A C-based synthesis system, Bach, and its application (invited talk)

Conference Paper · January 2001

DOI: 10.1145/370155.370309 · Source: DBLP

Akihisa Yamada
Morita Holdings Corporation
28 PUBLICATIONS **161** CITATIONS

Andrew Kay
Sharp Laboratories of Europe
16 PUBLICATIONS **142** CITATIONS

A C-based Synthesis System, Bach, and its Application

Takashi Kambe*, Akihisa Yamada*, Koichi Nishida*, Kazuhisa Okada*, Mitsuhisa Ohnishi*,
Andrew Kay**, Paul Boca**, Vince Zammit**, Toshio Nomura**

* Integrated Circuits Development Group,
Sharp Corporation
2613-1, Ichinomoto-cho, Tenri, Nara 632-8567, Japan
E-mail: {kambe, yamada, k_nishi, okada, ohnishi}
@icg.tnr.sharp.co.jp

** Sharp Laboratories of Europe,
Edmund Halley Road, Oxford Science Park,
Oxford OX4 4GB, United Kingdom
E-mail: {akay, ppb, vince, tosh}@sharp.co.uk

Abstract

In system LSI design, a desirable system is one that allows the designer to describe, partition, and verify systems, and to generate circuits efficiently. In this paper, we describe a C-based system LSI design system called Bach which we have developed. Using the example of an MEPG-4 video codec design, we summarize its design flow, effects and current issues.

1 Introduction

Recently a number of behavioral synthesis methods have been described. These methods enable the generation of an RTL description from a behavioral one without any a priori hardware structure[1]. With such tools the designer should be able to concentrate properly on algorithms and high-level architecture, and quickly see the consequences (to the circuit) of each structural change. However, most modern applications are implemented as a combination of several communicating processes, whereas existing high-level synthesis approaches are dedicated to individual module-level design: they cannot handle whole circuits which consist of multiple communicating processes.

For computationally intensive circuits such as image and video processing, however, we need a design environment where we can construct and verify the hardware algorithm for the whole system and synthesize it too. To design hardware efficiently at a high level, several synthesis methods using C-like input languages have been proposed[2-9]. Most of those approaches are still for module level design and not suitable for whole system-level design, because they do not have communication model.

Regarding input languages, since ANSI C/C++ does not support parallelism or bit-accurate operations, some extend it[3-5,9] while others introduce class libraries that support these features[2,6-8]. To exchange designs among these tools, several groups are working on the standardization of a C-based system-level design language[10,11]. Although SpecC[10] has been proposed as a system-level design language, no hardware synthesizers have been developed for it yet. SystemC[11] is gaining support in the EDA community. However, it has a timed semantics and so is not suitable for high-level design.

In this paper, we describe a new EDA environment, called *Bach,* for VLSI design. Its distinctive features are:

- It has a C-based user language, with untimed semantics, suitable for large-scale circuit design.
- It has a synthesizer that can compile a program (written in this language) describing the untimed behavior of hardware into RTL VHDL. It can also automatically generate interface circuits for data transfer between processes.
- It has a simulator that handles bit-accurate operations at the C level and is 10-100 times faster than HDL simulators.

Bach's input language, *Bach C,* is based on the ANSI C language, with extensions to support explicit parallelism, communication between parallel processes and bit-width specification of data types and arithmetic. The semantics for parallelism and communication based on CSP[12] and `occam` [13].

Circuits synthesized using Bach consist of a hierarchy of sequential threads, all running in parallel and communicating via synchronized channels and shared variables. Using Bach, the user can develop parallel algorithms, explore the design space of architectural choices and generate complex circuits in a much shorter time than previously. There is an interactive simulation environment for source-level simulation and debugging. Bach also provides a tool for converting Bach C into ANSI C for fast validation.

We have already applied the Bach system to several commercial designs, among which we describe the design of an MPEG-4 video codec in this paper. We show its design flow and effect. We also describe future work.

2 Issues in Conventional Design Process

General issues of conventional methods can be summarized as follows:

1) Most EDA tools available commercially do not facilitate automated high level design and synthesis of communicating multiple processes. Hardware for communication between subsystems has to be designed manually.
2) Validation of the design has to be done in both software (e.g. C) and hardware (e.g. VHDL) languages.

