Embedded System Protocol Design Flow based on SDL: From Specification to Hardware/Software Implementation

Daniel Dietterle
IHP microelectronics GmbH, Wireless Communication Systems
PO Box 1466, D-15204 Frankfurt (Oder), Germany
dietterle@ihp-microelectronics.com

ABSTRACT

SDL (Specification and Description Language) is popular for communication protocol design. SDL tools allow simulating and verifying SDL models. In this paper, we show how SDL models can be transformed into hardware/software implementations for embedded systems. Our design flow contains a lightweight operating system integration layer and a cosimulation framework that supports hardware/software partitioning.

The design methodology has been applied to an implementation of the IEEE 802.15.3 MAC protocol. We present results from a prototypical system including a protocol accelerator.

Categories and Subject Descriptors

I.6.3 [**Simulation and Modelling**]: Applications

General Terms

Protocol engineering, embedded systems

Keywords

SDL, IEEE 802.15.3, protocol accelerator

1. INTRODUCTION

Advances in system-on-chip (SoC) design and wireless communication technology enable the development of tiny, battery-powered sensor nodes that can be worn on the human body forming a wireless body area network (BAN). This creates new opportunities for novel applications and products, for instance long-term health monitoring [7], [8]. Application and communication protocol designers are faced with the need to deliver reliable systems within a short development time to be successfully on the market.

This paper presents a design flow for complex embedded system design that addresses

- reliability by the use of the formal language SDL,
- short development time by evolving the original, abstract SDL model towards an efficient software implementation, and
- low-power consumption by the support for hardware / software partitioning.

Our design flow shall give engineers guidelines and a framework for efficient application or protocol design. We advocate the use of application-specific hardware accelerators for processing-intensive and time-critical tasks when their software implementation would require high clock frequencies and, consequently, would increase power consumption.

The proposed design flow has been applied to the medium access control (MAC) protocol design of a wireless communication platform. This platform shall provide wireless connectivity for devices in a personal communication sphere and support multimedia applications.

The capabilities of the platform are to be demonstrated with a medical application. A number of battery-powered sensor nodes measuring various bioparameters, such as heart rate, temperature, or ECG are attached to the human body and form a wireless network. The body area (sensor) network forms the basis for long-term health monitoring of chronically ill patients.

The signals measured by the sensor nodes are locally analyzed (preprocessed) and evaluated within the node or network. Communication with a remote medical center is only initiated in the case of emergency or upon request. An application scenario and the hardware architecture are shown in Fig. 1.

The heart of the communication platform is the LEON2 processor, that runs the protocol and application software. LEON2 is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture [4]. A single-chip implementation of the platform is pursued, integrating digital baseband processing and the RF front-end on chip.

In 2003, the IEEE has standardized a medium access control (MAC) and physical layer specification for high data rate wireless PANs, known as IEEE 802.15.3[1]. We see this standard as a good candidate for future wireless applications and an enabling technology for low-power, wireless multimedia communications for a number of reasons:

The IEEE 802.15.3 MAC protocol offers an isochronous data service, supporting multimedia traffic as well as industrial applications with requirements for guaranteed transmission opportunities. To save energy, devices may go into power-save mode. Synchronized power-save sets ensure

SIMUTOOLS 2008, March 03-07, Marseille, France

DOI 10.4108/ICST.SIMUTOOLS2008.2929

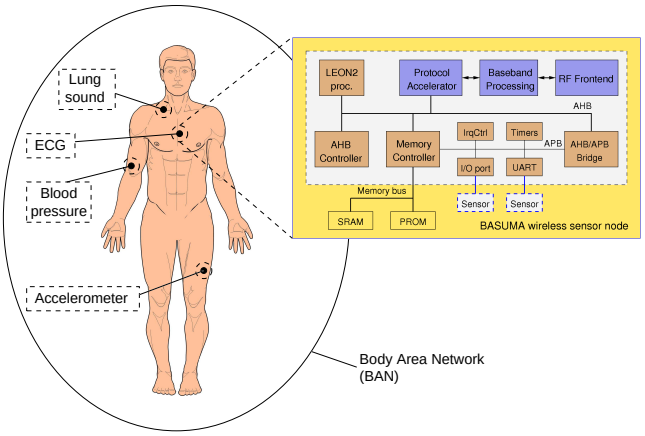

Figure 1: Body area network and hardware architecture of the wireless communication platform [19].

