An SRAM Prototyping Tool for Rapid Sub-32nm Design Exploration and Optimization

Mudit Bhargava¹, Satyanand Nalam², Benton H. Calhoun² and Ken Mai¹
¹Carnegie Mellon University, Pittsburgh, PA; ²University of Virginia, Charlottesville, VA

Abstract—SRAM design in scaled technologies increasingly requires circuit innovations such as read/write assist techniques or alternative bitcells to ensure even basic functionality. However, the lack of a quick mechanism for understanding the impact of these circuit level changes on system level metrics makes accurate assessments of new circuit techniques difficult. Thus, we introduce Virtual Prototyper (ViPro), a tool that helps circuit designers explore this large design space by rapidly generating optimized virtual prototypes of complete SRAM macros. ViPro does this by allowing SRAM component specification with varying levels of detail – from ‘black-box’ descriptions to complete netlists – and by incorporating those components into a hierarchical model that captures circuit and architectural features of the SRAM to optimize a complete prototype. SRAM designers can use ViPro to generate base-case prototypes, which provide starting points for design space exploration, or to assess the impact of a low level circuit innovation on the overall SRAM design.

I. INTRODUCTION AND RELATED WORK

While process scaling has enabled ever-larger embedded memories, scaling trends such as process variability, device leakage, and soft error susceptibility make memory design increasingly difficult. In particular, shrinking device sizes lead to increasing variations that lower read and write noise margins and therefore reduce the functional yield of SRAM [1]. In the face of such scaling effects, the best way to design efficient, robust SRAMs at the 32nm process technology node and below remains largely an open question. Researchers have proposed a number of circuit techniques and alternative bitcells to deal with problems such as variation and leakage [2][3][4], but they tend to address only certain individual components of the memory. A change in any one of the key memory circuits will alter the optimal circuit topologies, partitioning, and architecture for the entire memory. This makes it difficult to choose a particular circuit technique or an alternative bitcell without an evaluation of the global benefits and overheads. However, such a holistic evaluation for accurate comparison of available design options is not possible early in the design and would require the designer to create complete SRAMs optimized for each technique, which increases design time and reduces productivity. Thus, there is a need for a methodology through which designers can generate and evaluate prototypes at every step of the SRAM design process that account for both circuit level issues and system level metrics.

To address this problem, we present a rapid Virtual Prototyper (ViPro) tool, which can create virtual prototypes of a complete SRAM macro, even early in the design process when many design details are missing. The prototype is “virtual” since some of the components of the SRAM may not be explicitly specified with a real schematic or netlist. Three key features of ViPro allow it to build optimized virtual prototypes beginning in the early design phases. First, ViPro uses a hierarchical and editable model of the SRAM to capture architectural features of the design (e.g. number of banks, bank size, word-size). This model comprises components (e.g. sense amplifier, decoder, bitcell) that are available in ViPro’s library. Second, ViPro allows the designer to specify these components with varying levels of detail – from ‘black-box’ descriptions (e.g. fixed numbers for figures of merit (FoMs), such as energy and delay), to analytical functions of various parameters (e.g. device sizes, environmental conditions), and eventually to accurate transistor-level netlists. Finally, ViPro uses a layer of abstraction to separate technology-independent design information and technology-dependent simulation details, to make components in its library reusable in any process technology. These features enable a designer to use ViPro to generate a re-optimized prototype when exploring a new process technology or evaluating different design options, such as assist circuits or alternative bitcells. When a complete set of components is available for a given hierarchical model, ViPro can generate fully functional SRAMs to act as base-case reference designs targeted at different points in the design space (e.g. low power, high speed).

There are several memory design and FoM characterization tools available, but unlike ViPro they do not support integrated circuit-system co-design. At one end of the spectrum are architecture-level modeling tools like CACTI [5], used by computer architects to obtain quick estimates of SRAM FoMs such as access time, power, and area. CACTI makes fixed, built-in assumptions regarding the SRAM circuit components and optimizes at the architecture level only. At the other end of the spectrum are transistor-level optimizers [6][7] that are good at choosing optimal device characteristics (e.g. W/L, Vth, Vdd etc.) for a given circuit topology, but are not helpful in choosing an optimal circuit topology or micro-architecture.

