Advanced Silicon Prototyping in a Reconfigurable Environment

Matt Aubury, Ian Page, Dominic Plunkett, Matthias Sauer and Jonathan Saul

Hardware Compilation Group

Oxford University Computing Laboratory

Parks Road

Oxford OX1 3QD, United Kingdom

email: mpa,page,jsaul@comlab.ox.ac.uk

Abstract

A flow is proposed which offers a programming approach to the systems design of application specific micro-controllers. Tools have been developed for compilation and cosimulation, and a reconfigurable board has been designed which can be used for rapid prototyping. The final design can be compiled into a structural VHDL netlist for a standard cell ASIC process.

1 Introduction

Innovation in the market for low volume embedded systems is currently stifled by a number of factors: the tools are expensive, development times are long, non-recurring engineering costs are high. These factors prevent companies from producing embedded products quickly, cheaply, and in low volumes, which is especially significant for small companies with limited resources.

The aim of the work described here is to produce a "one-stop" evaluation and development kit for low and medium volume designs of application-specific embedded micro-controllers. This minimises the time from initial idea to silicon and allows system designers with little knowledge about hardware issues to design ASICs. The kit consists of a PC-based development card containing field programmable gate arrays (FPGAs), a micro-controller based on the ARM7TDMI core, and codesign software and FPGA tools. This work is part of an EU-funded collaborative project between ARM, Atmel-ES2 and Oxford University, known as the Aspire project.

The development card is aimed at rapid prototyping of designs containing an ARM core and custom hardware. The software tools consist of a C++ software compiler and a hardware compiler from a C++–like language. Using two similar languages instead of a "software" and a "hardware" language makes it easier for the designer to explore different hardware/software partitioning trade-offs. The output of the hardware compiler is initially FPGA netlists, which are evaluated on the evaluation board. When the design has been finalised, the hardware compiler produces a VHDL description of the hardware, which is then fed directly into Atmel-ES2's design flow, resulting in an ASIC containing an ARM core and custom hardware. The development card can also be used as a low volume product in its own right.

Hardware/software codesign is an extensive area of research (see e.g. [14]). A number of researchers focus on specification techniques for automatic partitioning [9], [4], [6]. The Ptolemy project [7], [10] focuses on signal processing applications and uses data flow models for specification. Other systems, like CASTLE [5] approach the problem from the hardware design automation perspective. Cosimulation [3] and its integration with synthesis [15] has also gained attention by other groups. We believe that, especially for the purpose of designing application specific micro-controllers, codesign should be mainly a programming task [13], where hardware compilation supports cosimulation and the design of correct hardware [8].

This paper describes the design flows and development card, using a simple video application

This work has been supported by the European Commision under grant OMI ASPIRE No. 20451

as a case study. Section 2 describes previous work on hardware compilation which was used as a basis for this project, and Section 3 describes the Aspire design flow. Conclusions are drawn in Section 4.

2 Background

The Aspire design flow is based on manual partitioning of system specifications in C++ into a hardware and software part. An imperative programming language, Handel-C, was designed to make this process easy to use for a programmer. This language implements a subset of C that is reasonable for hardware compilation augmented with language constructs that are essential for hardware implementations. It provides the normal structures of conditionals, loops, and assignment together with parallel composition, and CSP-like commands for communications.

One feature of Handel-C which is of particular note is the simple timing semantics. The description is at the register transfer level, and each primitive statement takes exactly one clock cycle to execute. This gives control of the scheduling and allocation of the resources to the programmer, rather than to the compiler. Features of the compiler are:

- Compilation into Xilinx XNF or structural VHDL netlists.
- Simulation. Two simulators are available. One is a simple gate level simulator. The other generates code for an abstract machine which interprets gate-level netlists. This machine is implemented as a C function and can be included in other simulators as will be demonstrated in 3.2.
- Estimation. The compiler gives feedback on overall cycle time (in fact combinational logic depth) and area required.

