GTX: The MARCO GSRC Technology Extrapolation System*

http://vlsicad.cs.ucla.edu/GSRC/GTX/

Andrew E. Caldwell, Yu Cao,[1] Andrew B. Kahng, Farinaz Koushanfar, Hua Lu,[2]
Igor L. Markov, Michael Oliver, Dirk Stroobandt[3] and Dennis Sylvester[4]

UCLA CS Dept., USA; [1] UC Berkeley EECS Dept., USA; [2] UCLA EE Dept., USA;
[3] Ghent University ELIS Dept., Belgium; [4] Synopsys, Inc., USA

{caldwell,abk,farinaz,imarkov,oliver}@cs.ucla.edu; ycao@eecs.berkeley.edu;
hua@ee.ucla.edu; dstr@elis.rug.ac.be; sylvest@synopsys.com

Abstract

Technology extrapolation — the calibration and prediction of **achievable design** in future technology generations – drives the evolution of VLSI system architectures, design methodologies, and design tools. This paper describes initial experiences with development and use of GTX, the MARCO GSRC Technology Extrapolation system. GTX provides a robust, portable framework for *interactive* specification and comparison of modeling choices, e.g., for predicting system cycle time, die size and power dissipation. We use GTX to reveal surprising levels of uncertainty (*modeling* and *parameter sensitivity*) in widely-cited cycle-time models that drive recent roadmaps. We also describe new SOI and bulk device models that have been developed for GTX, as well as studies of power dissipation and delay uncertainty under various implementation assumptions for global interconnects.

Keywords

Technology Extrapolation, VLSI, Expert Systems.

1 Introduction

Leading-edge VLSI system design aggressively exploits new process technologies, circuit techniques, design methodologies and design tools. It is thus difficult to predict the envelope of *achievable design* – e.g., with respect to power, speed, area, manufacturing cost, etc. – for a given behavior or function, in a given (future) process technology. On the other hand, such *technology extrapolation* activity directly influences the evolution of future VLSI system architectures, design methodologies, and design tools. Via roadmapping efforts such as the International Technology Roadmap for Semiconductors (ITRS) [8], technology extrapolation also influences levels of investment in academic research, career choices for faculty and graduate students, as well as private-sector entrepreneurial activity.

Highly influential technology extrapolation systems, developed 5-10 years ago, are due to Bakoglu and Meindl (SUSPENS) [2], Sai-Halasz [15], and Hewlett-Packard Laboratories (AIM) [12]. More recent "second-generation" systems include GENESYS [6], RIPE [14] and BACPAC [17, 18], along with Roadmap-related efforts [8, 7] and innumerable internal projects throughout industry and academia. Typically, each system provides a plausible "cycle-time model" and estimates of die size and power dissipation, based on a small set of descriptors spanning device/interconnect technology through system architecture. In Section 2.2, we observe that (i) these systems are often incomparable, (ii) they are "hard-coded" (hence it is difficult to assess their quality and to explore changes through modeling choices), and (iii) their development has entailed a near-total duplication of effort. These observations motivate efforts toward an entirely new level of technology extrapolation capability. Our GSRC Technology Extrapolation (GTX) system has been developed with the goals of *flexibility*, *quality* and *prevention of redundant effort* in mind.

The GTX system addresses these goals by providing an open, portable *framework* for specification and comparison of alternative modeling choices. A fundamental design decision in GTX is to separate model specifications from the derivation engine. This separation is achieved by a human-readable ASCII grammar. As domain-specific knowledge is represented independently of the derivation engine, it can be created and shared by multiple users. Additional extension mechanisms allow specialized prediction methods, technology data sets, and even optimization engines to be encapsulated and shared within GTX; this further reduces the amount of effort that is diverted from actual creation of best-possible prediction models.

Section 2 reviews relevant previous work in VLSI technology extrapolation and puts the GTX goals in perspective. Section 3 describes the architecture and implementation of GTX. As an example of increased possibilities for technology extrapolation, Section 4 assesses the *parameter sensitivity* and *modeling sensitivity* of several widely-referenced cycle-time models. Our analyses reveal surprising levels of uncertainty and sensitivity to modeling choices. We also analyze a new model for SOI and bulk devices and study the power dissipation and delay uncertainty under various implementation assumptions for global interconnects.