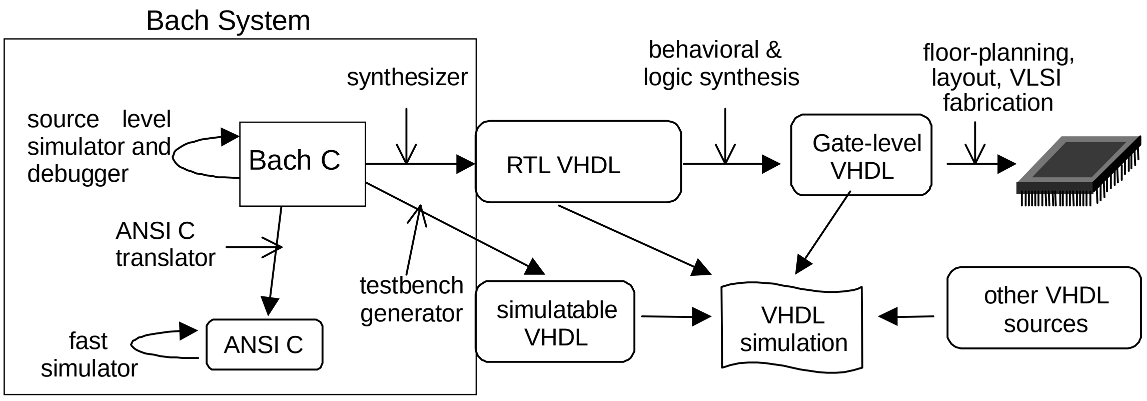

Figure 1: Design Flow with Bach

3) The test-bench has to be created manually in two versions, for software and hardware languages.
4) If memory resources such as RAMs and ROMs are to be shared, we may have to design arbitration circuits manually to avoid access conflicts.

3 Bach System

The Bach system was developed in order to design efficient system VLSIs more quickly, taking into account the issues described above.

Bach C has the statement **par** added to support explicit parallelism (concurrency). It also supports the declaration **chan** for synchronized channel communication, as in CSP or `occam`. Using **par** and **chan**, we can specify the behavior of a complete system, including communications between parallel subsystems.

The Bach synthesizer automatically generates interfaces for channel communications or external storage (RAM/ROM). It determines the timing of communications of synchronized channels to ensure data transfer, if possible. Otherwise, handshake circuits are created for synchronized channels.

The designer may build a test-bench in Bach C to test any particular design. Bach can generate a VHDL test-bench with the same simulation behavior, so that the resulting circuit can be tested without having to manually recode the test-bench (see Figure 1).

3.1 Bach C Language

Bach C supports almost all constructs in ANSI C, and adds a few more which are specially tailored for simulation and hardware description.

3.1.1 Untimed Semantics

The semantics of Bach C is untimed, which means that you *cannot* tell from examining the source in which clock cycle any particular operation will occur. This means that the designer does not have to worry about timing issues and also that Bach can apply various types of hardware optimization to the design. The Bach synthesizer ensures that data is never lost due to timing differences. Readers should compare this to the semantics of VHDL and the Handel hardware compiler[14,15] which assumes an exact timing to each statement.

3.1.2 ANSI C features

Bach C supports almost all ANSI C constructs, including **`while, if, switch, do, for, static, struct, typedef`**, multi-dimensional arrays, strings and all integer arithmetic, binary, logical and type coercion operators. Floating point may be used in test-benches. Union and pointer types are not supported.

3.1.3 Basic data types

Bach C allows each data type to be represented by a given number of bits, and these can be signed or unsigned. The arithmetic rules for these types extend the usual C notion of automatic type coercion.

This example declares two variables, `a` and `b`, of widths 24 and 16 respectively. Inside the loop the value of `b` is automatically extended to 24 (signed) bits to allow the subtraction to occur:

```
int#24 a=(101*100)/2;
unsigned#16 b=1;
while (a) { a -= b++; }
```

The simulation and synthesis tools support arithmetic to any number of bits of precision, e.g. a 128-bit bus. Unlike SystemC, there is no syntactic difference between big numbers and small ones.

3.1.4 Bit-manipulation operations

Several bit-manipulation operations have been added to allow certain operations to be performed more succinctly and cheaply. The symbol @ is used for concatenation of two bit strings. The grab operator `[..]` is used to select a subset of bits from an expression. For example:

```
b = a[23..8] ; // b=16 top bits of a
a = a[0..23] ;  // reverse bits of a
b = b[7..0]@b[15..8];
 // to swap byte order of b
```

3.1.5 Stream types

Input and output streams are supported to allow interaction with files and the console (for test-bench simulation only). Integers can be written and read to arbitrary precision, in binary, octal, decimal or hexadecimal.

