

# EXPERIMENTS WITH THE HI-PASS DSP SYNTHESIS SYSTEM

Phillip Duncan, Shobana Swamy, Steve Sprouse, Daniel Potasz, Rajeev Jain

*UCLA, Los Angeles, California 90024*

William Cammack, Neal Gafter, Yiwan Wong, Wanda Gass

*Texas Instruments, Dallas, Texas 75265*

## ABSTRACT

Hi-PASS is a CAD system for synthesizing maximally parallel architectures to implement real-time DSP algorithms. The target DSP applications are the class for which desired sample rates are too high for time sharing of hardware to be feasible. Hi-PASS accepts a C code description of the design to be synthesized and produces a structural description of the final design that can be fabricated using standard cells provided by the Lager IV silicon assembly system [5]. Intermediate steps in the Hi-PASS flow consist of conversion of the C code into a flowgraph, optimization of the flowgraph using heuristic searches, retiming for optimal placement of pipelining registers, and mapping of the flowgraph to a structural description. Synthesis experiments carried out with Hi-PASS fall into two categories: 1) Application classes where the algorithm can be mapped onto standard functions such as FIR and IIR filters, for which dedicated functional compilers exist, and 2) Applications where the algorithm cannot be mapped to standard functional blocks, and which requires a genuine architecture synthesis approach with many levels of optimization. Results are presented for synthesis of FIR filters in the first category, for which Hi-PASS synthesizes architectures of comparable quality in terms of area and throughput to those generated by a dedicated functional compiler; and for an edge detection image processing algorithm in the second category, where Hi-PASS provides significant optimization gains. These experiments demonstrate the capabilities of Hi-PASS as a synthesis system for high performance DSP ASICs.

## 1. INTRODUCTION

Hi-PASS [1] is a CAD system for synthesis of maximally parallel DSP architectures. It is targeted at applications where the sample rates are close to the achievable clock rate in the given technology and dedicated parallel architectures are essential for real-time processing. The steps in synthesis with Hi-PASS are 1) translation of a procedural DSP algorithm description into a dataflow graph; 2) arithmetic and logic optimization of the flowgraph; 3) optimal bit-level register placement using retiming to achieve a desired clock rate; 4) mapping of the flowgraph into a standard cell description suitable for automatic layout generation. The program modules implementing these steps are integrated using OCT [2], a specialized CAD database system developed at UC Berkeley. The program flow in Hi-PASS is described in section 2.

Existing synthesis techniques [3-4] are mostly aimed at time-shared datapath architectures. The critical issues in these approaches are scheduling and hardware allocation, so that a common datapath is optimally shared between the various algorithmic operations within the constraints placed by the precedence relationships in the flowgraph. In contrast, in the applications targeted by the Hi-PASS synthesis system, the desired sample rates are so high that time sharing is not feasible. Since dedicated operators are required for all algorithmic operations, the more critical issues are 1) how to minimize the number of hardware operators required (to reduce area and minimize the critical path) and 2) how to optimally place the available registers (or add

new ones) so as to meet high performance clock period specifications. The results of these optimization steps for the benchmark examples are discussed in sections 3 and 5, and comparisons with a dedicated functional compiler for the FIR filter synthesis experiments are presented in section 4. The paper concludes with discussion and analysis of the synthesis results in section 6.

## 2. HI-PASS SYSTEM FLOW

Hi-PASS is a CAD toolset used to synthesize ASIC architectures from C code descriptions of DSP algorithms. Figure 1 shows the system flow in Hi-PASS, as seen by the user in the X Window System graphical user interface. The C code is parsed and translated into a simplified representation in the 'Compile' module, where loop unrolling, control flow removal by evaluation, and other steps of symbolic interpretation are carried out. The simplified C code is then translated into a single-assignment intermediate format in the 'Interpret' module. This intermediate code is converted to an OCT flowgraph. The user then runs the heuristic optimization module HOPS, which performs arithmetic and logic optimization using a set of production rules. The Retiming module is executed next, which optimizes register placement down to the bit level, in order to meet the user-specified clock constraint, if a feasible solution exists. At the end of these optimization steps the user will have an optimized maximally parallel architecture. If the architecture meets the user's specifications, the OCT flowgraph may then be mapped into a structural description by clicking on the Flow->SMV icon in the user interface. The resulting structural netlist consists of standard cells from the LagerIV silicon compilation system [5] configured into parametrized macrocells. The Stdcell place and route tools in LagerIV are then run to obtain a standard cell implementation of the synthesized architecture.