*This research was supported in part by Cadence Design Systems, Inc., Synopsys, Inc. and the MARCO Gigascale Silicon Research Center. Dirk Stroobandt is a Postdoctoral Fellow of the Fund for Scientific Research (F.W.O.) – Flanders; his work on GTX was performed during his stay at UCLA as a visiting researcher. Andrew Caldwell is now with Simplex Solutions, Inc.

2 Related Work and GTX Goals

2.1 VLSI Technology Extrapolation

A number of previous systems attempt to forecast and estimate the performance of microprocessors. Given that GTX can *flexibly* accommodate the addition of new rules and inference chains, a baseline GTX implementation is intended to encompass these previous models. Four systems – SUSPENS, GENESYS, RIPE and BACPAC – are especially noteworthy.

SUSPENS [2] is the forerunner for most technology extrapolation systems. SUSPENS predicts the clock frequency, chip area and power dissipation. It ignores on-chip cache and memory structure, as well as details of multi-layer interconnect structure and clock distribution. SUSPENS is also oblivious to such DSM effects as scaling and noise.

GENESYS [6] offers both a GUI for MS Windows (95, 98, NT) and a command-line interface (it has no Web interface). The GENESYS output file is divided into four main sections: device/material, circuit, interconnect, and system. The device section contains information concerning device parameter calculations such as device capacitance and drain currents. The circuit section is broken into four parts: area, capacitance, delay and energy. The interconnect portion provides information on the interconnect structure of each wiring tier, as well as results of certain repeater insertion optimizations. System-level outputs include throughput, maximum clock frequency, CPI, and delay times for random logic and interconnects.

RIPE [14] explores the effect of interconnect design and technology tradeoffs on IC performance. Default input data are extracted from the NTRS roadmap. Memory and the multilayer interconnect structure are taken into account. No estimations of noise or reliability are available; other limitations are in the modeling of electromigration, non-ideal scaling, etc. The RIPE executable, available via Web interface, can be used in two basic modes: (i) given global wire parameters, RIPE estimates frequency, power dissipation and wiring efficiencies; and (ii) detailed "wiring strategy". The user can choose between the two modes, but cannot add new parameters and rules.

BACPAC [18] is based on a system-level performance model that consists of smaller-scale analytical models. The innovations of BACPAC compared to earlier models include attention to power dissipation, on-chip memory, process variation, and other effects. BACPAC is applicable to both ASICs and microprocessors. It attempts to enhance the accessibility of technology extrapolation via a Web-based interface; users can enter parameter values and receive relevant technology predictions. However, the derivation flow is mostly fixed, and users cannot add new parameters and rules. BACPAC does not capture architectural attributes or system reliability.

Previous work on (general) artificial intelligence systems include Design Sheet [13], TkSolver [19], and UniCalc [1]. Although these systems are very powerful, their generality may impose unnecessary overheads for VLSI technology extrapolation.

2.2 GTX Goals in Perspective

With respect to the previous systems for technology extrapolation, we make the following observations.

1. Different systems may predict the same "parameter" (e.g., microprocessor clock frequency), yet be incomparable due to differing sets of inputs and assumptions, as well as lack of documentation and visibility into internal calculations.

2. Each system typically offers exactly one "inference chain" for any given output of interest (e.g., cycle time). Furthermore, this inference chain can involve a large spectrum of modeling choices. The *quality* of such modeling choices cannot be assessed since the system is "hard-coded", and no exploration of modeling sensitivity or robustness is possible.

3. The hard-coded nature of previous systems also means that they are inflexible: the user cannot define studies of other system parameters, and interaction with the system is limited.

4. Finally, development of previous systems has entailed near-total duplication of effort – since each system attempts to bound the same envelope of achievable design – in gathering, interpreting, and systematizing data and models. Redundant efforts are made even though no single entity – EDA vendor, system house, or academic group – can achieve "best-possible modeling" of all aspects of technology and design.

These observations motivate three key goals as we seek a new level of technology extrapolation capability.

Flexibility. To experimentally determine model sensitivity and robustness, users must have the ability to (i) (interactively) edit available inference chains and collections of "rules," (ii) define new parameters and rules, and (iii) request specific types of studies, such as parameter optimization or trade studies. GTX inherits the flexibility of AI constraint-programming and design support systems, while retaining VLSI domain-specificity and avoiding unreasonable implementation complexity. Support for interaction (GUI, session management, etc.) is an implicit requirement.

