A Powerful System Design Methodology Combining OCAPI and Handel-C for Concept Engineering

Klaus Buchenrieder, Andreas Pyttel, Alexander Sedlmeier
Infineon Technologies AG, Corporate Development
D-81730 Munich, Germany
{Klaus.Buchenrieder|Andreas.Pyttel|Alexander.Sedlmeier}@infineon.com

Abstract

In this paper, we present an efficient methodology to validate high performance algorithms and prototype them using reconfigurable hardware. We follow a strict top-down Hardware/Software Codesign paradigm using step-wise refinement techniques. Starting from a performance evaluation on the data-flow level using the OCAPI system, we partition the simulated high-level data-flow description into hardware and software modules. The hardware parts, described in Handel-C, are compiled and mapped to Xilinx Virtex 2000E FPGAs, and the software is executed on a PC processor that hosts the Virtex boards. Hardware/software interfacing and communication between processor and FPGA is established via the PCI bus by shared memory DMA transfers.

This paper presents the methodology and illustrates the method with an example of a channel coder.

1 Introduction

Only few commercial design environments support the design of complex digital hardware/software systems at the highest level, i.e. during the design exploration phase. There is, however, a number of non commercial HW/SW Codesign approaches [4] [5] [6].

Another system is OCAPI [3] [7], developed at the IMEC institute, which supports the development of complex systems-on-chip. OCAPI's design methodology is object-oriented and design issues are solved with rich C++ libraries. Key features of OCAPI include:

- Algorithmic design of complex systems by means of dataflow modeling
- Full support for fixed-point modeling, including automation of refinement
- Incremental refinement design style of behavior and data types
- Architecture design at RT level using FSMD modeling

In our approach, we use OCAPI for system level exploration in combination with the Handel-C [1] based design flow for the prototype implementation.

Our example consists of blocks that implement FIR-filters, upsampler, slicer, coder, and decoder. All modules can be well described with OCAPI, that supports data-driven applications for modelling and simulation. By changing *dfix* parameters (Section 2), the necessary bit-width, fractional part, etc. of the model can be evaluated.

The simulation of designs in OCAPI is easy. However, processing a large number of data is time consuming, but the simulation can be accelerated through mapping the design to an FPGA. In OCAPI, the system-level description can be mapped to VHDL and subsequently synthesized resulting in an FPGA implementation. This approach, however, is tedious and sometimes error-prone.

Since our model is based on C++ descriptions with arithmetic functions, such as sine, cosine and logarithm, which cannot be easily implemented by an FPGA, we have chosen an integrated hardware/software approach.

In that, we implement all functions that require floating-point arithmetic and computationally intensive functions in software, which runs on the PC processor, while other functionality, such as FIR filters, coders, decoders, which work with fixed-point integer arithmetic, and which processes a high data-rate, in hardware (on the FPGA).

A hardware description based on C seems to be ideal, since OCAPI is based on C++, and fosters a combined hardware/software approach. Handel-C has been chosen for a number of reasons:

- The C based description facilitates the mapping of the OCAPI functions to Handel-C
- Handel-C allows to describe functions that run in parallel and provides channels as a powerful mechanisms for synchronized, data-driven communication (between parallel sections of code).
- OCAPI's *dfix* data-type is not available in Handel C. However, a fixed-point data type can be easily realized, because Handel-C supports bitstrings with individual length.
- The communication between PC and FPGA (RC1000 board / PCI bus) is supported on the FPGA side by Handel-C macros, that facilitate access to the interfaces.

- By means of the HW-SW interface and the channel mechanism, FIR filters with loadable coefficients can be implemented (Trigger to reload by PC).

2 Data-Flow Performance Analysis with OCAPI

OCAPI is a design environment and methodology targeted for digital ASIC design. It is based on C++ to capture the system behavior at the system level described with data flow semantics. The object-oriented data-flow representation is incrementally refined to include detailed system architecture for the RT level. In subsequent steps, the RT level is translated to VHDL. Since OCAPI focusses on hardware design, we used OCAPI's data-flow features to evaluate high-level system properties. Initially, there is no timing information and the description expresses the circuit at the highest abstraction level. Each actor is simulated with a separate C++ class, while the communication channels are implemented by a generic queueing class. In that, actor execution is decided by a firing rule, that detects the presence of tokens in the input queues. We didn't follow the OCAPI design flow for the lower abstraction levels. Instead, we smoothly migrated to the Handel-C flow (Section 3.3). This is more suitable for HW/SW designs, because the translation from the data-flow graph representation to a synthesizable FSMD (finite state machine with data path) is easier and more efficient with Handel-C.