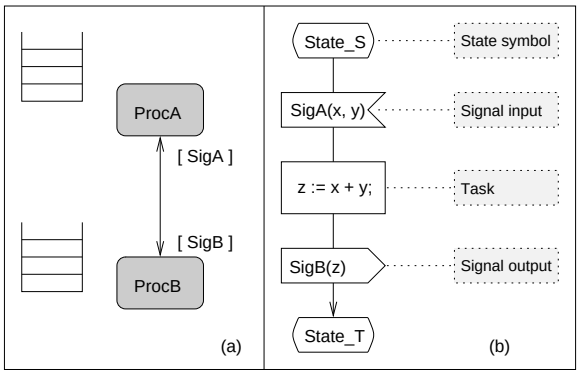

Figure 2: (a) SDL processes with signal route and emphasised input queue. (b) Sample SDL transition.

that all devices in the set wake up at the same time. The used channel access scheme is time-division multiple access (TDMA). This way, the protocol can be easily used with ultra wide band (UWB) transceivers, that do not provide a channel sensing capability. Contention access based on channel sensing is optional.

The specified data rates in the standard range from 11 to 55 Mbit/s, however the same MAC protocol can be used with much higher or lower data rates, as well. While high-rate transceivers typically consume more *power* than low-rate transceivers, still the overall amount of *energy* can be less than for low-rate transmissions provided that the packets are long enough.

We have modeled the MAC protocol in SDL [2]. SDL is a formal language that allows systems to be modeled, simulated, and implemented. It is a popular language for protocol modeling (cf. [9], [10], [11]). Formal verfication can be employed to design reliable systems.

The remainder of this paper is organized as follows. We start with a brief introduction to SDL. In Section 3, our protocol design flow—from an abstract SDL model until hardware / software partitioning and hardware design—is presented. This is followed by results from protocol accelerator design and software implementation. Finally, we present our conclusions.

2. SPECIFICATION AND DESCRIPTION LANGUAGE (SDL)

In a way, SDL can be considered as a programming language with a graphical user interface that offers high-abstraction level programming elements to the designer. The SDL description of system behavior is based on communicating extended finite state machines (CEFSM) that are executed concurrently.

State machines are represented by SDL processes. Processes communicate with each other and the system environment by exchanging asynchronous signals that may carry any number of parameters. SDL also provides timers that can be configured to generate signals at defined points in time. Each process in an SDL system contains a FIFO (First-In-First-Out) input buffer (with infinite space) into which the received signals and timer events are queued. This is shown in Fig. 2 (a).

In Fig. 2 (b), a typical SDL transition is shown. Transitions are triggered by receiving a signal from the signal input queue of the process. During state transition, the SDL process may perform computations, send any number of signals to other processes, set/reset timers, call procedures, and, finally, settle in the next state. Only then, a new signal can be consumed from the head of the input queue.

Telelogic TAU SDL Suite [3] is a tool that allows modeling, simulating, validating, and implementing SDL systems. The CAdvanced code generator translates the SDL model into C code. This C code contains definitions of the required SDL structures such as processes, signals, etc., but also the state machine implementations of the SDL processes in the so-called PAD (process activity description) function. In this function, SDL transitions are triggered depending on the current state and signal input.

The generated code is independent of the underlying operating system (OS). It can be the basis for a system simulation or implementation.

3. MAC PROTOCOL DESIGN FLOW

Our adopted MAC protocol design flow (see Fig. 3) including hardware/software co-design is described in more detail in the following sections. It can be applied not only to communication protocol implementation, but to any embedded systems application development as it puts special emphasis on reliability and efficiency.

3.1 Protocol modeling in SDL

The starting point of our design flow was the IEEE 802.15.3 MAC protocol specification [1].

In a wireless network operating according to the IEEE 802.15.3 standard, there is one piconet coordinator (PNC) and a number of associated devices. The PNC broadcasts beacon frames at regular time intervals. These beacons contain, among others, information about the time of the next beacon and when other devices may access the channel in the same superframe, i.e. in the time until the next beacon.

A channel time allocation list, which is part of the beacon frame, announces the time slots that are reserved exclusively for the devices. Additionally, the PNC may define a contention access period (CAP) following immediately after

