HI-PASS: A COMPUTER-AIDED SYNTHESIS SYSTEM FOR MAXIMALLY PARALLEL DIGITAL SIGNAL PROCESSING ASICS

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

UCLA, Los Angeles, California 90024 USA

Neal Gafter, William Cammack, Yiwan Wong, and Wanda Gass

Texas Instruments, Dallas, Texas 75265

ABSTRACT

Hi-PASS, a CAD system for DSP architecture synthesis, has been developed to automatically produce maximally parallel VLSI designs for real-time applications. The target DSP applications are the class for which desired sample rates are so high that time sharing of hardware is not feasible. Hi-PASS accepts a C code description of the design to be synthesized and produces a structural description of the final design, which can be fabricated using standard cells provided by the Lager IV silicon assembly system. Hi-PASS is a collection of modular tools, several of which contain new techniques within the field of synthesis. C code is converted into a flowgraph using a technique known as symbolic interpretation, including new methods for handling certain types of input-dependent control flow. The HOPS program is a heuristic search flowgraph optimizer tailored for the arithmetic intensive structures necessary for real-time DSP applications. The high throughput rates necessary for these applications also require that retiming be used to provide very short inter-register delay times. A new retiming tool has been developed which allows for computationally efficient bit-level retiming.

1. INTRODUCTION

Hi-PASS 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 four main CAD tools in synthesis with Hi-PASS are a) translation of a procedural DSP algorithm description into a dataflow graph; b) arithmetic and logic optimization of the flowgraph; c) optimal bit-level register placement using retiming to guarantee a desired clock rate; d) mapping of the flowgraph into a standard cell description suitable for automatic layout generation. These tools are integrated using OCT [1], a specialized CAD database system developed at UC Berkeley.

Existing synthesis techniques 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. By contrast, in the proposed 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 a) how to minimize the number of hardware operators required (to reduce area and minimize the critical path) and b) how to optimally place the available registers (or add new ones) so as to meet high performance clock period specifications. The optimization techniques being developed in Hi-PASS address these problems, as briefly described below.

2. HI-PASS SYSTEM FLOW

Hi-PASS consists of set of independent tools which are used to synthesize ASIC architectures from C code descriptions of DSP applications. Figure 1 shows the system flow of the Hi-PASS toolset. The C code is parsed and translated into an IPL (Intermediate Procedural Language) format. The IPL code is then converted into an OCT flowgraph using a program called ipl2oct. These first two steps will be discussed in section 3. The next step in the synthesis process is datapath optimization. Section 4 presents the HOPS optimization system. The optimized flowgraphs are then retimed to guarantee meeting performance constraints, if a feasible solution exists, while minimizing the number of pipelining registers necessary. A special technique for bit-level retiming has been developed for this project and is presented in section 5.

The Hi-PASS synthesis environment produces VLSI implementations by using the standard cell generation facilities of the LagerIV silicon assembly system [2]. Hi-PASS is linked to LagerIV using a program called flow2smv which produces an OCT structure master view that describes a standard cell design. Flow2smv is discussed in section 6.

Execution of the Hi-PASS tools is controlled from the X-window user interface shown in Fig 2. Each of the core tools discussed above is activated by the associated icon in the system flow diagram. Icons for utility functions such as file editing, database viewing, and statistics review are also available to the user. In the following sections a small example will be used to demonstrate the system flow of Hi-PASS. The C code for the example is shown as Fig. 3.

0-7803-0532-9/92 $3.00

3. C TO DATAFLOW GRAPH CONVERSION

The conversion from a C program to a dataflow graph occurs in two steps. First the C code is processed using a technique known as symbolic interpretation. The output of the symbolic interpretation program is single assignment, straight line code in a text form called IPL (Intermediate Procedural Language). All control flow constructs are removed during symbolic interpretation. The code in Fig. 4 represents the IPL generated by symbolic interpretation of the C program in Fig. 3. Since the control flow has been removed by evaluation during the generation of IPL, it is then a straightforward task to convert the IPL into a dataflow graph. Figure 5 shows the dataflow graph resulting from the example IPL code.

