DAC — A Silicon Compiler System for High Performance DSP ASIC
Y Hu, J V McCanny and M Yan
IFI Institute of Advanced Microelectronics
Department of Electrical and Electronic Engineering
The Queen’s University of Belfast, Belfast BT9 5AH, Northern Ireland

Abstract

An overview is given of an application-specific silicon compiler system called DAC for the automated design of high performance DSP ASIC chips. This system consists of a number of application-specific silicon compilers and a set of universal utilities for the running and programming of these compilers. It provides an environment to systematically develop application-specific silicon compilers.

1: Introduction

As the field of Application Specific Integrated Circuits (ASIC) develops, automated synthesis of VLSI chips for a variety of Digital Signal Processing (DSP) applications has attracted considerable interest. Experiences in the design of high performance ASIC show that the efficiency and performance of a chip can be obtained by using high parallelism in its architecture design and full-custom method in layout strategy. Based on this knowledge, a specially designed silicon compiler is then the solution for rapidly generating a range of DSP VLSI chips. There has been a growing body of work on such compilers in the past number of years [1-3].

In this paper, an application-specific silicon compiler system, DAC (Dsp Asic Compilers), is presented. This system has been targeted to automatically generate a number of full-custom DSP ASIC chips as required based on a series of high performance systolic VLSI architectures developed in this laboratory. Its applications range from specific functions such as VQ, FIR, IIR, DCT and DFT to more general computations like matrix multiplication and vector inner product computation [4-6]. To fully utilise the advantages of these architectures, a number of design issues such as clock distribution and floorplan have to be considered and therefore an application-specific compiler is required for each of applications included in the system. There is also a set of universal utilities in DAC for systematically developing ASIC compilers as well as a user-friendly interface for the running of these compilers.

At present two compilers for high performance inner product array and vector quantisation chips have been implemented. In the following, an overview will be given of the system and this is followed by description of how the end-user can rapidly obtain VLSI chip design and how the compiler designer can add a new compiler to it. Then the IPA and VQ compilers will also be presented.

2: System structure

Figure 1 System structure of DAC

DAC basically consists of a graphics-based menu-driven shell, a library manager with the full-custom designed cell...
library, a number of utilities and application-specific programs which comprise various compilers with each implementing a particular DSP function. Supporting packages dealing with data access, layout composition, and communication between programs are shared by these blocks. The general structure is illustrated in Figure 1.

The DAC shell is a powerful environment. It manipulates all the application-specific programs and compiler utilities. Under its control, the screen is divided into three viewports. The first is for displaying the menu which is organised in a tree structure with each vertex corresponding to a sub-menu and each leaf to an application-specific program or utility routine. The menu contents can be easily changed or expanded by modifying the compiler definition file. The second is used as a graphics terminal which displays both textual and graphic output produced by application-specific programs and utilities.

Each of the DSP functions included in DAC system is realised by a compiler which has three application-specific programs, namely, the functional simulator, the core assembler, and the test pattern generator. These are scheduled by the shell and run as child processes of the shell. The system can then be expanded to contain new application-specific compilers by developing such three programs and modifying the compiler definition file without any change of the system software itself.

Shared utilities are those routines which are common to all the application-specific compilers. These include 1) a screen-oriented input utility for the input and validation of parameters, 2) a silicon assembler for the connection of the core and padding, 3) a test program translator for converting the test pattern to a program in a particular test language, and, 4) a design data processor for estimating performance and generating the input/output signal timing and design description. These are programmed in a general way and used by all the compilers. They can greatly reduce the cost of developing a new application-specific silicon compiler.

Cell layouts are designed by using conventional CAD tools based on λ-symbolic method and imported into the library in CIF format. They can be linearly scaled when the compiler uses them. Their electrical attributes concerned are extracted from simulation of SPICE and also stored in the library. The library manager handles transactions from and to the library. It has two interfaces. One is interactive under the control of the shell for the compiler designers to maintain the library and expand the system. The other is in the form of subroutines which are appropriately called by the utilities and application-programs to access the data stored in the library. The library collects all the cells together so that they can be fully utilised by different compilers.

There are also several supporting modules in DAC which are used to access CIF files, compose chip layout and program core assembler, functional simulator and test pattern generator. The use of these modules is shared by the utilities and application-specific programs for different compilers.

The DAC system software is based on GPR graphics package and programmed with PASCAL and C language. It runs on an HP-APOLLO workstation.

