SDL/Virtual Prototype Co-design for Rapid Architectural Exploration of a Mobile Phone Platform

Shadi Traboulsi, Felix Bruns, Anas Showk, David Szczesny, Sebastian Hessel, Elizabeth Gonzalez, and Attila Bilgic

Institute for Integrated Systems, Ruhr-University of Bochum
D-44780 Bochum, Germany
{shadi.traboulsi, felix.bruns, anas.showk, david.szczesny, sebastian.hessel, elizabeth.gonzalez, attila.bilgic}@is.rub.de

Abstract. In this paper we present a new hardware/software co-design methodology for embedded systems, where software components written in Specification and Description Language (SDL) execute on a soft-model of a hardware platform, a so called Virtual Prototype (VP). The proposed approach enables fast exploration of different hardware and software design options at high level of abstraction in order to make early system design decisions. We prove our approach by considering the Long Term Evolution (LTE) communication stack as a use case for the architectural exploration of our mobile terminal. The open source L4/Fiasco microkernel is deployed as a Real-Time OS to run the modem application represented by the LTE SDL-modelled protocol stack. We profile and analyze the system performance by measuring average and maximum packet processing times under various hardware and software conditions. Thereby, we are able to rapidly obtain an efficient design point that provides 80 % packet processing speedup against other unoptimized implementations while meeting the required timing constraints and maintaining a good balance between area and power consumption.

Key words: hardware/software co-design, rapid system prototyping, design-space exploration, mobile terminal, SDL.

1 Introduction

The complexity of embedded systems is increasing over time due to the integration of more and more functionalities into a single chip, which is supported by the advances in fabrication technology. Nowadays, these systems are made up of heterogeneous architectures consisting of a broad range of IP modules like embedded processors, accelerator blocks, interface modules, a memory subsystem, and a communication infrastructure through which these blocks can interact for the exchange of data and synchronization.

Embedded systems like mobile phones have serious power constraints because of their limited battery life time. Moreover, these systems often have real-time characteristics. For instance, communication devices have to process a certain number of packets per second in order to guarantee no degradation in the system's efficiency and quality of service. In addition, reducing the chip area is an important aspect in chip design due to its impact on chip cost. The metrics introduced above (power, performance and area) influence the selection of an appropriate system architecture based on given constraints and their trade-off figures. System constraints, together with the degree of system's complexity, determine the effort needed to optimize hardware and software components of the system.

To be able to define system bottlenecks at an early stage of system development, describing hardware and software at high level of abstraction is mandatory. SystemC, a recent hardware modelling language, has gained a lot of attention for describing hardware components at high level of granularity for the purpose of design exploration [1]. On the software side, formal description techniques such as the Specification and Description Language (SDL) provide simplicity and modularity needed to cope with complex modern software applications.

In this paper we present a new methodology that bridges software and hardware development as both impact the fulfilment of system constraints. The proposed methodology combines modular and abstract software design using SDL with system-level hardware design based on virtual prototyping (VP). This allows for optimizing and customizing the whole system at a high level of abstraction, where simulations are faster and the potential for optimization is larger compared to low level detailed implementations.

The rest of the paper is organized as follows. In the following section we give an overview about co-design and modelling methods from literature. Section 3 describes our SDL/VP co-design approach. An abstract design of a mobile phone system with its VP and software stack is demonstrated in Sects. 4 and 5. As a case study, we demonstrate in Sect. 6 the usage of the proposed methodology for architectural exploration of the reference mobile system. Section 7 concludes the paper and gives an outlook on future work.

2 Related Work

There is currently much research being done in the area of Hardware/Software (Hw/Sw) co-design. Many approaches have been proposed that rely on different concepts. Interplay of processes describing a whole system is modelled in [2] using performance networks, where system workload and services of resources are described as event and resource streams, respectively. These two stream types interact in performance components that are connected to a network to analyze resource loads. This method can only be used for initial steps of system partitioning and mapping of tasks into specific resources. It is very abstract and does not provide any mechanism for evaluating the impact of hardware and software variations on system performance.

Several co-design tools are developed throughout research. Cosyma [3] is an environment that enables a C-based description of an embedded system's tasks and their interconnection. It transforms the system behaviour into an Extended Syntax (ES) graph, which is then used for determining the mapping of system tasks into software and hardware resources. Software parts are converted to C code and hardware blocks are generated in a HardwareC language. Estimations for both software and hardware metrics are then obtained by simulating the object code with Register Transfer Level (RTL) hardware models and their synthesized counterparts, respectively. A similar approach is followed in [4], where hardware partitions are memory-mapped with an interface through interrupt lines to the controlling software and estimations are computed in cycles per byte for comparison with predefined cost and performance parameters. Another work in [5] uses Co-design Finite State Machines (CFSM) to describe both hardware and software parts of a system. Iterative evaluation is then performed for the sake of system partitioning. The next step involves conversion of software partitions into C routines and mapping the hardware parts into an abstract description, which is then refined through logic synthesis to represent the final implemented hardware. Another language suitable for co-design is called SpecChart [6]. Development of hardware and software estimators for this language is carried out by researchers. Particularly, software estimation is based on a generic processor model, while hardware estimations are based on several area models [7, 8]. These tools perform system partitioning at very abstract level in a first step, while evaluation is performed at low level in the second step. Hence, they are able to make very accurate estimations. However, they suffer from low simulation speed, which in turn slows down the convergence of finding a suitable system configuration.