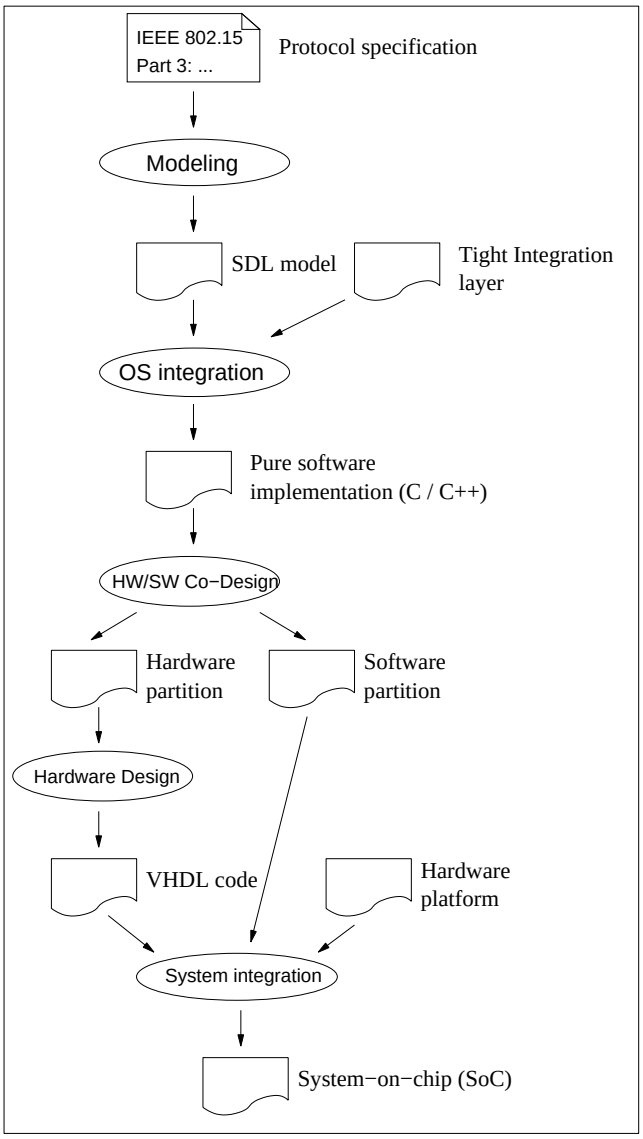

Figure 3: MAC protocol design flow

the beacon. In this CAP, all devices can access the channel using a random backoff procedure. Data communication among the associated devices is peer-to-peer. An example of a piconet and a beacon frame is shown in Fig. 4.

IEEE 802.15.3 MAC frames consist of a header followed by an arbitrary length payload field. Furthermore, a 32-bit frame check sequence (FCS) is calculated over the payload and added to all transmitted frames. The CRC-32 algorithm is used for this purpose.

In most cases, especially in the case of command frames, the sender expects the receiver to acknowledge the correctly received frame immediately, i.e. exactly 10 microseconds after the end of the frame. If an acknowledgement frame is not received, the sender will retransmit the frame.

We have modeled all the necessary protocol functionality in SDL using Telelogic TAU SDL suite [3]. By extensively simulating the model we could validate the correct behavior of our model.

In our model, the protocol functionality has been divided

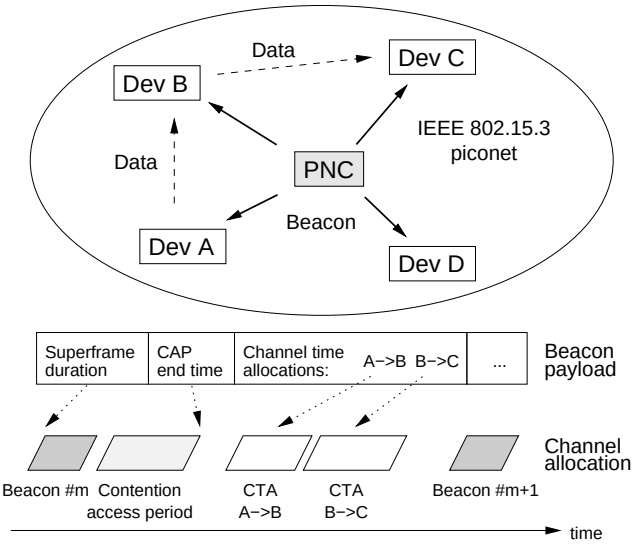

Figure 4: IEEE 802.15.3 network topology and superframe structure (example)

into a number of SDL processes, similar to an object-oriented design approach. Each process is responsible for a well-defined functionality. The SDL processes can be grouped into three conceptual service layers and one management plane, as shown in Fig. 5. Additionally, within each layer we identified those processes that are only needed for devices that are capable to act as PNC. This layering approach and the separation of PNC-specific functionality further enhances the clarity of the model and gives first hints for the hardware/software partitioning.

The lowest service layer is called *Transport Engine*. It provides the ability to receive and transmit service data units (SDUs) to the upper service layers. This means that any upper-layer process does not have to deal with the exact timing of transmission and reception, retransmission, fragmentation and reassembly, and so on. The timers in this layer require an accuracy of 1 microsecond.

On top of the *Transport Engine*, the *Core Services* are placed. The processes in this layer are responsible for maintaining the piconet operation. The *Synchronization* process, for example, observes the reception of beacons and takes action if the beacon was lost in several consecutive superframes.

The highest service layer contains the management processes. These are, for example, the *StartPiconet*, *Scan*, *AssocServer*, or *AssocClient* processes. Their behavior is defined in the IEEE 802.15.3 standard. Note, that the *Core Services* and *MLME Processes* do not require timers with an accuracy of 1 microsecond, but millisecond timers are sufficient.

A more detailed description of our SDL model can be found in [15].

