Flexible codesign target architecture for early prototyping of CMIST systems

Kalle Tammemäe, Mattias O'Nils, Ahmed Hemani
Electronic System Design Laboratory
Royal Institute of Technology
Electrum 229, Isafjordsgatan 22-26, S-16440 Kista, Sweden
e-mail: nalle@ele.kth.se

Abstract. We present an architecture for rapid prototyping of selected control-dominated HW parts resulting from HW/SW cosynthesis. Our codesign toolkit AKKA[1], which targets to design of control intensive systems, consists of design space exploration, estimation, partitioning, cosimulation and verification subtools in common environment. For fast prototyping of HW parts an EVC board with Xilinx XC4013 FPGA is used. We describe an multifunctional architecture, programmable into the FPGA, consisting of reusable interface/memory section and synthesized functional part. Usability of architecture in client-server relationship is described.

1 Introduction

The major trends in the digital industry are the shortening of design time and growing complexity of the systems. Era of tens of millions of gates per chip has just started. The solution lies in moving from register-transfer description style of design to the functional and system description style, which is possible only by using more intelligent synthesis tools or by narrowing the application domain. The latter effectively limits the design space at system level and makes it possible to concentrate only on real target architectures. Selected application domain in our case is a telecommunication, an area where Control and Memory Intensive (CMIST) tasks are typical. Quite often pure software solutions are acceptable, but for satisfying some stringent timing constraints speeding up of some key functions is essential. From this point of view, we consider the hardware part of the overall system as a coprocessor. This architecture enables the designer to find a trade-off between flexibility, speed, and design time. Another issue is the design validation, which can be solved exploiting coverification methodology on all design flow stages.

In this paper we cover design flow for embedded CMIST with emphasis on estimation, partitioning, coverification, and rapid prototyping, as implemented in AKKA's design environment. We use C/C++ as functional level specification language due to its popularity among industrial users and extended GNU compiler as base for solving codesign subtasks. Toolkit AKKA (Figure 1) consists of frontend design analysis tools, interactive design browsing facilities, HW/SW partitioner and target code generation subtools. After partitioning, the design can be validated against original specification

with several cosimulation techniques. Backend tools for HW are industrial synthesis systems like Synopsys or SYNT and Xilinx XACT. The final SW code, depending upon the selected core processor, can be generated by GNU C/C++ compiler itself or by proper cross-compiler.
The subtools are presented to the user through a graphical user interface based on Tcl/Tk.

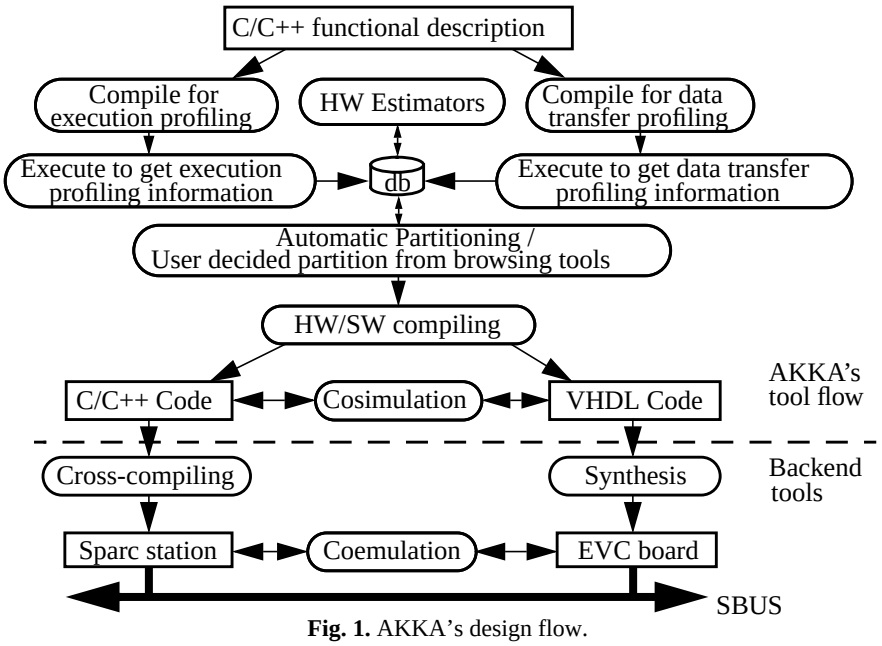

Fig. 1. AKKA's design flow.