Typically in the concept engineering design phase of a product, the specification of algorithms is done with floating point precision. However, the architecture, on which these algorithms are performed is precision-limited and relies on a fixed-point representation. OCAPI provides a fixed-point data type, called *dfix*, for design and analysis. The *dfix* type is an object that carries information about the word length, the quantization behavior, the number of fractional bits, two's complement, rounding and overflow behavior. For example, an error message is generated in case of an overflow. Thus the designer knows when to increase the wordlength.

Corresponding to the *dfix* data type arithmetic operations are defined. When working with fixed-point arithmetic, it is important to have an efficient representation of values for simulation of operations. For this purpose, on simulation level all operations are performed with floating point arithmetic. Only when assigning the value to a signal, the quantization is performed.

Data representation in *dfix* is convenient because Handel-C allows for an almost one-to-one mapping to a bit-string type.

3 Methodology for Implementation

After the system is fully modelled and analyzed with OCAPI, three steps lead to a prototype implementation for a combination of an FPGA (dedicated logic) and a processing element, e.g., microprocessor or DSP. In the first step towards implementation, an OCAPI description is partitioned into executable C and Handel-C. Hereby, computationally intensive functions are mapped to software and data intensive computations are targeted for FPGA. In the second step, interfaces are generated for communication via shared memory and synchronization of hardware and software components. Finally, descriptions of OCAPI targeted for hardware implementation are translated to Handel-C.

3.1 Partitioning OCAPI into Hardware and Software Modules

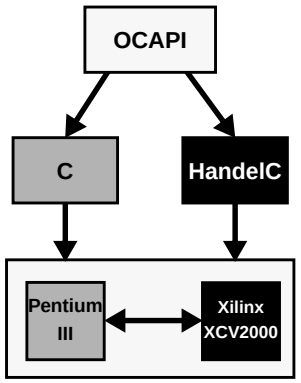

Fig. 1. OCAPI to HW/SW modules

The OCAPI dataflow description contains functions, which perform computationally expensive mathematical operations as required for calculations of filter coefficients using raised root cosine (Section 5) and white noise. These functions cannot be easily mapped onto an FPGA structure and hence are implemented in software.

Other functionality like the filters, which process a high amount of data, are efficiently realized using FPGA for higher execution speed. The partitioning process is driven by the complexity and computational cost of individual modules. Figure 1 provides an overview of the mapping process.

3.2 Interface Generation

For the communication and synchronization of hardware and software, two basic interfaces are supported. The shared memory interface and the synchronized point-to-point interface. Large data transfers are conducted via shared memory. This memory is accessed through an optimized PCI-DMA transfer mechanism from the software side. Block transfer is predominant for reconfigurable hardware. A semaphore like mechanism, supported by a C library, synchronizes the access to the shared memory. The point-to-point interface serves as control port for communication from the software to the hardware domain, and conversely as status-port. Clearly, the implementation is software mastered, so that the configuration of the

FPGA and the calculation of computationally expensive functions can be directly obtained from the initial OCAPI model. Library modules exist for both the shared memory and the point-to-point communication interfaces.

3.3 Mapping OCAPI Models to Handel-C

Handel-C [1] [2] is aimed at compiling high-level algorithms directly into gate-level hardware. It is an extended subset of C syntactically in compliance with conventional ANSI C. It contains parallel constructs to exploit inherent parallelism, variable bitwidth for variables and data-structures. In addition, a channel construct expresses synchronized communications between parallel branches of code.

The *Handel-C environment* consists of a simulator and a compiler targeting reconfigurable FPGA circuitry. The *Handel-C simulator* can display contents and the status of all variables (registers) in a program or design for every clock-cycle. The timing semantics of the simulator is straight forward, in that assignments require exactly one clock-cycle and expressions or control logic consume no cycles to evaluate. One or more cycles are necessary for communications over channels, since those are implemented with a blocking scheme, that waits until sender and receiver become ready. The simulator can visualize the actual timing behavior as number of cycles over simulation steps. For convenience, programmers can also visualize the source code executed at each clock-cycle as well as the program state at any time.