3.2 Operating system integration

The validated SDL model is the basis for the MAC protocol implementation by an automatic transformation. The effort of re-implementing the protocol in C/C++ would be too high and error-prone compared to an optimization approach where inefficient SDL concepts in the model are replaced by

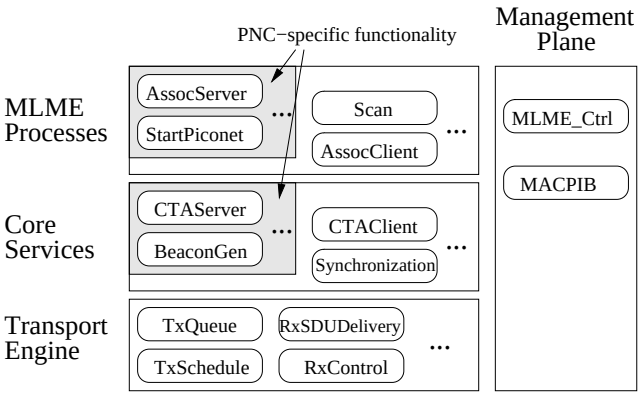

Figure 5: Functional layering of the SDL processes of the MAC protocol model

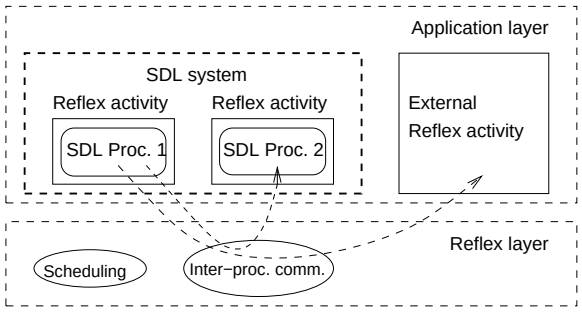

Figure 6: Tight Integration approach.

equivalent functions with less overhead. Additionally, the time to achieve a fully tested implementation is considerably shortened.

The next step is to target the SDL model to an operating system (OS), in our case the Reflex OS [13]. For this purpose, we developed a so-called Tight Integration model for Reflex. This replaces the SDL run-time environment with a tailored, very efficient OS integration layer.

Reflex is a tiny, event-flow oriented OS for deeply embedded systems [12]. Although quite similar to TinyOS [14]—the operating system most often used for wireless sensor nodes—we believe it is better tailored for our system because of its earliest-deadline-first process scheduling strategy. Whereas TinyOS tasks run to completion before any other task is scheduled, time-critical tasks (activities) will interrupt lower-priority activities, in Reflex. Such a behavior is difficult to achieve in TinyOS. We have ported Reflex to the LEON2 processor.

The mapping of SDL concepts to Reflex concepts is straightforward. In our approach, each SDL process is mapped directly to a Reflex activity. Our OS integration layer also contains support for SDL timers. Furthermore, it is possible to completely get rid of dynamic memory management (*malloc/free*) by statically allocating signal pools, from which SDL signals are taken at run-time. The Tight Integration model targeted for the operating system Reflex is shown in Fig. 6.

The required memory space for the operating system Reflex, the integration layer, and a simple SDL system was measured to be about 20 kbytes for a system targeted for

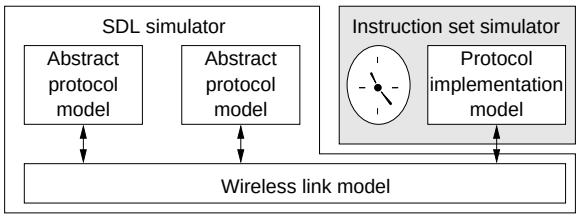

Figure 7: Co-simulation framework [17]

the LEON2 processor. Further details on the Tight Integration model can be found in [16].

3.3 Hardware/software co-design

Some of the MAC protocol functionality underlies tight timing constraints, for instance acknowledgment (ACK) frame transmission has to start exactly 10 microseconds after the end of a received frame. With a pure software implementation this would require a processor clocked at a very high frequency leading to high power consumption. Therefore, some protocol functions need to be realized in hardware. The functionality that is mapped to the hardware partition will then be designed using the hardware description language VHDL. This will be the focus of Section 4.

To identify bottlenecks in the pure software implementation and to estimate the required clock frequency to meet all timing constraints, we performed a profiling of the software. For that purpose, the software was simulated using the LEON2 instruction set simulator (ISS) TSIM [5]. TSIM allows profiling of individual functions. This way, we can identify functions that are most often called or that consume most of the processing time.

In order to see whether the protocol implementation meets its timing requirements, we couple the ISS with the SDL simulator that simulates a body area network on an abstract time basis (cf. Fig. 7).

The SDL simulation as well as the TSIM simulation both have their own simulation time. In order to guarantee semantically correct co-simulation runs, both simulations must be synchronized. In the case of the ISS, time advances at each processed instruction, while our (abstract) SDL simulation does not consume time when transitions are simulated. Only when there are no more active transitions, the simulation time can advance to the next scheduled SDL timer or external event. This means that the instruction set simulation can process as many instructions until the next SDL event is scheduled or the ISS emits an SDL signal to the SDL simulator.

