Assignment 3

Aly Sultan 001057787

EECE 7368

Professor Gunnar Schirner

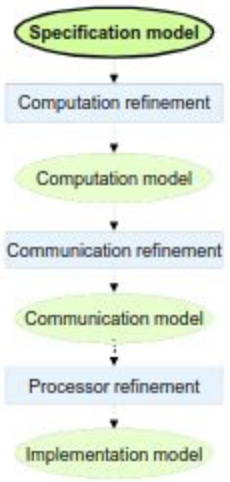
Problem 3.1: Goals of System Level Design Languages (10 points)
In class, we discussed six goals of an SLDL. Please select two out of these six which are in your consideration the most important for system level design. Name each goal, and describe its importance and influence in system level design. You may use an example each to illustrate your point.

- Goals
  - Executability
    - Validation through simulation
  - Synthesizability
    - Implementation in HW and/or SW
    - Support for IP reuse
  - Modularity
    - Hierarchical composition
    - Separation of concepts
  - Completeness
    - Support for all concepts found in embedded systems
  - Orthogonality
    - Orthogonal constructs for orthogonal concepts
    - Minimality
  - Simplicity

Of these six goals I find **Modularity** and **Executability** to be the most important for an SLDL. **Modularity** allows varied composition of heterogeneous hierarchical components which is integral to design exploration. Additionally, hierarchies lead to separation of concerns, which allows us to abstract away unnecessary details when assembling complicated designs which emphasizes focus only at the current level of abstraction. An example of this would be a convolution accelerator that contains different PEs and Channels. The composition of PE's is independent of the internals of each PE, so we can abstract away the details of the PE and focus only on their interaction. As a result we have created a hierarchy that results in separation of concerns. From the programmer's perspective at the PE level, the details of the PE don't matter, only their composition. The programmer can then begin design exploration to determine which optimum configuration of PE's will lead to the best performance. However, to be able to determine performance, the SLDL must have support for executability in order to determine preliminary performance metrics. Additionally, **executability** allows functional verification which is necessary before any design exploration happens (You have to know if the design works before you refine it)

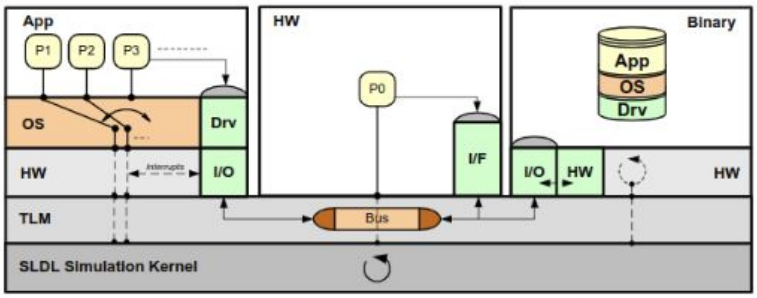
Problem 3.2: Specification Model (15 points)
Writing a good specification model is essential to a successful design process. We discussed in class how to write a specification model as input to a top down system level design flow. Name 3 requirements of a specification model, explain each requirement and outline what influence it has to the design process.

- **Functional and executable**
  - "Golden Model" (first functional model in the design flow) - Reference for all other models
- **High abstraction level**
  - No implementation details
    - Unrestricted exploration of design space
- **Separation of communication and computation**
  - channels and behaviors
- **Pure functional**
  - no structural information
- **No timing**
  - exception: timing constraints



A specification model should be as 1) **High level as possible**, and as a result it should contain no implementation details. This allows the design space for implementation to remain completely unrestricted in later stages of the top down design flow. Additionally, a specification model should be the 2) **Functional "golden model"** meaning that in order to validate functional correctness at any given point in the flow, the behavior of the model down the chain should be functionally identical to the specification model. If that doesn't happen, then that particular model is in violation of the the expected specifications required by the system. Finally, a specification model should 3) **Separate communication and computation** due to the fact that later points in the design flow require independent manipulation of compute and communication elements with a dependency between them. This is due to the fact that some elements related to computation refinement (like allocation and scheduling) need to be determined before the designer/ tool can be allowed to reason about the communication between computing elements.