Symbolic interpretation is closely related to the compilation technique of partial evaluation. The basic technique compiles a program with some of its inputs or internal data elements known; evaluating the parts of the program that it can and leaving for run-time only the parts of the execution that are dependent on unspecified inputs. When this technique is applied to VLSI architecture synthesis, the code left for "run-time" is actually what is implemented as an ASIC design. Symbolic interpretation subsumes constant folding, copy propagation, procedure integration (inlining), loop unrolling, and other conventional compiler optimizations.

Input dependent control-flow has always presented a problem for most symbolic interpretation techniques. The tools developed for this project include new methods for handling certain types of input-dependent control flow.

4. HOPS DATAPATH OPTIMIZER

Since the high-throughput DSP applications being targeted by the Hi-PASS system require real-time processing at very high data rates and consist of arithmetic intensive operations, it is natural that a key problem associated with the architecture design issue is the optimization of the datapath. Previous work [3][4] focuses on optimization of time-shared datapaths where each physical functional unit processes multiple algorithmic operations. A primary measure of optimization quality for time-shared datapaths is the degree of resource utilization. In contrast, the Hi-PASS system focuses strictly on optimizing the area and critical path for fully-parallel architectures to achieve the highest possible throughput with no resource sharing overhead.

HOPS (Heuristic search OPtimization System) is a datapath optimization system developed to handle flowgraphs consisting of arbitrary combinations of arithmetic and logic operations. It produces functionally equivalent flowgraphs that better meet the high performance requirements of the targeted applications. One of the keys to obtaining high performance datapaths is making maximum use of all opportunities to optimize the arithmetic portions of the system. Much work has been published on methods to optimize single or multi-level gate logic, but in general these methods fail to produce adequate results when the system to be optimized includes a high percentage of arithmetic elements.

The basic concept of HOPS is to generate a search tree of functionally equivalent graphs using a modifiable set of arithmetic and logic transformations. A heuristic search strategy is then employed to direct the traversal of the search tree using a cost function based on estimates of area and speed. Several search strategies have been investigated including branch and bound, A*, and best first [5]. HOPS is the first system to propose search techniques developed within the artificial intelligence community to optimize fully parallel DSP architectures. New techniques have also been developed for pruning redundant sections of the search tree. Figure 6 represents a small search tree for the presented example. The solution found by search in this case is Node D, which reduces the hardware to two adds and a fixed shift versus two adds and two multiplies for the original graph at Node A.

After HOPS has been executed the designer can view changes which have occurred using a bar chart representation of the number of occurrences of each particular type of operator before and after optimization. Figure 7 shows the hardware utilization graph for a more complex application than the example design used in this paper. For each module type listed across the bottom the dark bar represents the number of occurrences before optimization and the light bar shows the results after optimization. Figure 7 shows that for the given application the number of ADDer modules increased but the number of MULtiply modules was reduced to zero.

5. BIT-LEVEL RETIMING

The high throughput rates required for the applications being targeted by the Hi-PASS system demand that the critical path of the architecture be optimized to yield very short inter-register delay times. It is a well known fact in the design of DSP architectures that this goal can be achieved by pipelining (inserting extra registers into the computation paths). This technique can reduce the critical path length and thus reduce the clock period. A more general form of pipelining is known as retiming [6]. Retiming allows existing registers to be moved within the circuit as well as adding extra registers, thereby providing more freedom to reduce the critical paths. In order to achieve the high performance goals of Hi-PASS an automatic retiming tool has been implemented.

In addition to implementing the methods presented by Leiserson [6], an important extension to the retiming model has been developed for the Hi-PASS system. In high performance fully-parallel DSP circuits it is often necessary to pipeline bit-parallel arithmetic operators at the bit level by inserting registers in operator ripple paths. To accomplish this using Leiserson's retiming model requires each n-bit operator be expanded into n distinct nodes. Since the number of nodes in the graph grows by a factor of n, the execution time of the retiming algorithms increases dramatically. The extension to Leiserson's model developed for Hi-PASS allows bit-level retiming to be performed without increasing the number of nodes by a factor of n. In some cases this retiming enhancement results in over an order of magnitude reduction in algorithm run time.