3: End-user environment

![Diagram of Compiler running procedure](image)

Figure 2 Compiler running procedure

The end-user interface for the whole system has been programmed with a much higher level than that of conventional CAD tools. Figure 2 shows a general running procedure of a compiler in DAC.

When the system is activated, it prompts the user to select from a list of DSP functions for which it contains functional silicon compilers. The user can then start a particular compiler by picking up a menu item corresponding to his application. After entering the compiler, no detailed structural descriptions are required from users for the input of his requirements (unlike...
conventional silicon compilers). Instead, only a set of dedicated parameters at high level is needed and the input of these parameters is carried out by filling a table on graphic screen. Explanations about the parameters will appear as the user requests. This set of parameters has been carefully selected by the compiler designers to define the application as well as structural information of the VLSI design in an adequate and necessary way. These are based on the knowledge in a number of fields ranging from original application algorithms, design of VLSI architectures to silicon layout designs.

After the parameters are entered, the user can invoke the simulation program to simulate the design. This is important where the round-off error is concerned. At this stage, the user can go back to the beginning and try other different configurations until satisfied results are achieved.

Once the user decides to take the design, the compiler will then go further to generate the core layout (in CIF format), test patterns and the design description. The core can be assembled with pads to form a complete chip and the test pattern can be translated to a test program on request. As in the first step, everything in the above process is carried out by picking up corresponding items from a particular menu. A high performance application specific VLSI chip design can be obtained virtually in a matter of a few minutes.

4: Compiler design procedure

The process of expanding the DAC system to contain a new application-specific silicon compiler follows three main steps, i.e. to design cell layouts and put them into the library, to modify the compiler definition file of DAC system and create some definition files for the system utilities, and to develop three application specific programs. In the following, we describe the procedure of developing a new compiler to the DAC system.

First of all, cell layouts for the new application have to be designed by using conventional VLSI CAD tool and it is required that the final layout data is in CIF format. The DAC’s library manager is then used to import the cell layouts into the library. Transistor count, power consumption, highest clock frequency and technology length unit $\lambda$ of these cells are also entered into the library at this stage.

A parameter definition file is then created for this compiler. The definitions include parameter names and descriptions, their maximum, minimum and default values and the conditions governing the inherent relations among these parameters. DAC’s parameter input utility will later be able to automatically ensure that these conditions are not violated when the parameters are entered in the use of the compiler.

Before writing the application-specific programs, the compiler definition file needs to be modified. This includes definition of the menu items, the application-specific program names and the inputs and outputs for the programs and DAC utilities. When the shell invokes them, the input and output file pointers will be passed to them as standard input and output of the child process. It is noted that, as the three programs are run as child processes of the shell, their development is independent of DAC system software and therefore will not affect the operation of the existing compilers.

A logic model library has been built up in DAC to support the programming of bit-level systolic array simulators in general programming language like PASCAL or C. The work to date has shown that programming a simulator of a bit-level systolic architecture is quite straightforward. It can be done by using several hundred lines of programs in addition to calling the sub-routines in the logic model library. Another choice for the designer is to write a program to generate an ELLA or VHDL simulation file according to the parameters supplied and the simulation is done by invoking a corresponding simulator on the background.

There is a powerful layout composition package in DAC which can be called in C and PASCAL for the programming of the silicon assembler. This package is based on $\lambda$-symbolic method and allows the assemblers to access $\lambda$-based design rules so that they can be programmed technology-independently to some extent. The functions this package provides include access of cell layout from the library, composition of new cell layout cell, placement of cells and arrays, routine between cells under the support of an internal switch-box router. These subroutines are internally connected to the DAC shell and therefore, composed layout can be instantly displayed on
the screen in graphics form once it is created. This provides a good environment for the debugging of the assemblers under development.

Test pattern generators are highly architecture-dependent and the designers have to make their decision of how the chip will be tested and then program the generators accordingly. If the generated patterns have the format recognised by DAC they can be translated into test programs of a particular test language.

Finally, the designer has to prepare design description files for the compiler to finish the development of a new application-specific compiler. This file should include information on architecture, data format and timing, pad definition, and so on.