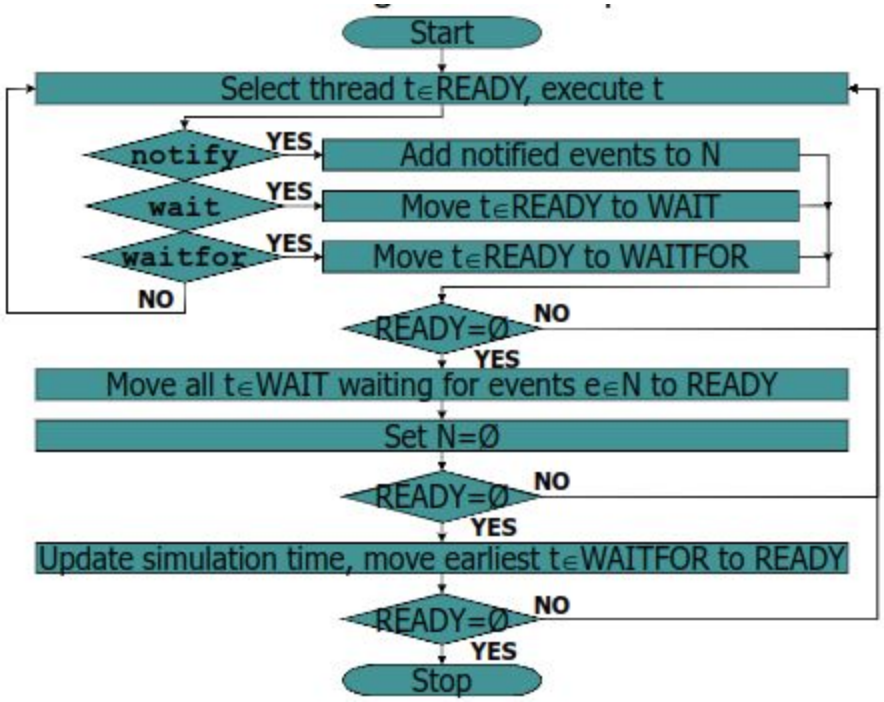
Problem 3.3: System Model (15 points)
Assume you are tasked with writing the drivers on top of a Linux operating system for a custom hardware device for video compression. The interface to the hardware accelerator and its interface is already specified (e.g. Memory Mapped Registers, DMA channels, functionality). The application is also available in C. How would you model the system to allow driver development while the integrated chip is not available? Describe how you at which abstraction level to capture the drivers and the accelerator. Describe your reasoning (advantages / disadvantages) for your selected abstraction level.



- **CPU model**
  - Source-level timing, energy, .. back-annotation
  - OS & processor models
- **Hardware/IP model**
  - Functional model
  - Timing, energy, … back-annotation
- **ISS model**
  - Cycle-accurate [GEM5]
  - Functional [QEMU] + timing, energy, … back-annotation

There are generally two approaches to driver development without fabrication or using an FPGA. One approach is to use a host compiled simulation with a simplified model of the operating system used, a HW abstraction layer (For interrupts), a TLM representing the interconnect between the OS and the custom accelerator and finally the actual accelerator captured with a modeling language like SystemC or SpecC. The other approach is to an ISS that simulates the target processor via dynamic binary translation on which an actual OS runs. All other abstractions (HW layer, TLM, and SystemC accelerator) are the same as the host compiled simulation. I would choose to develop the driver using the ISS model due to its **increased accuracy** at the disadvantage of **increased simulation time**. In the ISS model the driver is developed within an actual version of the operating system with the interface to the TLM representing the interconnect accessible via a hardware abstraction layer. This development method has the advantage of actually developing a driver within a real kernel that will eventually be deployed on the target system. Any details related to the OS running on that target processor will likely be represented. Developing the driver in an OS model may not result in a driver capable of running within a real operating system given the simplification of many OS elements in the OS model.