The Handel-C compiler generates EDIF or VHDL files for Xilinx and Altera FPGAs. All features known from conventional C-compilers, including expansion of macro expressions and macro procedures, are available. Here, Handel-C provides a macro concept similar to the *#define* directives used by standard C preprocessors.

OCAPI's queues and the *dfix* representations can be easily described in Handel-C. The OCAPI queuing mechanism is mapped to the Handel-C channels described above, while the implementation of the *dfix* fixed-point representation is supported by the variable bitwidth of integer data.

4 System Design Environment.

The system design environment encompasses the execution environment and the HW/SW system prototyper platform. The system prototyper platform, embedded in the execution environment, consists of a RC1000-PP board [2] (Figure 2) plugged into the PCI slot of a Pentium III PC, running at 800 MHz. The RC1000-PP board hosts a Xilinx Virtex XCV2000E FPGA and four banks of memory with 2 MBytes each. This allows, to prototype, i.e., telecommunication subsystems, with embedded memories or provide sets of local test-stimuli. All four memory banks are accessible by the FPGA and any device on the PCI bus.

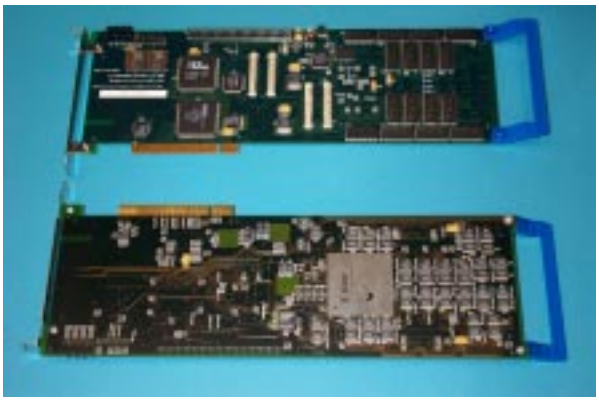

Fig. 2. The RC1000-PP Board

Communication between the reprogrammable Xilinx FPGA and any device connected to the PCI bus is conducted via the PLX PCI9080 PCI controller located on the RC1000-PP board. Different methods of data transfer from the host PC or the environment to the FPGA are available: (1) Bulk transfers of data or test vectors between FPGA and PCI bus are performed through the memory banks Mem0 to Mem3. (2) Streams of bytes are most conveniently communicated through the unidirectional 8-bit control- and status-ports (Figure 3).

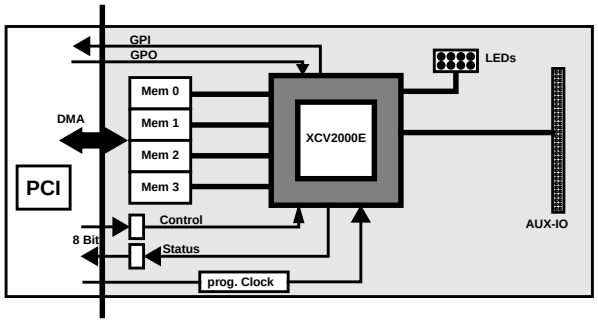

Fig. 3. Schematic view of the RC1000-PP Board

The RC1000-PP board is supported with a macro library that simplifies the process of initializing and talking to the hardware. This library comprises driver functions with the following functionality:

- Initialization and selection of a board
- Handling of FPGA configuration files
- Data transfer between PC and the RC1000-PP board
- Function to help with error checking and debugging

These library functions can be included in a C or C++ program, that runs on the host PC and performs data transfer via the PCI bus.

