Précis: A Design-Time Precision Analysis Tool[1]

Mark L. Chang and Scott Hauck
Department of Electrical Engineering
University of Washington, Seattle, WA
{mchang,hauck}@ee.washington.edu

***Abstract*-Currently, few tools exist to aid the FPGA developer in translating an algorithm designed for a general-purpose-processor into one that is precision-optimized for FPGAs. This task requires extensive knowledge of both the algorithm and the target hardware. We present a design-time tool, Précis, which assists the developer in analyzing the precision requirements of algorithms specified in MATLAB. Through the combined use of simulation, user input, and program analysis, we demonstrate a methodology for precision analysis that can aid the developer in focusing their manual precision optimization efforts.**

I. INTRODUCTION

One of the most difficult tasks in implementing an algorithm in an FPGA substrate is dealing with precision issues. Typical general-purpose processor concepts such as *word size* and *data type* are no longer valid in the FPGA world, which is dominated by finer-grained computational structures, such as look-up tables. Instead, the designer must use and implement bit-precise data paths.

The difficulty is in the translation of a software algorithm into a hardware implementation that is precision-optimized for FPGAs. This task requires extensive knowledge of both the algorithm and the target hardware. Unfortunately, there are few tools that aid the would-be FPGA developer in this translation. In this paper, we discuss our work in filling that gap by introducing a developer-oriented tool for the design-time analysis of the impact of precision on algorithm implementation.

II. BACKGROUND

At the head of the development chain is the algorithm. Often, the algorithm under consideration has been implemented in some high-level language, such as MATLAB, C, or Java, targeted to run on a general purpose processor, such as a workstation or desktop personal computer. The most compelling reason to utilize a high level language running on a workstation is that it provides infinite flexibility and a comfortable, rich environment in which to rapidly prototype algorithms

This tool flow requires the developer to first convert a software prototyped algorithm into a hardware description. From this hardware description language (HDL) specification, various stages and intermediate tools are used to perform simulation and generate target bitstreams, which are then executed on reconfigurable logic.

A simple conversion without precision analysis would most likely yield an unreasonably large hardware implementation. On the other hand, if the algorithm actually requires more precision for some data sets than the data path provides, the results obtained from the algorithm could potentially be incorrect due to unchecked overflow or underflow conditions.

Therefore, within the HDL description, it is important that the developer determine more accurate bounds on the data path. Typically, this involves running a software implementation of the algorithm with representative data sets and performing manual fixed-point analysis. At the very least, this requires the re-engineering of the software implementation to record the ranges of variables throughout the algorithm. From these results, the developer could infer candidate bit-widths for their hardware implementation. Even so, these methods are tedious and often error-prone.

Unfortunately, while many of the other stages of hardware development have well-developed tools to help automate difficult tasks, few tools can automate HDL generation from a processor-oriented higher level language specification. And while there are C-to-Verilog[1] and C-to-VHDL[2] tools in existence, they do not offer such "designer aids" that would help with precision analysis of existing algorithms implemented in a high level language.

III. PRÉCIS

In order to fill this void in hardware development tools, we are developing *Précis*, a design-time precision analysis tool. Précis utilizes MATLAB as an input specification for algorithms and is designed to interact with the developer in order to assist them in making the best choices regarding data path precision. Currently, Précis aids the developer by providing a constraint propagation engine, simulation support, range finding capabilities, and performing precision slack analysis.

Précis is designed to complement the existing tool flow. It is not meant to be an HDL generator, a MATLAB-to-HDL

[1] This is an abstract of the full paper: M. L. Chang, S. Hauck, "Précis: A Design-Time Precision Analysis Tool", *IEEE Symposium on Field-Programmable Custom Computing Machines,* 2002.

converter, or an optimizing compiler of any sort. Instead, it is meant to provide a convenient way for the user to interact with the algorithm under consideration. Our goal is for the knowledgeable user, after interacting with our tool, to have a much clearer idea of the precision requirements of their data path. It is our belief that the developer of the algorithm, with suitable software assistance, can perform much better precision analysis and optimization than a fully automated tool could ever achieve.

In the following sections, we describe in brief detail the constituent parts of Précis. We ask that readers refer to [12]—a previously published paper—for a more detailed review of our work.