Finally, memory compilers [8][9] generate memories based on user-defined parameters, but they are more a deliverable from memory design teams rather than a tool for memory design teams. ViPro fills the gap between these tools by providing an optimization and design space exploration tool for SRAM that supports circuit-system co-design.

An analogous design tool for the design and optimization of high speed links was previously presented [10] and relies heavily on analytical expressions that are specific to high speed links. ViPro targets another complicated mixed signal design problem, SRAM, with a more generalized approach. While ViPro can support analytical modeling to accelerate the optimization process, it also supports a fully simulation based approach or a mixed modeling/simulation framework.

II. DESIGN METHODOLOGY USING ViPRO

Fig 1 shows the structure of ViPro, which comprises two main sub-systems: a characterization engine (CE), and a hierarchical meta-compiler (HMC, ‘meta’ to distinguish it from a true compiler that produces complete designs). The HMC implements a hierarchical model of the memory that allows a...
designer to define components of the memory with varying levels of detail and accuracy. The CE provides a technology-agnostic framework for generating data from simulation about those components, so that ViPro can operate in any process technology. Since the CE produces the building block components for the HMC, we begin by describing the CE.

A. Characterization Engine (CE)

Characterization of a circuit or process refers to using simulations to evaluate FoMs of interest for that circuit or process. The CE is a custom interface (wrapper) for a conventional SPICE simulation environment for characterizing the process technology and various circuit blocks that can be used as components of the SRAM. It uses simulation templates that abstract out the process dependencies from the simulation setup. Then, by combining these templates with process-related data, it produces the simulation-ready netlist and directives for that particular process and runs it in SPICE. Post-processing (e.g. Matlab) scripts that are coupled with the templates extract useful information from the resulting data. The CE also allows the designer to link groups of related simulations, which permits a design to remain tied to the analyses that illuminate the associated trade-offs and design issues, even when the underlying process technology changes.

The CE has two uses within ViPro. First, using simulation templates, it characterizes a technology or process through device-level simulations (e.g. I-V curves, FO4 delay, FET gate and junction capacitances, leakage current characteristics). Scripts extract these parameters for later use by the HMC to evaluate the system level FoMs. Second, the CE includes a library of simulation templates of SRAM components that can be characterized in terms of global (e.g. energy and delay) and component-specific FoMs (e.g. noise margins for a bitcell and offset for a sense amplifier). Designers can expand this library by adding simulation templates for new circuits of interest.

B. Hierarchical Meta Compiler (HMC)

The HMC contains and manages a hierarchical model of the memory (HMM) that acts as a prototype of the design. A HMM is implemented using object-oriented programming in Matlab. Fig 2 depicts an example hierarchy. For demonstration, we show only two levels of hierarchy that support low-capacity, single macro memories. Components in the hierarchy are implemented as classes with properties defined for parameters (device sizes, voltages, etc.) and FoMs (energy, delay, area, etc.). The method in a class that defines an FoM of the component supports varying levels of detail. For example, the method can assign constant values to the FoMs, equivalent to treating the component as a black box with estimated behavior. The methods can alternatively use analytical expressions or macro-models to compute the FoMs from the defined parameters. For instance, the energy of a wordline driver can be roughly estimated using $C V^2$ calculations, with the value of $C$ determined using gate and junction capacitance values of the transistors driven along the wordline. These capacitances are in turn estimated from simulations in the CE. Delay of components can be calculated using logical effort calculations. Finally, components can be specified with transistor level netlists from which the CE can directly obtain FoM data across different values of the input parameters.