3 The ASPIRE design flow

The design flow developed in the Aspire project (see Figure 1) is shown in Figure 1. In this section it will be explained step by step and accompanied by the development of a case-study to show what kind of design decisions were made. The example that we have chosen is intended to be simple enough to be easily understandable and complex enough to illustrate the main features of

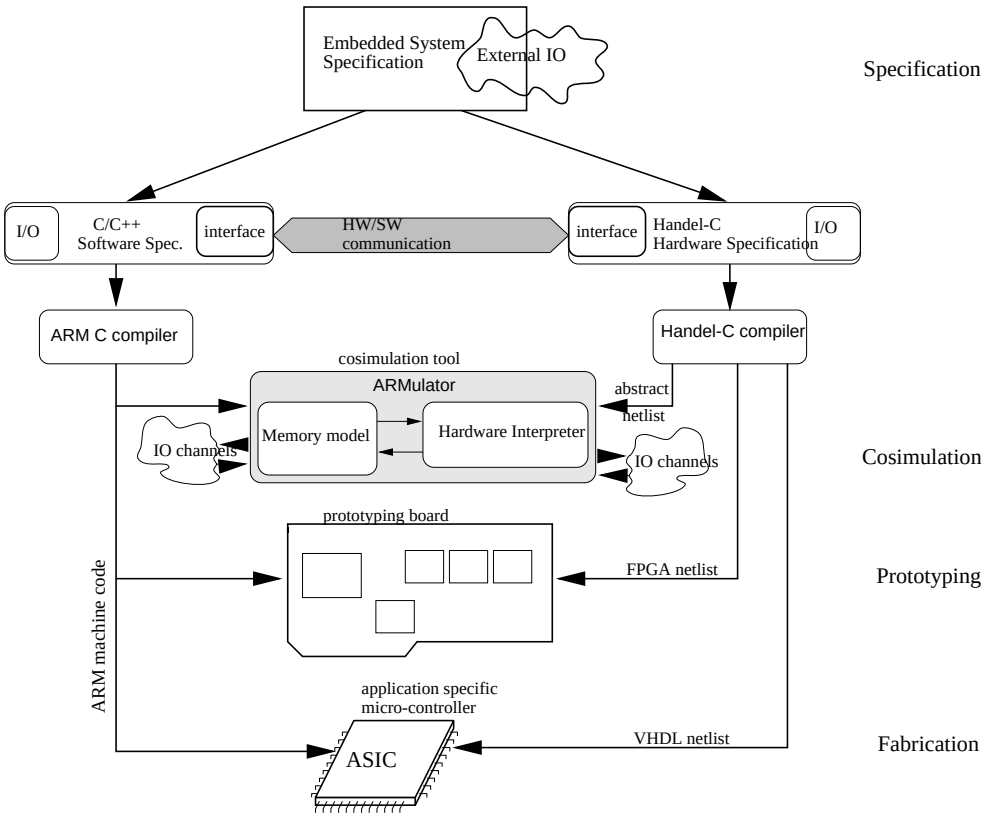

Figure 1: Overview over the Aspire design flow

embedded systems: hardware/software collaboration, communication with external devices and real time requirements. The hardware and software parts of the system are specified concurrently in similar languages together with a hw/sw communication interface. This interface is verified in a cosimulation step. The partitioned design is prototyped on a special prototyping board. This step allows the verification of the external interface to the real environment. If confidence about the correct behaviour of the design is obtained, the Handel-C compiler generates a VHDL netlist for a standard cell ASIC process.

We decided on an example from the field of video processing: a decoder for run-length encoded digital video signals. The system decodes pairs (c, n), where c represents a colour (red green or blue) and n the number of scan-consecutive pixels of that colour, into appropriate video signals which are displayed on a monitor. Figure 2 shows a block diagram of the application.

