sondrel

The 10 practical steps to model and design a complex SoC

White Paper

Author: Piyush Singh

Date: 10 February 2022

Content

Overview

It is important to model an SoC well in advance to avoid costly over design or insufficient performance and to create a hardware emulate on which representative end user applications can be run. Detailed architectural modelling provides reasonable estimates of the performance, power, memory resources, and the NoC (Network on Chip) configuration that will be required along with an indicative size of the die and what it is likely to cost. With this information, a customer can decide whether to proceed with the design, if it needs to be adjusted or even cancelled. Sondrel™ has created unique, proprietary modelling flow software, initially for use with Arm® and Synopsys® tools, that dramatically reduces the time to do this from months to a few days, which Sondrel claims to be an industry first for a services organisation. This article discusses how modelling is used in the ten steps of modelling and designing a complex SoC architecture.

Sondrel's new tool

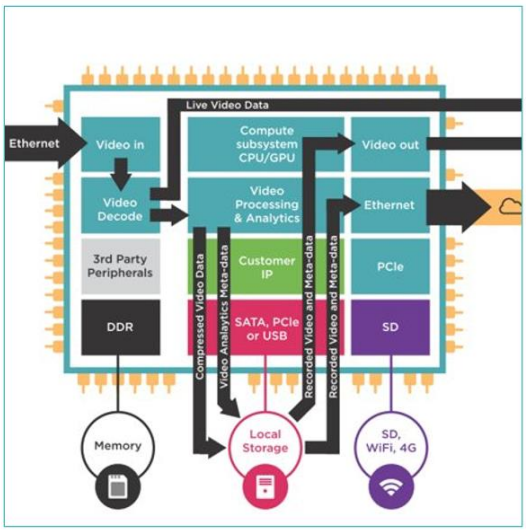
Modelling tools are available as standard items from leading vendors but what Sondrel does is to wrap the vendor's offerings with its own custom flow. The vendor's tools are limited in terms of automation and ways that they can be adjusted but Sondrel's new modelling flow tool adds a framework with a much greater number of settings that can be tweaked by the Sondrel Systems Architect who is working on the project. This is added using hooks into the vendor's software that are provided for this very purpose. Typically, users create customisation wrappers that are specific to the designs that they work on if not already present in a library of an ever-growing number of such wrappers. However, because Sondrel works on a wide variety of projects for a plethora of customers, it has defined a methodology and flows that are unique and broader in scope so that they can be used for almost any architectural exploration project.

The biggest benefit of the modelling flow's dramatic reduction in the time it takes to create a model and run simulations, is that Sondrel can provide customers with data on the likely performance of a proposed ASIC in a matter of few days to determine if the architecture proposed gives an appropriate set of numbers. If not, it is very easy and quick to run variants of the model simply by changing the settings of the existing model to decide which is the best one for the customer's application use case. Running each variation takes anywhere between a few minutes to an hour, so the whole process of model creation and running variants can still be done in a few days.

For comparison, converging on a candidate architecture without Sondrel's modelling flow tool would rely heavily on static spreadsheet modelling which would take several weeks and then each variant of the model to evaluate different architectures would each take weeks as each variant model would have to be created from scratch. Overall, that could total a number of months.

How an ASIC is modelled

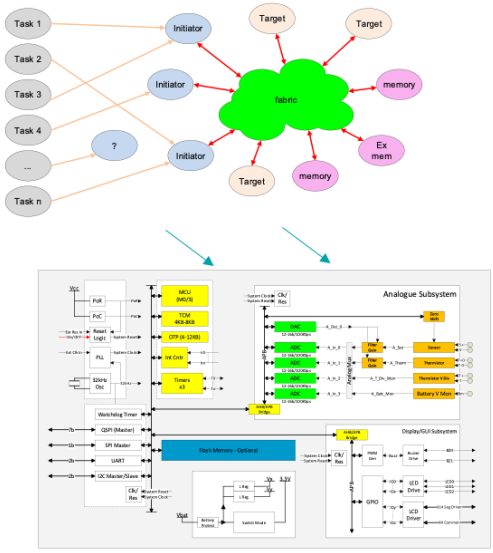
- Multiple 3rd party IP Components
  - CPU, GPU, DSP, SystemIP
  - Custom Hardware
- Multiple embedded processors
  - Multi-core clusters
- Off-chip Storage e.g. LPDDR
  - Performance and Power implications
- Standard Peripheral Interfaces e.g. MIPI, I2C, USB
- Multiple Operating Modes
  - e.g. MIPI-RX Global Shutter vs Rolling Shutter
  - e.g. output via Ethernet, PCIE, MIPI-TX