Quality. GTX seeks adoptability in the sense of having an easy learning curve and providing much "value" in the form of high-quality embedded data, embedded models, and user interface. We aim for a system that can be continuously improved to have "best-possible models" across the entire scope of technology extrapolation. Since no single group can achieve this alone, we require an open-source mechanism that is conducive to distributed ownership and maintenance.

Prevention of redundant effort. To avoid redundant effort, GTX is meant as a "permanent repository of first choice" for rules and data (calibration points) related to technology extrapolation. Beyond the open distribution mechanism noted above, *adoptability* (by academics open to collaboration, or by companies with proprietary data and firewalls) and *maintainability* become key concerns. A lower bound for adoptability is a platform-independent implementation that subsumes the functionality of all previous "hard-coded" systems. This recognizes the proprietary nature of user data and offers usability behind firewalls, with frequent releases to update the state of model/data collection. GTX also applies to any domain of semiconductors, VLSI or VLSI CAD, and is extensible to models of arbitrary complexity.

3 The Structure of GTX

GTX establishes a clear separation between *knowledge* and *implementation* (Figure 1). Knowledge is represented independently from its implementation in a serializable public-domain format. It contains data (*parameters*), the models (*rules*) that can operate on them and studies (*rule chains*), a collection of rules to obtain a par-

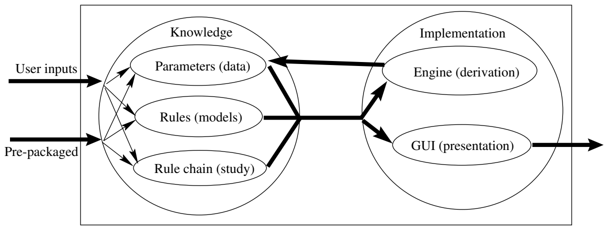

Figure 1: Schematic view of the GTX framework.

ticular result. The implementation then consists only of a *derivation engine* and a graphical user interface (*GUI*).[1] The engine can load *modules* of parameters, rules and a rule chain and automatically operate on them. The result of the operation is new data. *Known studies* are supplied in pre-packaged rule chains; additional modules can be written and shared by users.

3.1 Parameters, Rules and Rule Chains

As previously mentioned, the values of interest are encapsulated in *parameters*, and potential inferences between them in *rules*. Each rule accepts as inputs a fixed collection of parameters, and its evaluation computes a single output parameter. The collection of available rules and parameters is naturally viewed as a *bipartite digraph* in which an edge extends from a rule to a parameter if the parameter is the output of the rule, or from a parameter to a rule if the parameter is an input to the rule.

Two or more rules may compute the same output (i.e., alternative models of the same value), and the above digraph may contain cycles. However, any particular calculations must avoid such irregularities to prevent value conflicts and infinite loops. This is supported through the notion of a *rule chain* – an acyclic subgraph of the graph of available rules and parameters such that no two rules compute the same output.[2]

3.1.1 Parameters

Parameters are the common base on which rules of different types operate. The main attributes of a parameter are its name, data type and its units. In order to obtain the goal of high reuse-ability of rules and parameters, the parameter names have to be carefully chosen so that they are easy to understand. Also, we must ensure that no physical attribute receives two different names in GTX and that no GTX parameter name is used for two different physical attributes. Therefore, we have devised strict rules for the parameter names [4]. The grammar for parameters is specified at our website [10]. Following is a very simple example representing the chip edge length.

```
#parameter dl_chip
#type double
#units {m}
#default
  1e-2
#description
  chip edge length
#endparameter
```

3.1.2 Rules

GTX supports the following types of rules.

ASCII rules provide a closed–form expression language that allows calculating the output from the input using common mathematical functions or operations, interpolation or table lookup, and if–then–else. There is no program flow and therefore no iteration per se, but *vector* operations are provided that allow common computations such as sums.

External executable rules cause the engine to invoke a specified executable file (e.g., a PERL script), passing the input values on the command line or through a file. The external executable saves its output into a temporary file to be read by the engine. External executable rules allow the inclusion of executables for which source code is not available or for computations that cannot be expressed in ASCII rules.

Code rules are another option for rules too complex for ASCII rules. However, they are hard–coded into the engine itself and require recompilation of the engine code. Therefore, they are appropriate only when execution speed is an issue.