In our framework, the co-simulation is controlled by the SDL simulator from Telelogic. The SDL simulator processes transitions in the SDL model and queries the environment for input signals by calling the function *xInEnv()*. It can also output signals to the environment through the function *xOutEnv()*. Together with the functions *xInitEnv()* and *xCloseEnv()* this is the interface that the tool vendor provides to interact with external software components.

The instruction set simulator TSIM is not only available as a stand-alone application, but also as a library. We use the library version in our approach and link this library to the SDL simulator. It is possible to control the ISS by calling functions provided by the library, for instance to initialize

the simulation, load an executable, proceed for a specified duration, or query the current simulation time.

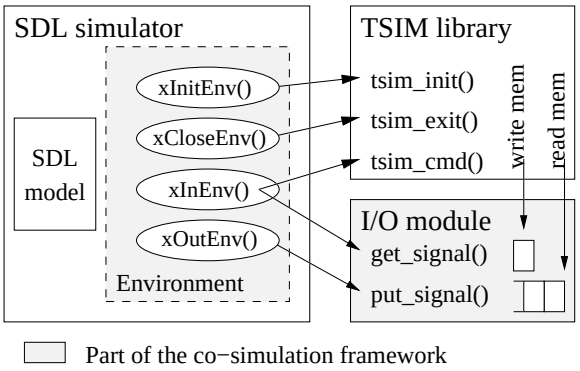

Figure 8: Relationships between the SDL simulator, TSIM library, and the I/O module in the co-simulation framework

At simulation start, the *xInitEnv()* function is called by the SDL simulator. From this function, the ISS is initialized and the application to be simulated by it is loaded. The SDL simulator then simulates all active transitions at timestamp 0. When there are no more transitions, it calls the *xInEnv()* function to check whether there are any external signals as inputs for the SDL model.

As a parameter of the *xInEnv()* function, the timestamp of the next SDL timer that is going to expire is passed. Since there are no active transitions and no other sources of signals that could trigger a transition before the indicated timestamp, it is safe to advance TSIM until it reaches this point in time in its simulation. The call to continue TSIM is made from within *xInEnv()*.

However, it is possible that the external (i.e. TSIM simulated) system sends a signal to the SDL system during that simulation. In this case, TSIM stops immediately and control resumes in the *xInEnv()* function. Here, the current TSIM simulation time is read and the abstract SDL simulation time is advanced to reflect the same point in time. If a signal was sent from the external system, this signal is input into the SDL model — this is the purpose of calling the *xInEnv()* function by the SDL simulator. Otherwise, the SDL time will have reached the expiration time of the next timer and a new transition becomes active.

When all active transitions have been simulated — without advancing the abstract SDL time — *xInEnv()* is called again. Consequently, the ISS will be resumed. With this approach, the SDL simulation time cannnot advance ahead of the time of the instruction set simulator, which might lead to a signal sent from the ISS to the SDL simulation too late.

It is also possible that signals are sent to the TSIM system from the SDL simulation in the course of processing SDL transitions. For that purpose, a signal queue has been implemented that stores these signals and can be read from the application simulated in the ISS. Whenever a signal is written into that queue, an interrupt request is created so that the application — after it has been resumed from *xInEnv()* — will first read these signals and process them, in turn.

The communication interface between TSIM and the SDL simulator has been implemented as an additional I/O module to TSIM. It is mapped into the LEON2 memory space and can thus be accessed from the application simulated by TSIM. This I/O module is also linked to the SDL simulation and provides functions to write into the before-mentioned signal queue and to check whether the external system sent a signal into the SDL simulation.

```
xInitEnv():
    Initialize TSIM
    Load application into TSIM
-----------------------------------------
xCloseEnv():
    Exit TSIM
-----------------------------------------
xInEnv(Time_For_Next_Event):
    TSIM_time = Get current TSIM time
    If TSIM_time < Time_For_Next_Event Then
        Continue TSIM until Time_For_Next_Event
        SDL system time = Get current TSIM time
    Else
        SDL system time = Time_For_Next_Event
    End if
    If there is a signal from TSIM application
        Copy signal from I/O module into
          new SDL signal SignalIn
        Send SignalIn into SDL model
    End if
-----------------------------------------
xOutEnv(SignalOut):
    Put a copy of SignalOut into signal queue
      in I/O module
```

Figure 9: Pseudo code implementation of the environment functions

Figure 8 schematically shows the relationships between the SDL simulator, TSIM library, and the I/O module. The pseudo code implementation of the four environment functions is given in Figure 9. An example simulation run for an SDL system that sets a timer periodically every 5 seconds and sends a signal *SigA* to the external system upon expiration of the timer is illustrated in Figure 10. The external system responds to the received signal with the signal *SigB* exactly one second after it received *SigA*.