Most chips are created from compute elements such as CPUs, GPUs, DSPs and system IP along with custom hardware, IP and software. Off chip DDR memory storage is usually required as well. There are also multiple operating modes as determined by the end user application, i.e., settings within those hardware blocks that modify what they do. This can have implications on the memory traffic and access patterns that they initiate that go to the DDR controller via the NoC.

ASIC from Abstract view to Specification

- A Typical ASIC consists of a set of application level tasks
  - Tasks need to be mapped to IP via HW/SW partitioning
- A set of IP: Processing Systems, Peripherals, Interfaces, Accelerators, Caches
- Storage: On and Off Chip
- Interconnect Fabric: Bus, NoC
- Constraints
  - Performance requirements
  - Process Node, Package etc
- Validate Specification BEFORE implementation starts
  - Modelling can help achieve this

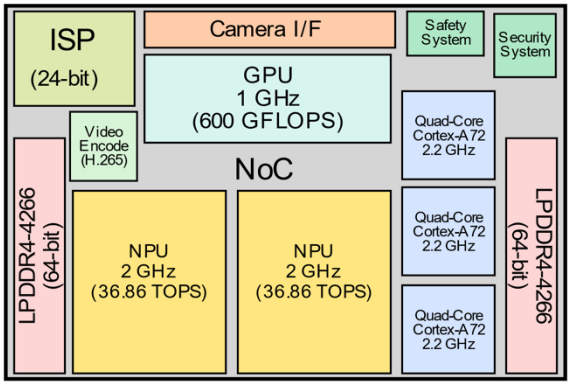


This starts as a set of application tasks, i.e., exactly what is it that the ASIC will have to do which are on the diagram of the abstract view (top right) as grey circles. These run-on initiators (blue circles) that are representations of hardware blocks that generate memory traffic. Targets (light orange) receive the memory traffic and do something with it. These are memory devices either on the die or off-chip like DDR.

There are a number of steps to convert an abstract view to a specification as shown in the bottom right of the illustration as a schematic diagram. This requires a modelling methodology that refines the abstract view to create a specification.

Interconnect fabric

- The wiring to connect all IPs. Unique for every SoC
  - Specification is dynamic. Cannot be finalized before layout
- Global Reach (all over chip)
  - Design significantly impacted by floor planning
  - Timing affected by pipelining often refined several times during implementation
- Many Interfaces / parameters
  - Most connected component in an ASIC
- Interconnect fabric have evolved from simple busses to more scalable complex Network-On-Chips
  - EDA technology to generate functionally correct interconnect has significantly improved
- Both Interconnect and Memory Sub-system matter for performance
  - Modelling can help validate performance early.

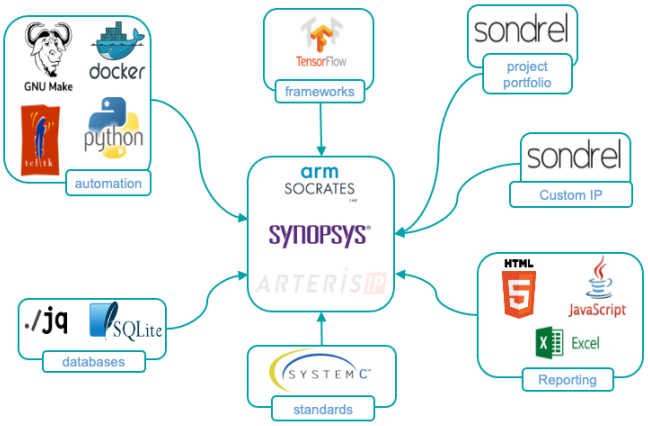


Central to all of this is the interconnect fabric – effectively the wiring – that connects all the compute and memory elements on the SoC together.