These types provide a reasonable expressive power and facilitate easy updates to GTX with new models. The following is an example of an ASCII rule computing the chip edge length from the chip area. The `#output` and `#inputs` sections declare the types and units of output and input parameters. The formula in the `#body` section specifies the evaluation of the rule.

```
#rule BACPAC_dl_chip
#description
  rule from BACPAC for the chip edge length
#output
  double {m} dl_chip;    // chip edge length
#inputs
  double {m^2} dA_chip;  // chip area
#body
  sqrt(dA_chip)
#reference
  BACPAC
#endrule
```

3.1.3 Rule chains

The GTX user indicates to the engine which of the currently available rules should be evaluated, by providing a simple list of those rules. The order in which rules are executed forms the *rule chain*, and is decided by the engine based on the relations between the rule inputs and outputs. If we would have a rule "BACPAC_dA_chip" that computes the chip area, e.g., as a function of number and size of the gates, then the chip edge length could be computed by executing the following rule chain

```
BACPAC_dA_chip
BACPAC_dlchip
```

3.2 Engine Structure and Operation

For each parameter, the engine maintains zero, one or more values. Values can be set by default, loaded from files, entered by the user or computed. Multiple values can be computed by *sweeping*, i.e., evaluating rules over multiple combinations of input parameters. When instructed to evaluate a rule chain, the engine clears values that can be computed by rules of the chain. For each combination of values of primary inputs of the chain, the engine evaluates rules in topological order and adds their output values to respective collections of values, unless some constraints fail. A faster algorithm is possible to produce all derivable *sets of values*, but with our simple algorithm the inputs of any particular value can be recovered (e.g., for minimization along a rule chain).

3.3 Graphical User Interface

The GUI is implemented with the cross–platform toolkit wxWindows; we have run it successfully on Windows 95/98, Windows NT, Solaris and Linux. At any given time, the user may view (i) current parameters, or (ii) current rules, or (iii) current rule chain, or (iv) values of parameters in the current chain. When a particular parameter or rule is selected, its details are shown and can be edited. The chain view shows all rules in the chain and helps the user to add new rules to the chain. The values view shows both inputs to and outputs of the current chain. The inputs may be edited. This view permits invoking the chain and observing the output, sweeping over

[1] Currently, engine and GUI form a single executable. However, our implementations can also be used in an "engine server," supporting multiple GUI clients connecting to the server over a network.

[2] Except for a *constraint*, a special kind of rule for calculations with constraints on the input parameter values. See [4].

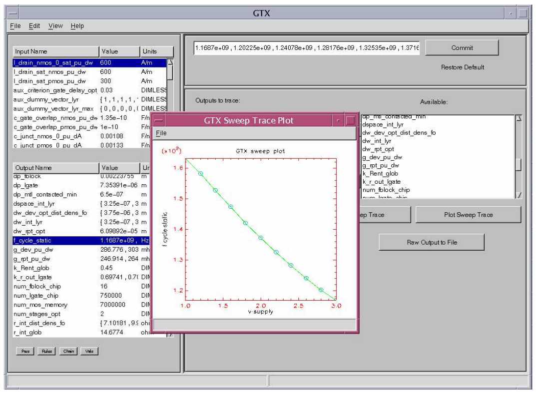

Figure 2: Screenshot of GTX GUI.

multiple input values, observing the trace of such a sweep (including optimization) and plotting (see Figure 2). In addition to the four views, the GUI handles extensive file I/O and interactive addition of new parameters and rules.

4 Results of GTX Studies

The flexibility of GTX makes it particularly useful as a development tool for adding new rules that model a very particular part of the design behavior, as an emulation tool for existing estimator tools, as a comparison tool between different estimation methods and as an evaluation tool for those methods. In this section, we highlight some of these abilities of GTX.

4.1 Sensitivity Analyses of Cycle-Time Models

Detailed evaluation and comparison of prediction models starts with model implementation in GTX. Motivated by their prominence in roadmapping, our first experiments focus on *cycle-time models*: we have implemented the models from SUSPENS [2] (with extensions of Takahashi et al. [20]), BACPAC [18], and Fisher et al. [7] within GTX,[3] and reproduced published results with each model. Our implementations are tuned to ensure maximal interchangeability of the GTX rules for each model, allowing extensive evaluation of various model sensitivities.[4]

Our experiments address two basic types of sensitivity: *parameter sensitivity* and *model (or rule) sensitivity*. The former describes the influence of changes in the primary input parameters to the model, while the latter describes the influence of changes in the estimation model itself. (We do not aim to make value judgments about or compare the models; rather, our goal is to show the value of being able to try variant estimation methods.) We perform the following experiments:

1. For the same primary inputs, compare the results for different models (model sensitivity).

2. For each model, vary the input parameters by +/- 10% and note the difference in the resulting clock frequency (parameter sensitivity).

