C++ based system design of a 72 Mb/s OFDM transceiver for wireless LAN

Diederik Verkest, Wolfgang Eberle*, Patrick Schaumont,
Bert Gyselinckx, Serge Vernalde

IMEC, Leuven, Belgium
* also Ph.D. student at Katholieke Univ. Leuven

Abstract

This paper describes the system-level design process followed for the implementation of a 72 Mb/s OFDM (Orthogonal Frequency Division Multiplexing) transceiver for 5 GHz wireless LAN (Local Area Network) that is realized in 0.18 μm CMOS technology. The starting point is a high-level specification using the general-purpose programming language C++. By making use of a set of class libraries developed internally at IMEC, architectural trade-offs can be easily explored. The open nature of a C++ based design environment supports the re-use of previously designed building blocks and allows designers to extend the typically supported design-flow eliminating the need for manual generation and correction of synthesis and verification scripts. Automated HDL (Hardware Description Language) code generation from the C++ descriptions creates the link to standard synthesis tools and back-end flows.

Introduction

OFDM forms the physical layer for upcoming broadband wireless LAN standards such as IEEE 802.11a (1) and ETSI Hiperlan/2 (2). A throughput of 72 Mb/s after coding within a 20 MHz bandwidth requires spectrally efficient modulation schemes up to 64-QAM (Quadrature Amplitude Modulation). The design discussed in this paper includes a parameterized interpolating equalizer architecture, clock offset tracking, and a robust programmable acquisition, covering all transmission modes from BPSK (Binary Phase Shift Keying) to 64-QAM extending capabilities of classic OFDM signal processing (3) to fast burst communication in a multi-path indoor environment.

The traditional design process of such a telecom system starts at the level of MATLAB (4) models. Extensive simulations allow to decide on the algorithms and system parameters that meet the system requirements. From this high-level algorithmic specification, HDL coding is started without any intermediate design steps. All decisions related to architecture and implementation are coded immediately in the RT (Register Transfer) level HDL code suited for synthesis. Because of the low simulation speed of the RT-level HDL code, exploration of architectural trade-offs is limited and algorithmic changes imply a long and cumbersome iteration over the MATLAB specification. To bridge this gap from algorithmic specifications to HDL code, IMEC developed OCAPI (5): a C++ based design flow consisting of a set of class libraries that support the designer in making architectural trade-offs starting from an algorithmic data-flow level specification and ending with automatic generation of synthesizable HDL code. The use of object-oriented programming techniques supported by the general purpose programming language C++ (6) allows to abstract away low-level details of the implementation that are irrelevant when deciding upon architectural issues, resulting in a more efficient design process. A process of incremental refinement allows the designer to gradually introduce the implementation detail required for an efficient implementation of the circuit.

The next sections describe details of the architecture and implementation of the OFDM transceiver, details of the C++ based design flow and its support for architectural trade-offs, IP (Intellectual Property) re-use, and HDL code generation. Finally, results of application of this design flow to the OFDM transceiver are discussed and conclusions presented.

OFDM transceiver architecture and design

A. Principles of orthogonal frequency division multiplexing

The OFDM transceiver is intended for a wireless indoor LAN. The indoor propagation channel is frequency selective due to multi-path fading, with dips up to 30 dB, meaning that the transmission in these frequencies will be dramatically degraded. OFDM is a technique that exploits this frequency diversity to improve the communication performance. OFDM allows the transmission of data on different orthogonal carriers, which are created by means of an FFT. The carriers are modulated (PSK or QAM) independently and can be considered as narrow band signals that see a flat fading channel. This makes it easier to estimate the channel characteristics and results in a simpler, frequency domain, equalization in which the coefficients can be updated independently per narrow band.