Transaction level models (TLM) [9] have the advantages of decoupling functionality from communication between system blocks to meet short simulation times. TLMs are used for different purposes depending on the level of abstraction they are applied at [10]. In [11], TLM is used in a trace-based simulator, where processing times of software routines running on a processor or Central Processing Unit (CPU) are expressed as delay functions and memory reads/writes are mapped to bus latencies. This mechanism only allows for identification of hardware architecture timing bottlenecks without any indication on meeting absolute constraints. Moreover, this method overly abstracts software representation and hence does not account well for impacts from the software side. Ptolemy [12] is a design framework that targets the modelling, simulation and design of embedded systems by considering different models of computation, however, with the main focus on specification and code generation. Click [13] is an approach for specifying packet processing functionalities in an efficient way, however, without providing means for evaluation of their performance on specific system architectures.

In contrast to the above concepts, which either focus on software or hardware modelling, our methodology is based on a medium level of abstraction, where modelling of full software functionality is abstracted using SDL and the description of virtual hardware prototyping models is based on SystemC/C++.

3 The Design Methodology

The design problem of a parameterizable System-on-Chip (SoC) can be formulated as a search problem. As software and hardware are the two main components that build a whole SoC, each design point (d_n) can be defined as a combination of software and hardware parameters reflecting their architectural design variations. As an example, we assume that both hardware and software designs depend on two parameters each, and that each of these parameters has two possible assignments. Therefore, we will have in this case $2^4 = 16$ different design points forming a so called search space (S) as shown in Fig. 1. Notice that out of these points we have only six design points that meet our system's multi-objective functions, and hence they form a solution space (R).

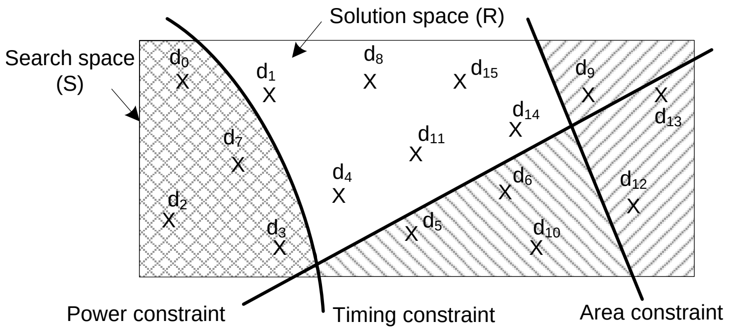


Fig. 1. System design problem viewed as a searching problem.

Searching for design points belonging to the solution space in complicated SoCs cannot be done in an exhaustive manner due to an exponential number of design points. Moreover, estimating performance metrics for each configuration requires costly simulation and analysis of the system. Some evolutionary algorithms are used in [14] to reduce searching complexity. As finding an efficient heuristic is not the focus of our work, we use a simple heuristic that first considers the hardware dimension and then performs software variations only on promising hardware design points: those that are close or meet the system objective functions. This approach does not necessarily lead to an optimal solution, but to a local optimum that fulfils the given constraints.

The proposed flow for SDL/VP co-design is depicted in Fig. 2. Starting with system specifications and intended functionality, an initial partitioning of system tasks between hardware and software can be made. The design of software applications is carried out in SDL with an abstract style of modelling, from which C code is generated. SDL applications are then integrated together with a selected Real-Time Operating System (RTOS) and the required device drivers forming the whole software part of an embedded system. On the other hand, a VP representing the hardware system architecture can be constructed from off-the-shelf building blocks like processors and proprietary modules such as hardware accelerators, in addition to the memory sub-system, on-chip interconnects and Input/Output blocks. The binary of the software component is then co-simulated with the developed VP. During simulation, profiling takes place by monitoring special system events based on selected target figures like software processing time, power consumption and memory usage. Analysis of simulated hardware and software design variations can then be performed by comparing the evaluation results against given system constraints. In case of the constraints are not met, further configurations are applied to hardware and/or software components before starting another iteration to evaluate the new system design point. Another possibility is to migrate some system tasks from software to hardware and vice versa to satisfy performance and area constraints, respectively.