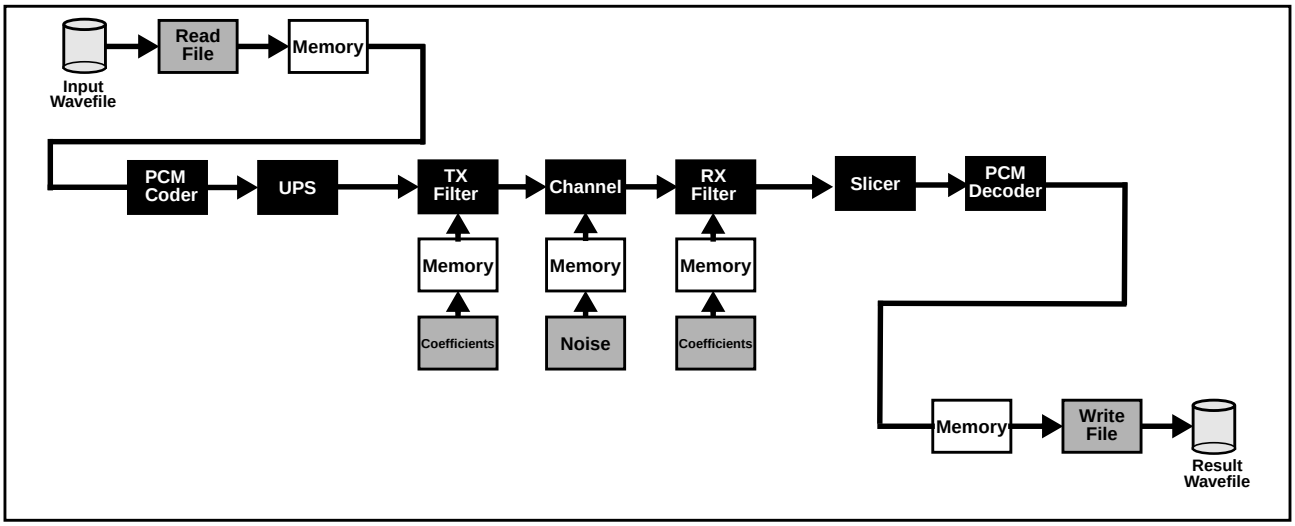

Fig. 4. Block Diagram of the Channel Coding Example

The execution environment provides the connection to the programmed part of the system and the processor, test stimuli, debugging aids and the environment of the embedded system.

5 Signal Transfer Example

In the example, the content of a waveform file is PCM-coded, upsampled, filtered, and decoded as shown in Figure 4. At the end of the processing chain, the result is written back to a new waveform file.

The entire system is realized in hardware (black boxes) on the FPGA and in software (grey boxes) that runs on the host PC. The data exchange between hardware and software modules is established via shared memory (white boxes). The functions to read and write the waveform files, and those, that perform complex mathematical operations, as the calculation of the filter coefficients and the generation of noise, are implemented in C and executed by the host. Other functional blocks, that process a large amount of data, are described in Handel-C and mapped to Xilinx Virtex 2000E FPGA.

The waveform files contain 100000 samples of a 1 kHz sine wave sampled at 44.1 kHz, 16 bit. These data are copied to the RC1000 memory by a C-routine, using a 32-bit memory word for 2 values.

Data are read from the memory and transformed to 16 PCM-symbols by the coder unit. The coder transforms a 2-bit value to one of the four symbol values (-3, -1, 1, 3). The upsampler increases the sample rate by the factor 4, inserting 3 values of 0. These values are filtered by the TX-filter, sent through the channel to add noise, and filtered again by the RX-filter. In the slicer unit, the values are sampled down by 4 and compared with a threshold of 2 or -2. According to the threshold values the received values are mapped to one of the four symbols. These values are written to the memory and saved in a new waveform file. For both filters, the coefficients reside in registers and therefore dynamic changes are possible. This is different from other approaches, which support only fixed coefficients, e.g., Xilinx Coregen.

This channel model allows to simulate different scenarios that occur in real signal processing applications. Both, TX filter and RX filter are implemented as *square root raised cosine filter.* ISI (inter symbol interference) freeness is fulfilled, if the Nyquist criterion is satisfied [8]. Another potential source of bit errors is the quantization noise, that occurs in calculations with too little accuracy.

As described in Section 3.3, parallel constructs to exploit inherent parallelism, allow for easier implementation of filters with higher throughput. In our example, we implemented both filters as sequential and as parallel designs.

6 Results

The combination of OCAPI and Handel-C into a single high-level design environment allows for the modelling and analysis of complex systems with a flexible link to implementation. Additionally, OCAPI enables designers to optimize the number of bits required by the databus in the implementation. OCAPI's *dfix* typing and Handel-C's bitstring casting bridges the gap. In general, OCAPI's submodels such as functional blocks and flow-buffers directly translate to parallel functions and channels in Handel-C.