What is interesting is that it has global reach but not yet fixed shape in advance of the start of the design. Its shape is influenced by the floor plan and layout of all the blocks on the SoC. Because of that we have a different set of restrictions imposed on the NoC, one of which is the timing requirement because it does not have a fixed shape and can therefore span long distances from one end of the die to the other which can cause issues when it comes to timing closure. Because of this, the specification of the NoC is not completely finalised until much later in the design flow when the floor plan is actually realised. At that point, information from the floor plan is fed back into the NoC design in order to refine it as an iterative process. Modelling enables us to have a starting point for the NoC design.

Modelling uses and advantages

- There are many reasons to use models
  - Optimization of architecture
  - Specification Validation
  - Performance estimation
    - Power estimation
    - Throughput limitations
  - SW Development. HW/SW Co-design
  - System Integration
    - Modelling of interactions with external objects
- The choice of Modelling strategy is critical to success
  - Speed / Accuracy / Visibility
  - Tools Support
  - License costs
- Modelling stack relies on many tools and technologies



Modelling is used to come up with an estimate of what the architecture is going to be and play *what if* scenarios to work out the best configuration parameters of that architecture to arrive at a validated specification that summaries what the power and performance requirement will be. This can also be used for early software development by customers as they have something to try the software on, but it has to be born in mind that it does not have timing information so it does not show how fast the answers are computed.

As can be seen in the graphic on the right, the modelling stack requires several technologies to come together. In the middle are the EDA tools and around them Sondrel wraps its own custom tool and methodology on to Synopsys tools to create an all-encompassing environment of the modelling solution.

Various types of modelling

At the lowest level is **Dataflow modelling** which are programmes that are written in MATLAB, Python or C/C++ that are intended to capture the steps of an algorithm and what are the data processing pipelines to take data through to an answer but they do not contain any timing information.

Loosely timed or fast models are for early software development as they run executable code but they are not appropriate for architecture exploration because they do not have strict definition of timing between processes.

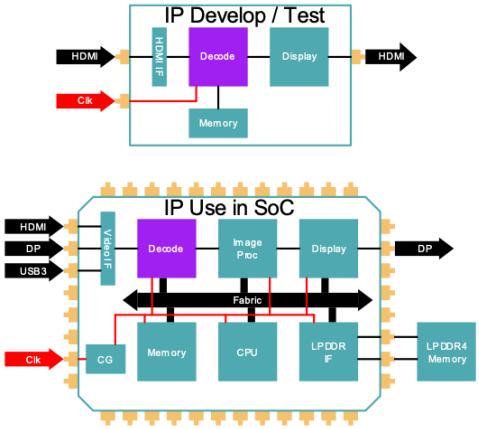
Most useful are the **Approximately Timed models** with a refined version being the **Fast Timed models** that are much better because they contain transaction level tracing to trace the timing of transactions and every transfer in the level of detail required to do architectural exploration and analysis.

At the highest, most detailed level are the **Cycle Accurate RTL simulation models** but this is not appropriate for architectural exploration as it is too slow as they model all signals at every clock event.

Why do we need performance exploration

- IP blocks are typically verified in isolation
  - Ignores external limitations
- IP Integration in chips is sometimes not straightforward
  - Master/Slave interfaces not matching
- Performance will be impacted by other IP
  - Clocking strategy
  - Shared Interconnect Fabric
  - Shared Memory
  - Complex memory timing and interactions
  - Interface adaptation
- Performance of the chip may be lower than expected
  - Less than the sum of the parts
  - Increasingly difficult to predict and verify

Performance of an interconnect is defined by whether a system using this NoC behaves correctly or not. i.e. Will a displayed video image freeze, the loud speaker click, or the embedded web-browser be too slow?