3. For each rule out of one rule chain (model), replace one rule by a rule from another model that computes the same parameter and record the difference in clock frequency (model sensitivity).

[3] We have also used executable rules to link the IPEM executable [5] to GTX.

[4] Although "we have implemented the models" sounds straightforward, it is tremendously difficult to truly reimplement other researchers' models. This is a difficulty that the GTX framework seeks to remove once and for all. By enabling the building of new and variant rules on top of existing ones, GTX permits "reuse without understanding", and thus lowers the barrier to entry for those wishing to pursue technology extrapolations.

model	t_l (ps)	t_g (ps)	f_c (MHz)
BACPAC	893	115	745
Fisher	1162	204	659
SUSPENS	665	–	1505

Table 1: Logic stage delay t_l, global delay t_g and overall clock frequency f_c for interconnect models.

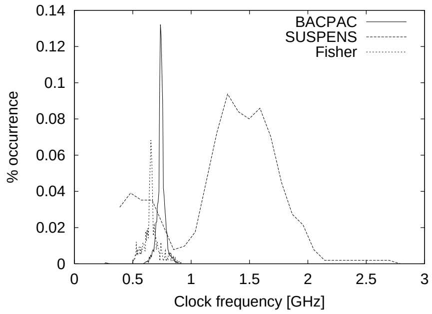

Figure 3: Parameter sensitivity: BACPAC, Fisher and SUSPENS.

For all experiments and models, we use a common primary input (PI) parameter base derived for 0.25 μm technology and mainly following the default parameter values of BACPAC (additional PIs for other models are tuned to these parameter values). Despite the common parameter base, our initial model sensitivity assessment of the SUSPENS, BACPAC, and Fisher models shows very different values for respective predictions of logic stage delay (t_l), global delay (t_g), and overall clock frequency (f_c) (see Table 1).[5] A more detailed type of model sensitivity analysis tests the sensitivity of a given model to "hybridization" with other models. In other words, we take the rule chain for a single model and replace exactly one rule by an equivalent rule (or set of rules) from another model. Details of such experiments are in [4].

Parameter sensitivity studies that vary *single* PI parameter values in each model's evaluation, changing each PI value by +/- 10%, are also detailed in [4]. More extensive studies *simultaneously* change more than one parameter value, again by +/- 10%. Since this produces three values for each parameter and since there are between 15 (SUSPENS) and 46 (BACPAC) primary inputs, it is not possible to sweep over all possibilities, and we therefore sweep over smaller parameter subsets (up to 7 parameters at the same time). Figure 3 plots the relative occurrence of clock frequency values in small intervals that result from the sweeping. If we say that a more "robust" (to changes of its input parameters) model is one with a narrower and higher peak, then BACPAC would seem to be the most robust, and SUSPENS the least robust.[6]

4.2 New Device Models

Apart from reimplementing existing cycle-time models in GTX, we have also developed new device models, both for bulk Si and Silicon-on-Insulator (SOI) devices. The SOI module assumes

[5] SUSPENS does not have a model for global delay on chip. We believe this was compensated by taking into account more stages but we chose to use a number of stages equal to that of the other models to maintain interchangeability. While the very high 1.5 GHz frequency predicted by SUSPENS is largely due to the lack of a global interconnect model, the logic stage delay is still significantly different from the other models.

[6] In general, we find BACPAC to be much less sensitive to either hybridization with other models, or variation of input parameter values. This does not necessarily imply that BACPAC is a better model (e.g., if a model predicts a clock frequency of 700 MHz independent of any input parameter value, then this is "robust" but not practical or correct). Note also that sweeping over more than 7 parameters at once will widen the peaks shown in the plot.

	Bulk Si		SOI	
	P (W)	%	P (W)	%
Logic + local wires	26.20	46.18	28.99	43.91
Global interconnects	2.20	3.88	2.60	3.93
I/O drivers + pads	11.71	20.65	13.35	20.22
Clock distribution	7.93	13.98	9.65	14.62
Memory	0.94	1.66	0.86	1.31
Short Circuit	7.68	13.54	10.21	15.47
Leakage	0.067	0.12	0.359	0.543
Total power	56.74	100.00	66.03	100.00

Table 2: Different components of power consumption for Bulk Si and SOI microprocessors.

partially-depleted SOI (PD-SOI) technology and is based on popular BSIM3SOI models [3]. The use of these modules will allow GTX users to more completely explore the future design space. Both modules have been compared to BSIM3 HSPICE runs, with results matching within 10%. Again, more details are in [4].

