SASIMI '90

Synthesis And Simuation Meeting
and International Interchange

23rd - 25th October, 1990
Kyoto, Japan.
CAD System for an Application-Specific DSP Processor Design

L. G. Chen, L. G. Jeng, K. T. Chao, D. J. Lin, and C. T. Chao
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan 10764, R.O.C.

Abstract

A synthesis system called DSPDA is proposed for application-specific DSP processor design. This system takes a C program for describing the signal processing algorithms and performs behavior-to-architecture transformation, flow-graph optimization, and architectural synthesis including module selection, control steps scheduling and hardware allocation, then transfers the RTL structure to physical design. A dedicated bit-sliced data path cluster is generated by the system and the layouts can be further generated through the Genesis system.

1 Introduction

Nowadays, synthesis for IC design is getting to mature at the register-transfer (or functional) level. Many available algorithms on RTL synthesis are presented. Most of the proposed systems describe on synthesizing a general purposed circuit in terms of ALUs, registers, buses, and controllers. Some systems, for example, SPAID[1], Parasifal[2], and Cathedral[3], emphasize on the application specific architectural design. As the satisfactory development on this architectural level, the design on CAD systems are getting to a higher level. This progress would make more tightly coupled between CAD tools and system designers. While discussing with several industrial engineers and department researchers, some practical requirements of DSP designers are founded. The DSP designs always use C language to develop the algorithms and programs to simulate and evaluate the function and performance, even for real-time simulation on parallel processing systems. Could the CAD system directly extract or translate the behavioral description from the developed programs? The DSP chips have a restricted throughput rate and latency requirements to meet the sampling rate of the specific applications (ex. 8KHz in speech, 14.3MHz in video, ...etc). Could the synthesized systems efficiently match this demand? The designers usually have much experience and knowledge in their specific field...Could the CAD system accept their idea during synthesizing?

The goal of the research presented in this paper, as well as related research at National Taiwan University, is to explore these questions to theDSP designers. Toward this goal, we have developed an application-specific CAD system for DSP design, as shown in figure 1. A C-like language is developed as the behavioral description. The user's simulated program can be directly acceptable. Many useful blocks are supported and users can use a graphical flow graph representation to enter their design. The simulated result will point out whether the description needs to be modified or not. Two different design styles are then used to process the data. The style-specific tools are still under developing now, and will be ignored in this presentation. The transformation procedure provides general language compilation optimization and special parallel optimization. The former includes in-line, expansion, constant folding, deadline elimination, and other basic optimization. The latter emphasizes on tree-height minimization on both associative and nonassociative arithmetic operations. The useful kernel finding algorithm and binary tree inserting algorithms are developed for this optimization.

In order to select an optimal module set to meet the dedicated throughput rate with least cost from many well-defined modules in the cell library, a module set selection procedure is introduced. The area-performance product is used as the designed cost, where the area includes the data path area and that of

---

*This work is supported in part by National Science Council under Grant NSC79-0404-E002-34 and NSC78-0404-E002-47.*
Synthesis and Simulation Meeting and International Interchange

1. Fig. 1. The block diagram of the DSP/CAD System at NTU

2. Fig. 2. The C (C-diag) description of the video codec algorithm (a), and (b) the standard C simulation program. The translated CDFG (a) is shown in (b).

3. Fig. 3. An example for Arithmetic Minimization.

4. Fig. 4. Tree-High Minimization for associative operator (a) and nonassociative operator (b).

5. Fig. 5. Hardware Model Example.
the controller. The performance depends on the cycle time and number of control steps. Therefore, the system automatically decides an optimal clock cycle by judging the minimum available latency of the DSP application and the delay of the valid modules. The experimental results show that the module selection procedure can provide a good direction in higher level design. Basically, the scheduling and allocation procedures are developed by using heuristic list scheduling and clique-partition/graph-covering allocation algorithms. Finally, the physical design tools and others (ex. ATPG) are based on SCS (Silicon Compiler System, Co.) tools.

The full system provides friendly interactive fashion to allow user involve their decision during synthesizing. A pre-processor of the low bit-rate video codec system is designed as the tested example. This paper also describes a set of examples, including several benchmarks, to further illustrate their effects.

2 Input Behavioral Description

A computer software compiler is regarded as a black-box, if you write an algorithm with ordinary computer languages such as C and Pascal. In the same meaning, if the algorithm is fed into a silicon compiler and implemented on a dedicated ASIC design, the ASIC must be designed in the following steps:

1. Algorithm Transformation.
3. Datapath Design.

Design steps 2 and 3 have been intensively studied to reduce the design effort by many previous work[4]. So far, however, few tools have been developed to cope with step 1. It seems that step 1 implies the most information for hardware design such as data processing sequence and hardware facilities in the input algorithm since it is located at the most top level. Therefore, a good synthesizing system should include all these steps. As compared to other general purpose design system, the DSPDA system is dedicated to generated special purpose ASICs which executed fixed procedures, such as a signal processing algorithm.