6. INTERFACE TO LAGER IV STANDARD CELL DESIGN

LagerIV is a chip design system which consists of a set of layout generators and a set of cell libraries [2], all driven by a structural description in the OCT database format. The Hi-PASS system interfaces to the LagerIV toolset as a means of automatically generating ASIC layouts. The flowgraph representation, which consists of functional operators and lacks nets for clock/supply lines, is translated to an OCT structure master view (SMV) consisting of specific leaf and macro cell instances from a standard cell library with explicit nets for all clock/supply lines and floorplanning hints for the LagerIV layout generators. The layout generated for the example flowgraph is shown in Fig. 8. It consists of full-adder, register, and hardwired constant leafcells and implements the optimized flowgraph of Fig 6, Node D.

7. CONCLUSION

Hi-PASS is an integrated toolset that provides automatic design flow from a C language input to an ASIC layout. The problems in designing fast dedicated DSP architectures have been investigated and CAD tools have been developed to aid in two optimizations: to reduce hardware and to reduce critical path delays. Significant contributions have been made in the areas of symbolic interpretation, datapath optimization, and retiming.

8. REFERENCES

[1] P. Moore, "The General Structure of OCT," *Oct Tools Distribution 3.0*, Electronics Research Laboratory, UC Berkeley, 1989.

[2] C. Shung et al., "An Integrated CAD System for Algorithm-Specific IC Design," *IEEE Trans. on CAD*, pp. 447-463, May, 1991.

[3] N. Park and A. Parker, "SEHWA: A Program for Synthesis of Pipelines," *23rd Design Automation Conference*, 1986.

[4] J. Rabaey, 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] R. Korf, "Search: A Survey of Recent Results," in *Exploring Artificial Intelligence*, pp. 197-237, 1988.

[6] C. E. Leiserson et al., "Optimizing Synchronous Circuitry," *3rd Caltech Conf. on VLSI*, pp. 87-116, 1983.

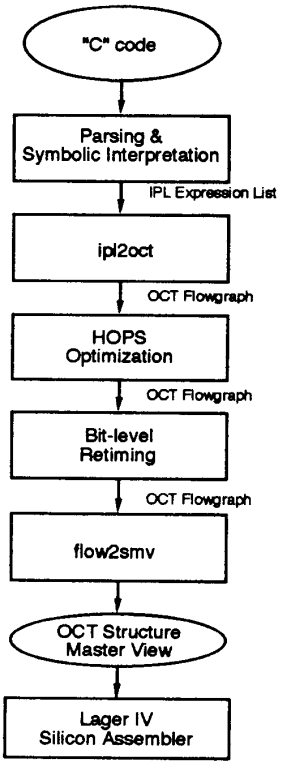

Figure 1: Hi-PASS System

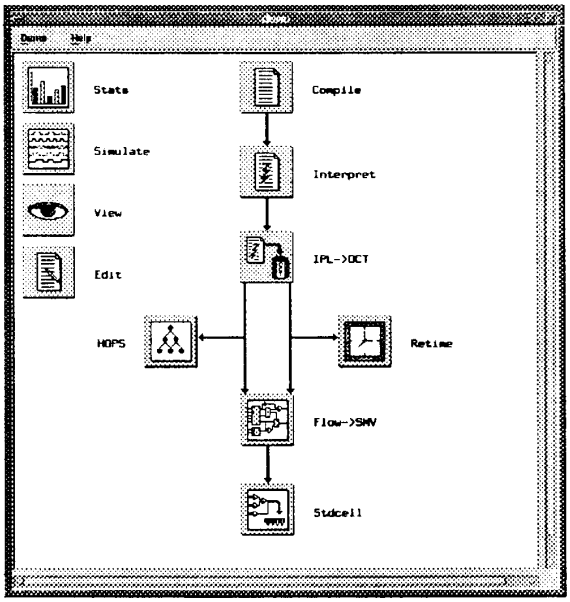

Figure 2: Hi-PASS User Interface

```
#define SIZE 3
process(in, z)
int in, *z;
{
  static int x[SIZE];
  int i;
  x[0] = in;
  *z = 2 * in;
  for(i=SIZE;i>=1;i--) {
    x[i] = x[i-1]; }
  for(i=1;i<=SIZE;i++) {
    if(i==1) {
      *z = *z + x[i]; }
    else {
      *z = *z + 2*x[i];}}}
```