Comparison between Bulk Si and SOI

One of the primary design considerations for PD-SOI is the *floating body effect*. In short, since the transistor body is isolated from the substrate it must be modeled as an additional floating node. Depending on the switching history of the device and its capacitances, the body voltage can fluctuate, which leads directly to changes in the threshold voltage and subsequent I_{dsat} (saturation drain current) variation.

The SOI models in GTX calculate a range of possible I_{dsat} values, depending on the expected switching activity of the system. The steady-state body voltage is calculated based on reverse-biased diode leakage and substrate current due to impact ionization. This body voltage is then used to calculate a new V_{th} and I_{dsat} which are used in other GTX modules to calculate key parameters such as cycle time, leakage power, etc. Currently, the SOI module ignores the impact of capacitive coupling on body voltage.

A second source of variability that we investigate is *dynamic delay*. This phenomenon occurs due to the presence of large coupling capacitances between same-layer interconnects. Switching on adjacent wires can lead to variation in expected stage delay.

Our first study with these new models assesses the influence of device technology on clock frequency and power. In the best case[7] the clock frequency increases from 1.03 GHz for bulk Si to 1.31 GHz for SOI, and in the worst case from 867 MHz (bulk Si) to 1.05 GHz (SOI). The power results presented in Table 2 show a 16% rise in power for an SOI system. However, it should be noted that the SOI-based design exhibits a 24% higher clock frequency than its bulk Si counterpart (for the nominal case). We would expect a 24% rise in dynamic power but due to smaller SOI device capacitances, this increase is reduced by a third. The SOI leakage power is substantially larger than bulk since a positive body voltage acts to reduce V_{th} and exponentially increase off-current.

The sensitivity of both device models to their input parameters is shown in Figure 4. In this figure, several technology related parameters were varied (including T_{ox},V_{dd}, and L_{eff}) by +/- 10%, and best and worst-case scenarios were examined. A few points should be clarified. First, SOI devices seem to have slightly less sensitivity to input parameter changes in this case; this could be due to the lower V_{th} for SOI which makes I_{dsat} less dependent on V_{dd}. Second, the process spread (between best and worst-case) is larger for SOI due to the floating body effect. This increased uncertainty

[7] Best case for SOI refers to the largest I_{dsat} realizable due to the floating body effect combined with zero effective coupling capacitance due to dynamic delay. Best case for bulk only considers dynamic delay effects.

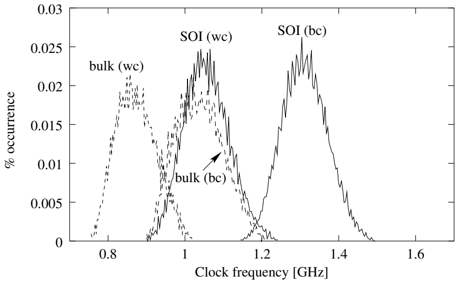

Figure 4: Parameter sensitivity for bulk Si and SOI device models.

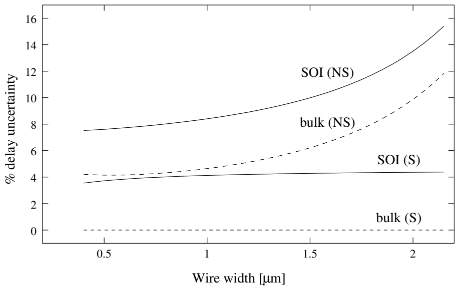

Figure 5: Delay uncertainty for staggered (S) and non-staggered (NS) repeater topologies using bulk and SOI. Results are for a 1.5cm global wire with four 100X repeaters.

eats into the advantage that SOI offers. This sort of analysis will help designers better quantify the price-performance tradeoffs of SOI technology. Again, the variation in the case of bulk silicon is due to dynamic delay only. Other interesting studies using the SOI module in GTX could explore the delay implications of constraining power (either leakage or dynamic) to be identical to the bulk Si case.

4.3 Delay Uncertainty Studies

Our last illustration of GTX capabilities is a delay uncertainty study that investigates the extent of the variations described above. The question we pose is: for a global interconnect, what type of repeater and wire topology is needed to limit/reduce the delay uncertainty due to floating body and dynamic delay effects?