Furthermore, with TSIM it is also possible to model the behavior of hardware components that are connected to the LEON2 processor via the on-chip bus. This enables the simulation of the system with protocol tasks mapped to hardware. To achieve this, the corresponding functions are removed from the software model and put into a hardware component. This allows us to study the new timing behavior of the protocol implementation and optimize the hardware/software partitioning until all timing constraints have been met and the required clock frequency is acceptable.

As an outcome of the hardware/software partitioning we have identified the frame reception and transmission procedure, superframe timing control, immediate acknowledgment handling, and parts of the transmission queue to be designed in hardware. In other words, all the low-level, time-critical and processing-intensive tasks of the channel access mechanism have been mapped to the hardware partition. This corresponds well to the lowest service layer, *Transport Engine*, in our SDL model (see Fig. 5). The remaining pro-

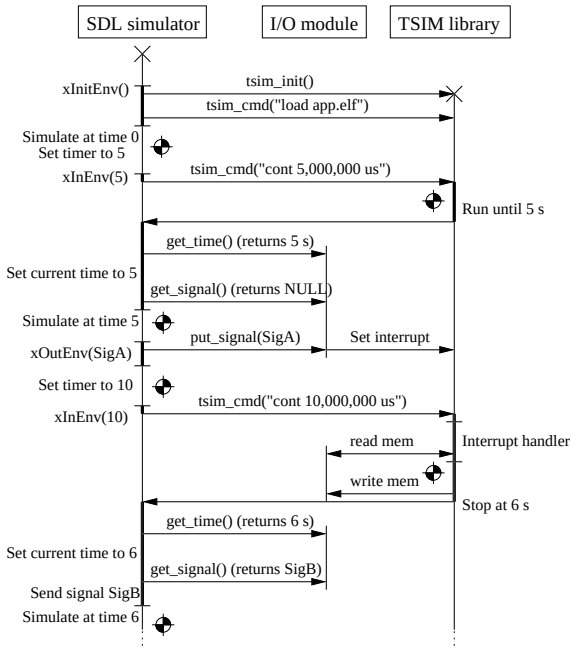

Figure 10: Example co-simulation run

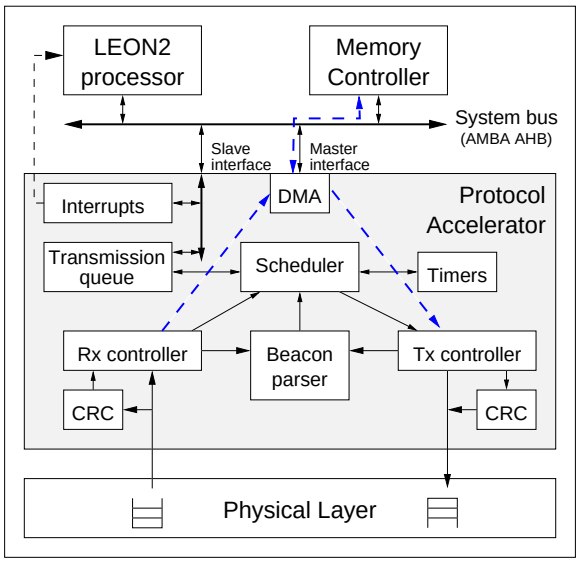

Figure 11: Hardware architecture of the protocol accelerator (direct memory access data path highlighted).

tocol functionality is handled by the LEON2 processor.

3.4 Hardware design

The protocol functionality that has been mapped to hardware in the previous step has been designed in VHDL. An SDL-to-VHDL compiler for rapid prototyping of SDL systems has been reported by Bringmann [23]; we have, however, decided to manually design the algorithms in VHDL for efficiency reasons. The use of hardware accelerators as an addition to a general-purpose processor has been reported previously in the literature (cf. [21], [22]) for wireless MAC protocol implementations.

As we are targeting system-on-chip (SoC) implementations, the protocol accelerator becomes a block of our SoC hardware platform, attached to the on-chip AMBA high performance bus (AHB). Our hardware platform based on the LEON2 processor including the protocol accelerator is shown in Fig. 1.

Additionally, the protocol accelerator has got an interface to the physical layer implementation, such that the payload of received and transmitted frames passes through the accelerator and can be processed on the fly. As the hardware accelerator was designed to be a bus master, it can access the memory to store or read frame data independently of the processor.