The Hi-PASS design flow provides simulation and verification capabilities at various stages of the synthesis process: 1) C code simulation of the input algorithmic description; 2) Bit-level functional simulation of the synthesized architecture at the structural level using the LagerIV simulator THOR; 3) Linear switch-level simulation of the netlist extracted from the standard cell layout using the LagerIV simulator IRSIM. These verification tools can be accessed through the 'Simulate' icon in the user interface. Design statistics on the optimization results are obtained by clicking on the 'Stats' icon. The 'View' icon provides access to various methods for displaying intermediate results, and the 'Edit' icon provides a quick method for editing the original C program if modifications are needed. In the following sections, the system flow and the functionality of the individual CAD modules will be discussed using benchmark examples.

## 3. FIR FILTER SYNTHESIS

An 11-tap FIR filter for sinc compensation in an A/D converter has been synthesized using Hi-PASS. The input to the system is a C description which uses an optimized canonical signed digit (CSD) representation for the tap coefficients. Design statistics for the results of Hi-PASS synthesis is shown in Table 1. Using the same C description, different sinc filters were synthesized with different clock period constraints. With no additional pipelining, and using the same clock

period as obtained by critical path analysis in the retiming module, the design 'sinc1' was synthesized. Designs 'sinc2', and 'sinc3' were obtained by specifying a smaller clock period, forcing the retiming module to pipeline at the bit-level, and provide 1 clock period of latency. Figure 2 gives an idea of the trade-off involved between clock speed and number of register bits for bit-level pipelining in Hi-PASS for this example. All designs are in 2.0 $\mu$ mwell CMOS using LagerIV standard cells, and clock speeds are as simulated from the layout using IRSIM in the linear mode. The most significant optimization performed by Hi-PASS on this test case is bit-level pipelining of the ripple carry adders in the CSD multipliers.

#### 4. COMPARISON WITH A DEDICATED FUNCTIONAL COMPILER

The synthesis results from the sinc FIR filter example were compared with results reported for the FIRGEN compiler which was also developed at UCLA. A sinc compensation filter described in [6] was chosen as the test case. One point to note is the differences in architecture style produced by Hi-PASS and FIRGEN. FIRGEN uses carry-save addition for each of the FIR taps followed by one pipeline register and a final vector-merge adder. This design style produces a filter fixed at one stage of added latency and whose speed is determined almost solely by the carry propagation time of the vector-merge adder. Hi-PASS uses carry-ripple adders throughout the design and uses bit-level pipelining to increase operating speed. The approach of Hi-PASS is much more flexible than that of FIRGEN and allows for extra performance to be gained at the expense of extra latency.

Table 1 presents the comparison results for the sinc filters generated by Hi-PASS and FIRGEN. The first three designs show alternatives created using Hi-PASS and specifying different desired clock speeds to the retiming tool. The fourth design is the FIRGEN compiled filter reported in [6]. The clock speeds listed in the table are simulated results using IRSIM. One important result is that on average the Hi-PASS designs are smaller than the FIRGEN design even though FIRGEN uses a tiled datapath layout and smaller, hand crafted leafcells. This result is due to the fact that FIRGEN designs contain a much larger number of register bits because of the use of carry-save arithmetic.

These comparison results show that a generalized synthesis approach can be competitive with functional compilers with the benefit of not requiring individual tools to be written for every possible DSP function. In addition a large degree of flexibility is gained by using the bit-level pipelined architecture style. A range of clock speeds may be obtained with a relatively smooth trade-off of performance versus area.

#### 5. GENERAL ALGORITHM SYNTHESIS

The previous section demonstrated Hi-PASS's capability in generating cost-effective implementations for structured DSP algorithms which are normally synthesized by function-specific silicon compilers. This section focuses on the synthesis of user-defined DSP algorithms which consist of non-homogeneous, varied combinations of arithmetic and logic operations. Since these algorithms cannot be mapped directly onto standard functional blocks, computer aided analysis and optimization are essential.

There currently exists no reliable method for predicting the maximum throughput achievable if a given user-defined algorithm is to be synthesized with a particular silicon technology. Hi-PASS can be used as a feasibility analysis tool to determine if a maximally parallel architecture implementation of a user-defined algorithm can meet the throughput requirements. This is accomplished by eliminating overhead due to resource sharing and through utilization of heuristic optimization. In addition, pipelining overhead is minimized through optimization of register placement and register sharing using retiming techniques. This is demonstrated in the example application described below.