There are several codesign systems using C as specification language and targeting to set of general microprocessors or DSPs and FPGAs, for instance, UMIST [8],CoDeX [6], BOAR [7]), and PRISM-I [5]. Our approach differs from others in terms of the overall methodology, partitioning algorithm used, cosimulation strategy, and selected prototyping platform [2][3]. In this paper we concentrate on early prototyping of HW, using synthesis, cosimulation and coemulation.
In this paper coprocessor is viewed as a *server* servicing the SW, which is running on the main processor as the *client*. Such notation is taken from [11], expressing unambiguously master-slave relationship between SW and HW.

2 AKKA's Codesign Framework

AKKA's toolkit components perform the following activities:

- System **profiling** for collecting data about the subpart features (execution time in SW, function complexity, amounts of data transferred between subparts, etc.). In addition to standard function profiling capabilities built-in into compiler, we are analyzing code until loop granularity.
- **Estimation** of HW execution time and cost or number of the gates. If profiling does not give enough accurate SW estimation time, this can also be calculated by the estimator.

- **Partitioning** into HW and SW, taking into account profiling data, estimation results, and designer's own preferences as well. We use modified knapsack stuffing algorithm for maximizing available HW utilization.
- **Interface synthesis** depending of the estimated channel characteristics and bandwidth requirements.
- **Cosimulation** of the SW and HW descriptions.
- **Coemulation** of the SW and synthesized HW on an early prototyping systems.
- **Synthesis** for implementation.

C/C++ entry language enables visualization of design in the form of a tree-like call graph, where nodes are loops or functions, branches express passing the control. We use interactive graph browser daVinci [13] for visualization, browsing, and inserting designer preferences for directing the partitioner. Probable candidates for accelerating in HW coprocessor environment should have as few as possible input/output parameters, because data transfer time can easily offset the estimated speedup in coprocessor. An FPGA at present does not have enough resources to compete with SW processor data pipes and caches.

2.1 Code Generation and Coverification

The tool-kit allows verification in all the different design phases. After partitioning, cosimulation of the hardware and software components is fully supported. Design parts, selected for implementation in HW, are converted into synthesisable behavioural VHDL. Depending on the design phase, IP socket based interface is generated for cosimulation or SBus oriented interface for coemulation. We are using a Xilinx XC4013 FPGA based Engineer's Virtual Computer board [4], connected to the SBus of the workstation, for early prototyping of selected HW parts. This board can be optionally complemented with 2MB local memory, allowing coverage of a wide set of possible HW architectures. Coverification sequence is presented in Figure 2.

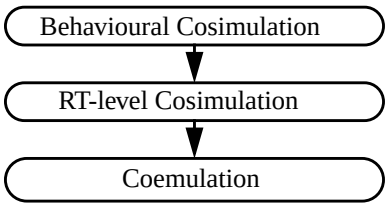

Fig. 2. Verification stages.

2.2 Verification Stages

During cosimulation the data between HW and SW parts is transferred using IP-socket. A benefit of such mechanism is the possibility to run HW and SW on different workstations or servers, balancing the machine load. Another benefit is a partial similarity to SBus 32-bit data transmissions which simplifies interface generation.

Behavioural Cosimulation. At behavioural level the cosimulation of HW and SW parts is done by direct IP-message send/receive primitives.

RT-Level Cosimulation. During RT-level cosimulation the full target interface (SBus)

protocol is simulated from HW side, thus establishing real clock-level simulation.

Coemulation. Hardware is programmed into FPGA and real interface is exploited for data transfer between HW and SW.

3 Target Hardware Architecture

Due to CMIST nature of our applications and GNU C++ compiler approach, flip-flop based registers are not suitable for implementing register files of generated HW, because that would result in wasting of limited CLB resources. Instead we use the memory features of Xilinx 4000 family CLB's, where for implementing 32x32 bit register file only 32 CLB-s are used (in case of using flip-flops the equivalent file size is implementable on 512 CLB-s, near 100% of XC4013 flip-flops). The proposed programmable architecture (seen in Figure 3) consists of three main blocks - local on-chip memory, memory interface and user module. Each of these blocks corresponds to a VHDL entity. From the fixed modules the middle one is the most critical one, accomplishing the time-critical multiplexing of memory between SBus (on SW side) and user module. The user module, actually one big FSM with datapath, is generated directly from GNU compiler intermediate register transfer language description of the design. Every superstate of FSM consists of the following subfunctions - data preparation, execution, storing a single result, and determination of the next state. For operations, a set of 32-bit multiplexed functional units is used.