Problem 3.4: Simulation Performance (10 points)
Two virtual platforms are captured in an SLDL (e.g. SystemC, SpecC). Both models are identical except that they differ in their timing annotations. Platform A terminates at 125.048.350 time units with simulating 5.123 waitfor statements. Platform B terminates at 645.053.000 time units with simulating 1.736 waitfor statements. For which simulation does the designer have to wait less (shorter real-time)? State the underlying reasons for the performance difference.



Simulated time is completely detached from simulation time. The fact that Platform B takes a longer amount of simulated time has no bearing on the amount of time it actually spends running. Given the increased number of waitfors in platform A, it can be assumed that timing in Platform A is at a finer granularity. If all behaviors and events remain the same between platform A and B, platform A should take longer to longer than platform B to simulate. This is due to SpecC's discrete event simulation semantics where all behaviors that wait on a waitfor are parked until none are left, then the waitfor queue is sorted and the closest available behavior to run in simulated time is chosen. Since A has more waitfors then the process of adding behaviors the queue, sorting it and choosing the next available behavior will happen more frequently and thus **A should take longer to simulate in real-time** than B and **the designer will wait for less time when simulating platform B.**

Problem 3.5: System-On-Chip Environment (SCE) (20 points)

Report on model complexities (File→Statistics), the time it takes for simulating the design and encoding delays (maximum/worst-case delay over all frames) after each design step. The encoding delays are a product of the simulation, hence encoding delay is in virtual time. The encoding delays are printed as part of the simulation.

Design name ..... : vocoderSpec
Formatted lines of code ..... : 9086
Formatted code size ..... : 257557 bytes
Number of behaviors ..... : 107
... classified as 'leaf' ..... : 76 (71.0%)
... classified as 'sequential' ..... : 9 (8.4%)
... classified as 'concurrent' ..... : 11 (10.3%)
... classified as 'fsm' ..... : 10 (9.3%)
... classified as 'other' ..... : 1 (0.9%)

Simtime:
0.00 ms/ frame
Total: 163 frames * 0.00ms = 0.00ms
Realtime:
0.691s

Design name ..... : vocoderArch
Formatted lines of code ..... : 11893
Formatted code size ..... : 322270 bytes
Number of behaviors ..... : 156
... classified as 'extern' ..... : 2 (1.3%)
... classified as 'leaf' ..... : 111 (71.2%)
... classified as 'sequential' ..... : 16 (10.3%)
... classified as 'concurrent' ..... : 12 (7.7%)
... classified as 'fsm' ..... : 13 (8.3%)
... classified as 'other' ..... : 2 (1.3%)

Simtime:
4.17 ms/ frame
Total: 163 frames * 4.17ms = 679.71ms
Realtime:
0.702s

Design name ..... : vocoderSched
Formatted lines of code ..... : 12049
Formatted code size ..... : 325535 bytes
Number of behaviors ..... : 156
... classified as 'extern' ..... : 2 (1.3%)
... classified as 'leaf' ..... : 110 (70.5%)
... classified as 'sequential' ..... : 25 (16.0%)
... classified as 'concurrent' ..... : 4 (2.6%)
... classified as 'fsm' ..... : 13 (8.3%)
... classified as 'other' ..... : 2 (1.3%)

Simtime:
5.26 ms/ frame
Total: 163 frames * 5.26ms = 857.38ms
Realtime:
0.575s?

Design name ..... : vocoderNet
Formatted lines of code ..... : 12206
Formatted code size ..... : 331892 bytes
Number of behaviors ..... : 157
... classified as 'extern' ..... : 2 (1.3%)
... classified as 'leaf' ..... : 110 (70.1%)
... classified as 'sequential' ..... : 26 (16.6%)
... classified as 'concurrent' ..... : 4 (2.5%)
... classified as 'fsm' ..... : 13 (8.3%)
... classified as 'other' ..... : 2 (1.3%)

Simtime:
5.26 ms/ frame
Total: 163 frames * 5.26ms = 857.38ms
Realtime:
0.577s?