The DSP application chosen as an example consists of an edge detection algorithm which analyzes two dimensional gray-scaled images (which can be of any width and height) in order to determine the location of moving automobiles on a relatively straight road. The algorithm processes continuous frames of data, and by storing the results of various intermediate calculations during the processing, can determine the distance (range) of the automobiles from a fixed point and the relative speed (range rate) at which the automobiles are approaching or moving away from that fixed point. Since the algorithm must compute the range and range rates of multiple automobiles in real-time, the speed at which the frames of data must be received and processed by the algorithm is too high for a strictly software implementation. Thus, a hardware implementation of the algorithm has been developed using the Hi-PASS system.

The initial OCT flowgraph is generated by symbolic interpretation and translation of the 82 lines of C code describing the DSP algorithm. Heuristic optimization is then run in HOPS using a set of production rules for flowgraph transformations. These production rules are specifically tailored for arithmetic and logic optimization. Additional technology dependent production rules are also employed in the optimization process. A sample of the production rules is shown in Figure 3. HOPS sets up a search tree of functionally equivalent flowgraphs, using the transformations specified in the production rules. A heuristic search strategy is employed to direct the generation, traversal, and pruning of the search tree using a cost function based on estimates of area and speed. The user specified run-time parameters in HOPS include cost thresholds to limit search depth.

The run-time statistics of HOPS in terms of the cost function at every internal search step is shown in Figure 4. The results in terms of total transistor counts per operator type before and after HOPS optimization are shown in Figure 5. The 15.3% reduction in total transistor count for the design represents a significant optimization gain.

The optimized flowgraph is fed to the retiming module. Automatic bit-level pipelining allowed the clock constraint of 88ns to be met, which was impossible with only word-level pipelining. The resulting flowgraph was then mapped to a structural description in flow->SMV, and a standard cell layout was generated using LagerIV. THOR simulation from the structural netlist verified that the functionality was the same as the C language algorithmic specification. IRSIM simulation in the linear switch mode from the layout extracted netlist verified that the clock constraint of 88ns is indeed met in the synthesized design. A design summary is shown in Table 2.

#### 6. CONCLUSION

Hi-PASS is an integrated toolset that provides automatic design flow from a C language input to an ASIC layout. It synthesizes maximally parallel DSP architectures for target applications where the sample rates are close to the achievable clock rate in the given technology. Experiments have been carried out in DSP synthesis using Hi-PASS for two sub-classes in this application area: 1) Those which can be mapped onto standard functional blocks, where Hi-PASS has been shown to produce designs equivalent in quality to those produced by dedicated functional compilers; 2) Those which contain a non-homogeneous set of arithmetic and logical operations and need a general synthesis approach, for which Hi-PASS has been shown to produce significant optimization gains in terms of hardware and critical path delay reduction. These design experiments demonstrate the capabilities of Hi-PASS as a synthesis system for high performance DSP ASICs.

#### 7. ACKNOWLEDGEMENTS

This work was supported in part by grants from the University of California MICRO program and Texas Instruments.

## 8. REFERENCES

- [1] P. Duncan et al., "Hi-PASS: A Computer-Aided Synthesis System for Maximally Parallel Digital Signal Processing ASICs.," to be presented at *International Conference on Acoustics, Speech and Signal Processing*, 1992.
- [2] P. Moore, "The General Structure of OCT," *Oct Tools Distribution 3.0*, Electronics Research Laboratory, UC Berkeley, 1989.
- [3] N. Park, A. Parker, "SEHWA: A Program for Synthesis of Pipelines," *23<sup>rd</sup> Design Automation Conference*, 1986.
- [4] J. Rabacy, M. Potkonjak, P. Hoang, and C. Chu, "HYPER - Design Synthesis for High Performance Real Time Systems," *Proc. IEEE International Symp. on Circuits and Systems*, May 1990.
- [5] C. Shung et al., "An Integrated CAD System for Algorithm-Specific IC Design," *IEEE Trans. on CAD*, May 1991.
- [6] R.Jain et al., "FIRGEN: A CAD system for Automatic Layout Generation of High-Performance FIR Filters," *IEEE 1990 Custom Integrated Circuits Conference*, 1990.