B. Transmit and receive path

In this section we briefly describe the transmit and receive path of the modem (see (7) for more details). Fig.1 shows the architecture of the transceiver IC containing all digital signal

0-7803-6591-7/01/$10.00 © 2001 IEEE IEEE 2001 CUSTOM INTEGRATED CIRCUITS CONFERENCE

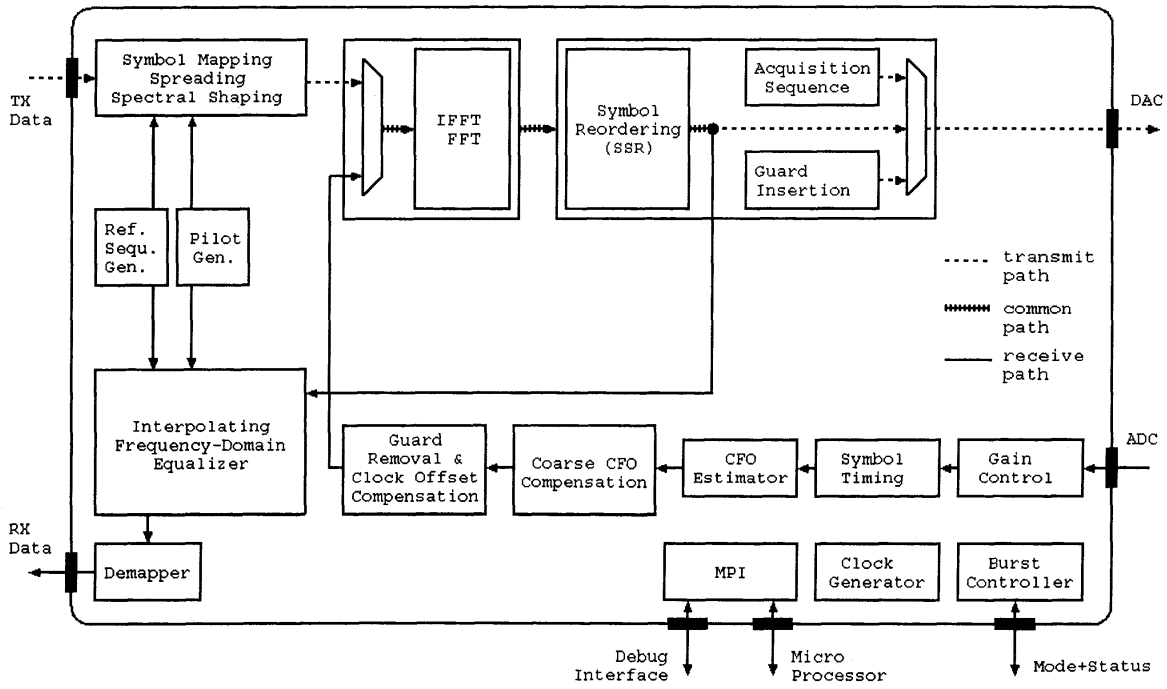

FIGURE 1. Architecture of the OFDM transceiver. The transmit path – from TX data input to DAC output – is shown at the top. The receive path – from ADC input to RX data output – is shown at the bottom. Both share the (I)FFT and symbol reordering blocks.

processing of an OFDM WLAN physical layer except channel (de)coding. It achieves BPSK, QPSK, 16- and 64-QAM transmission up to 72 Mb/s after and 54 Mb/s before coding as required by current standards. The IC has a synchronous FIFO-like host interface and communicates to 8-bit DAC and 10-bit ADC I/Q pairs in the front-end. All data-path and parameter RAM, 78 Kb in total, is integrated on chip.

Depending on the modulation scheme, up to six bits are mapped on the sub-carriers. Reference symbols and/or a rotating BPSK pilot pattern are inserted into the data stream. Spreading allows frequency diversity and a programmable spectral mask performs pulse pre-shaping at OFDM symbol level.

The FFT, symbol reordering, pilot and reference symbol generation make up approximately 14.5% of the hardware and are common to transmit and receive path. The 64-point (I)FFT performs the actual OFDM (de)modulation. It is based on recursive decomposition. The symbol reordering (SSR) rearranges carriers within an OFDM symbol by means of two single-port RAMs and a set of address generators. It transforms bit-reversed (I)FFT data to linear order, in transmit mode it inserts the guard interval and the acquisition preamble, while in receive mode it interleaves the carriers.

In the receive part, symbol-spaced timing synchronization determines the FFT frame start, carrier frequency offset and clock offset. The rest of the circuitry is powered up only on successful detection of the frame start – in listening mode only 2% of the chip is active. The adaptive equalizer sequentially processes the individual carriers, compensating amplitude and phase distortion due to the multi-path channel and removing the phase errors due to group delay variation and carrier frequency offset. A demapper provides hard decisions in case of BPSK or QPSK or, for the other modulation schemes, two times six bits soft output to be used by the subsequent channel decoder.