3.1.6 Parallelism and communications

The **par** keyword is used to make a collection of sub-processes execute concurrently. The sub-processes may be arbitrary compound processes, and may themselves contain further **par** statements. When a **par** statement executes, all of its sub-processes are executed concurrently. The **par** statement terminates when all the sub-processes have terminated. Unlike VHDL and SystemC, parallelism is *not* restricted to the top-level.

There are two ways to communicate between different concurrent processes (threads). The first is by synchronous channels, declared with the keyword **chan**. Each channel transfers data of a given type synchronously from one thread to another. The sender uses the function **send(**ch,v**)** to send the value v down a channel ch of the same type. The receiver can use **receive(**ch**)** to receive the value. Both sender and receiver must be ready in order for the transfer to occur. If they are not ready, they just wait. The test **ready(**ch**)** is available to the receiver to check whether data is waiting.

In this example the values 9,..,0 are sent from the first sub-process to the second, which prints them to the standard output in hexadecimal (where HEX is a macro defined to be 16):

```
chan int#4 ch;
par{{
   int#4 x=10;
   while(x--) send(ch, x);
   }{
   int y;
   do{y=receive(ch);
      putint(stdout,HEX,0,y);
     }while (y);
   }}
```

The second method uses asynchronous channels, denoted by the keyword **achan**. Each **achan** may be written to and read from as many threads as desired, and so behaves as a global variable. However, since it is asynchronous, the exact time of each read or write *cannot* be predicted at source simulation time.

In the following (unlikely) example, the three assignments could be scheduled in any order:

```
achan a = 99;
par {a=f(0); a=f(1); b=a;}
```

achans are cheaper to implement than **chan**s in most cases, but the price is that the code may be slightly harder to understand. Both **achan**s and **chan**s may be used as the interface between the Bach circuit and the outside environment. Syntactically, this is achieved by declaring them as arguments to the top-level circuit description function.

3.1.7 Call by reference

Bach C does not have pointers; nevertheless, it is often useful to pass parameters to functions by reference. This is always the case for **chan**, **achan** and array, but by default simple variables are passed by value. We borrow some syntax from C++ for call by reference, using the ampersand (&) in the function declaration. Note, as shown in this example, that dereferencing in the procedure body occurs automatically (as in C++), and does not require the dereference operator (*). Similarly, the caller does not require the reference (&) operator for its arguments:

```
void swap(int &a, int &b) {
  int temp = a;
  a = b;  // no '*' needed
  b = temp;} // no '*' needed
void main(void) {
  int a=1, b=2;
  swap(a,b);} // no '&' needed
```

3.1.8 Timing Specification

Although untimed semantics may be sufficient to specify behaviors of computationally intensive subsystems, the notion of real time is essential for embedded systems. Bach supports C **pragma** statements for specifying certain timed behaviors explicitly; for example, for I/O synchronization and throughput specification.

3.1.9 Controlling resource use

It is possible to give instructions to the Bach synthesizer to control the way that resources are used. These instructions do not change the external or simulation behavior of the program. We give them using pragmas. The most important pragmas in this class are those which control the mapping of arrays to external RAM or ROM.

3.2 Synthesizer

The Bach system has a behavioral synthesizer[16] which compiles a parallel algorithm into several communicating modules, and generates I/O interfaces between them. Its output is RTL circuits in VHDL which is synthesizable with commercial logic synthesis tools, e.g. [17].

Generally, synchronous channels, **chan**s, are implemented with the handshake protocol circuitry. Users can specify the level of handshaking (e.g. receiver ready, sender ready) for different I/O with the aid of pragmas.

Asynchronous channels, **achan**s, are mapped to shared storage (registers, RAM or ROM) and their interfaces are automatically generated. If there exists conflict among shared storage, arbitration logic will be generated.

Finally, a *top-hierarchy* is generated to bind together all the modules and resource entities and to propagate clock and reset signals.

3.3 Simulation and Validation

As well as the source-level simulator and debugger, Bach can convert Bach C code into ANSI C for fast behavioral validation. Also Bach C test-benches can be converted into VHDL and used for validating the generated circuit.