Using inheritance in the HMM classes allows properties common to a branch of the hierarchy to filter down to the leaf node of that branch. Parameters affected by multiple components are defined as properties of a top-level class. This captures the interactions between different blocks when optimizing the overall design. For example, the offset of the sense amplifier ($V_{OFFSET}$) and the strength of the bitcell’s access transistor both influence the optimum number of rows ($Num_{Rows}$) in the memory array. To reflect this, $Num_{Rows}$ is defined in the top-level SRAM class that filters down to both the sense amplifier and the bitcell classes. The local component methods in these two classes define the relationship between $Num_{Rows}$ and $V_{OFFSET}$, and the drive strength of the bitcells. This allows top-level optimization while keeping track of local dependencies. As the design progresses, more of these local parameter relationships are defined, and the virtual prototype that the HMM represents becomes increasingly more accurate.

C. Tool flow

This section describes interaction of a user with ViPro. The designer provides the following inputs:

- Process technology models
• Top-level memory specifications – Capacity, word-size, supply voltage, operating temperature, etc.
• Constraints on metrics like energy, delay and area (optional – may be defined locally or globally)
• Component specifications (optional) – black-box estimates, analytical models or CE simulation templates

Fig 3 shows the steps involved in using ViPro for generating virtual prototypes. As shown in the figure, ViPro is designed to support multiple iterations to produce an optimized design. The designer leverages the virtual prototype and accompanying information produced after an early iteration to explore the design space by changing circuit or process options.

**Step 1: Process Characterization**

The first step is to characterize the process technology being used through device-level simulations (discussed in II.A) and needs to be executed only once per process technology.

**Step 2: Characterize SRAM components**

In the second step (discussed in II.A), the CE characterizes components from the library (once per technology) and any new circuits provided by the user. Any components unavailable from the CE must be defined using black-box estimates or analytical models. As more components are added to the CE library, the accuracy of the virtual prototypes improves.

**Step 3: Generate optimized virtual prototype**

a) **Base-case prototype**: ViPro can generate optimized base-case prototypes targeted towards either high-speed or low-power by using existing compatible components. The base-case prototypes provide a convenient starting point for a designer interested in creating a more optimized custom design. They can also act as ‘control’ memories for comparison with virtual prototypes that are produced later in the iterative design process. In an unfamiliar technology, the base-case designs can help the designer by immediately identifying the features of the design that need the most attention by reporting which local constraints (e.g. bitcell read or write margin) are not met after optimization.

b) **Custom prototype**: ViPro generates virtual prototypes (HMMs) and calculates their FoMs by combining components specified by the designer with the components characterized using the CE. For example, the energy and area of a prototype is calculated by summation, and the delay is calculated by adding the delay of all the components in the critical path. If multiple options exist (e.g. multiple decoder topologies in the library), ViPro evaluates each option and selects the best.

Next, as per designer-specified constraints, ViPro finds the configuration for the prototype for optimal energy or delay. This optimization can be specified for a wide range of parameters or knobs (e.g. array dimensions, device sizes, \( V_{\text{TH}} \), and voltages), and optimization objectives (e.g. energy, area, delay and yield). ViPro supports existing optimization tools as plug-ins to solve the optimization problem. As outputs, the tool produces a custom virtual prototype along with FoM information and parameter values.

**Step 4: Designer guidance**

At this point, the designer intervenes in the tool flow by exploring different circuit options (e.g. read/write assist techniques, alternative bitcells, etc.) to further optimize the design. By changing the component specifications and running ViPro iteratively, designers can quickly generate and compare several virtual prototypes (demonstrated in sections III.A and III.B). Alternatively, the designer can exploit the technology-agnostic nature of ViPro to compare prototypes for different process or device options. This kind of comparison is especially useful for many “fab-less” companies that have to choose between different processes for their design. Thus, for example, when porting an existing design to a new technology, the designer can quickly see how the optimum configuration of the design changes (demonstrated in section III.B). This technology agnostic feature also allows for process-circuit co-design, since the optimal circuit and architecture selections will change in response to process alterations.