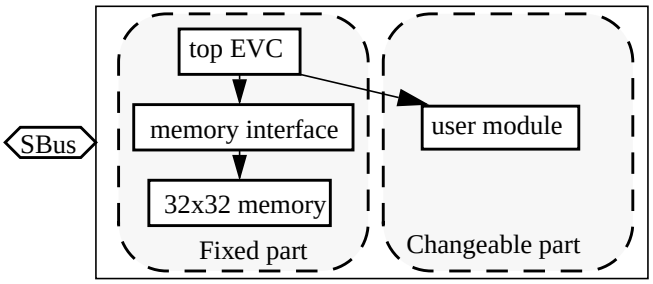

Fig. 3. Target architecture.

Our target architecture (seen on Figure 4) consists of up to three mutually exclusive controllers with a common datapath and a register file. The number of controllers is limited by Xilinx FPGA CLB peculiarities, limiting one-level multiplexer to 4-to-1. Preferably only register addressing mode is implemented. If register indirect mode is also needed, then we have to restrict number of functions to two, preserving the same speed i.e. number of multiplexer stages, and visibility of registers. Client (as SW) uploades initial data into register file and initialises selected function. After completion signal "done" can be polled out from server (HW) status word, informing client, that

results for downloading are ready.

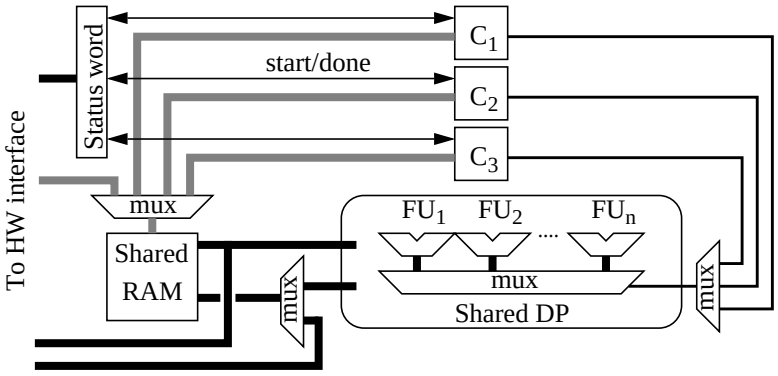

Fig. 4. Mapping multiple functions to FPGA.

Problems so far are:

- the limited number of gates in Xilinx XC4013 FPGA (13'000 by handbook [12])
- limited size of register file (until 32 of 32-bit registers are reasonable for implementing the register address mode)
- limited width of Xilinx hard macros (8- and 16-bits), whereas 32 is preferred for most of applications and better compatibility with GNU compiler.

4 Results

We have tested elaborated codesign methodology and multifunctional programmable architecture speeding up some of ATM F4 [9] and D-AMPS [10] channel decoder time consuming loops. Used clock frequency on FPGA board (25 MHz SBus clock) does not allow to achieve remarkable speedups during prototyping, instead we can measure time spent in HW and estimate correct speedup when real ASIC with higher clock frequency is used. In following code segments, expressing several transformation stages of SW and HW are provided. We use for example one critical loop from D-AMPS demonstrator (Figure 5). Address 64_{10} in Figure 6 is assigned to status register, here bit 0 represents signal *start* and bit 1 signal *done*.

```
for (i=0; i<constr_len; i++)
        {
          temp >>= 1;
          cnt ^= temp & 1;
        }
```

Fig. 5. C code of selected loop to HW in D-AMPS channel decoder.

```
senddata(0, constr_len);     // Reg0 := constr_len
senddata(1, temp);           // Reg1 := temp
senddata(2, cnt);            // Reg2 := cnt
senddata(64,1);              // Start HW
while(!(recvdata(64) & 2)); // Wait until hardware done
cnt = recvdata(4);           // result :=Reg4
```

Fig. 6. Code inserted into SW instead of original loop.

Generated behavioural VHDL (seen in Figure 7) is simulated and converted into RT-level VHDL (like user module in Figure 3), where clock dependency is already fixed. During conversion register file size is specified and functional units are allocated. RT-description consists of concurrent statements, instantiating functional units, multiplexers, interconnections, and of finite state machine, controlling the function execution and implementing start/done protocol with SW.

```
WHILE (statevar /= exit_state) LOOP
  CASE (statevar) is
  WHEN state_0 =>
    statevar := state_1;
  WHEN state_1 =>
    temp:= SHR (temp,1);
    var_79 := temp and 1;
    cnt := cnt xor var_79;
    i := i + 1;
    IF (i < contst_len) THEN
      statevar:= state_1;
    ELSE
      statevar := exit_state;
    END IF;
  WHEN exit_state =>
    statevar := exit_state;
  END CASE;
END LOOP;
```

Fig. 7. Partial behavioural VHDL code of loop.