The input behavioral description language (C-dsp) of the DSPDA system is a C-like language. The main advantages of C-like language is the popularity of the language which is friendly to DSP designers and the efficient compilers for simulation. The main disadvantage is its lacking of describing the relative signal delay timing. In order to describe the signal processing applications, C-dsp includes both those features of C that correspond closely to the capability of signal processing and some extended features for describing the related signal delay timing. For example, C-dsp includes two major extensions of C to describe the relative timing behavior of digital signal processing: the delay operator $z^{-1}$ and the system timing constraints. For behavioral simulation, we provide a translator that converts the C-dsp description into standard C program, which can be directly acceptable by other C compilers.

Some useful modules are built in the DSPDA system. For example, the canonical realization form of the FIR (Finite Impulse Response) and the IIR (Infinite Impulse Response) is built in this system. User can directly call these functional modules by providing the designed parameters. In some cases, the designer may wish to pass structural or timing constraints to the compiler, or the designer may want to calculate the bit-width by S/N ratio for data type of integer, fixed point and floating point[5]. The hardware structural and timing constraints can be defined by '#' in C-dsp, which will pass the information during synthesis. The C-dsp description language also allows the description of equations, delays, decisions, repetitions, hierarchy, structural modularity and bit-width specifications. Figure 2 shows an example for video codec system description which will depicted detailedy in section 6.

3 Flow Graph Transformation

The C-dsp description of a DSP algorithm is equivalent to a signal flow graph, in which nodes represent instances of functions and arcs represent the path followed the signals. Very often, the flow graph specified by the designers' programs does not meet the performance specifications or results in an inferior realization. The application of optimizing transformation is well known from optimizing software compilers[14]. In the flow graph transformation, the translator performs a variety of routine tasks and simple optimizations, including parsing, constant folding, in-line expansion, building the symbol table,
and eliminating dead codes. Because the flow graph reflects our target architecture, several optimizations are of note.

The first is called arithmetic minimization which is the reduction of the atomic (kernel) arithmetic operations. Since a single-cycle multiplication and addition are common in the DSP chips, the aim of arithmetic minimization is to reduce the number of multiplications and additions which are called the atomic operations in our flow graph. The problem is solved by the concept of the kernel finding algorithm in the multiplier level logic minimization[6]. This optimizing process is aiming at saving the hardware resource by reducing the atomic arithmetic operation. One example in figure 3 explains the procedure and result effect of the arithmetic minimization.

The second is the tree-height minimization. This concept is derived from the multiprocessor implementation of a parallel algorithm. In that case, the process of the minimal tree-height can save the computation time. Every arithmetic expression is parsed as a binary tree to obtained minimal tree-height is must. The optimizing process of tree-height minimization saves the computation time by reducing the level of computation trees but does not reduce the number of arithmetic operation nodes. Figure 4 explains the tree height minimization algorithm working on the associative operator and the nonassociative operator as the top of the tree.

The flow graph contains both the data flow and the control flow. The flow graph is represented as a block structure by the structural control nodes. Each branch or loop or if-then-else decision is forked as a block oriented flow graph which specified by its own block number. This block structure is good for user to partition the flow graph and provides the mutual exclusive information between blocks. The loops are unfolded as a block and all multiple jumps to jumps are eliminated in the optimization process.

4 Modules and Clock Cycle Selection

The design problem of architecture synthesis consists of three major tasks: module selection, control step scheduling and hardware allocation[7]. In this section, a module set and clock cycle time selection procedure is presented. The selection procedure bases on scanning the local optimal value of clock cycle time with associated candidate module set while minimizing the chip area-speed product. The cost of chip area consists of the weight of data path and controller.

The architecture of DSP system is constructed on a candidates module set which consists of functional units and other modules and the design criterion of pipelined or nonpipelined system latency. The set of modules which implement all of the atomic operation nodes in a data/control flow graph is called a candidates module set. Thus, initial selection of modules and clock cycle time to meet the design criterion is important to a final result and saving the repetition processing time. By the meet-in-the-middle design strategy, the modules (functional unit, ALU, register, RAM, PLA ...) can be realized at different specific operator implementation (e.g. carry-look-ahead vs. ripple-carry). The modules are stored in a library which the high level design system can instance whenever being needed.