Figure 11 shows the main components of the protocol accelerator. The tasks performed by each of the main components are listed below:

- In receive direction, to retrieve frame data from the physical layer byte by byte, perform filtering and CRC check, and to store the data at a given memory location by means of direct memory access (components *Rx controller*, *CRC*, and *DMA*).
- In transmit direction, to retrieve frame data from a memory location, calculate and append the check sum, and to push the data to the physical layer (components *Tx controller*, *CRC*, and *DMA*).
- To signal a successful reception or transmission of a frame to the processor by an interrupt (component *Interrupts*).
- To analyze received and transmitted beacon frames and extract information on channel time allocations (component *Beacon parser*).
- To manage a queue of frames that are to be transmitted and to select an appropriate frame for transmission (component *Transmission queue*).
- At the start of a time slot or following a frame transmission, to query a new frame from the queue and, in the case that the frame must be acknowledged by the receiver, wait for the acknowledgment frame (components *Scheduler* and *Timers*).
- To perform the backoff procedure in the contention access period (components *Scheduler* and *Timers*).
- To send an acknowledgment at the right time upon reception of a frame that needs to be acknowledged (components *Scheduler*, *Timers*, and *Tx controller*).

3.5 Integration and test

The final step in the design flow as shown in Fig. 3 is the integration of the hardware and software implementations and a test of the complete system.

The hardware accelerator registers are accessible through memory mapped I/O, a special memory region is reserved for the accelerator. Additionally, interrupts are used to signal events from the protocol accelerator to the processor. More details on the protocol accelerator design and its interface to software are described in [18].

Table 1: FPGA resources used by the MAC protocol system.

Resources	LEON2 system		Difference
	Original	With accel.	
4 input LUTs	11,582	24,034	12,452
Occupied slices	6,828	14,365	7,537
Block RAMs	20	22	2
Equiv. gate count	1,427,060	1,681,651	254,591

We have fabricated the LEON2 processor system including the protocol accelerator in a 0.25 μm CMOS technology. Prior to the tape out of this chip, we have tested the complete digital system on two FPGA boards connected by wires.

4. DESIGN RESULTS

First, we have implemented the LEON2 system and protocol accelerator on a Xilinx Virtex-II FPGA. This FPGA is the core of the GR-CPCI-XC2V hardware development board [6]. The development board also provides RAM, external interfaces, and is clocked at 40 MHz. We have successfully tested the complete MAC protocol implementation, i.e. the protocol software running on the LEON2 processor and the protocol accelerator, by connecting two such boards with wires. This emulates a network of two devices. The wires couple the boards below the MAC layer, data symbols are transferred serially at a rate of 20 Mbit/s.

Table 1 shows the usage of FPGA resources of the same LEON2-based system with and without the protocol accelerator. No effort has been made so far to optimize the hardware design, these are results from a prototypical implementation. Our synthesis results for a 0.25 μm CMOS technology show that the protocol accelerator occupies an area of 1.8 mm^2. The estimated power consumption of the complete chip is 15 mW/MHz [20].

In the software part, dynamic memory allocation has been completely avoided. There are statically allocated pools, from which SDL signals are taken. By limiting the number of signals sent into SDL model from the environment—the application program or protocol accelerator interrupt handlers—an upper bound for the number of signals that can be allocated by the SDL processes could be determined and the sizes of the signal pools have been chosen accordingly.

The overall memory requirements for the software implementation are roughly 160 kbytes in program memory (*text* segment) and 50 kbytes of RAM (*data* and *bss* segments). These relatively high numbers are caused by the complex protocol behavior, including functions necessary for acting as PNC, and the 32-bit target processor. Although some effort has been spent on reducing the memory consumption, there is still plenty of room for optimization. This has not been the major focus so far, rather to develop a first working system. The share of the operating system and SDL tight integration layer is less than 20 kbytes of program memory.

5. CONCLUSION

We have presented a methodology for the design of complex embedded applications and communication protocols. An SDL model, which can be simulated and formally verified, is the first step in the design flow. We use this initial model throughout the design process; it is subject to optimizations, integration with an operating system, hardware/software partitioning, and, finally, serves as software implementation.

We have developed a cosimulation framework that supports hardware/software partitioning by coupling an SDL simulation with an instruction set simulator. The effects of mapping software tasks to hardware can be measured. Application-specific hardware accelerators relax the timing constraints and take processing-intensive tasks from the processor.

The proposed methodology has been applied to the design of an IEEE 802.15.3 wireless MAC protocol hardware/software system. It has been influenced by practical experiences made during this development process. The successful test of this complex system lets it appear to be suitable for the design of other wireless, embedded system applications that need to be reliable and low-power.

6. ACKNOWLEDGMENT

This work was partly funded by the Federal Ministry of Economics and Technology (BMWi) of Germany under grant no. 01 MT 306.

7. REFERENCES

[1] IEEE Standard 802, "Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks," 2003.

[2] ITU-T, "ITU-T Recommendation Z.100. SDL: Specification and Description Language," 1999.

[3] Telelogic AB. (2004). Telelogic Tau SDL Suite [Online] Available: http://www.telelogic.com/products/tau/sdl

[4] Gaisler Research AB. (2006). LEON2 Processor [Online]. Available: http://www.gaisler.com

[5] Gaisler Research AB, "TSIM Simulator User's Manual," 2006.