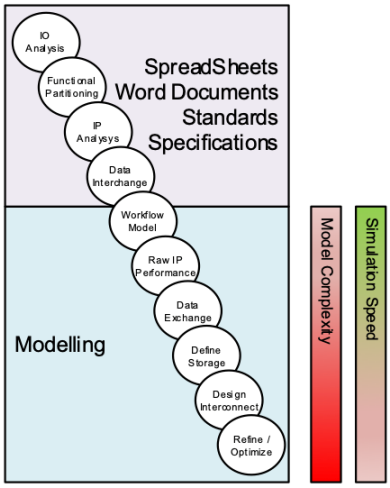


Typically, IP blocks are developed in isolation (top right illustration) but when a block is integrated into an SoC, there are other hardware blocks that also generate their own traffic profile and pattern (bottom right illustration). These impose a restriction on interconnect structure and the resources of the memory components. Performance exploration is used to decide on an appropriate size for the interconnects and memory subsystem by modelling the memory traffic patterns that are generated by all the subsystems as if they were running on the real system to a reasonable approximation. This enables the impact on the memory controller to be seen well in advance of the hardware or software being available which gives an indication of what the performance of the system will be well in advance that can help with the design and further downstream flows as the work progresses.

10 Steps to Architecture Success

The complex task of modelling becomes manageable by breaking it into stages and then

- Break the problem into stages
  - Divide and Conquer methodology
- Increasing detail at each stage
  - Reduced options to consider
- Use Modelling to answer questions
  - Each Model phase answers specific issues
- Focus resources as process progresses
  - Avoid wasted simulation time
  - Reduce data set to be analyzed
  - Reduce simulation size / count to offset slow down
- Consider issues in logical order
  - Simplifies cause and effect analysis
  - Reduces iterations required
- Iterate as soon as required
  - Identify issues early and fix before advancing
  - Reduces wasted design effort
- **Stages trade off accuracy versus speed**



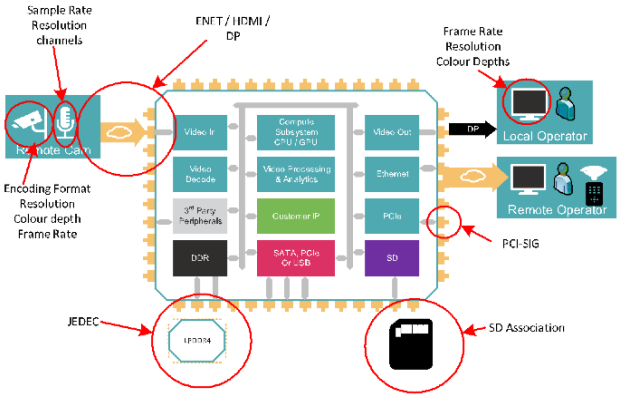
refining each stage with greater level of detail which can reduce the number of options that have to be considered. Each model phase enables this to be done to hone the focus as the increasing level of detail enables drill down to specific issues. This elimination process ensures that resources such as simulation time are not wasted on what would be dead ends and instead used on the reduced data set. At all times, issues must be considered in logical order so that cause and effect is clear and issues are fixed before advancing to the next iteration to reduce wasted design effort

The first four steps can be done on paper or on a spreadsheet by calculation to try and understand what are the input/output dataflows into the SoC and what their characteristics are.

The last six steps are simulation based where software models are constructed and simulations run to generate results that inform about the system.

Step 1: System OI Analysis

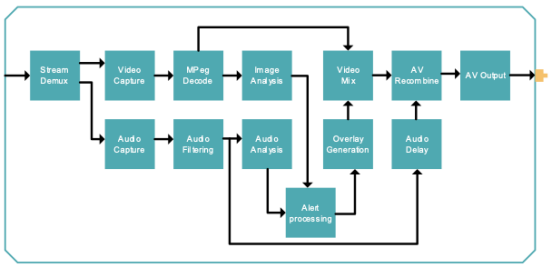
- Data Content
  - What does it represent
  - What will we do with it
- Applicable Standards
  - Data Structure
    - MPEG etc
  - Data Interfaces
    - DP, USB3, ENET etc
- Determine IO Constraints
  - Rates / Burstiness
  - Buffer requirements
  - Latency
  - Timing
  - Formatting



This determines what the data is and what are the I/O constraints such as burstiness, latency, timing, and data formatting to decide on the buffer requirements that is captured in a spreadsheet.