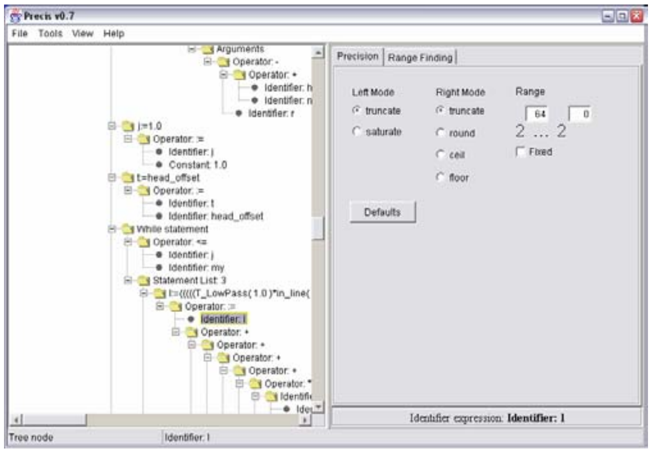
A. MATCH front-end

The front-end of Précis comes from Northwestern University in the form of a modified MATCH compiler[3,4]. The MATCH compiler understands a subset of the MATLAB language and can transform it into efficient implementations on FPGAs, DSPs, and embedded CPUs. It is used here primarily as a pre-processor to parse MATLAB source. The MATCH compiler was chosen as the basis for the MATLAB code parsing because no official grammar is publicly available for MATLAB. We are not constrained to using the MATCH compiler, though, as our tool may be updated to accommodate an alternate MATLAB-aware parser.

The MATCH compiler remains a work in progress and is currently being marketed by AccelChip[5].

B. Précis application

The main Précis application is written in Java, in part, due to its relative platform independence and ease of graphical user interface creation. Précis takes the parsed MATLAB code output generated from the MATCH compiler and displays a GUI that formats the code into a tree-like representation of statements and expressions. An example of the GUI in operation is shown in Figure 1.



```
Arguments
  Operator: -
    Operator: *
      Identifier: h
      Identifier: n
    Identifier: r
j=1.0
  Operator: =
    Identifier: j
    Constant: 1.0
t=head_offset
  Operator: =
    Identifier: t
    Identifier: head_offset
While statement
  Operator: <=
    Identifier: j
    Identifier: my
  Statement List: 3
    l=(((((T_LowPass(1.0)*in_line(
      Operator: =
        Identifier: l
        Operator: +
          Operator: +
            Operator: +
              Operator: +
                Operator: *
                  Identifie
                    Ide
```

Left Mode: (●) truncate ( ) saturate
Right Mode: (●) truncate ( ) round ( ) ceil ( ) floor
Range: 64 0 — 2 ... 2 — [ ] Fixed

Identifier expression: **Identifier: l**

Identifier: l

Figure 1. Screen capture of the Précis GUI.

The left half of the interface is the tree representation of the MATLAB code. The user may click on any node and, depending on the node type, receive more information in the right panel. The right panel displayed in the figure is an example of the entry dialog that allows the user to specify fixed-point precision parameters, such as range and type of truncation. With this graphical display the user can then perform the various tasks described in the following sections.

C. Propagation engine

A core component of the Précis tool is a constraint propagation engine. The propagation engine simulates the effects of using fixed-point numbers and fixed-point math in hardware. This is done by allowing the user to (optionally) constrain variables to a specific precision by specifying the bit positions of the most significant bit (MSB) and least significant bit (LSB). Variables that are not manually constrained begin with a default width of 64 bits. Typically, a user should be able to provide constraints easily for at least the circuit inputs and outputs.

The propagation engine traverses the expression tree and determines the resultant ranges of each operator expression from its child expressions. This is done by implementing a set of rules governing the change in resultant range that depend upon the input operand(s) range(s) and the type of operation being performed. For example, in the statement `a=b+c`, if `b` and `c` are both constrained by the user to a range of `2^15` to `2^0`, 16 bits, the resulting output range of `a` would have a range of `2^16` to `2^0`, 17 bits, as an addition conservatively requires one additional high order bit for the result in the case of a carry-out from the highest order bit. Similar rules apply for all supported operations in both the forward and backward directions. A more complete study of propagation and its effects upon hardware synthesis can be found in [6]. We plan to continue development of our own propagation tool to a similar extent.

The propagation engine can be used to get a provable upper bound of the growth rate of variables through the algorithm. This will allow the user to see a conservative estimate of how the input bit width affects the size of operations down stream.