4 Real LSI design: MPEG-4 video codec

The most recent application of Bach to a real LSI design for consumer products is MPEG-4. We show its design flow and current issues to be solved in this section.

4.1 Design Flow

We started from an ANSI C program developed by algorithm designers.

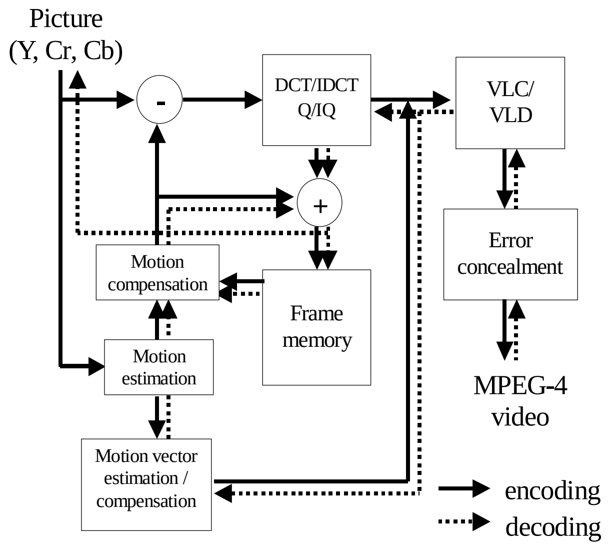

Figure 2: Data flow of MPEG-4 video codec.

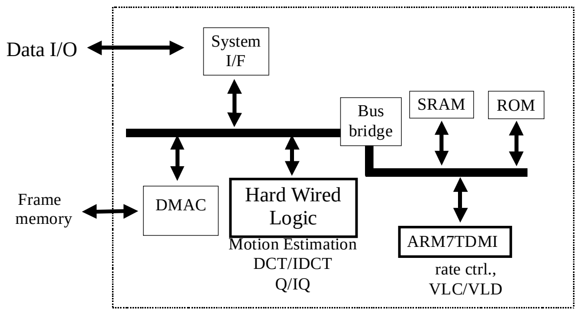

Figure 3: Architecture diagram for MPEG-4 video codec.

a) Architectural Design (HW/SW partitioning)

Figure 2 shows an overview of the MPEG-4 video codec algorithm, which includes rate control and error concealment suitable for SW as well as motion estimation and DCT suitable for HW. We partitioned the given algorithm into the HW and SW parts and implemented the hard-wired logic. The SW part executes on an ARM7 (see Figure 3).

b) Bach C design

Although Bach C supports most data types and constructs such as **if**, **while**, and **for** available in ANSI C, it does not support pointers and recursive function calls. Thus we had to modify the code to allow us to use the Bach simulator. Then we refined the code for circuit optimization. The main tasks were:

- to insert **par** to specify block partitioning explicitly;
- to specify the bit width of each variable and operation;
- to modify loops such as **for** and **while** into efficient ones.

Loops have an impact on the registers and functional units used and also affect the performance of synthesized circuits. Thus we analyzed the area and performance of the synthesized circuit and modified the loops accordingly.

The simulation time at the Bach C level is about 100 times faster than the corresponding VHDL simulation. Thus checking functionality at the Bach C level avoids a lengthy design loop.

c) Bach C compilation (Behavioral synthesis)

The Bach C code is compiled into RTL circuits using the Bach synthesizer, meeting the given constraints. To achieve this, the compiler uses a logic-synthesis tool to estimate the delay and area of each functional unit in the design. The estimates are stored in a cache file and reused.

Since the compilation time of the Bach synthesizer was very short, we were able to explore different architectural designs.

d) RTL simulation

The functionality of the synthesized circuit is verified at the Bach C level. We checked its performance and data transfer between external circuits through simulation of the generated VHDL.

e) HW/SW Co-verification

To verify the behavior of the HW and SW parts, the SW part executes on an ARM7, we used a co-verification tool and bread-board with the VHDL generated by the Bach compiler.