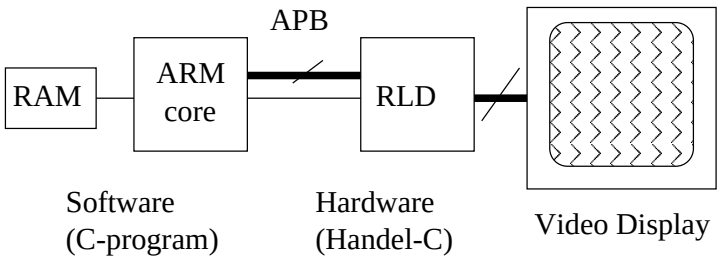

Figure 2: Block diagram of run-length decoder example

3.1 Specification and Partitioning

A programmer might start with an executable specification of the algorithm in a familiar programming language such as C/C++. Partitioning then is done manually by rephrasing time-critical parts of the code in Handel-C. Handel-C is a C-like language which makes the migration of code between the hardware and software domain easy. The display device can, for example, be modelled as a simple X-windows application. The pseudo code of an initial specification of the run-length decoder is shown in Figure 3. The call to `send_and_display_num_times` is intended to drive the video signals appropriately.

```
int main(void){
... /* declarations */
 while(/* for all frames */){
      for (y = 0 ; y < LINENUM ; y++){
        while((colour[i],number[i]) = read_data() != End_of_Line){
          send_and_display_num_times(colour[i],number[i]);
        }
        sync_with_end_of_line();
      }
      sync_with_end_of_frame();
 } /* terminate */
}
```

Figure 3: Pseudo code of run-length decoder specification

The time critical part in Figure 3 is the function `send_and_display_num_times()` because of real-time constraints due to the generation of video signals. We therefore decided to implement this function as a piece of application specific hardware using Handel-C. The result of the partitioning are two separate processes which will run in parallel in the final system. Note

that the system specified in the (sequential) C-program is different from the partitioned system in that we have introduced true parallelism. This enables us to deal with time critical aspects of the video signal generation independent from less time-critical external communication, such as memory accesses. However, we have to design an interface between the hardware and the software process.

A communication model that is conceptually simple to understand is offered by occam-like channel communication [12]. This then has to be implemented with the available resources, such as the peripheral bus of the micro-controller. From the software side the application specific hardware is assigned an address space and accessed like memory. The hardware process is a slave on the system bus and therefore needs to use the data bus to send back acknowledgement signals to the processor. The `send_and_display_num_times` call will be replaced by the `send` call to hardware (see Figure 4). On the hardware side, the interface is a process that monitors the

```
void send(c,n) {
  while(*HW_ADDRESS != ACKNOWLEDGE);
  *HW_ADDRESS = n << COLORBITS | c & COLORMASK;
}
```

Figure 4: Sending values to hardware

bus and sends the value on the data signals to some other process. It is specified in a Handel-C procedure as a non-terminating process (see Figure 5). Handel-C is designed such that every

```
macro proc bus_interface(bus,address,rx,tx,out_chan) {
  while(true){
    if (rx ) {
        var t;
        par{t = bus; tx = 1;};
        out_chan ! t;
        tx = 0;
    } else delay;
  }
}
```

Figure 5: Bus interface of the hardware

assignment statement takes exactly one clock cycle and other control statements add no other

cycles. Therefore it is possible to precisely specify the interface protocol to the system bus. The hardware process itself consists of several communicating processes running in parallel. We have already seen the interface process that monitors the bus. Apart from that there is a small buffer, a decoder process and the main video process. The outline of the Handel-C program is shown in Figure 6.

```
void main(void){
  ...  /* declarations */
  par {  /* main process */
       bus_interface(data_bus,address,busrx,buf,c4); //communication with ARM
       video_out(x,y,eol,eof);                       // video Generation
       fifo4(c4,c0);                                 //fifo buffer
       decoder(n_blank,counter,c0,r_on,g_on,b_on);   //decoder
  }
}
```

Figure 6: Main program specifying the hardware part of the system.

After this partitioning step we need to be sure that the system still implements the intended algorithm. This means that we have to verify the hardware/software interface and check whether the sequence of output signals generated produces the correct video signals. A first step towards this goal is the concurrent simulation of the hardware and the software programs described in the following section.