Design name ..... : vocoderTlm
Formatted lines of code ..... : 14378
Formatted code size ..... : 385456 bytes
Number of behaviors ..... : 180
... classified as 'extern' ..... : 2 (1.1%)
... classified as 'leaf' ..... : 116 (64.4%)
... classified as 'sequential' ..... : 27 (15.0%)
... classified as 'concurrent' ..... : 14 (7.8%)
... classified as 'fsm' ..... : 14 (7.8%)
... classified as 'exception' ..... : 4 (2.2%)
... classified as 'other' ..... : 3 (1.7%)

Simtime:
5.38 ms/ frame
Total: 163 frames * 5.38ms = 876.94ms
Realtime:
3.089s

Design name ..... : vocoderPam
Formatted lines of code ..... : 14121
Formatted code size ..... : 379778 bytes
Number of behaviors ..... : 173
... classified as 'extern' ..... : 2 (1.2%)
... classified as 'leaf' ..... : 116 (67.1%)
... classified as 'sequential' ..... : 27 (15.6%)
... classified as 'concurrent' ..... : 9 (5.2%)
... classified as 'fsm' ..... : 14 (8.1%)
... classified as 'exception' ..... : 2 (1.2%)
... classified as 'other' ..... : 3 (1.7%)

Simtime:
5.38 ms/ frame
Total: 163 frames * 5.38ms = 876.94ms
Realtime:
3.691s

Problem 3.6: Interpretation of Results (30 points)

a) **Leaf: and behavior:**
http://users.ece.utexas.edu/~gerstl/ee382v_f08/docs/SpecRM.pdf

(a) SCE provides statistics about model the complexities (File→Statistics). It reports the number of behaviors and classifies them as either 'extern', 'leaf', 'sequential', 'concurrent', 'fsm', or 'other'. Explain this classification (what is the meaning of each). For each category, briefly outline the properties (or potentials) a behavior of such a category has. Describe the importance of this classification with respect to the synthesis flow

Extern: Behaviors that contain behaviors/ channels imported from an external library

Leaf: According to **SCE Specification Model Reference Manual** a leaf behavior contains just plain ANSI-C code with the following rules:

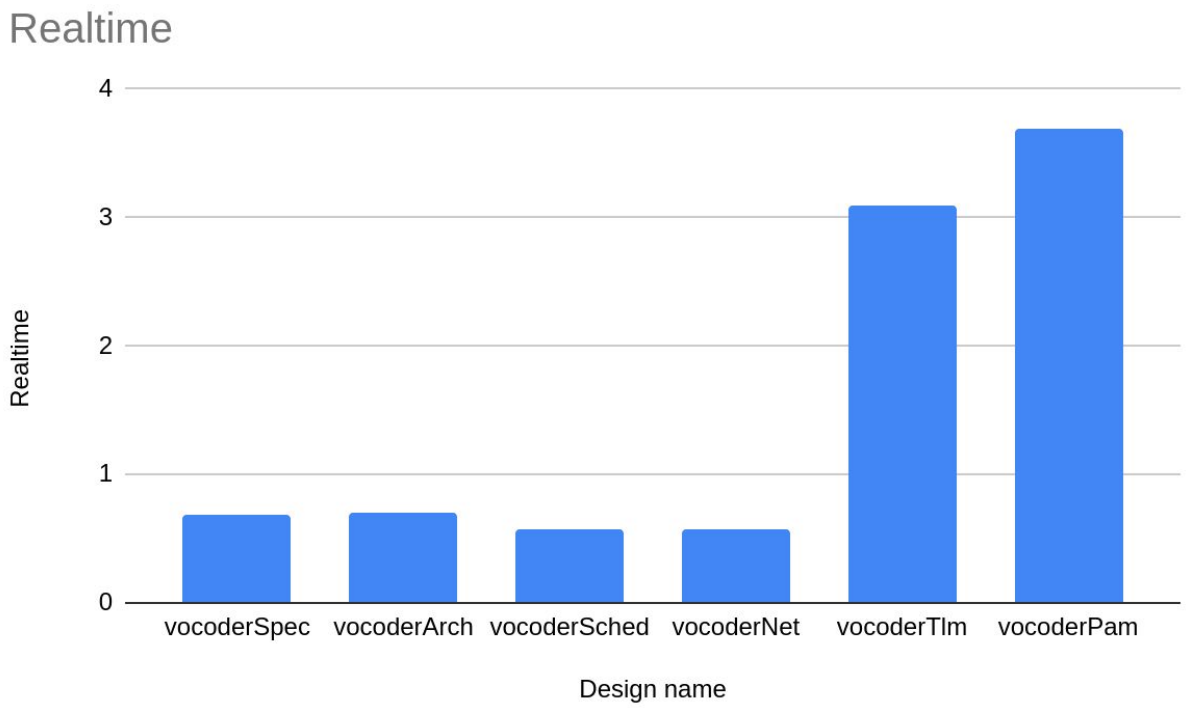
1. No References for ports
2. No Static variables
3. No Internal Behaviors or Channels
4. No SpecC statements like par
5. No piped variables
6. Only ANSI-C data types inside expressions and variables
7. No structs/ unions for leafs implemented in HW
8. No multidimensional arrays in HW
9. No pointers in HW

Any violation of the following will lead to the behavior being only implementable in software

Hierarchical Behaviors: According to **SCE Specification Model Reference Manual** a hierarchical behavior contains a composition of other hierarchical behaviors/ leaf behaviours determined by SpecC's behavioral hierarchy constructs in the code like sequential, par… etc. For example, behaviors classified as sequential will contain a sequential composition of hierarchical behaviors/ leaf behaviors. Same for par, fsm, pipe, etc...

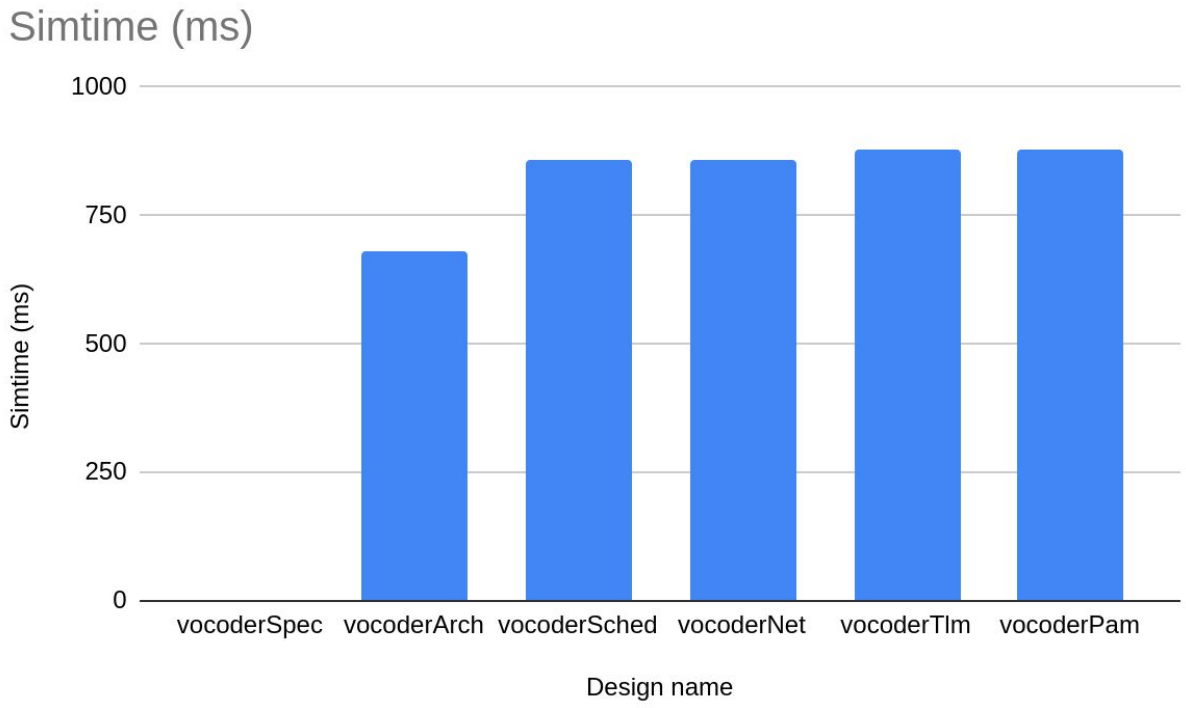
Other: Behaviors that fall outside of the previously defined classification, usually they are dirty behaviors that are implementable only in SW.