D. Simulation support

As previously mentioned, a typical step in precision analysis is the actual running of the algorithm in a fixed-point environment. Précis can automatically generate annotated MATLAB code to aid in fixed-point simulation of the user's algorithm. The user simply selects variables to constrain and requests that MATLAB simulation code be generated. The code generated by the tool includes calls to MATLAB helper functions that we developed to simulate a fixed-point environment. The simulation flow is shown in Figure 2.

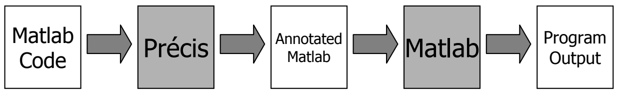


Figure 2. Code generation for simulation.

The purpose of these simulations is to determine the effects of constraining variables on the correctness of the implementation. Not only might the eventual output be erroneous, but the algorithm may also fail to operate entirely due to the effects of precision constraints.

Note that it is typically not sufficient to merely test whether the fixed precision results are identical to the unconstrained precision results, as this is too restrictive. In situations such as image processing, lossy compression, and speech processing, users may be willing to trade some result quality for a more efficient hardware implementation. Précis, by being a designer assistance tool, allows the designer to create their own "goodness" function, and make this tradeoff as they see fit. With the Précis environment, this iterative development cycle is shortened, as the fixed-point simulation code can be quickly generated.

E. Range finding

While the simulation support described above is very useful on its own for fixed-point simulation, it is only truly useful if the user can accurately identify the variables that they feel can be constrained. If the user does not really have an idea of where to begin, one place to start is utilizing the Précis range finding capability. The development cycle utilizing range finding is shown in Figure 3.

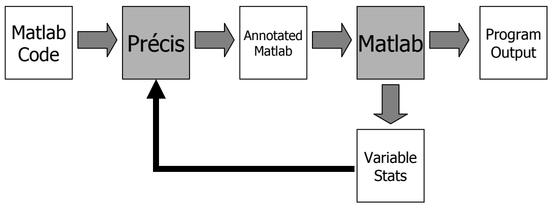


Figure 3. Development cycle for range finding analysis.

After the MATLAB code is parsed into the tool, the user can select variables they are interested in monitoring. Variables are targeted for range analysis and annotated MATLAB is generated, much like the simulation code is generated in the previous section. Instead of fixed-point simulation, though, Précis annotates the code with another MATLAB support routine that monitors the range of the values that the variables under question take on.

The user can then load the resultant range values discovered back into the Précis tool and (optionally) constrain the variables. The user now has an idea of what precision each variable requires for the sample data. Propagation can now be performed to determine the effect these precisions have on the rest of the system.

The results from this range finding method, however, are data set dependent. If the user is not careful to use representative data sets, the final hardware implementation could still generate erroneous results if the data sets were significantly different in precision requirements, even on the same algorithm. For this reason we will consider range-gathered precision information to be a lower bound. Given that the precisions obtained from propagation are a conservative upper bound, manipulating the difference between these two bounds leads us to another method of precision analysis—slack analysis.

IV. Slack analysis

One of the goals of this tool is to provide the user with "hints" as to where the developer's manual precision analysis and hardware tuning efforts should be focused. Ultimately, it would be extremely helpful for the developer to be given a list of "tuning points" in decreasing order of potential overall reduction of circuit size. This way, the developer could start a hardware implementation using more generic data path precision and iteratively optimize code sections that would give them the most benefit to meet constraints, such as time, cost, area, performance, or power. We believe this type of "tuning list" would give a developer a head start on precision analysis and put them on the right path of development faster than non-automated techniques.

As mentioned earlier, if the user performs range finding analysis and propagation analysis on the same set of variables, the tool would obtain what would amount to a lower bound from range analysis and an upper bound from propagation. We consider the range analysis a lower bound because it is the result of true data sets. While other data sets may require even lower amounts of precision, we know we need *at least* the ranges gathered from the range analysis. Further testing with other data sets may show that some variables would require more precision. Thus, if we implement the design with the precision found, we might encounter errors on output, thus the premise that this is a lower bound.