3.2 HW/SW-Cosimulation

Cosimulation is situated between pure hardware simulation and system prototyping on hardware platforms in that it avoids speed deficiencies of the former and implementation overhead of the latter. Cosimulation using industrial-strength software debugging tools offers additional convenience during debugging. Just as software can be “simulated” by running it on an abstract machine that emulates the target machine, hardware can be simulated on an abstract machine that uses gate-level “commands” like the elements in a hardware netlist. Concurrent simulation of the hardware and software parts of the system is achieved if we get the corresponding abstract machines to cooperate in the same way as their concrete counterparts do in the real system. This can be done either in a processor independent way, using some kind of operating system support for concurrent processes, or using an instruction level emulator for the target processor with the

hardware simulator included. The second approach allows for a more detailed verification of the communication protocol between hardware and software, as it provides accuracy at the level of memory cycles. The Handel-C program is compiled into a bit-level netlist comprising only combinational logic gates, flip-flops, memory elements and external IO connections. The simulation is done by interpreting the elements of this netlist as instructions of an abstract machine. At every clock cycle – the basic unit of time – the state of the network is updated according to the transfer function described by the combinational elements. Input and output can be modelled as "interactive" netlist elements, i.e. a combinational element with either no input wires (=external input) or no output wires (= external output). The actual sources and sinks of these elements can be specified in the Handel-C program and then be resolved by the simulator accordingly. This adds up to a simple but powerful cycle-based RT simulation model which leads to an efficient abstract machine capable of interpreting the netlist.

The simulation of the software part of the specification is done by ARM's configurable debugging tool *ARMulator*[1]. The ARMulator provides support for modelling ARM-based systems either instruction accurate, cycle-accurate or timing accurate. In this paper we will describe the cycle accurate cosimulation strategy, because it is the most appropriate for our purpose of verifying the hardware/software interface.

In the final application specific micro-controller, the individually designed piece of hardware that has been added to the standard microprocessor core, is integrated on the same piece of silicon as the CPU and therefore operated on the basis of the same system clock. We therefore decided on a synchronous simulation of software and hardware. For every memory cycle of the CPU one state update of the hardware simulator is carried out. Hence, simulation of the embedded system is accurate to the level of memory cycles.

ARM's instruction level emulator allows the user to customise the memory model in order to simulate the hardware environment in which the processor eventually resides. This is a very convenient starting point to integrate the hardware simulator. Read and write accesses to the hardware can be done by extending the standard memory access functions such that on an access to the memory space of the application specific hardware the appropriate communication takes place and the relevant signals of the memory bus are driven.

The ARM processor and the RLD/video generator module communicate through a peripheral bus [11]. The specification developed in Section 3.1 could be simulated without any changes and using the header files for the hardware prototype. The hardware/software interface worked

as expected in the simulation and the graphical display of the video output proved to be helpful in gaining confidence that the partitioned system specification describes the correct behaviour. The cosimulation turned out to be very fast: one frame of video, corresponding to approximately 82000 memory cycles was simulated in some 37 seconds on a SPARCstation 5, i.e. some 2200 memory cycles per second could be simulated. In applications like video processing it is very difficult, if not impossible, to tell whether the actual sequence of, for example, video frames is as expected if there is only numerical data available. Simulation may take prohibitively long on event-driven conventional hardware simulators to be of practical value. Prototyping therefore is essential to check the correct behaviour of the overall system.

Cosimulation as the first prototyping step could well establish confidence in the hardware/software communication as very accurate models of these were available. For embedded systems the collaboration with the external world is, however, crucial and can best be verified in a prototyping system which connects to the real environment of the embedded system.

3.3 Rapid Prototyping Environment

Next we will describe the Aspire prototyping board which is the next step toward an ASIC. The system can be verified in its real environment and verified at real-time or nearly real-time speed. The Aspire rapid prototyping board has been designed by Oxford University and built by ARM Ltd. Its purpose is to enable ASIC designs of the form of an ARM processor and custom logic to be quickly prototyped. The major aim of the board was to make it as close to the final silicon as possible. The board is PCI based so it can be fitted to a standard PC to enable easy use of standard high speed IO devices, such as video cards, digitisers and samplers. This was also chosen to avoid a communication bottleneck if real time video applications were to be prototyped. Main board features are

- ARM AT91 Micro Controller [2]
- Two 240pin uncommitted FPGAs on board with local shared SSRAM.
- PCI bridge on board to another FPGA which also contains a DMA controller.
- The board may be used ”stand alone” or fitted in a PCI work station.
- Expansion connectors for extra FPGAs and IO devices