(b) In Problem 2.3, you captured metrics (number of lines, simulation time and simulated time) for each model in the refinement process. Graph the result for each metric individually. The x-axis enumerates the models. The y-axis shows the actual metric. For each graph, explain the development of the actual metric over refinement stages (explain the trend)

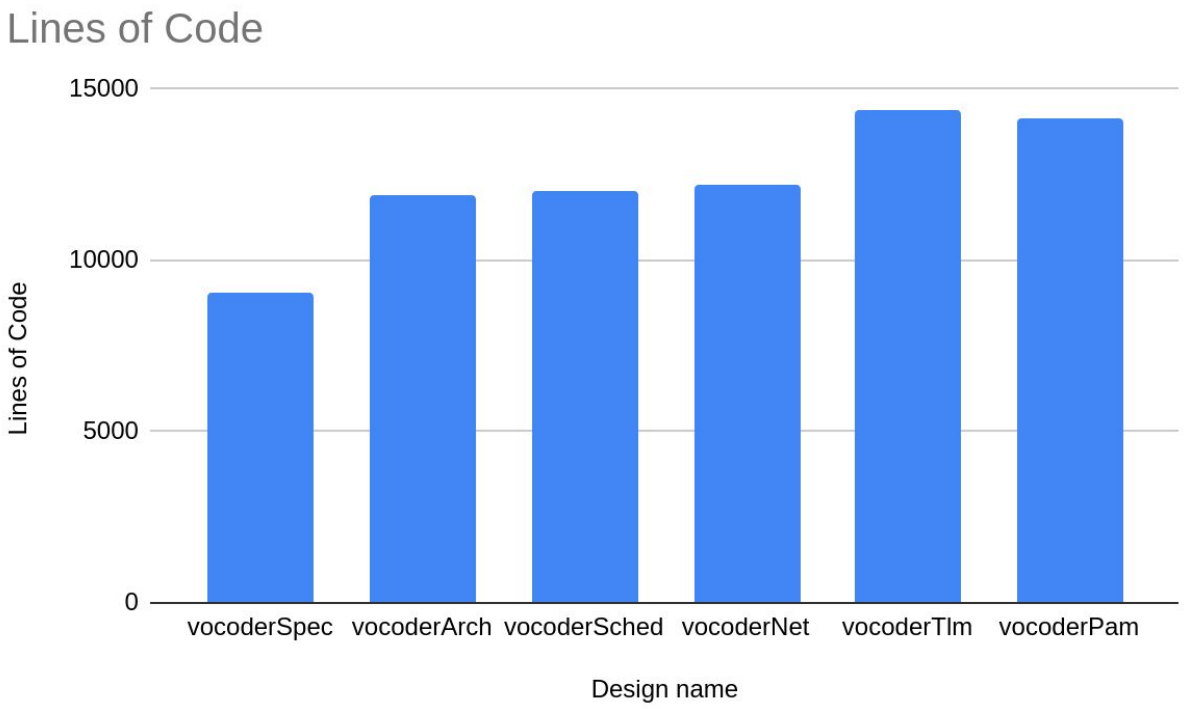


Real Time simulation usually increases as modes get more complex via back annotation. Usually complexity can be inferred from an increased number of lines. One exception to this rule however, is vocoderSched and vocoderNet. After back annotating static RTOS scheduling in vocoderSched simtime decreases. This is due to the elimination of behavioral hierarchy in the part of the design chosen to be implemented on a DSP. That elimination results in simplified execution due to the nature of SpecC's discrete event simulation semantics. Since SpecC doesn't have to manage varied behavioral hierarchies, picking behaviors to actually run becomes easier as that was determined statically by the scheduler and it doesn't have to be determined dynamically by SpecC. According to the **SCE Specification Model Reference Manual,** the output model of the network refinement stage is a BFM. vocoderNet maintains the same realtime speedup due to static scheduling despite mapping communication to busses and creating a BFM. This is confusing since the expectation for the BFM is that it is bus cycle accurate and hence it should take the longest amount of time to simulate. However, if network models don't include specifics regarding arbitration and lack of determinism as to when

communication starts this justified vocoderNet's runtime being lower than vocoderTlm and vocoderPam. If this assumption isn't true then a mistake has been made when following the tutorial for this assignment.

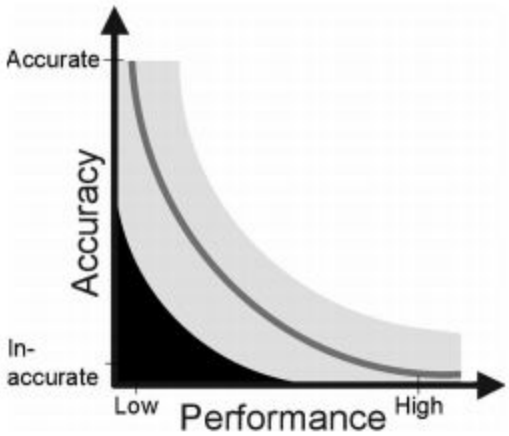