5: IPA compiler

Inner product computation is widely used in DSP application. The Inner Product Array (IPA) chips realised by this compiler performs the following computation

\[ P = \sum_{i=1}^{n} x_i y_i + c \]

The architecture of the chips is based on the Auto-rounding Inner Product Array [5] which consists of n bit-level pipelined bit-serial multipliers and an accumulator. Data enters the array in bit-serial word-parallel (bswp) manner with LSB first. Vector Y can be stationary in the array while X moves across it. Special conversion circuits [6] are designed so that vector X can enter the chip in bit-parallel word-serial (bpws) format to meet the requirements of communication interfaces. The output is always in bit-serial format. Parameters and their ranges for the IPA chips are shown in Table 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Max</th>
<th>Min</th>
<th>Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxwiz</td>
<td>vector X word size (bits)</td>
<td>32</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>vywiz</td>
<td>vector Y word size (bits)</td>
<td>32</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>vecdim</td>
<td>vector dimension</td>
<td>32</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>synwz</td>
<td>system word size (bits)</td>
<td>48</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>vxfrmt</td>
<td>vector X format</td>
<td>1(bswp) 2(bswp)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ipnum</td>
<td>number of IPAs</td>
<td>10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>lambda</td>
<td>tech. length unit(0.01um)</td>
<td>150</td>
<td>50</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 1 Parameters for IPA compiler

Implementation of the IPA chips requires 18 different cells which are designed by using Compass CAD tool from VLSI Technology Inc. in 1.5μm double-metal double-well CMOS technology with λ being set to 0.8 μm. As they are designed with full-custom method their areas are only 30-50 percent of those implemented by standard cells. Switch-level simulation of the circuit indicates that the IPA chip generated by using these cells can be clocked up to 40 MHz when the parameter lambda is assigned to 80.

IPA simulator is programmed in about 800 lines of C language code and the core assembler in about 900 lines under the support of DAC's shared modules. The good testability of bit-serial architectures [1] simplifies the test pattern generator. It generates pseudo-random inputs for the simulator and collects the output to produce the patterns.

6: VQ compiler

The Vector Quantisation compiler is another one currently available in DAC. Vector quantisation with squared error distortion measure can be regarded as the computation[8] of

\[ \text{Max}(\sum_{i=1}^{n} x_i y_i + e, j \in [0...m-1]) \]

where vector X is the sample and Y is the codevector, the code book has m entries with each having an index. The index with which the codevector yields the maximum inner product will be the quantised code.

VLSI chips generated by this compiler are based on the Inner Product Array with appropriate comparison circuits. They are categorized into three types of schemes. In the first architecture, each codevector is assigned to a separate VQ element. A full search of the codebook is achieved by propagating sample vectors across these cascaded elements whilst the inner product values of the sample vectors with each codevector is being computed. The best matches to the input vectors can then be located by choosing the maximum of these values. In the second architecture, the situation is reversed. Sample vectors are loaded and stored on site in the circuits, whilst codevectors are clocked across the VQ elements. Again, the maximum value of these inner product computations is used to determine the
index of the most suitable codevector. Both can be used to construct very high performance VQ systems for image or multi-channel speech coding. The third type of VQ chip is obtained by adding a bit-parallel to bit-serial conversion circuit to the second. It is well suited to constructing a single-chip VQ system (excluding codebook memory) for some low bandwidth applications such as speech VQ coding. Table 2 lists the parameters for this compiler.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Max.</th>
<th>Min.</th>
<th>Def.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vqtype</td>
<td>VQ chip type</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>vqcbz</td>
<td>codebook entries</td>
<td>no limit</td>
<td>4</td>
<td>512</td>
</tr>
<tr>
<td>vqdim</td>
<td>vector dimension</td>
<td>32</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>vqewz</td>
<td>code word size (bits)</td>
<td>16</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>vqwss</td>
<td>sample word size (bits)</td>
<td>16</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>vqsawz</td>
<td>system word size (bits)</td>
<td>24</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>vqiofm</td>
<td>index output format</td>
<td>(para.)</td>
<td>2(seri.)</td>
<td>2</td>
</tr>
<tr>
<td>lambda</td>
<td>tech. length unit (0.01um)</td>
<td>150</td>
<td>50</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 2 Parameters for VQ compiler

The three application programs for this compiler are realised in the same way as those of IPA compiler and they share most of the codes.

7: Conclusion

A DSP ASIC compiler system, DAC, has been described for the automated design of high performance DSP ASICs. Most of the work undertaken to date has been concentrated on the design of the system software and the supporting modules for the systematic development of application-specific silicon compiler and some compiler utilities are still under development. The experiences in implementing the IPA and VQ compilers show that this methodology is suited to design high performance ASIC chips whose architectures are parameterisable and well defined. Expansions for DAC to include DCT (Discrete Cosine Transform), FIR and IIR filter functions is currently being carried out. The possible new applications in the future will include the DFT (Discrete Fourier Transform), division/square root computation [8] and wave digital filter [9].