4.2 Current Issues

Since the target of the Bach system is currently hardware compilation, the following issues concerning HW/SW co-design of the MPEG-4 video codec had to be solved as follows:

a) HW/SW partitioning

The architectural design including HW/SW partitioning was done by hand.

b) System level evaluation

The performance, area and power consumption of the total system could not be evaluated at the Bach C level.

c) Interface circuit design between HW/SW

The Bach synthesizer generated the required AMBA bus I/F circuits automatically[18]. Note that, at present, Bach does not support other protocols.

d) IP reuse

It is sometimes more efficient to reuse optimized Intellectual Property (IP) than to synthesize new IP from scratch. Regarding the MPEG-4 design, DCT/IDCT IP optimized for fast circuits was used. Since the Bach synthesizer cannot introduce such IP into the synthesized circuit automatically, we modified the VHDL code manually.

5 Remarks and Future Work

We described an overview of the Bach system and MPEG-4 design as one of its applications. The current Bach system is hardware-design oriented and to establish the HW/SW co-design flow shown in Figure 4, we are developing the next generation Bach, which includes:

- System (HW/SW) partitioning,
- System level evaluation,
- Interface circuit generation, and
- Linkage to IP libraries.

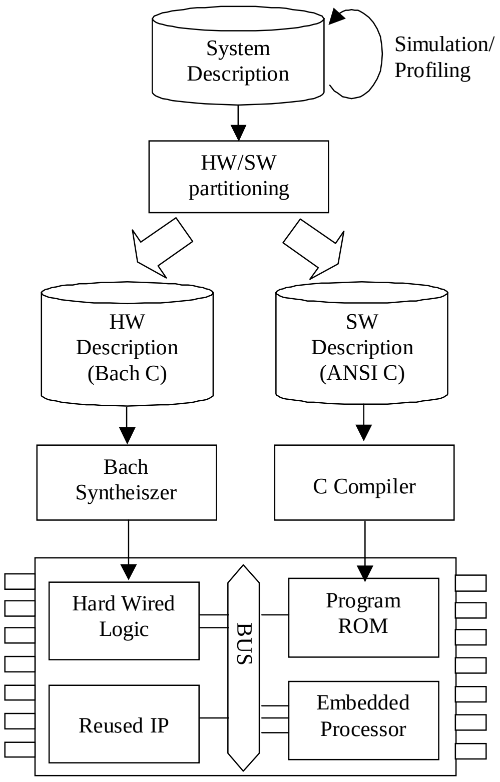

Figure 4: HW/SW co-design flow using Bach.

References

[1] D. Gajski, A. Wu, N. Dutt, S. Lin, *High-level Synthesis: introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.

[2] A. Ghosh, J. Kunkel, and S. Liao, "Hardware Synthesis from C/C++," in *Proc. of Date'99*, pp. 387-389, March 1999.

[3] G. Arnout, "C for System Level Design," in *Proc. of DATE'99*, pp. 384-386, March 1999.

[4] K. Wakabayashi, "C-based Synthesis Experiences with a Behavior Synthesizer, "Cyber"," in *Proc. of DATE'99*, pp. 390-393, March 1999.

[5] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, *Spec C: Specification Language and Design Methodology*, Kluwer Academic Publishers.

[6] http://www.cynapps.com/

[7] http://www.cleveldesign.com/

[8] http://www.frontierd.com/

[9] A. Yamada, K. Nishida, R. Sakurai, A. Kay, T. Nomura, T. Kambe, "Hardware synthesis with the Bach system," in *Proc. of IEEE ISCAS'99*, Vol. VI, pp.366-369, 1999.

[10] http://www.specc.org/

[11] http://www.systemc.org/

[12] C.A.R. Hoare, "Communicating Sequential Processes," Prentice-Hall, 1985.

[13] INMOS Ltd, "occam2 reference manual", Prentice-Hall International, 1988.

[14] I. Page and W. Luk, "Compiling occam into FPGAs," in *W Moore and We Luk (eds.), FPGAs*, pp. 271-283, Abingdon EE&CS Books, 1991.

[15] http://www.celoxica.com

[16] K. Nishida, K. Okada, M. Ohnishi, A. Kay, P. Boca, A. Yamada, T. Kambe, "A Behavioral Synthesizer for Hardware Compiler –Bach-," in *Proc. of IPSJ DA Symposium '99*, pp. 95-100, 1999 (in Japanese).

[17] "Design Compiler Reference Manual," ver. 1999.10, Synopsis, 1999.

[18] M. Ohnishi, K. Nishida, K. Okada, A. Yamada, T. Kambe, "A hardware/software co-design environment with hardware compiler Bach," in *Proc. of IPSJ DA Symposium 2000*, pp. 13-18, 2000 (in Japanese).