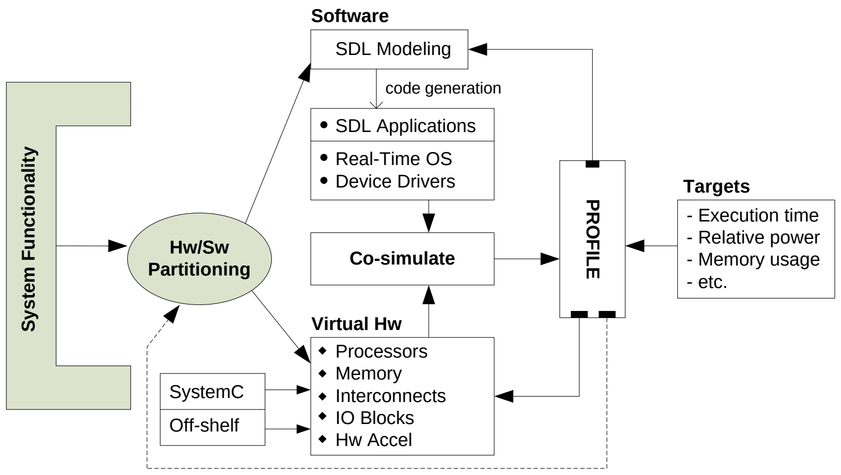


Fig. 2. Embedded system SDL/Virtual Prototype co-design flow.

The VP can be configured by selecting different hardware blocks to achieve a certain data or signal processing task. Thus, it can switch between candidates such as Digital Signal Processors (DSPs), Application Specific Instruction Set Processors (ASIPs) and general purpose processors. CPUs can also be customized by modifying their Arithmetic Logic Unit (ALU), pipeline architecture, cache associativity and size. The memory sub-system can also be investigated for various sizes, hierarchies and read/write latencies. The on-chip communication infrastructure can be configured to use different bus standards and arbitration schemes, or a network-on-chip topology. These configurations will help to evaluate the impact of hardware architecture on the metrics of interest. The design of different blocks in a VP is usually performed at relatively high abstraction levels based on SystemC/C++ which can be gradually refined as soon as the solution space is defined and reduced. This is especially required for area estimations which are dependent on transistor libraries and hence can be precisely determined only at low implementation levels.

Software partitions can be modelled in SDL in terms of communicating processes, where each process is specified as a set of interconnected abstract Finite State Machines (FSM). In this way, we achieve a clear and abstract way of modelling software by focusing on program semantics rather than the language itself. Modification of software at this level is much faster than at lower levels. For instance, different implementations of the algorithm can change the functionality per process and its interaction with other processes. Other optimizations at this level could be reducing SDL process communication overhead and memory accesses, and distributing software tasks among SDL processes in order to exploit locality of data and instructions. SDL modelling might also be useful to cope with software challenges associated with next-generation multi-core embedded systems. For example, the number of software threads and the way they communicate and synchronize, and the degree of parallelism is highly influenced by the number of available SDL processes and their interconnections.

The proposed co-design flow employs combined software and hardware modelling at reasonable abstraction level, allowing the capturing of accurate information for system-level analysis and identification of design points that meet specified objective functions. Moreover, it provides means for Hw/Sw partitioning in modern SoCs as well as for optimization of both software and hardware within the same design flow. This approach also provides implicit verification of software and bridges the gap between software and hardware design methodologies. After finding a suitable design point, hardware refinement towards pure VHDL can be made for final chip tape-out and SDL generated code can be optionally replaced with pure C-code for efficiency considerations. As a use case, we will present in the following sections the hardware and software design of a simple mobile phone system and its architectural exploration based on the proposed methodology.

4 Virtual Hardware Platform

We build a VP of a mobile phone platform using tools from VaST Systems Technology Corporation [15]. This platform is intentionally designed to be based on a multi-core processor, which enables us to perform future investigations about parallelism opportunities for high data rate communication standards. Within the context of this paper, this platform is used as a single-core system, where only one of the available cores is activated.

The architecture of our platform is inspired from the RealView PB11MPCore baseboard provided by ARM [16]. A simplified block diagram of our VP is depicted in Fig. 3. Standard building blocks like processor and interconnects, are taken from the model library provided by the tool vendor. An ARM11 MPCore processor realizes four ARM11 cores, which are representative for state-of-the-art processors in mobile phones [17]. Each of these processors is equipped with an L1 data and instruction cache and a local timer. A snoop control unit is used to