C. Implementation

Distributed control, based on token flow semantics, facilitates the on-chip communication. The use of token flow greatly simplifies the management of global timing constraints and allows concurrent design of the various blocks. In this chip, distributed control is combined with distributed local clock gating which reduces the average power consumption, simplifies individual block idle state control, and enables receiver sampling rate adaptation. Preamble, symbol and burst structure of the OFDM transceiver are programmable to operate the modem efficiently under varying channel conditions and service requirements.

TABLE 1. IC key figures

Technology	0.18 µm CMOS, 1.8 V core, 3.3 V I/O		
Nominal clock frequency	20 MHz		
Package	160 pin PQFP		
Equivalent gate count	431,000 = 100 %		
active in receive mode	416,000 = 96.5 %		
active in transmit mode	79,000 = 18.0 %		
equalizer	270,000 = 62.6 %		
FFT	42,000 = 9.7 %		
RAMs	78,000 = 18.1 %		
Die size	20.8 mm^2		
Measured power dissipation			
in IEEE mode at 20 MHz	3.3 V I/O	1.8 V core	Total
in transmit mode	156	43	199
in receive mode	66	146	212
in programming mode	35	81	116

The transmit delay is 150 cycles from data input to preamble output. The receive delay is 243 cycles from payload input to data output. The essential figures of the IC design are summarized in Table 1 and Fig. 2 shows the layout of the chip.

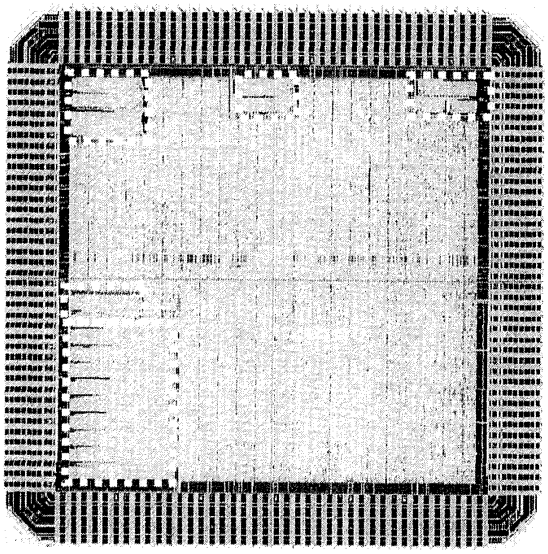

FIGURE 2. Layout of the OFDM transceiver in 0.18µm CMOS. The high-lighted rectangular shapes are embedded memories.

C++ based design flow

The demands of high-speed modems in terms of delay, area, and power dissipation are approaching the technological limits. Therefore, a joint optimization of algorithm and architecture is required to come to a feasible solution. Many of the blocks in the OFDM architecture are flexible to be able to accommodate several (sometimes emerging) standards. Investigation of reasonable parameter ranges and their interdependencies requires a fast high-level simulation model. On the other hand, the architecture must eventually be implemented in silicon and, consequently, a smooth path from the high-level simulation model towards a synthesizable RT-level description is vital. In this process, architectural exploration must be maximally supported.

The OCAPI technology supports the gradual refinement of an object-oriented C++ model starting from behavioral code. Its application to the design of the OFDM transceiver consists of several phases:

1. Behavioral description of the algorithm using a set of class libraries to express data-flow semantics;
2. Design partitioning in which functionality is being grouped in larger entities to be mapped onto single hardware units.
3. Scheduling of the operations inside each entity to get a clock cycle-true description and formal mapping to finite-state machines (FSMs) and signal flow graphs (SFGs) resulting in a register-transfer description;
4. Automatic generation of synthesizable RT-level VHDL code.

Fig. 4 shows the OCAPI design flow as part of the global ASIC design flow used for this design.