Figure 3: C Program

```
process(in; z;)
INT<16> in, z;
{
  STATIC INT<16>  x_0,x_1,x_2;
  INT<16> z_1,z_2,_var1;
  X_0 = in;
  z_1 = 2 * in;
  x_2 = x_1;
  x_1 = x_0;
  z_2 = z_1 + x_1;
  _var1 = 2 * x_2;
  z = z_2 + _var1;
}
```

Figure 4: IPL Code

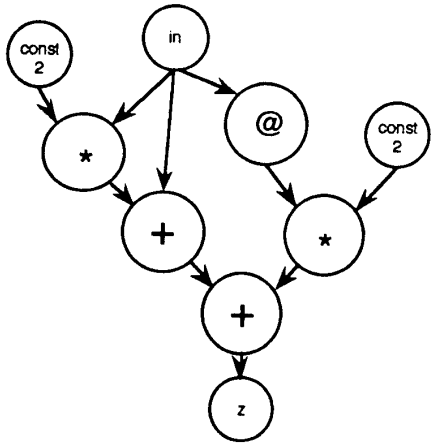

Figure 5: Unoptimized Graph

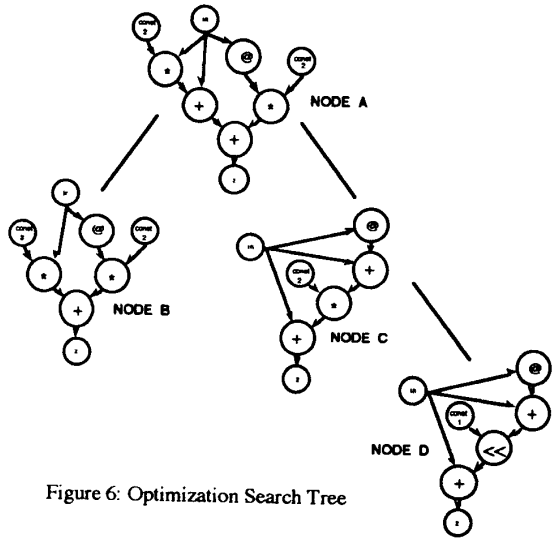

Figure 6: Optimization Search Tree

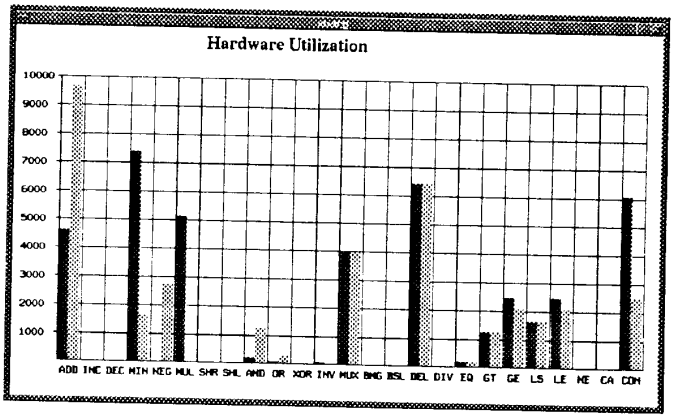

Figure 7: Optimization Results

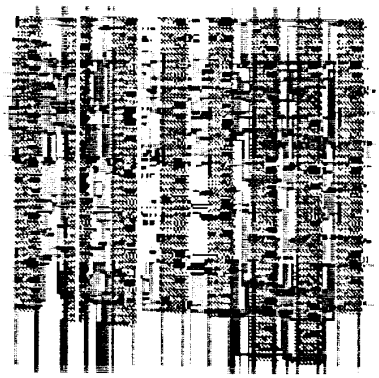

Figure 8: Standard Cell Layout