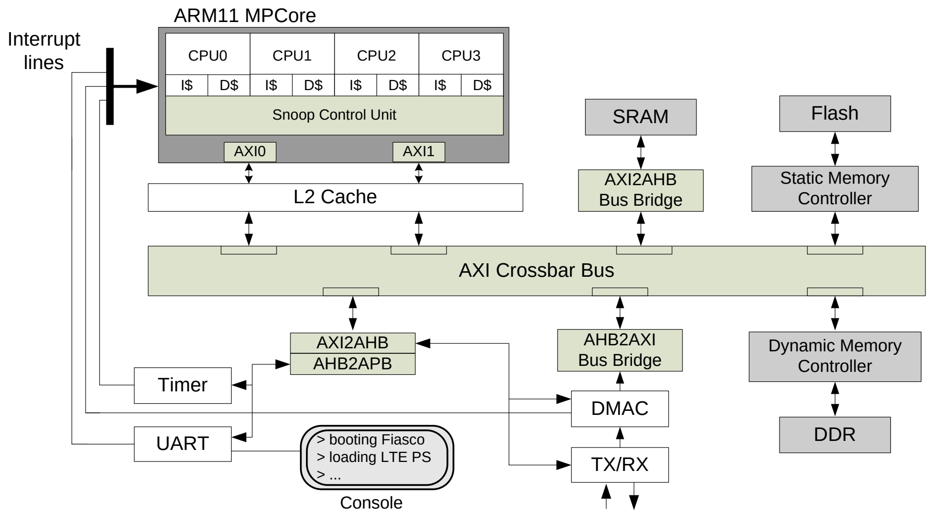


Fig. 3. Hardware architecture of mobile phone platform built in a Virtual Platform.

maintain the coherency between processors' local L1 data caches. A distributed interrupt controller (not shown in Fig. 3) is responsible for dispatching input interrupt lines into the corresponding cores. Each of the four cores features an eight-stage datapath pipeline and a Memory Management Unit (MMU) to support virtual memory. Moreover, the processor subsystem contains a L2 cache to improve the performance of data intensive applications.

The platform employs different kinds of memory blocks. Flash memory is used to hold the device firmware or boot code, which initializes the system : for example, configuring the L2 cache and memory controllers. The firmware then gives control to the operating system which runs from the RAM.

This Double Data Rate Synchronous Dynamic RAM (DDR) is used as an external main memory of the system. The operating system and applications running above are executed from this memory. The read and write latencies of this memory are adjusted according to the state-of-the-art mobile phone platform [18]. In addition, an internal SRAM memory offers low access latencies and hence can be used to store small and time-critical data and/or code. Static and dynamic memory controllers implement the interface protocol required for reading and writing data to Flash and DDR memories, respectively.

To off-load the processor, the Direct Memory Access Controller (DMAC) is used for performing efficient burst transfers from and to the IO blocks. The DMAC has an associated device driver software for configuration. The Transmit/Receive (TX/RX) is an IO block that implements the physical interface for transmitting and receiving data frames, for instance through an Ethernet port. Once a data frame is received, the TX/RX block signals the DMA requesting a data transfer to memory. After that, the DMA copies the data from the TX/RX block to a predefined location in memory. When the copy is completed, the DMA controller raises an interrupt notifying the processor that a radio frame exists in memory and is ready to be fetched and processed. Upon transmission, the processor triggers DMA transfers from memory to the TX/RX block by writing to its configuration register.

System components communicate and exchange data through a high speed crossbar bus. Other busses with different standards and speed rates are connected to the main bus using bus bridges. The bus bridge allows translation between distinct bus protocols. The AMBA High-performance Bus (AHB) is used for internal memory and the DMAC, whereas the Advanced Peripheral Bus (APB) is used to connect low speed peripherals, such as timer and UART. The timer can be used for the scheduling functionality of the operating system, however, in our case the core's internal timer is used for this purpose. User interaction is provided by a console connected to the UART.

5 Software Stack

The stack representing the software component of our mobile phone platform is made up of two parts. The Long Term Evolution (LTE) communication subsystem for decoding and processing of data packets, and the L4/Fiasco microkernel as an RTOS on top of which the protocol stack executes. These two software entities and their details are illustrated in the following subsections.

5.1 LTE Modem Application

The modem application in our case represents the layer 2 (L2) functionality of the LTE protocol stack. It is divided into uplink and downlink processing paths, representing the communication protocol from mobile phone to base station and vice versa, respectively. Both uplink and downlink consist of the Medium Access Control (MAC), the Radio Link Control (RLC), and the Packet Data Convergence Protocol (PDCP) sublayers [19]. As part of abstract software modelling, the SDL model of the modem application implements data plane sublayers in several concurrent processes communicating through signals. Control plane processing is not considered since it does not have much impact on the data processing time. Fig. 4 shows the processing tasks implemented in the SDL model.

When a transport block is received at the mobile terminal, MAC processing starts by applying the Hybrid Automatic Repeat Request (HARQ) process, which retransmits transport blocks for error recovery. When a correct transport block is decoded, header processing starts by decoding the MAC header to extract data like logical channel Identification (LCID) and the Service Data Unit (SDU) length. Afterwards, the downlink shared transport channel is mapped into corresponding control and traffic logical channels, which realize the interface with the RLC sub-layer. Consequently, it demultiplexes MAC SDUs into their corresponding logical channels. In the opposite direction, i.e. uplink processing, inverse operations are performed starting with multiplexing of MAC