On the other hand, propagation analysis is very conservative. For example, in the statement `a=b+c`, where `b` and `c` have been constrained to be 16 bits wide by the user, the resultant bit width of `a` may be *up to* 17 bits due to the addition. But in reality, both `b` and `c` may be well within the limits of 16 bits and an addition might never overflow into the 17th bit position. For example, if $c=\lambda-b$, the range of values `a` could ever take on is governed by λ. To a person investigating section of code, this seems very obvious when `c` is substituted into `a=b+c`, but these types of more "macroscopic" constraints in algorithms can be difficult or impossible to find automatically. It is because of this that we can consider propagated range information to be an upper bound.

Given a lower and upper bound on the bit width of a variable, we can consider the difference between these two bounds to be the slack. The actual precision requirement is most likely to lie between these two bounds. Manipulating the precision of nodes with slack can net gains in precision system-wide, as changes in any single node may impact many other nodes within the circuit. The reduction in precision requirements and the resultant improvements in area, power, and performance can be considered gain. Through careful analysis of the slack at a node, we can calculate how much gain can be achieved by manipulating the precision between these two bounds. Additionally, by performing this analysis independently for each node with slack, we can generate an ordered list of "tuning points" that the user should consider.

For this paper, we consider the reduction of the area requirement of a circuit to be gain. In order to compute the gain of a node with respect to area, power and performance, we need to develop basic hardware models to capture the effect of precision changes upon these parameters. One simple implementation that we have utilized is to provide simple weighting parameters for different operator types. Thus, for example, if an adder has an area model of *x*, it indicates that as the precision decreases by one bit, the area reduces linearly and the gain increases linearly. In contrast, a multiplier has an area model of *x^2*, indicating that the area reduction and gain achieved are proportional to the square of the word size. Intuitively, this would give a higher overall gain value for bit reduction of a multiplier than of an adder. Using these parameters, our approach can more effectively choose the nodes with the most possible gain to suggest to the user. We detail our methodology in the next section.

A. *Performing slack analysis*

For each node that has slack, we set the precision to the range-find value, the lower bound. Then, we propagate the impact of that change over all nodes and calculate the overall gain for the change, system-wide. We record this value as the effective gain as a result of modifying that node. We then reset all nodes and repeat for the remaining nodes that have slack. We then order the resultant list of gain values in decreasing order and present this information to the user in a dialog window. The user then can see which nodes to change to get the highest gain and in what order. It is then up to the designer to consider these nodes and determine which, if any, should actually be more tightly constrained.

V. Benchmarks

In order to determine the effectiveness of our slack analysis methodology, we allowed the tool to perform slack analysis with propagated and range-found values. To gauge how effective the suggestions were, we constrained the variables the tool suggested in the order they were suggested to us, and calculated the resulting area. The area was determined utilizing the same area model discussed in previous sections.

A. *Wavelet Transform*

The first benchmark we present is the wavelet transform. Further details can be found in [12]. The results of the slack analysis are shown in Figure 4.

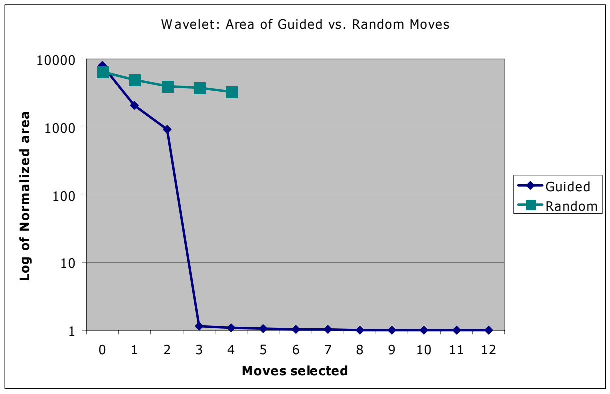


Figure 4. Wavelet area vs. number of suggestions implemented.

These results are normalized to the lower bound obtained by setting all variables to their lower bound constraints and computing the resulting area. This graph shows that between the upper bound and lower bound, there is a theoretical area difference of about four orders of magnitude. The slack analysis results suggested constraining the input image, then the low pass filter coefficients, and then the high pass filter coefficients. By taking the suggested moves in order and recomputing the order at each step, we were able to reach within 15 percent of the lower bound area of the system in three moves and within three percent of the lower bound in seven moves. At this point a typical user may choose to stop optimizing the system.

In order to determine the quality of our slack analysis, we chose five sequences of five random moves and plotted the average of the resultant area. As can be seen in the graph, our slack analysis achieves a much better result.