Step 2: Processing Analysis

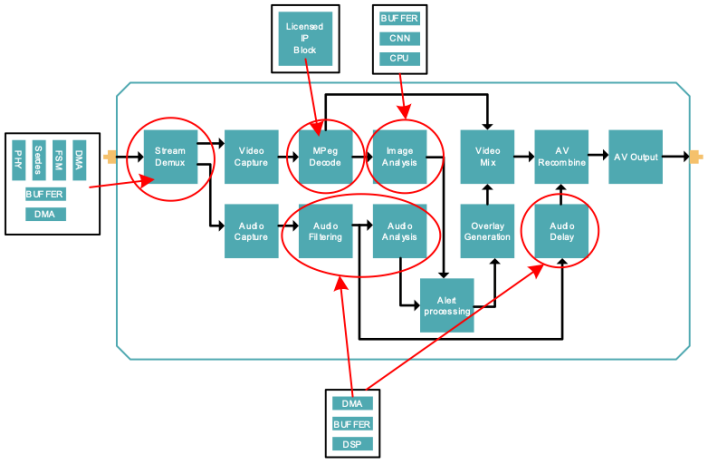
- Break the processing down into sub-tasks
  - Divide and Conquer Methodology
  - Group common pieces of functionality
- Define processing sequences
  - Virtual Pipelines
  - Key functionality
  - Operating modes
  - Constraints
- Define control mechanisms
  - SW / HW state machines
  - Flow control
    - Src / Dest synchronization etc
- This can be hierarchical
  - Refine each stage in isolation within constraints



This breaks the processing down into sub-tasks and groups parts of the SoC into common pieces of functionality. In the illustration, MPEG decode and image analysis are parts of an algorithm which needs to be performed on input data to generate the output.

Step 3: IP Analysis

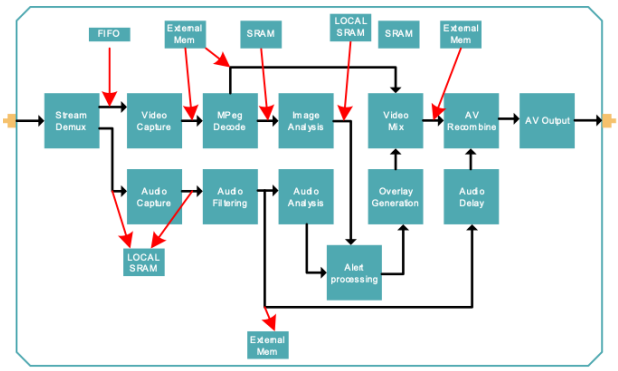
- Map Functions onto possible Implementations
  - Third party IP
  - Custom IP
  - HW / FW / SW
- Key Considerations
  - Performance
  - Cost
  - Area
  - Power Estimation
  - Latency / Throughput
  - Interface requirements



This step identifies what third party IP blocks will be required to perform the steps of an algorithm and how much memory and compute power they require from their datasheets that can be fed into the modelling environment to give a more accurate representation of what all the IP blocks will be doing.

Step 4: Data Interchange Analysis

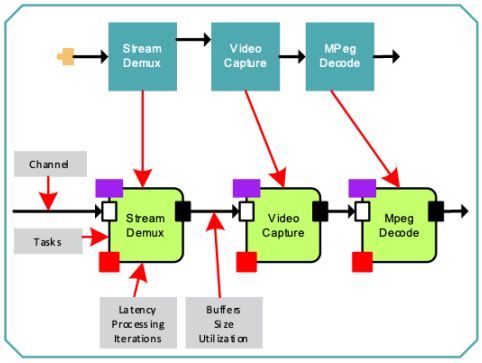
- How will data be exchanged ?
  - Local Buffers
  - System Memory
  - DMA
- What size buffers are required ?
  - Latency
  - Feedback
- Interfaces
  - System Bus protocol
  - Interface Widths / rates / burstiness
- Alignments
  - Natural data structure alignments



This covers the method of exchanging data in between parts of an algorithm such as on-chip SRAM or external DDR memory as well as FIFO which are small spaces of memory on chip. The decision between SRAM and DDR depends on the size of the data and how often it needs to be accessed with large pieces of data going to external memory and small pieces of data to SRAM or FIFO.

