article | Publisher: IEEE

[3] *CMOS-based cryogenic control of silicon quantum circuits* •Xiao Xue, Bishnu Patra, Jeroen P. G. van Dijk, Nodar Samkharadze, Sushil Subramanian, Andrea Corna, Brian Paquelet Wuetz, Charles Jeon, Farhana Sheikh, Esdras Juarez-Hernandez, **Brando Perez Esparza**, Huzaifa Rampurawala, Brent Carlton, Surej Ravikumar, Carlos Nieva, Sungwon Kim, Hyung-Jin Lee, Amir Sammak, Giordano Scappucci, Menno Veldhorst, Fabio Sebastiano, Masoud Babaie, Stefano Pellerano, Edoardo Charbon, Lieven M. K. Vandersypen. Nature volume 593, pages205–210 (2021) https://www.nature.com/articles/s41586-021-03469-4

[4] What is Library Characterization? https://www.synopsys.com/glossary/what-is-library-characterization.html

[5] *Intel Labs Quantum Computing Backgrounder*. https://download.intel.com/newsroom/2022/new-technologies/quantum-computing-backgrounder.pdf

[6] *What Intel Is Planning for The Future of Quantum Computing: Hot Qubits, Cold Control Chips, and Rapid Testing* , IEEE Spectrum, August 2020 https://spectrum.ieee.org/intels-quantum-computing-plans-hot-qubits-cold-control-chips-and-rapid-testing

[7] *Intel's New Chip to Advance Silicon Spin Qubit Research for Quantum Computing*, Intel Newsroom, June 2023, https://www.intel.com/content/www/us/en/newsroom/news/quantum-computing-chip-to-advance-research.html#gs.f3xkjm

[8] *Intel Takes Next Step Toward Building Scalable Silicon-Based Quantum Processors,* Intel Newsroom, May 1, 2024, https://newsroom.intel.com/new-technologies/intel-quantum-research-published-in-nature