B. *Probabilistic Neural Network: PNN*

Another benchmark we investigated was a multi-spectral image-processing algorithm designed for NASA satellite imagery that is similar to clustering analysis or image compression. More details can be found in [7], and a full description of the benchmarking results can be found in [12].

Again, all results were normalized to the lower bound area. As shown in Figure 5, the tool behaved similarly to the wavelet benchmark in that it was able to reach within five percent of the lower bound within six moves, where after additional moves serve to make only minor improvements in area.

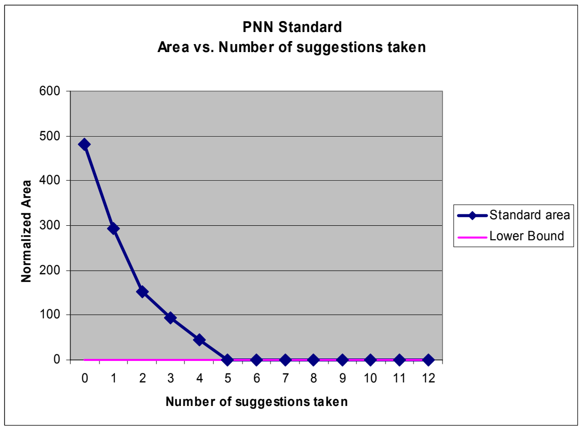


Figure 5. PNN area vs. number of suggestions implemented.

VI. Conclusions

In this paper we have demonstrated the need for precision analysis tools in the development cycle of software to hardware mapping. To direct the developer's efforts in hand-optimizing the precision of algorithms mapped to hardware, we have developed and demonstrated a tool, Précis, which allows the user to automate many tasks necessary for effective precision analysis. We have demonstrated how our tool can aid the developer in simulation of fixed-point math with automatic annotated MATLAB code generation. We have also developed MATLAB scripts that support range analysis of a user's MATLAB code in order to deduce a theoretical lower bound to the precision of selected variables. We have also presented a framework for propagation of precision range information over a MATLAB program. Finally, we have described our methodology of slack analysis and have shown how the suggestions provided by this methodology can be helpful in guiding the user in their manual precision optimization on real-world benchmarks.

VII. Acknowledgements

This research was supported by contracts with NASA and DARPA, and a grant from NSF. Scott Hauck was supported in part by an NSF CAREER award and a Sloan Research Fellowship.

VIII. References

[1] Synopsis CoCentric SystemC Compiler. http://www.synopsys.com/products/cocentric_systemC/cocentric_systemC_ds.html

[2] Celoxia Handel-C Compiler. http://www.celoxica.com/products/technical_papers/datasheets/DATHNC002_0.pdf

[3] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Chang, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden. "MATCH: A MATLAB Compiler for Configurable Computing Systems." Technical report CPDC-TR-9908-013, submitted to IEEE Computer Magazine, August 1999.

[4] P. Banerjee, A. Choudhary, S. Hauck, N. Shenoy. "The MATCH Project Homepage". http://www.ece.nwu.edu/cpdc/Match/Match.html (1 Sept. 1999).

[5] AccelChip, info@accelchip.com, http://www.accelchip.com.

[6] Mark Stephenson. "Bitwise: Optimizing Bitwidths Using Data-Range Propagation." Master's thesis. Massachusetts Institute of Technology. May 2000.

[7] Mark L. Chang. "Adaptive Computing in NASA Multi-Spectral Image Processing." Master's Thesis. Northwestern University, Evanston, IL. December 1999.

[8] Thomas W. Fry. "Hyperspectral Image Compression on Reconfigurable Platforms." Master's Thesis. University of Washington, Seattle, IL. May 2001.

[9] A. Nayak, M. Haldar, A. Choudhary, P. Banerjee, "Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs," Proc. Design Automation and Test in Europe (DATE 2001), Berlin, Germany. Mar. 2001.

[10] Kiran Bondalapati and Viktor K. Prasanna, "Dynamic precision management for loop computations on reconfigurable architectures," IEEE Symposium on Field-Programmable Custom Computing Machines, April 1999.

[11] George A. Constantinides, Peter Y.K. Cheung, Wayne Luk, "The multiple wordlength paradigm," IEEE Symposium on Field-Programmable Custom Computing Machines, April 2001.

[12] Mark L. Chang and Scott Hauck, "Précis: a design-time precision analysis tool," IEEE Symposium on Field-Programmable Custom Computing Machines, April 2002, in press.