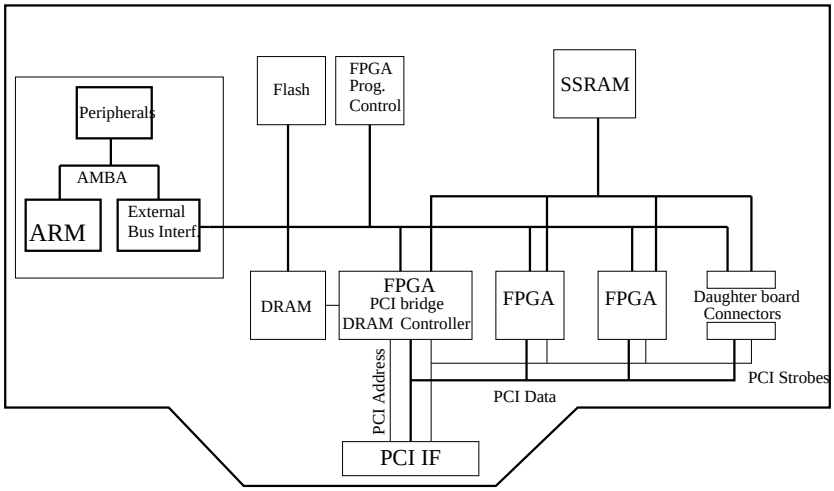

Figure 7: The Aspire ASIC Prototyping Board

The processor is a standard ARM-based micro controller [2]. It has an external bus interface (EBI) to connect to the memory system which also hosts the reconfigurable hardware. The only real difference in using the prototyping board is that the custom logic (in FPGAs) sits on the EBI where as on an ASIC it would normally sit on the peripheral bus. This does not cause a problem as a different set of interface macros is provided which hide these details from the user. The entire board is clocked from the same 33MHz clock. This clock is derived from the PCI bus clock and gives a synchronous system which models the ASIC implementation.

The board has support for two Xilinx 4000E,L,EX, or XL series parts in a 240pin package. More FPGAs may be added via the expansion connector. The FPGAs have a total of 80 interconnects though some are used for other device for example SSRAM. Also the FPGAs have 35 connectors to the EBI and 36 to the PCI interface. Connecting the FPGAs through a bus enables all of the FPGAs to share all the resources. Therefore the architecture is easily extendible and more reconfigurable hardware resources can be provided if necessary.

Synchronous SRAM is used as a local memory to the FPGAs because it is also available as macrocells on the final ASIC. Furthermore it is fast and enables one read per cycle without any complex timing arrangements.

The Handel-C program for the decoder was compiled into Xilinx XNF netlist format and placed and routed using Xilinx Xact software. The video output pins were directly connected to a TTL monitor. The software part of the system was compiled using ARM's C/C++ compiler. The system could be prototyped at real time, i.e. all components were running at the full final clock

speed. Prototyping by cosimulation was, in this case, accurate enough to establish the external communication interface. This gives some confidence in the cosimulation model. The prototyping system is easy to use in the sense that no deeper knowledge about hardware is required to implement the specified system. The compilation process is fast (in the order of a few minutes including place and route) so that it is possible to explore several design variants within a short time.

3.4 On to Application-Specific Silicon

The aim of the ASPIRE tools is to produce an ASIC without the user needing to know too much about ASIC design. The microprocessor core with the bus system is provided as a macrocell by ATMEL-ES2. The application specific hardware is connected to this core after compilation to a netlist of standard cells using the Handel-C compiler. This standard cell netlist is a structural VHDL description which can be used directly for placement and routing by ES2. As the bus interfaces are similar on the prototyping board and on the chip a standardised connection to the processor core can be provided. The user only has to ship the VHDL file together with a list of the external pin names and will get an application specific micro-controller back that can replace most of the prototyping board in the application.

This route has been verified with the RLD example which could be implemented as prototyped. Apart from using different header files which encapsulated the different bus interfaces and different naming conventions no changes to the prototyped code were necessary. The silicon manufacturer's design guidelines were guaranteed by the hardware compiler. A circuit level simulation of the system using standard logic simulators showed that the system is equivalent to the prototyped one and also revealed that cosimulation at this level is not a practical option for applications involving large amounts of data.