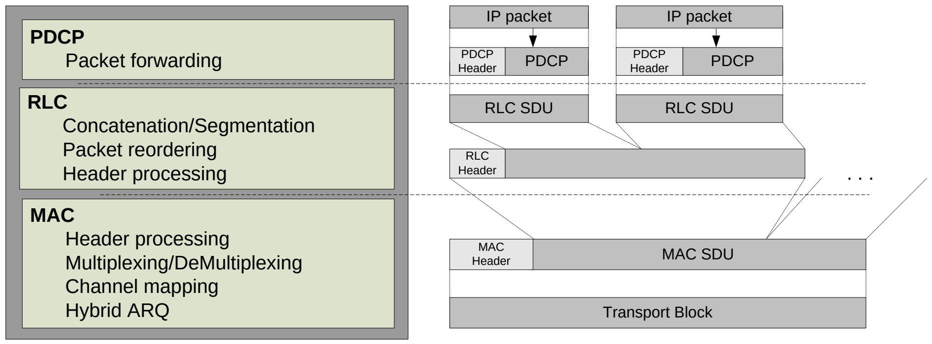


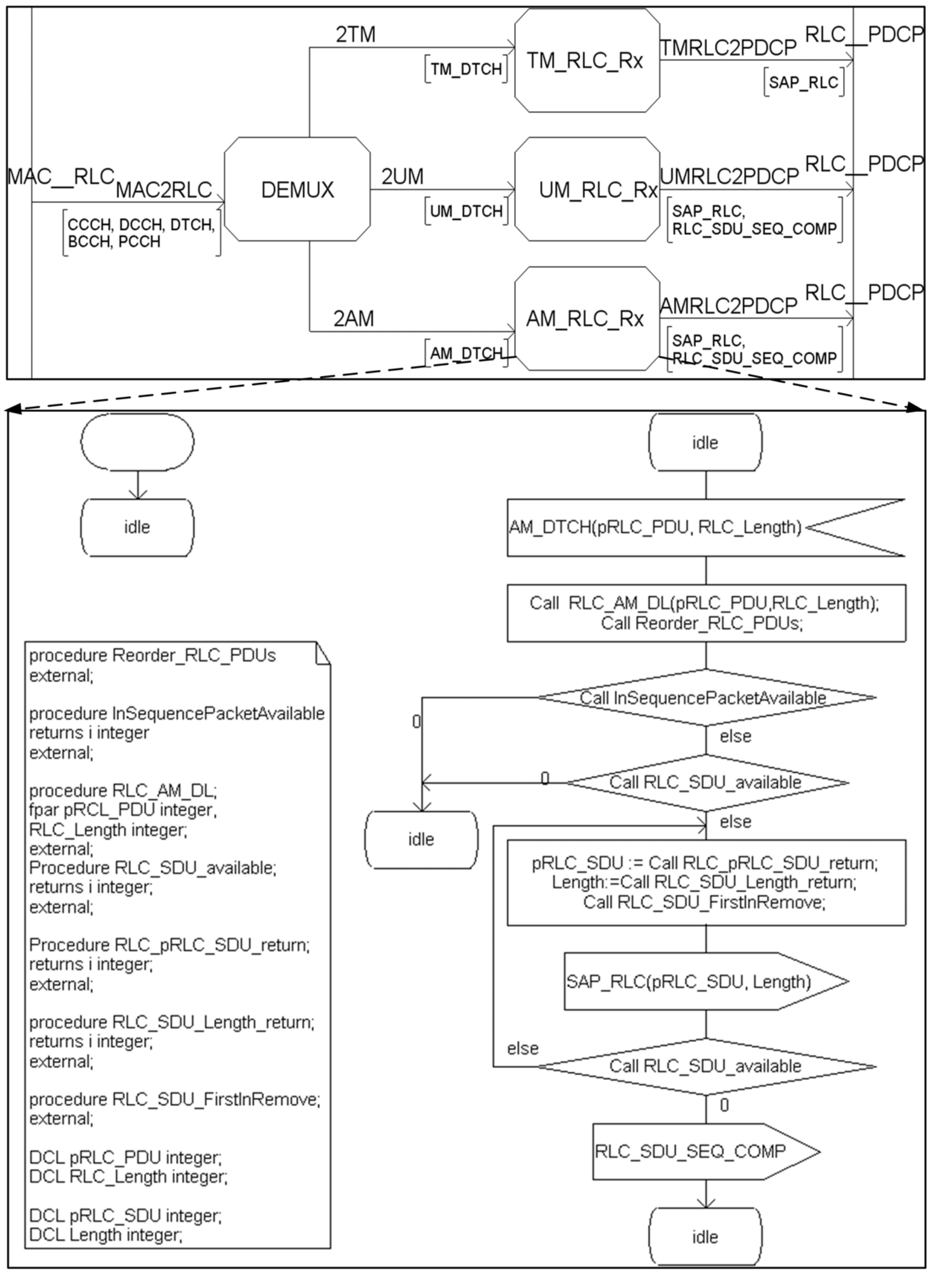
Fig. 4. Implemented functionality of the modem subsystem in SDL.