4.1 Utilization of XC4013 resources

Although most of the proposed architecture components are reusable between different controller functions, the datapath unit size is critical. Restricting bus width to 16-bit enables to exploit more Xilinx hard macros and implement multiplication operation in DP. The 32-bit bus allows to implement only simple operations like adders, subtractors, shifters, etc. For instance - well-known fibonacci function, which needs only one adder takes in total 281 CLB-s from 576 CLB-s available (32-bit bus width, 32x32 RAM, and medium optimization effort in Synopsys FPGA Compiler).

5 Conclusions and Future Plans

The used FPGA board has given us a good opportunity for early prototyping of designed hardware and testing its interface. For this purpose a special programmable architecture has been developed, which consists of fixed interface and changeable controller parts. Special attention was given to a good visibility of all internal registers and status bits. From the codesign point of view user can check the design partitioning quality and accuracy of the estimation just after compiling HW into uploadable bitstream. Without prototyping, the critical timing issues will be discovered too late, after silicon compilation stage. Thus, early prototyping inserts one additional validation stage before ASIC synthesis, enabling test with real-near clock frequencies.

We have tested our FPGA-targeted coverification methodology, and tools with two industrial demonstrators from telecommunication field. Results have validated current approach, although usefulness of methodology depends heavily from new Xilinx FPGA families, not available yet. We have done preliminary research on using more advanced XC4000E and XC4000EX family FPGAs instead of current one. Simple re-

placement allows us to shorten the executing time of FSM superstates in half, establishing thus more real-like coemulation.
Our prototyping target in the near future would be Xilinx XC6200 coprocessing optimized family which includes a built-in interface circuit, partial fast reconfiguration features, and large equivalent gate counts available. In case of XC6200, we can replace client-server relationship with true multimaster relationship, enabling shared memory for passing data between HW and SW.

References

1. Kalle Tammemäe, Mattias O'Nils, Axel Jantsch, Ahmed Hemani: "AKKA: a Toolkit for Cosynthesis and Prototyping". *IEE Digest No.96/036 of Colloquium on Hardware-software Cosynthesis for Reconfigurable Systems*, pp. 8/1-8/8, Bristol, February 22, 1996.
2. Axel Jantsch, Johnny Öberg, Peeter Ellervee, Ahmed Hemani, Hannu Tenhunen: "A software oriented approach to hardware-software co-design" *International Conf. on Compiler Construction*, CC-94, pp. 93 - 102, Edinburgh, Scotland.
3. Mattias O'Nils, Axel Jantsch, Ahmed Hemani, Hannu Tenhunen: "Interactive Hardware-Software Partitioning and Memory Allocation Based on Data Transfer Profiling". *Proceeding of International Conference on Recent Advances in Mechatronics*, Page 447-452, August, 1995.
4. "Engineers' Virtual Computer. User Guide EVC1s". *Virtual Computer Corporation*, Aug., 1994.
5. Peter M. Athanas, Harvey F. Silverman: "Processor Reconfiguration Through Instruction-Set Metamorphosis". *IEEE Computer*, March 1993.
6. Reiner W. Hartenstein, Jürgen Becker, Rainer Kress, Helmut Reinig: "CoDeX: A Novel Two-Level Hardware/Software Co-Design Framework". *VLSI Design'96*, Bangalore, India, Jan. 1996.
7. Sami Hoisko, Harri Hakkarainen, Kari Vihavainen and Jouni Isoaho: "Specification, Hardware Implementation and prototyping Environment for Image Processing Algorithms", *Proc. IEEE International Symposium on Circuits and Systems*, pp. 834-837, Atlanta, USA, May 1996.
8. John Forrest, Stephen Wright: "The Cosynthesis of C using Assembly Extraction". *IEE Colloquium of HW/SW Cosynthesis of Refconfigurable Systems*, Bristol, 1996.
9. Martin de Prycker: "Asynchronous Transfer Mode: Solution for the Broadband ISDN". *Ellis Horwood Limited*, 1991.
10. "EIA/TIA Interim Standard, Cellular System Dual-Mode Mobile Station - Base Station Compatibility Standard". IS-54-B, April, 1992.
11. Jan Madsen, Bjarne Hald: "An Approach to Interface Synthesis". *Proceedings of the 8th International Symposium of System Synthesis*, ISSS'95.
12. "The Programmable Logic Data Book". *Xilinx, Inc.*, 1994.
13. M. Frölich, M. Werner, "The Graph Visualization System daVinci - A User Interface for Applications", *Technical Report No. 5/94*, Department of Computer Science, University of Bremen, Sept., 1994.