4 Conclusion

A video application been successfully designed using the proposed Aspire design flow. Design improvements can be quickly evaluated on the development card, and the final solution can then be moved to an ASIC based on the same ARM micro-controller and custom hardware used on the development card. First silicon using this route will be available early in 1998.

In summary the Aspire design flow guides the designer through the following steps:

- The specification languages for hardware and software are very similar. This makes hardware/software trade-offs easy. The features of Handel-C allow explicit exploitation of parallelism and timing.
- Hardware/software interfaces can be encapsulated in separate processes and verified in a cosimulation step. Cosimulation nicely fills the gap between prototyping and pure hardware simulation as it avoids the implementation efforts of a prototyping setup and is nevertheless significantly faster (an approx. factor 100) than conventional hardware simulation. Using an industrial-strength debugging tool also offers all the convenience a programmer is used to during debugging. The hardware related aspects of software can also be debugged during cosimulation, with the model of the application specific hardware present.
- The whole system can be prototyped in real-time, or near real-time, conditions on the evaluation board. An evaluation of the system performance is obtained and the external IO behaviour is verified. The prototyping system is fast and easy to use and may serve as a stand-alone implementation for low-volume product series.
- Migration from prototyped system to final ASIC can be done with almost no knowledge of conventional ASIC design techniques. This lowers the investment required for an ASIC design and makes the Aspire design flow particularly attractive for small companies.

References

[1] ARM Ltd., Cambridge, U.K. *ARM Software Defelopment Toolkit, Programming Techniques*, version 2.0 edition, June 1995.

[2] ATMEL-ES2. *AT91 Market-leading 16/32-bit Microcontrollers*. ATMEL, 1996. Also at European Microprocessor Seminar.

[3] William D. Bishop and Wayne M. Loucks. A Heterogeneous Environment for Hardware/Software Cosimulation. In *Proceedings of the 30th Annual Simulation Symposium*, pages 14–22, Atlanta, Georgia, April 1997.

[4] R. Bündgen and W. Küchlin. Term rewriting as a tool for hardware and software design. In : [14], pages 19 – 40.

[5] R. Camposano and J. Wilberg. Embedded system design. *J. Design Automation for Embedded Systems*, 1(1):5 – 50, 1996.

[6] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. *Proceedings of the IEEE*, 85(3), March 1997.

[7] DSP Design Group. An overview of the Ptolemy project. Unpublished Memorandum, Dep. of EECS, Univ. California, Berkeley, March 1994.

[8] J. He, I. Page, and J. Bowen. Towards a provably correct hardware implementation of Occam. In G.J. Milne and L. Pierre, editors, *Proc. of IFIP WG10.2 Advanced Research Working Conference, CHARME'93*, volume 683 of *Lecture Notes in Computer Science*, pages 214 – 225. Springer-Verlag, 1993.

[9] R. B. Hughes and G. Musgrave. Design-flow graph partitioning for formal hardware/software codesign. In : [14].

[10] A. Kalavade and E. A. Lee. Hardware/software codesign using Ptolemy: A case study. In : [14], pages 397 – 413.

[11] ARM Ltd. Amba, advanced microcontroller bus architecture specification. available from `http://www.arm.com/`.

[12] I. Page and W. Luk. Compiling occam into field-programmable gate arrays. In Will Moore and Wayne Luk, editors, *FPGAs*. Abingdon EE & CS books, 1991.

[13] Ian Page. Constructing hardware-software systems from a single description. *Journal of VLSI Signal Processing*, 12(1):87 – 107, 1996.

[14] J. Rozenblit and K. Buchenrieder(eds.). *Codesign, Computer-aided Software/Hardware Engineering*. IEEE Press, Piscataway, 1995.

[15] C. A. Valderrama, A. Changuel, P. V. Raghavan, M. Abid, T. Ben Ismail, and A. A. Jerraya. A unified model for co-simulation and co-synthesis of moxed hardware/software systems. In *European Design and Test Conference*, 1995.