The tasks of module set selection and clock cycle selection are done before actual synthesis. First, in the estimation phase, we try to determine the lower bound on the amount and types of required hardware resources (functional unit and controller). These lower bounds will be used to delimit the search space. By inspection of the flow graph, we can find exactly how many operations of each type should be executed during the available latency (called T, represented by the total number of available control steps). Denote this number by O_i (for operations of type i). It is easy to see that the number of functional units of type i (denoted by F_i), necessary to schedule the given flow graph in the given available time, must satisfy the following relation:

$$F_i \geq \lceil \frac{O_i}{T} \rceil \quad \text{(1)}$$

The equality relation between the right side and the left side exists when the atomic operation node is uniformly distributed among every control steps. Since the precedence relation could block the uniform distribution of operation nodes and the number of functional units must be an integer, the number of functional units in the right side is greater than or equal to the left side which is the expected value of the uniform distribution of operation nodes.

The size and complexity of a controller is estimated by assuming that each control step corresponds to a microword which will be implemented in a PLA controller. In each microword there are three control fields to implement each control step: 's' bits are required to specify the next control step (the next state),
'o' bits are required to specify the microoperation to be performed, and 'd' bits are required to control the data path. The total amount of microcodes (μC) are the number of control steps multiplied by the width of the microword:

$$\mu C = T \cdot (s + o - d)$$

where
$$s = \log_2 T$$
$$d = \text{number of FUs}$$
$$o = \text{number of micro operations}$$

For modules in a module set with different delays, the relative timing of modules, which may be single or multi-cycle; determining the clock cycle time (τc) and the corresponding total number of control steps (τt) is a trade-off. Figure 6 shows the effect of the τc of a candidate module set on area-performance product. The clock cycle time, τc, multiplied by the corresponding total number of control steps, τt, is the total execution time of the algorithm. The delay of a functional unit (denoted as dopi) is expressed in terms of the clock cycle time as:

$$dopi = Ci \cdot τc,$$

Where Ci is the number of clock cycles to complete the execution of an atomic operation. Then, every τc is the local optimal clock cycle time of a certain target module set. Obviously, a clock cycle time, τc, corresponds to a number of control steps complete the algorithm and the number of the control steps affects the number of functional units (equation 1). This implies that the clock cycle time affects the total number of control steps as well as the number of functional units. We scan all the local optimal clock cycle time to find the clock cycle time, Tc, which minimizes the total number of control steps and makes the cost of functional units and controller within constraints. A heuristic cost function bases on the area-speed product is derived:

$$F_{cost} = \text{cost(hardware)} \cdot \text{time(execution)}$$

where
$$\text{cost(hardware)} = \mu C \cdot \text{Area}(\mu C) + \sum F_i \cdot \text{Area}(FU_i)$$
$$\text{time(execution)} = Tc \cdot T$$

Since all candidate module sets are enumerated sequentially and the local optimal clock cycle time is also scanned in a linear way, the complexity of the module set and clock cycle selection can be expressed as O(m \cdot T). Where m represents the enumerated candidate module set and T denotes the local optimal clock cycle time. If the module set is fixed in a predefined module library, then the m is a constant. In this case, the selection procedure is performed in linear time.
5 Scheduling and Allocation

The importance of the control step scheduling and hardware allocation is reflected in the amount of effort spent on lots of previous intensive studies. The scheduler accepts a flow graph and a set of modules which are used to implement the signal flow graph. Two approaches are taken by us to attach designer's experience: If the design is subject to a speed constraint, the scheduling algorithm will attempt to make sufficient operations run in parallel to meet the constraint. Conversely, if there is a limit on chip area, the scheduler can be asked to serialize operations to give the maximum speed consistent with the constraint. The scheduler is integrated by basing on two famous scheduling algorithms. The resource-constraint approach is implemented by using heuristic list scheduling [8] with extended features which can perform functional pipelining or non-pipelining scheduling. The time-constraint approach is implemented by utilizing the force directed scheduling algorithm [9] with the same extensions. Designer can specify the constraint either time or hardware resource according this applications. So far, the designer always can get the optimal solution in a few seconds.

Once the flow graph has been scheduled, an allocation program is applied. The tasks of data path allocation can be further divided into three subtasks: 1) operation assignment, 2) register allocation, and 3) connection allocation. Facet[10] developed a heuristic clique partition method to solve this problem which first does 2) register allocation, then 1) operation assignment by weighted clique partition, and 3) connection binding. In our allocation process, each step is solved by a weighted clique partition algorithm. The operation assignment is modeled as a clique partitioning problem. The allocator builds a graph where vertex represents an operation in the flow graph and an weighted edge exists between two vertices, if and only if, the two corresponding operations will not be performed in a same control step. The graph is then partitioned into a number of cliques which will be allocated on the same module. The register allocation problem is also modeled as a clique partitioning problem. A graph is built where each vertex represents a variable and an edge exists between two vertex, if and only if, the two corresponding variable can share a same register (i.e. they have disjoint lifetime). However, weights are associated with the edges of the graph to reflect the preference of register-sharing among variables. The same clique's registers are grouped into a register file. The interconnect allocation is performed in the same way. The full resulting interconnect clique will be allocated on a bus.