Step 5: Workflow Model (Transactional)

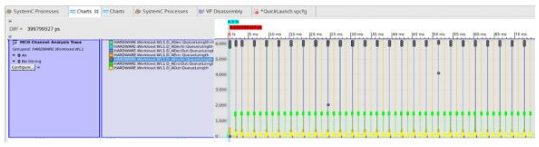
- Create TASK objects to model IP
  - Data Throughput
  - Latency
  - IO Timing (duration / start time)
- Specify Transactions between tasks.
  - Transaction can be data or control
  - Represented by size / time
- Specify channels between tasks
  - Virtual data / control paths
  - Can represent buffers for size analysis

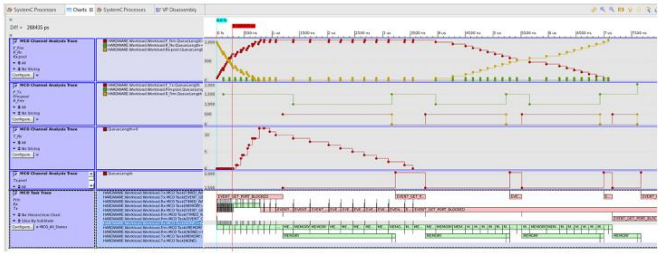


Now a software representation is created of what the different stages are. The first three blue blocks of the illustration in step 4, which form the conceptual view of the algorithm, are translated into the green boxes that represent the actual simulation objects and correspond to the different software stages of the algorithm. These require settings such as latency and processing cycles, and are joined by objects known as channels that indicate what the sequencing is.

Step 6: Simulate to verify processing

- Idealistic Simulation
  - All IP in perfect environment
  - Regulated by processing and IO Size / timing
- Interconnect Tasks using workflow
  - Infinite interconnect paths
    - No Contention
  - Channels monitor actual usage
    - Can use this to dimension HW
- Simulate to verify throughput
  - Identify traffic bursts / contention
    - When Data Required / Produced
  - Model traffic timing / IO duration
    - Based on width and clock rate
- Compute traffic statistics
  - Map requirements to functionality
  - e.g. Frame Rate
    - Total transfers per frame period
  - Estimate buffer requirements
  - Estimate latency

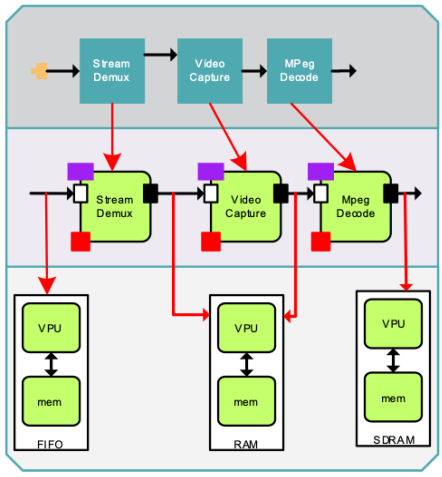




Having constructed all the simulation objects for the full algorithm, simulations can be run to see if the right sequencing of the algorithm has been captured. This can be visually checked as per bottom right image by using the visualisation tools that are available in the modelling environment.

Step 7: Quantify Data Interchange

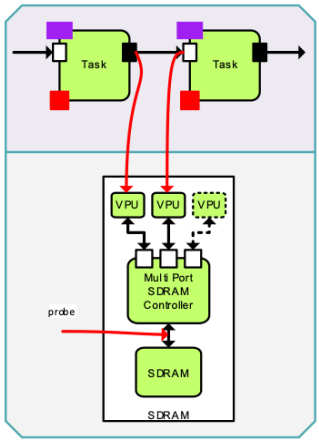
- Next step is to consider interface timing
  - Simulate “Unmapped” with timing
    - Blocks IO is independent
    - Indicates Total and peak Bandwidth
- Define communication domains
  - Domains operate independently
  - Used to model independent HW Entities
    - FIFOs
    - Shared Memory
  - Model domain as VPU + memory
- Assign each channel to a domain
  - Mapping is applied at simulation time
  - Domain may have many channels
- Evaluate each domain independently
  - Peak / Average bandwidth requirements
  - Contention between channels
  - Latency requirements



This is the next step on from the green blocks in step 5 to converge onto models of the hardware platform with VPUs (Virtual Processor Units) that will run the software of step 5, each with their own local memory. Here the interface timing can be considered and communication domains defined with their assigned channels and evaluated. It also enables the configuration of the VPUs to be verified as correct.