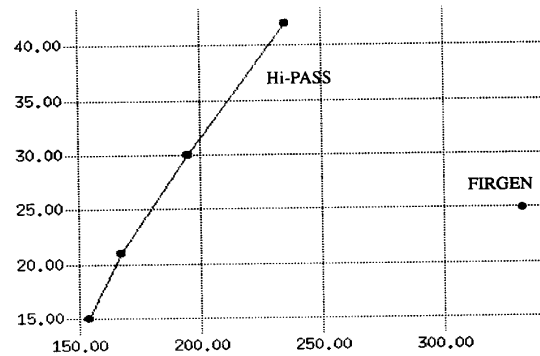


Figure 2: Clock Speed vs. Register Count for FIR example

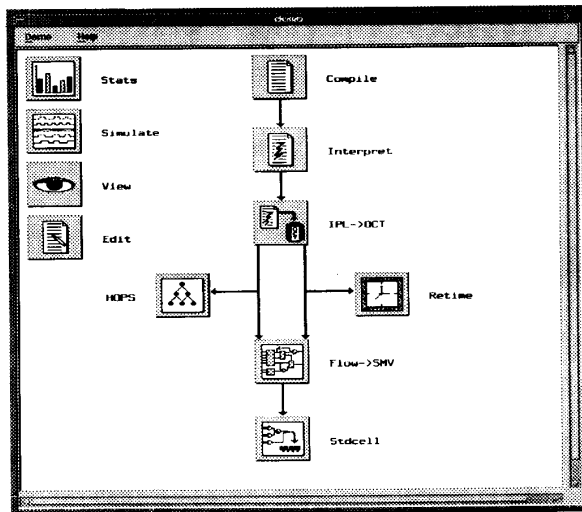


Figure 1: Hi-PASS User Interface

design name	Clock Speed (MHz)	Added Latency	Register Bits	Area(in mm <sup>2</sup> )
sinc1	15	0	154	8.3373
sinc2	30	1	195	8.7459
sinc3	42	1	235	9.7456
s_firgen	25	1	336	9.1524

Table 1: Statistics for FIR Filter Synthesis

```

; factor multiply over addition
(+ (* X ?Y) (* X ?Z)) ::= (* X (+ Y Z))
; convert self-addn. to left shift
(+ X X) ::= (<< 1 X)
; convert multiplication by power of two to a shift.
(* X 16) ::= (<< 4 X)
    
```

Figure 3: Sample Production Rules for Heuristic Optimization

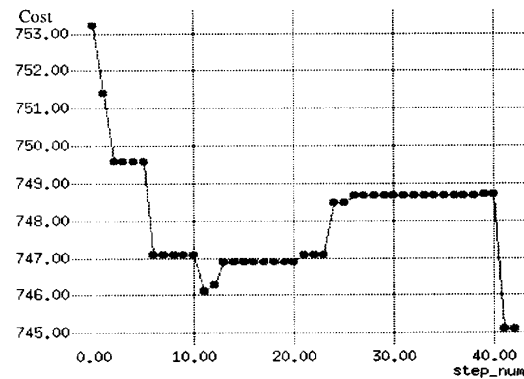


Figure 4: Cost function as graphed during Heuristic Optimization

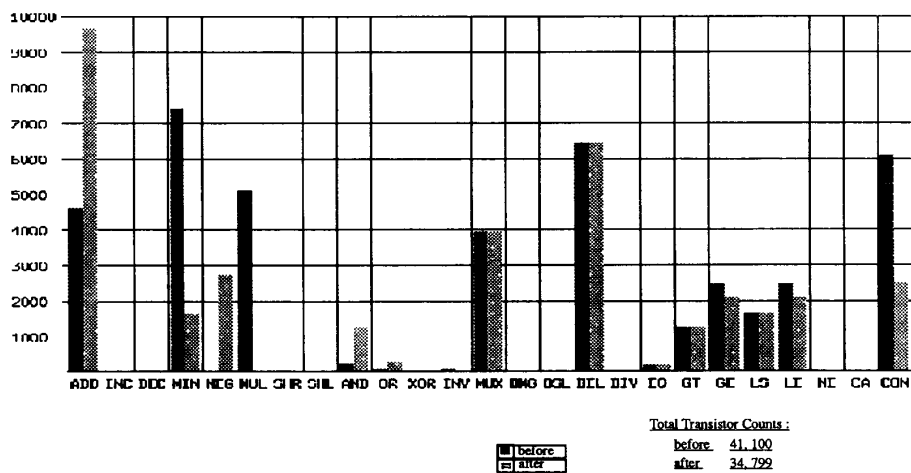


Figure 5: Transistor counts per operator type before and after optimization.

Table 2: Design statistics for general algorithm synthesis example.

Parameter	Value
Register Bits	498
Critical path	88ns
Transistor Count	35,875
Stdcell Instances	3,816
Area	30.31mm <sup>2</sup>