Finally, the scheduling and allocation results will map to the physical level design. The high level design result is feed into a translator to transform the design into the intermediate text format, which is directly acceptable by the SCS datapath design tools.

6 Experiment Result

In this section we will discuss the results we have obtained by running our synthesis environment. To demonstrate the proposed methodology, the design of a video codec system preprocessor is presented here. Video sources normally generate three signals: a red, a green and a blue component (R, G and B). Subsequent processing typically requires these signals to be converted into a luminance signal (Y) and two chrominance signals (U and V). Since the R, G and B signals are oversampled to 27 MHz and then converted to 8 bits by the RGB matrix at this rate. The actual signal transformation is then defined by the following formulae where a word-length of 16 is assumed in the matrix:

\[ Y = (77 \times R + 150 \times G + 29 \times B) / 256 \]
\[ U = (-44 \times R - 87 \times G + 131 \times B) / 256 \]
\[ V = (131 \times R - 110 \times G - 21 \times B) / 256 \]

The C-dsp description of the video codec preprocessor is present in figure 2. The coefficient multiplications (with 77/256, 150/256, and 29/256 etc) in the RGB matrix is defined in front of the description body, and a signal flow graph description is generated in figure 2. The result layout of the processor is given in figure 7. This example shows that the behavioral C-dsp description is suitable for dsp application and the whole design environment will perform effectively on a practical design.

Another example was chosen as a benchmark for the 1988 High Level Synthesis Workshop and has been studied carefully. The Control Flow Graph is obtained from [11]. We are using the version of [9] which corrected some errors in the original graph in [11]. This is a more substantial behavioral description that contains 43 operations (additions and multiplications) submitted to over 60 precedence constraints.
It is stated in [11] that multiplications require an execution time that is twice as long as that for additions. Here, they were assigned two and one control steps, respectively. The critical path is therefore 17 control steps long. It is noted that retiming makes it possible to reduce this path to 16 control steps. However, to allow easier comparison with other synthesis system, we have not exploited this transformation. We have run our design system twice: once for the nonpipelined case and the other for the pipelined case.

This system is implemented in C language on a SUN3/110. In table 1, a regular design using non-pipelined multipliers requires two clock cycles. In pipelined design, the multiplier is pipelined and requires two clock cycles to produce an output, and can be initiated every clock cycle. A comparison of the results obtained in SPAID[1], by K&K[11], by HAL[9], by SPLICER[12], by CATREE[13], and by ours is also shown in table 1.

### Table 1: The comparison table.

<table>
<thead>
<tr>
<th></th>
<th>DSP/CAD</th>
<th>SPAID</th>
<th>SPLICER</th>
<th>HAL</th>
<th>K&amp;K</th>
<th>CATREE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Regular</td>
<td>Pipelined</td>
<td>Regular</td>
<td>Pipelined</td>
<td>Reg</td>
<td>Pip.</td>
</tr>
<tr>
<td>Multipliers</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ALUS</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Max. Inputs</td>
<td>27</td>
<td>20</td>
<td>32</td>
<td>27</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>Memory</td>
<td>19</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Registers</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Registers</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>12</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>Clock Cycles</td>
<td>17</td>
<td>21</td>
<td>16</td>
<td>17</td>
<td>19</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 7 Conclusion

CAD system for application specific DSP processor design was presented in this paper. It generates dedicated DSP hardware from a C program in conjunction with VLSI synthesizer. The system employs a top-down VLSI design methodology starting from a system algorithm description with few hardware images.
As for the DSP algorithmic representation, the data/control flow graph is used which is very familiar to designers with DSP background which are the potential user of our system. The CDFG can also be defined hierarchically and the designer can benefit from predefined algorithm model (e.g. IIR, FIR) to compose his algorithm description.

In comparison with previous approaches, for all benchmarks available from the literature, our designs are as good as, if not better than, those by others. While our programs consumes only a few seconds.

Areas for improvement for the methodology described here and implemented in DSPDA system are concurrently under investigation. Some are 1) A architecture/algorithm partition method. 2) A architecture driven style specific synthesizer for higher performance.

References