SDUs and ending with header generation to form the transport block ready for transmission.

Data processing continues at the RLC sub-layer after data is passed from the MAC sublayer. First, the RLC header is decoded and then segmentation processing is applied by unpacking an RLC Protocol Data Unit (PDU) into RLC SDUs, or portion of SDUs. This process depends also on the size of the packets. If the transport block is small, due to bad channel conditions, the RLC SDU may be split among several RLC PDUs. As out-of-order packets might be produced during handover, packet order is corrected in RLC by reordering the packets based on the sequence number carried out in the RLC header. These operations summarize the unacknowledged mode of RLC processing. Fig. 5 illustrates the architecture of the RLC entity in the downlink direction. This entity demultiplexes packet data into different modes of RLC processing implemented in three different processes. The communication of these processes with the rest of the system is performed using SDL messages via the SDL channels. As an example, the Acknowledged Mode process *AM_RLC_Rx* is described using an Extended Finite State Machine (EFM). For space reasons, the LTE SDL system is not fully demonstrated in this paper.

The PDCP sub-layer is implemented as packet forwarding in our investigations, but will be extended in the future enabling complete L2 modelling in SDL. According to LTE specifications, two transport blocks should be processed in 1 ms leading to a processing budget of 0,5 ms per transport block. Performance analysis of LTE protocol processing in [20] shows that 13 % of this time is occupied by MAC and RLC sublayer processing. This means the timing requirements for our architectural exploration in section 6 must be set to 65 μs.

SDL modelling and code generation of the previously described protocol stack functionality is performed using the IBM Rational tool, SDL Suite [21]. By setting the operating system interface option to POSIX, we are able to generate code capable to run on L4/Fiasco microkernel, which supports the same API standard. The size of the SDL kernel itself accounts for 11,000 lines of code and hence will impose some overhead on the generated SDL software system.


2TM
TM_RLC_Rx
TMRLC2PDCP
RLC__PDCP
TM_DTCH
SAP_RLC
MAC__RLC
MAC2RLC
DEMUX
2UM
UM_RLC_Rx
UMRLC2PDCP
RLC__PDCP
UM_DTCH
CCCH, DCCH, DTCH, BCCH, PCCH
SAP_RLC, RLC_SDU_SEQ_COMP
2AM
AM_RLC_Rx
AMRLC2PDCP
RLC__PDCP
AM_DTCH
SAP_RLC, RLC_SDU_SEQ_COMP
idle
idle
AM_DTCH(pRLC_PDU, RLC_Length)
Call RLC_AM_DL(pRLC_PDU,RLC_Length); Call Reorder_RLC_PDUs;
Call InSequencePacketAvailable
0
else
Call RLC_SDU_available
0
else
idle
pRLC_SDU := Call RLC_pRLC_SDU_return; Length:=Call RLC_SDU_Length_return; Call RLC_SDU_FirstInRemove;
SAP_RLC(pRLC_SDU, Length)
else
Call RLC_SDU_available
0
RLC_SDU_SEQ_COMP
idle

```
procedure Reorder_RLC_PDUs
external;

procedure InSequencePacketAvailable
returns i integer
external;

procedure RLC_AM_DL;
fpar pRCL_PDU integer,
RLC_Length integer;
external;
Procedure RLC_SDU_available;
returns i integer;
external;

Procedure RLC_pRLC_SDU_return;
returns i integer;
external;

procedure RLC_SDU_Length_return;
returns i integer;
external;

procedure RLC_SDU_FirstInRemove;
external;

DCL pRLC_PDU integer;
DCL RLC_Length integer;

DCL pRLC_SDU integer;
DCL Length integer;
```

Fig. 5. The RLC downlink block architecture and its Acknowledged Mode process behavior.

5.2 The L4/Fiasco Microkernel

The modem application in our mobile phone system runs on top of L4/Fiasco based operating system [22]. The latter is composed of two layers, the L4/Fiasco microkernel and the L4 runtime environment (L4Re) as shown in Fig. 6. The selection of such a modern operating system which adopts the concept of microkernels is due to the fact that microkernels can act as robust RTOSs. For instance, microkernels aim at running only the most necessary functionality in the processor privileged mode. Hence, it requires smaller code size which reduces complexity and percentage of errors in the privileged mode. Moreover, microkernels offer good isolation characteristics by separating the communication subsystem from untrusted components like freeware applications. In addition, it supports virtualization by allowing the execution of general purpose operating systems like embedded Linux, where mobile applications like calendar and video codec can run, together with a proprietary RTOS responsible for executing the communication protocol in a mobile phone. The services of each layer of the deployed OS will be illustrated in the following.