In most cases, the initial description of the algorithm will use floating-point signals and operations. For efficient hardware implementation, these operations have to be transformed into fixed-point arithmetic. Because this refinement is orthogonal to the other refinement steps, the signal representation can be chosen freely throughout the refinement. This allows for mixed data type (floating- and fixed-point) simulation throughout the design, which in turn allows fixing the representation of signals very late in the design cycle. In addition, it allows re-use of the same test-benches throughout the design trajectory. The operator-overloading feature of C++ allows the designer to make this floating- to fixed-point exploration with minimal code changes. As illustrated in Fig. 3, only the declarations of the signals need to be changed slightly, no other code changes are required.

```
dfix a(42.2);          dfix a(4,10,2);
dfix b(23.6);          dfix b(2,10,2);
dfix c(0);             dfix c(0,10,2);

c = a + b;             c = a + b;
cout << c;             cout << c;
// prints " 65.8" .    // prints " 65.8" .
```

FIGURE 3. Floating-point to fixed-point refinement is supported very elegantly by exploiting operator-overloading features of C++. Only the signal declarations need to be changed in the left-hand side floating-point code to obtain fixed-point code at the right-hand side, which in this case behaves identically as the floating-point code.

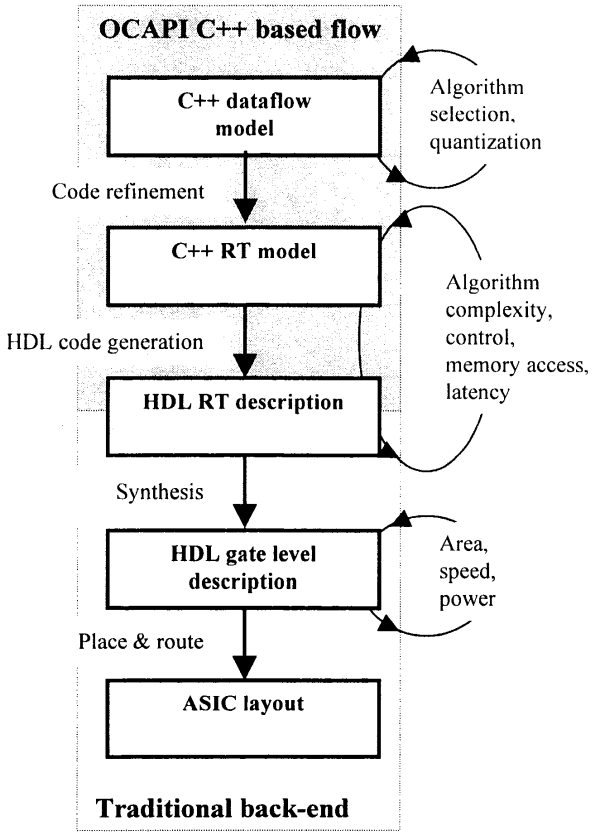

FIGURE 4. C++ based design flow on top of traditional back-end flow. The OCAPI flow covers algorithm exploration down-to register-transfer implementation in an open C++ environment

A. OCAPI flow

In this section, the refinement steps are explained in more detail.

In many single-carrier applications, control is limited to the basic granularity i.e. a data symbol. However, OFDM offers an additional higher level of granularity, the OFDM frame, which is reflected in its architecture. It consists of a low number of control-intensive building blocks. Because the parameters of these blocks are adapted on the fly between transmission bursts and the scheduling of the blocks is parameter dependent, an architecture with centralized control is not feasible. Instead, a data-flow processor architecture is used in which each block has a firing rule that determines the condition for execution of the block. When the firing rule is true, the block is executed without timing constraints or scheduling limitations. OCAPI offers a class library that contains support for the modeling of data-flow semantics. Each block is modeled by instantiating a data-flow object. The data-flow objects are connected using data-flow queue objects. The class library contains a scheduler that will execute the specification by testing the firing rule of each data-flow object. When the firing rule is true, the corresponding block behavior is executed and new tokens are produced.

At this level of abstraction, performance simulations are done to verify system parameters such as bit-error rate, packet-error rate, signal-to-noise ratio, etc. Also when floating-point to fixed-point refinements are done, simulations are performed to verify the implementation loss due to quantization errors.

In the next step, the design is partitioned into entities that correspond to the final hardware units. Blocks with related control functionality are grouped together to reduce interdependencies. This reduces the number of data queues required to communicate control parameters between blocks. This reorganization is accomplished by manual rewriting of the code and verification by simulation. At this level, data-transfer and data-storage optimizations are done. This process is explained in more detail in section C below.