A key insight here is that as the designer runs the tool multiple times and compares several virtual prototypes, his understanding of the trade-offs involved in the design increases. Thus, he finalizes more and more components, which improves the accuracy of the prototype. Ultimately, as the design nears completion, all the components in the memory are specified in terms of circuit netlists.

**III. USE CASES FOR ViPro**

ViPro has the following key uses.

• **Base-case generation**
• **Evaluation of different circuit topologies**
• **Quick re-evaluation with process/technology change**

The following examples demonstrate these use cases.

A. **Base-case generation and circuit topology evaluation**

We can use ViPro to generate control memories that can be used for comparison with improved designs. Here we show an example where ViPro is used to create virtual prototypes of
control SRAMs with different design specifications. For this example we use 32nm Predictive Technology Model (PTM) [12]. The memory size required is 16 kbit with a word-size of 16 bits. We use ViPro to explore prototypes with all possible array configurations, i.e. combinations of number of rows and columns, and vary the column mux (interleaving) from 1 to 32 (Fig 4). Single-core array configurations are compared for their energy per access and delay characteristics. The figure suggests that for energy critical applications, we should choose number of rows and columns as 512 and 32 respectively. But for high-performance applications, a prototype with number of rows and columns both equal to 128 is more appropriate.

ViPro also provides component-wise detailed characteristics. By reviewing the component metrics, the designer can quickly prioritize the design effort of certain components to achieve the desired optimal prototype.

B. Evaluation of different circuit topologies

ViPro can be used for rapid re-optimization of the design and complete evaluation of the effects of different circuit component topologies (e.g., bitcells, sense amps) on macro-level design parameters (e.g., access time, bit density). The following example shows how a designer can quickly evaluate the impact of a novel circuit component on the global metrics without the need of a complete design.

Using ViPro, a designer explores two sense-amplifier (SA) designs, one of which resolves faster at the cost of increased offset and power. For the given delay and power constraints, the optimal SRAM prototypes using each of the design options have different array configurations (number of rows and columns). Only one of them fits the specified footprint on the chip. Hence, without implementing the design, one of the SA options can be ruled out by the designer.

C. Re-evaluation with process/technology change

By rerunning the characterization step, ViPro can rapidly re-optimize the virtual prototype when the process technology underlying a design changes. This allows for the complete evaluation of a process change and its effects on macro-level design parameters (e.g., access time, bit density). This tightens the iteration loop allowing for process/circuit co-design.

ViPro can be used to generate prototypes across a range of technologies (real and predictive) to understand how scaling will affect SRAM design. This helps the designer to redesign the memory and satisfy the overall requirements.

We use predictive models [12] to demonstrate this use of ViPro. Using the architecture and component topologies used by the base-case prototype, we generate virtual prototypes for a range of constraints on the access time (delay) of the memory, and obtain the optimal E-D curves shown in Fig 5. We do this for a commercial 45 nm technology, and for 32 nm and 22 nm PTMs. We assume bitcell and Vdd0 Scaling. In this example, we have a single core array with small swing bitline architecture. The memory size required is 16 kb, and the word-size is 16 bits. Access time may be reduced by reducing the number of rows and making the bitlines shorter. But this in turns increases the number of columns and increases the total capacitance that is charged and discharged per clock cycle, resulting in increased energy per read/write operation.

Fig 5 shows the pareto-optimal trade-off curves for energy per access versus access time. For an energy constraint of 2 units, the optimum array configuration varies across technologies. This information can be used by the designer to rapidly make critical design decisions that are influenced by the array configuration, such as floor planning.

![Fig 5: Analysis of optimal E-D curves using ViPro across technologies.](image)

### IV. Conclusions

In this paper, we have presented ViPro, an SRAM design tool that aids SRAM designers by quickly generating virtual prototypes at every step of the design process. We have shown three main uses of ViPro. First, it can generate base-case memories. Second, it can evaluate the impact of different circuit options on the overall design. This speeds up designer response to SRAM circuit and architectural innovations and helps quickly make critical decisions. Finally, it can be used to re-optimize a design for different process technologies or device types.

### REFERENCES