[6] Pender Electronic Design GmbH, "GR-CPCI-XC2V Development Board User Manual," 2005.

[7] E. Jovanov, A. Milenkovic, C. Otto, and P. C. de Groen, "A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation," in *Journal of Neuroengineering and Rehabilitation*, 2(1):6, 2005.

[8] R. Bults, K. Wac, A. Van Halteren, D. Konstantas, V. Jones, and I. Widya, "Body Area Networks for Ambulant Patient Monitoring Over Next Generation Public Wireless Networks," in *Proc. 3rd IST Mobile and Wireless Communications Summit*, 2004.

[9] C. Drosos, M. Zayadine, and D. Metafas, "Embedded real-time communication protocol development using SDL for ARM microprocessor," in *Dedicated Systems Magazine*, Q1 2001, pp 37–43.

[10] M. Hännikäinen, J. Knuutila, T. Hämäläinen, and J. Saarinen, "Using SDL for Implementing a Wireless Medium Access Control Protocol," in *IEEE International Symposium on Multimedia Software Engineering (MSE 2000)*, 2000, pp 229–236.

[11] E. Grass, K. Tittelbach-Helmrich, U. Jagdhold, A. Troya, G. Lippert, O. Krüger, J. Lehmann, K. Maharatna, K. Dombrowski, N. Fiebig, R. Kraemer, and P. Mähönen, "On the Single-Chip Implementation of a Hiperlan/2 and IEEE 802.11a Capable Modem," in *IEEE Personal Communications*, vol. 8, no. 6, December 2001, pp. 48–57.

[12] K. Walther, R. Hemmerling, and J. Nolte, "Generic Trigger Variables and Event Flow Wrappers in Reflex," in *ECOOP — Workshop on Programming Languages and Operating Systems*, 2004.

[13] J. Nolte, "Reflex - Realtime Event FLow EXecutive," Available from http://www-bs.informatik.tu-cottbus.de/38.html?&L=2, 2006.

[14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.

[15] D. Dietterle, I. Bababanskaja, K. Dombrowski, and R. Kraemer, "High-Level Behavioral SDL Model for the IEEE 802.15.3 MAC Protocol," in *Proc. of the 2nd International Conference on Wired/Wireless Internet Communications (WWIC)*, P. Langendörfer, M. Liu, I. Matta, and V. Tsaoussidis Ed. Lecture Notes in Computer Science, Vol. 2957. Springer-Verlag, Berlin Heidelberg New York, 2004, pp. 165–176.

[16] G. Wagenknecht, D. Dietterle, J.-P. Ebert, and R. Kraemer, "Transforming Protocol Specifications for Wireless Sensor Networks into Efficient Embedded System Implementations," in *Proc. Third European Workshop on Wireless Sensor Networks (EWSN 2006)*, Lecture Notes in Computer Science, Vol. 3868. Springer-Verlag, Berlin Heidelberg New York, 2006, pp. 228–243.

[17] D. Dietterle, J.-P. Ebert, G. Wagenknecht, and R. Kraemer, "A Wireless Communication Platform for Long-Term Health Monitoring," in *Proc. PerCom Workshops 2006*, 2006, pp. 474–478.

[18] D. Dietterle, J.-P. Ebert, and R. Kraemer, "A hardware accelerated implementation of the IEEE 802.15.3 MAC protocol," *13th European Wireless Conference*, submitted for publication.

[19] IHP GmbH. (2006). BASUMA - Body Area System for Ubiquitous Multimedia Applications [Online]. Available: http://www.basuma.de

[20] Z. Stamenković, D. Dietterle, G. Panić, W. Bocer, G. Schoof, J.-P. Ebert, "MAC Processor for BASUMA Wireless Body Area Network," in J. G. Delgado-Frias (ed.), *Proc. 5th IASTED International Conference on Circuits, Signals and Systems*, 2007.

[21] T. H. Meng, B. McFarland, D. Su, and J. Thomson, "Design and implementation of an all-CMOS 802.11a wireless LAN chipset," IEEE Commun. Mag., vol. 41, no. 8, Aug. 2003, pp. 160–168.

[22] M. Haroud, L. Blazević, and A. Biere, "HW accelerated ultra wide band MAC protocol using SDL and SystemC," in *Proc. IEEE Radio and Wireless Conference (RAWCON'04)*, IEEE, 2004.

[23] O. Bringmann, A. Muth, F. Slomka, W. Rosenstiel, G. Färber, and R. Hofmann, "Mixed Abstraction Level Hardware Synthesis from SDL for Rapid Prototyping," in *Proc. 10th IEEE International Workshop on Rapid System Prototyping (RSPŠ1999)*, 1999, pp. 114–119.