In the third step, the operations inside each block are scheduled to obtain a cycle-true description. Loops are unrolled and local variables are translated to registered variables. Local counter constructs result in state machines with parameter dependent branching. Because the token flow separates functionality of a block from connectivity, we can co-simulate blocks at different abstraction levels. This supports a scheduling process through gradual refinement: individual blocks can be refined separately and independently of each other and simulated with the non-refined blocks (at high abstraction level) resulting in superior simulation speed. The result is described in OCAPI by refining the program into FSMDs (Finite State Machine with Datapath). The operations per branch are grouped in a SFG per state and transition. This level of description is semantically equivalent to an RT-level description.

In the final step, the FSMD-level description is flattened. Each block now consists of one FSM with one SFG assigned to each state-transition pair such that synthesizable VHDL code can be generated automatically from the RT-level C++ code. For each block (FSMD) of the transceiver, a synthesizable RT-VHDL file is created. For the overall OFDM chip, a system net-list is generated that connects the different FSMD blocks. Finally, the C++ test vectors are translated into FSMD-level and system-level test-bench vectors to allow verification of the synthesis results.

The synthesis of the final chip is a fully automated process using elaborate scripting (generated automatically from the C++ code base).

B. *Verification in the OCAPI flow*

During the design process, simulation-based verification is used extensively to track correctness of the design refinements. C++ based simulation is used during the system-level design phases, VHDL-based simulation is used during the synthesis and back-end flow.

The design of test benches is done in C++. The mixed abstraction level simulation capabilities of OCAPI (floating-point/fixed-point and data-flow/FSMD) allow the construction of an end-to-end test-bench in which the architecture of the design under test is gradually refined while keeping the remainder of the test-bench unchanged. The transition from C++ based simulation to HDL based simulation is done by the automatic generation of HDL level test-benches and test vectors that correspond to the C++ test benches. HDL test-benches are generated both at the block and at the system level.

In case of the OFDM design, separate block-level tests were performed to compare the generated RT-level VHDL code with the RT-level C++ code for each block. Automatic cross-checking of the two simulation results is achieved through the use of automatically generated scripts. The generated VHDL description of the complete system was verified by comparing the VHDL simulation results with the system-level data-flow simulation results.

C. Memory exploration

Since multi-carrier systems introduce a second hierarchy of signals (the OFDM symbol/frame), they work effectively on vectors. As such, memory optimizations are much more important than in single carrier systems to obtain efficient implementations in terms of power and size. Memory exploration was performed initially at the C++ data-flow level by introducing storage models and access profiling, and later at C++ RT level for accurate feedback after synthesis using the VHDL code generation. As an example, we discuss the memory selection in the equalizer circuit.

The equalizer coefficient memory is both read and written in each clock cycle. We investigated several trade-offs with respect to RAM partitioning, including a single dual-port RAM (DPRAM) of 256 words, two single-port RAMs (SPRAM) of each 256 words, two single-port RAMs with 128 words, and four single-port RAMS of 64 words. With respect to area and power, the solution with two SPRAMs of 128 words gains at least a factor of 2.25 for the power-area product compared to the other solutions for the same throughput. When more but smaller RAMs are used, the fixed area overhead becomes dominant. DPRAMs have a severe initial power and area penalty.

Memory optimizations were also performed at the level of the data queues between the blocks. Data-flow transformations of the code inside each block were used to modify the production and consumption order of tokens, thereby reducing the required buffer size in between functional blocks. The data-flow simulations are very useful in providing feedback during this exploration as they provide statistics about size and occupancy of the data-flow queues.

D. *Re-use of IP blocks in OCAPI*

The open nature of a C++ based design flow such as OCAPI allows a very efficient re-use of IP blocks. The (I)FFT of the OFDM transceiver was already available as an IP block (because it was developed using a standard, VHDL based, design flow prior to availability of the OCAPI flow). The strategy to integrate such an IP block in the OCAPI models, includes two activities. First, the existing VHDL code is encapsulated to make it compliant to the data-flow semantics of the rest of the system. This is done by manually adding VHDL code to the controller. Secondly, a data-flow abstraction of the IP block is constructed for use in the system-level simulation. To that effect, the internal structure of the FFT implementation is maintained and reflected in the system-level data-flow model of the FFT. The implementation of the FFT controller is abstracted into a data-flow scheduler that organizes the data-flow between the sub-blocks of the system-level FFT model. Finally, an extra controller block is added to hook up the FFT internal scheduler to OCAPI's data-flow scheduler. This process allows for an easy re-use of already synthesized blocks in the OCAPI flow.