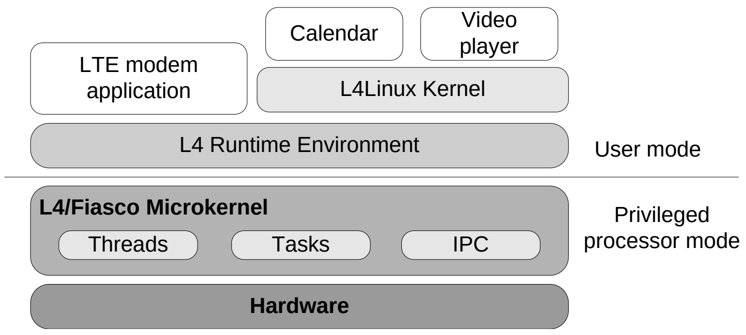


Fig. 6. The L4/Fiasco based operating system.

The L4/Fiasco microkernel is the only component running in processor privileged mode and is responsible for managing the underlying hardware. Based on its nature, it provides a minimal set of mechanisms like tasks, threads, and Interprocess Communication (IPC). Fiasco kernel services are implemented in terms of kernel objects. A task comprises an address space where one or more threads can execute. Multiple threads within a task are scheduled by Fiasco's priority-based and preemptive scheduler. An IPC kernel object provides the basic communication mechanism in L4- based systems and is used mainly for transmitting arbitrary data between threads. On the other hand, the L4Re offers a basic set of abstractions and services, which are useful to implement user- level applications on top of the L4/Fiasco microkernel. It consists of a set of libraries mainly responsible for memory and IO resource management.

In this work, we consider only the SDL model of the protocol stack as an application to execute directly on top of the L4Re and the microkernel, which in turn provides real-time capabilities.

6 Architectural Exploration Case Study

As a case study we apply the proposed methodology for exploring the impact of different hardware and software architectural parameters on system performance. The objective of this exploration is to analyze the mobile phone platform. So, the focus here is to see how the execution time of the protocol stack is influenced by different selected parameters. Particularly, the main goal of this case study is the customization of processor and memory subsystem in order to meet the timing constraint of 65 μs previously derived in subsection 5.1.

The parameters selected for system exploration are core frequency, memory latency, cache size, and the number of threads realized by our SDL model. These parameters have impact on system metrics. As the power of a CMOS-based chip grows linearly with the operating frequency, higher processor frequencies will result in more power consumption. On the other hand, this will improve the speed of our system as the processor will be able to handle more instructions within a fixed period of time, thus shortening the protocol stack execution time. Note that the frequency of a processor is not the only deciding factor for system performance. Actually the latter depends on the nature of the application being executed since the processor has always to communicate with memory. Therefore, it makes sense to investigate the impact of memory response or read/write latency on our objective function. This parameter representing memory throughput has also a linear impact on power consumption. Another important aspect is to find a suitable L1 cache size. Actually, large cache sizes should shorten the packet processing time by exploiting both spatial and temporal locality of program code and packet data. This however happens at the expense of increasing the chip area resulting in a higher fabrication cost. As an architectural software design parameter, we increase the thread density of our protocol stack SDL model. Threads have impact on systems memory and performance as each thread is assigned a dedicated stack in memory and has to be managed by the operating system. However, threads are also useful for exploiting concurrency and parallelism in a system as long as the application allows it. Although we know how these design parameters impact the system, it is still unclear how huge this impact is. This is going to be illustrated within the rest of this section. This study will also allow us to see the trade-off between different metrics, which contributes to making our final design decisions.

Figure 7 shows the profiling setup and demonstrates how the processing times are measured. The whole system is simulated with a cycle-approximate level of accuracy using CoMET from VaST. We run the generated code of the SDL-modelled protocol stack on top of the L4/Fiasco based operating system, which in turn executes on the designed hardware platform. In this study, IP packets' payload is generated in software. The data is then passed to the uplink data plane processing which outputs a valid transport block as it would be received from a base station via the air interface. After that, the downlink part of the stack is triggered to process the transport block before it signals the IP data generation process to start further iterations. In this way we are able to investigate both processing paths at the same time.

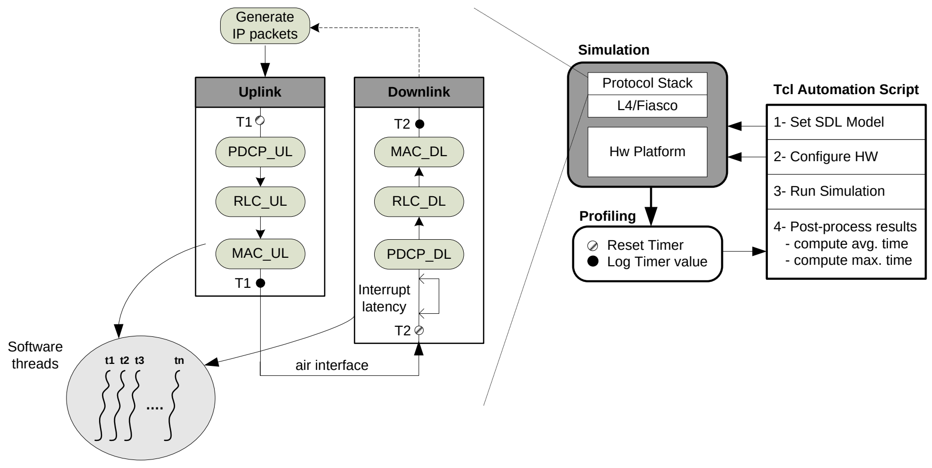


Fig. 7. Simulation and profiling setup for evaluation of several design configurations with respect to the execution time.