Step 8: Data Physical Exchange

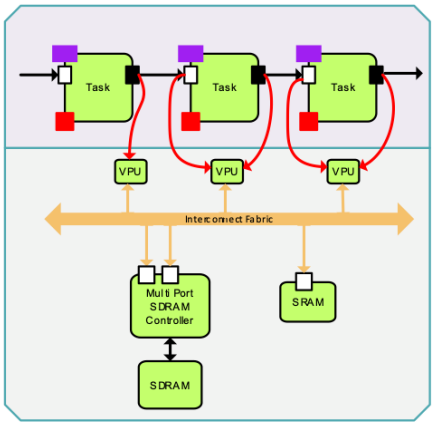
- Add memory HW simulation
  - Refine Domain Models
  - Assign Correct memory types
  - Extend domains with multiple ports for SDRAM etc
    - Ideal 1 per client
- Evaluate CORE memory performance
  - Ignoring Interconnect Fabric Contention
  - DRAM controllers buffer multiple transactions
- Evaluate interleaving / prioritization etc
- Detect issues for memory controllers
  - Address alignment
  - Burst size
  - Masked Operations



Here the memory that was available to each VPU is now remodelled as being connected to external memory via a common memory controller. This gives a more accurate representation of the connectivity of all the VPUs and memories in the final system.

Step 9: Implement Interconnect

- Implement interconnect
  - Include estimated timing (pipelining)
  - Evaluate impacts on performance
    - Latency / Bandwidth
- Typically ARM NIC, Arteris NoC
  - Configured using Generator
    - Using data from analysis
  - Generate FT Simulation model
  - Import Model into PA Ultra
  - Map tasks to FT transactors / VPU's
    - 1 per physical interface
- Re-simulate using workflow
  - Slower Simulation
    - Use optimized workflow set
  - Verify performance and margin
  - Adapt Interconnect to meet performance
  - Iterate to previous stages as required



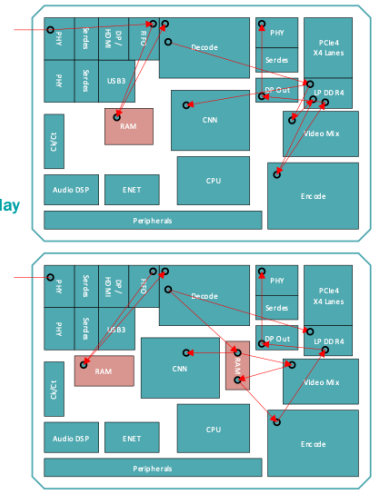
Now the interconnect fabric is added and instead of the direct connections between the VPUs and the memory controller, these are replaced by the interconnect fabric and the effects of this on the timing and performance evaluated. The interconnect fabric is then adjusted to meet the performance required with previous stages being redone to achieve the required results.

Step 10: Optimize Performance

Many further optimizations that can be evaluated using a Model

- Optimize memory architecture
  - Partition memory
    - Reduce active power
  - Additional Local Buffering
    - Reduce Latency
- Physical Memory Fragmentation
  - Reduce interconnect latency
- Prioritization
  - Ensure critical tasks not delayed
- Burst Sizes
  - Performance / latency tuning
- Data Alignment
  - Avoid expensive memory access clashes

*Effect of splitting memory
*Long paths = big pipeline delay
*Physically local memory can reduce latency and power dissipation



Now, with a good working model, by simply adjusting settings, various simulations can be run to identify bottlenecks, what constraints there are in the system, and which parameters should be adjusted to improve the throughput and reduce the latency of the SoC. These take a few minutes to an hour to run so that it is very easy and quick to test variants.

Further information about Sondrel can be found at www.sondrel.com