A second form of re-use is required when using macro cells such as memories. In that case, the high-level behavior of such a macro cell is encapsulated in a C++ class and special code generation methods are added to the C++ class definition to instantiate the specific macro during VHDL code generation.

In addition, re-use of specifications is supported efficiently by a C++ based design flow. The OFDM transceiver implementation described in this paper is a second-generation system that was implemented starting from the OCAPI specification of an older 80Mb/s OFDM transceiver that was not standard compliant (8). A complete redesign for the 16/64-QAM requirements led to a new bit-true specification. The distributed control scheme allowed efficient incorporation of the 16/64-QAM functionality and extensive re-use of the C++ data-flow and RT-level code base. Table 2 gives an overview of re-use of blocks between the first- and second-generation OFDM transceiver.

E. *Extensions to the design flow*

Apart from efficient support for re-use, the openness of a C++ programming environment allows designers to accommodate deviations from the standard flow easily. The error-prone process of writing synthesis, verification, and test scripts can be completely automated by adding a few extra lines of C++ code

TABLE 2. Re-use of blocks between first- and second-generation OFDM transceiver.

Functional block	Re-use first-generation block
Burst controller	100 %
Clock generator	100 %
Demapper	10 %
Equalizer	0 %
FFT	90 %
Mapper	10 %
Micro-processor interface	100 %
Pilot generator	0 %
Reference symbol generator	50 %
Symbol reordering (SSR)	50 %
Synchronization (CFO)	100 %
Clock offset	10 %
Timing synchronization	100 %
Top level description	0 %

to the descriptions. Iterating through the design then becomes a process of changing the specification code and running through the (automated) code and script generation process again. It is this capability of extending a typically supported design environment with personal productivity tools that allows for major productivity gains. Because of the object-oriented nature of the OCAPI environment all meaningful design concepts (FSM, state, transition, clock, register, etc.) are modeled as objects and hence can be readily manipulated by the designer. To that end, the designer adds extra methods to the already defined classes (much in the same way that one would traditionally write complex scripts to browse through design databases to extract relevant information). These methods have direct access to the objects of interest. This technique has been used, for example, to automatically perform:

- verification tasks such as checking consistency of connections;
- design for test tasks such as introduction of scan chains and connecting scan chains that operate on the same clock
- hierarchy related tasks such as clock-tree routing.

In the context of the OFDM transceiver design, this capability was also use to completely automate the loop from RT-level C++ code changes to VHDL synthesis for the individual blocks. Typically, timing problems are discovered after synthesis of the VHDL code generated by OCAPI. These need to be resolved by (sometimes small) architectural changes in the RT-level C++ code. The complete process of VHDL code generation, simulation of the generated VHDL code, comparison of simulation results with the original RT-level C++ simulations, and generation of the required scripts for synthesis has been automated such that the complete loop from

TABLE 3. Statistics related to use of OCAPI for OFDM design

C++ data-flow	11,000 lines
C++ architecture	18,067 lines
C++ system test-bench	1,599 lines (+11,000)
RT VHDL code	50,884 lines
Gate level VHDL	465,828 lines
Extensions to OCAPI	9,741 lines of C++

architectural change in the RT-level C++ code, to synthesis-ready VHDL code for the block takes only minutes.

F. *Statistics related to the use of OCAPI*

Table 3 gives an overview of the complexity of the different descriptions made for the OFDM transceiver design using the OCAPI flow.

The difference in line counts between the semantically equivalent RT-level C++ code and RT-level VHDL code is due to the abstraction mechanism offered by OCAPI. The class libraries used to represent a design at the RT-level C++ code encapsulate the concept of FSMDs (FSM with Data-path) in a very concise manner. In contrast to HDL descriptions, where the complete behavior of the FSMD has to be expressed using the event-driven simulation semantics of the HDL, OCAPI provides for a declarative type of description.

Because the system model is used for an end-to-end simulation, the OFDM transceiver is instantiated twice in the model. One of the two instances, however, is not refined towards an implementation and should be counted as part of the test-bench. Hence the extra 11,000 lines in the row corresponding to the test-bench in Table 3.