In the study, we maintain a fixed pitch of 2.56 μm and a line length of 1.5 cm. The wire thickness is set at 1.9 μm and the repeaters have a W/L (NMOS) of 100. Delay uncertainty is calculated as $(T_{wc} - T_{bc})/T_n$ where T_n is the delay without dynamic delay or floating body effects, T_{wc} is the worst case delay, and T_{bc} the best case delay. Our study is set up so that the interconnect repeater interval and line width can be optimized subject to constraints on delay uncertainty and on peak coupling noise. (For example, a reasonable bound for delay uncertainty in global interconnect design is 10% (too much variation will make static timing results unreliable), and a peak noise limit of 15% of V_{dd} is also typically desired.)

In Figure 5 we plot the delay uncertainty against wire width for four cases: SOI and bulk Si with and without the use of staggered repeaters. Staggered repeaters are introduced in [9] and exploit offset repeater placement on adjacent global lines to eliminate the impact of dynamic delay. In the bulk case there is no uncertainty (at least due to the two effects we consider) and in the SOI case all delay uncertainty is due to the floating body effect. As wire width increases, the non-staggered repeater configuration exhibits more delay uncertainty since the coupling capacitance becomes a greater portion of the total capacitance. The floating body effect in SOI is also seen in Figure 5: the calculated value of about 4% is entirely due to variation in buffer drive current, and is in line with numbers reported by IBM for their SOI technologies [16].

As would be expected, our study also shows that the use of more repeaters limits dynamic delay uncertainty since the ratio of wiring capacitance to device capacitance decreases [4]. In general, with a small number of repeaters and a fixed pitch, reducing the wire width decreases the delay uncertainty and also leads to reduced power dissipation with minimal overall delay penalties. The benefits of staggered repeaters in meeting noise peak constraints are also easily assessed, e.g., with repeater W/L = 100 and a 20% V_{dd} noise margin constraint, the upper bound on repeater spacing due to the noise constraint is 2600 μm in the non-staggered case, and 12000 μm in the staggered case. Finally, since buffer insertion has large power implications, wire sizing combined with staggered repeaters seems to be the best strategy to reduce uncertainty. Overall, studies like this show the utility of GTX in searching over global interconnect solutions given various degrees of freedom (driver sizing, line width, etc.) and constraints (as in, e.g., [11]).

5 Conclusions and Ongoing Work

We have described the architecture and implementation of GTX, the MARCO GSRC Technology Extrapolation system. GTX has the potential to change how we extrapolate the impact of new process and design technology: it can provide a "living roadmap" that incorporates – and serves as a repository for – essentially unlimited forms of domain knowledge.

Initial studies with GTX have assessed the *modeling sensitivity* and *parameter sensitivity* of several influential cycle-time models, notably those of BACPAC [18] and Fisher [7]. These analyses reveal surprising levels of uncertainty and sensitivity to modeling choices in the technology extrapolations that drive roadmapping and R&D investment. Our reimplementation of these previous models reveals the value of not only GTX's flexibility, but also its standard mechanisms for interchange of data and models. We have also developed new models for bulk Si and SOI devices for the GTX context. Easily-implemented studies clearly contrast bulk versus SOI in terms of system power, as well as delay uncertainty for critical paths (i.e., isolating the impact of SOI variation in drive current). Other studies point out the reduction in delay uncertainty and peak noise that accrues from use of a staggered-repeater design technique for global buses. These experiences demonstrate the ease with which GTX enables new studies as well as new uses for existing models.

Part of our ongoing work seeks to improve the user-friendliness of GTX and especially its GUI: we hope to remove all barriers to development and use of new models in GTX. A key consideration is the maintainability of parameters and rules. To this end, planned extensions include (i) a convenient library mechanism for parameter names, with searching options based on GTX naming conventions; (ii) name spaces that allow unrelated details of one study (e.g., auxiliary or intermediate parameters) to remain hidden within other studies; (iii) a naming convention for rules, to facilitate exchange and reuse among different modules; (iv) a platform (grammar) to define studies in GTX so that users can easily see what a given rule chain calculates; and (v) a framework for sharing results of studies in GTX while protecting proprietary IP.