Using a fixed-point representation of 16 bit with a fractional part of 9 bit, filters with different roll-off factors [8] have been investigated. Using a roll-off factor of 0.3 or higher, no bit errors occurred. A roll-off factor of 0 pro-

duced 8.15 % bit errors. Further investigations have been done with the factor 0.1 and 0.2. The results are described in table 1. To investigate the impact of quantization noise, the fixed-point representation of 16 bit with a 7 bit fractional has been tested. In this case, an amount of 54% bit errors were produced, already at a roll-off factor of 0.3.

Roll-Off	Bit Errors
0	8.15 %
0.1	3.15 %
0.2	0.09 %
0.3	0 %

Table 1: Evaluation of different Roll-Off Factors

The FIR filters are realized with 16 coefficients and 16 bit fixed-point (9 bit fractional).

In our investigation, we implemented the multiply and addition operations of the filters as sequential and parallel circuits.

In the sequential realization, the arithmetic operations (multiply and add) to generate the output value are implemented by a sequential for-loop, where each iteration performs one multiplication and one addition. One more step is needed to clear the accu. The total cycle time depends on the number of taps (2*nooftaps+1) and is 33 for 16 filter-taps.

In the parallel version, all multiplications are done in one cycle. The additions of the intermediate results require another cycle. Here, the total number of cycles is always 3 (together with the step to clear the accu), independent of the number of taps.

	Built-in mults	LUTs	Slices	%	Frequency
Sequential XCV-2000E	None	1806	1440	7	35 MHz
Parallel XCV-2000E	None	9521	6194	32	35 MHz
Sequential XC2V-2000	6 of 56	1161	1139	10	70 MHz
Parallel XC2V-3000	96 of 96	1822	1853	11	60 MHz

Table 2: Design Results

The design results are shown in table 2 (first 2 rows) for a Xilinx Virtex 2000E FPGA. Additionally, the table shows design results for Virtex-2 FPGAs for comparison (row 3 and 4). These FPGAs have dedicated multiplier units and therefore the resulting clock frequency is about twice the value of the Virtex 2000E.

Our example, that processes 100000 samples, takes about 2 seconds of software execution time using the FPGA based hardware/software prototyping platform in the parallel version. In contrast, the high-level simulation in OCAPI needs more than 30 minutes for 100000 samples. Thus we gain a significant speed improvement.

7 Conclusions and Future Steps

In this paper, we illustrated the combination of C++ and Handel-C into a single design methodology for specification, system synthesis and FPGA realization. To demonstrate the usefulness of our approach, we designed and evaluated a PCM-Channel filter system. In the design method presented, we followed a strict top-down Hardware/Software Codesign paradigm. Starting from the OCAPI model, we derived software modules and a Handel-C description for synthesis through a partitioning step. The hardware parts were subsequently compiled and mapped to Xilinx Virtex 2000E FPGAs, while the software parts were executed by the PC processor. As an example, sound files were PCM coded and sent through the filters, monitoring the ISI influence of different roll-off factors. The results obtained clearly show the industrial usefulness of the approach.

Future work will focus on system design with OCAPI-xl, which is a C++ library extension of OCAPI targeted for the system design of heterogeneous hardware/software architectures. OCAPI-xl modeling grants improved representations and thus results in a more accurate partitioning of descriptions into hardware and software modules. Furthermore, system partitioning through extensions of platform-based codesign techniques as found in the Cadence Cierto VCC (Virtual Component Codesign) tool are explored.

8 References

1. M. Bowen: Handel-C Language Reference Manual, Embedded Solutions Ltd., 1999
2. R. Kress, A. Pyttel, A. Sedlmeier: FPGA based Prototyping for Product Definition, FPL 2000, Villach , Austria.
3. S. Vernalde, P. Schaumont, I. Bolsens: An Object Oriented Programming Approach for Hardware Design", IEEE Computer, Society Workshop on VLSI'99, Orlando, April 1999
4. G. DeMicheli, M. Sami: Hardware/Software Co-Design, Kluwer Academic Publishers, 1995, ISBN: 0-7923-3882-0
5. D.D. Gajski, F. Vahid, S. Narayan, J. Gong: Specification and Design of Embedded Systems, Prentice Hall, 1994, ISBN: 0-13-150731-1
6. W. Wolf: Object-oriented co-synthesis of distributed embedded systems, ACM Transactions on Design Automation of Electronic Systems, 1(3), 1996
7. C++ based hardware design of complex digital systems, Course Notes, IMEC Belgium, 2000.
8. E. Herter, W. Lörcher: Nachrichtentechnik. Hanser Verlag München, 2000.