Conclusions

In this paper, we presented the design process of an OFDM transceiver for wireless LAN. A C++ based design flow, OCAPI, was used to make a behavioral data-flow model of the system and gradually refine it to a C++ RT-level description from which synthesizable VHDL code was generated automatically. This C++ based design flow bridges the gap between MATLAB-like specifications and traditional hardware design entry at VHDL or VERILOG level. It allows a designer to efficiently make architectural trade-offs, it supports re-use and incremental design, and the openness of a C++ environment allows the designer to extend the standard design flow with personal productivity tools.

Apart from two generations of OFDM transceivers, several other ASICs have been designed with the OCAPI flow such as an arithmetic coder of a Wavelet compression system (9), a HFC cable modem (10), and a flexible up-down converter (11). Some of these designs were implemented as data-flow processors much like the OFDM transceiver presented in the

current paper. Others were implemented as VLIW architectures containing a centralized controller orchestrating a number of data-paths. In addition, systems containing embedded software, such as an ADSL modem, have been modeled by extensions of the OCAPI environment (12).

We are currently extending the OCAPI environment to cope with complete embedded systems containing functionality that needs to be partially implemented in hardware and partially in software. The extended environment, OCAPI-xl, supports a unified specification model that is unbiased towards a software or hardware implementation and supports a refinement path from the unified specification to both hardware and software implementation. OCAPI-xl is currently being used to design the MAC layer of the OFDM transceiver.

Acknowledgements

The authors would like to thank the following people who contributed to the design and test of the chip and the development of the OCAPI design environment: Veerle Derudder, Liesbet Van der Perre, Geert Vanwijnsberghe, Mario Vergara, Luc Deneire, Radim Cmar, and Luc Rijnders. We gratefully acknowledge the support of National Semiconductor and the Flemish IWT for the design and production of the chip and the development of the design environment.

References

(1) IEEE 802.11a "Supplement to Part 11: WLAN MAC and PHY specification".

(2) ETSI TS 101 475 "BRAN; Hiperlan Type 2; Physical layer".

(3) C. Mandl, M. Bacher, G. Krampl, F. Kuttner, "0.35 µm COFDM receiver chip for DBV-T", *Proceedings of the International Solid-State Circuits Conference*, pp. 76-77, February 2000.

(4) MATLAB, http://www.mathworks.com/

(5) P. Schaumont, S. Vernalde, L. Rijnders, M. Engels, I.Bolsens, "A design environment for the design of complex high-speedASICs", *Proceedings of the 35th Design Automation Conference*, pp. 315-320, June 1998.

(6) D. Verkest, J. Kunkel, F. Schirrmeister, "System level design using C++", *Proceedings of the Design, Automation and Test in Europe Conference*, pp. 74-81, March 2000.

(7) W. Eberle, et al., "A digital 80 Mb/s OFDM transceiver IC for wireless LAN in the 5 GHz band", *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 74-75, February 2000.

(8) W. Eberle, et al., "A digital 72 Mb/s OFDM 64-QAM OFDM transceiver for 5 GHz wireless LAN in 0.18 µm CMOS", *Proceedings of the IEEE International Solid-State Circuits Conference*, February 2001, in press.

(9) B. Vanhoof, M. Peon, G. Lafruit, J. Bormans, M. Engels, I. Bolsens, "A scalable architecture for MPEG-4 embedded zero tree coding", *Proceedings of the Custom Integrated Circuits Conference*, pp.65-68, May 1999.

(10) P. Schaumont, R. Cmar, S. Vernalde, M. Engels, "A 10 Mb/s upstream cable modem with automatic equalization", *Proceedings of the 36th Design Automation Conference*, pp. 337-340, June 1999.

(11) R. Pasko, L. Rijnders, P. Schaumont, S. Vernalde, D. Durackova, "High-performance flexible all-digital quadrature up and down converter chip", *Proceedings of the Custom Integrated Circuits Conference*,. pp. 43-46, May 2000.

(12) D. Desmet, M. Esvelt, P. Avasare, D. Verkest, H. De Man, "Timed executable system specification of an ADSL modem using a C++ based design environment: a case study", *Proceedings of the International Conference on Hardware/SoftwareCodesign, pp. 38-42*, May 1999.