Other current efforts are aimed at applying GTX to elucidate specific issues in technology extrapolation; examples include (i) noise effects on clock jitter and skew, and (ii) routability effects (e.g., via blockage) on global interconnect distribution. Other enhancements to the engine include "smart sweeping" that can sweep large rule chains less exhaustively, while intelligently retaining output values of interest (e.g., max and min); Monte-Carlo styles of sweeping are also contemplated. A major enhancement will be the addition of *implicit computations*, a new rule type that would have the output variable in the #body expression and would solve for the expression==0. Improved global optimization and visualization of, e.g., response surfaces are also in the pipeline.

Finally, we are actively seeking collaborations with groups in industry and academia who are willing to contribute models of particular aspects of technology and design as well as feedback that will allow us to continually improve both the engine and the GUI.

Acknowledgments

We thank Professors Ken Rose, James Meindl, Scott Wills and Kurt Keutzer for fruitful discussions, and Dr. Phil Fisher for providing MS Excel spreadsheets from the ITRS98 effort.

References

[1] A. B. Babichev, O. B. Kadyrova, T. P. Kashevarova, A. S. Leshchenko and A. L. Semenov, "UniCalc, A Novel Approach to Solving Systems of Algebraic Equations," *Interval Computations* N2 (1993), pp. 29-47.

[2] H.B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Chapter 9, Addison-Wesley, 1990.

[3] BSIM3SOI, version 2.2, UC Berkeley.

[4] A. E. Caldwell, Y. Cao, A. B. Kahng, F. Koushanfar, H. Lu, I. L. Markov, M. Oliver, D. Stroobandt and D. Sylvester "GTX: The MARCO GSRC Technology Extrapolation System," Technical Report CSD-200006, UCLA CS Dept., March 2000.

[5] J. Cong and D.Z. Pan, "Interconnect Delay Estimation Models for Synthesis and Design Planning," *ASP-DAC*, 1999, pp. 97-100.

[6] J. C. Eble, V. K. De, D. S. Wills and J. D. Meindl, "A Generic System Simulator (GENESYS) for ASIC Technology and Architecture Beyond 2001," *Proc. ASIC Conf.*, 1996, pp. 193-196.

[7] P. D. Fisher and R. Nesbitt, "The Test of Time: Clock-Cycle Estimation and Test Challenges for Future Microprocessors," *IEEE Circuits and Devices Magazine* 14(2) (1998), pp. 37-44.

[8] Semiconductor Industry Association (SIA), "International Technology Roadmap for Semiconductors," December 1999. `http://www.itrs.net/`

[9] A. B. Kahng, S. Muddu, E. Sarto and R. Sharma, "Interconnect Tuning Strategies for High-Performance ICs," *Proc. DATE*, 1998.

[10] MARCO GSRC Technology Extrapolation Initiative, `http://vlsicad.cs.ucla.edu/GSRC/GTX/`

[11] K. Rahmat, O. S. Nakagawa, S.-Y. Oh, and J. Moll, "A Scaling Scheme for Interconnect in Deep-Submicron Processes," *Intl. Electron Devices Meeting. Technical Digest*, 1995, pp. 245-248.

[12] P. Raje, "A Framework for Insight into the Impact of Interconnect on 0.35-um VLSI Performance," *Hewlett-Packard J.*, 1995, pp. 1-8.

[13] S. Y. Reddy and K. W. Fertig, "Design Sheet: A System for Exploring Design Space," *Proc. Artificial Intelligence in Design*, 1996, pp. 347-366.

[14] Rensselaer Interconnect Performance Estimator (RIPE), `http://latte.cie.rpi.edu/ripe.html`

[15] G.A. Sai-Halasz, "Performance Trends in High-Performance Processors," *Proc. IEEE*, Jan. 1995, pp. 20-36.

[16] G.G. Shahidi et al., "Device and Circuit Design Issues in SOI Technology," *Proc. CICC*, 1998, pp. 339-346.

[17] D. Sylvester and K. Keutzer, "Getting to the Bottom of Deep Submicron," *Proc. ICCAD*, 1998, pp. 203-211.

[18] D. Sylvester and K. Keutzer, "System-Level Performance Modeling with BACPAC – Berkeley Advanced Chip Performance Calculator," *Proc. SLIP*, 1999, pp. 109-114, `http://www.eecs.berkeley.edu/~dennis/bacpac/`

[19] Universal Technical Systems, Inc., `http://www.uts.com`.

[20] S. Takahashi, M. Edahiro and Y. Hayashi, "A New LSI Performance Prediction Model for Interconnection Analysis of Future LSIs," *Proc. ASP-DAC*, 1998, pp. 51-56.