The trend of the graph is that with more details (Back annotations for timing an allocation) the longer the simulation time. vocoderSpec is completely untimed and flexible to allow for later refinements to all subsequent models to be done. More back annotation leads to greater accuracy but increased simulation time (see previous graph).



The baseline number of lines is determined in the Spec model. All other models after that introduce additional timing details via back annotation and thus increase the number of lines of code above the baseline. Generally more lines of code with a higher density of back annotation leads to increased complexity, which leads to increased realtime for the simulation. However, there are cases such as vocoderSched and vocoderNet that violate this rule due to the simplification of the SW side of the model via flattening the behavior hierarchy into sequential components, therefore more lines of code did not translate to increased complexity.

(c) The graphs for simulation time (how long does it take to execute the model) and simulated time (what does the model report as a time within the simulation) show a trade-off. This trade-off is called the TLM trade-off. Describe the TLM-tradeoff with a few sentences.



TLM tradeoff.

The TLM trade-off: https://ieeexplore.ieee.org/document/4294043

Transaction level models introduce a tradeoff between lower simulation time and lower accuracy. This implies that for many TLM models there is a trade off between speed and accuracy.

Between the PAM and TLM, the TLM offers higher simulation speed and technically lower accuracy. However, the results between PAM and TLM where identical. Additionally, if the output BFM of network refinement is bus cycle accurate, then it should have taken the longest and had the greatest accuracy, however, simulating it was very fast. Since I am unsure what the runtime of a cycle accurate model is given that we were not expected to follow the tutorial to RTL, I am unsure which model is actually accurate. Unfortunately, due to my findings, I am unsure of what level of detail each of these models actually has after being produced by the tool.