As our objective function is to meet the timing requirements with respect to the protocol stack execution time, timer tags are integrated into the stack's generated code enclosing both uplink and downlink processing paths. Timing measurements are performed by waiting for the timer tags during simulation to start and stop corresponding timers leading to the measurements of the uplink and downlink processing times. A Tcl automation script controls the whole evaluation process which starts by selecting the SDL model which adheres to the software design parameter. Afterwards, it adapts the hardware architecture according to the assigned hardware parameter values and runs the simulation where the performance profiling is also made. The evaluation ends by post- processing the recorded timer values. Since different packet processing iterations have different processing times, mainly due to different hardware states, we collect the processing time for hundreds of iterations and post-process them to compute the average and maximum processing time per packet.

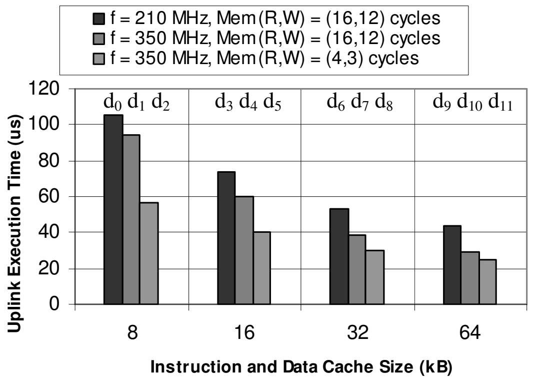
Having four design parameters each with wide range of values will result in a huge number of combinations or design points. To reduce the number of design points, we select parameter values within the range, which is acceptable for embedded systems. For instance, the core frequency can be assigned to two possible values 210 MHz and 350 MHz. However, the (read,write) memory latency parameters are assigned to (16,12) or (4,3) cycles at a reference frequency of

100 MHz. For the cache size, we consider four distinct values (8, 16, 32, and 64 kbytes) which are realistic candidates in a typical mobile device. For the software parameter, we design SDL models with different number of running threads (1, 2, 6, and 17) by distributing the functionality of the MAC and RLC sublayers into a higher number of SDL processes. In this use case, we are not concerned with efficient mapping of SDL processes into concurrent worker threads. This analysis will be carried out later on when dealing with multicore architectures in order to explore and exploit parallelism in the modem application.

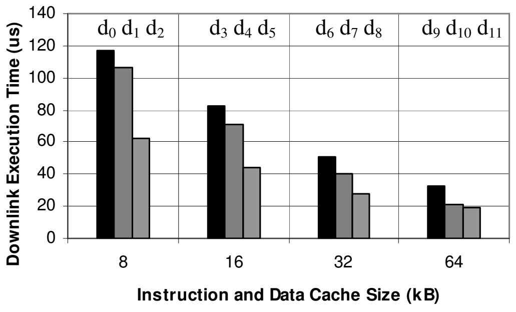
We apply the heuristic of considering design points with hardware variations under stable software conditions. Particularly, we set the number of threads to one and vary all other hardware parameters. As a second step, we vary software parameters only on the set of design points showing good results from the first evaluation step. The uplink and downlink processing times corresponding to design points d_0 to d_{15} considered in the first evaluation step are depicted in Fig. 8. In this figure the design points are grouped according to their cache size.

From the results, we can notice that the achieved processing times are shortened with higher data caches reaching $44\,\mu$s for uplink and $33\,\mu$s for downlink at core frequency of 210 MHz and memory (read,write) latency of (16,12) cycles. By increasing the processing frequency to 350 MHz, the execution time of both uplink and downlink is reduced by 10 % and 35 % at a cache size of 8 kB and 64 kB, respectively. This shows that for small caches with large number of misses, the system performance is limited by the memory response rather than processor speed. This is justified in the third bar where the memory latency is reduced to 25 %, leading to 40 % reduction in processing time at 8 kB cache size, and only 10 % reduction at 16 kB. In plot (c), the maximum execution time per design configuration point is depicted. Design points with a cache size of 32 kB and 64 kB are the only configurations which fulfil or almost fulfil the timing constraints. On the other hand, a small gain of $9\,\mu$s can be observed by doubling the cache size from 32 kB to 64 kB under the same frequency and memory latency. This shows that design points with a 64 kB cache size are not worth to consider due to their area overhead in comparison with the performance gain they can bring. As a result, only design points based on 32 kB cache, i.e. $d_6 - d_8$, are considered in the next evaluation step.

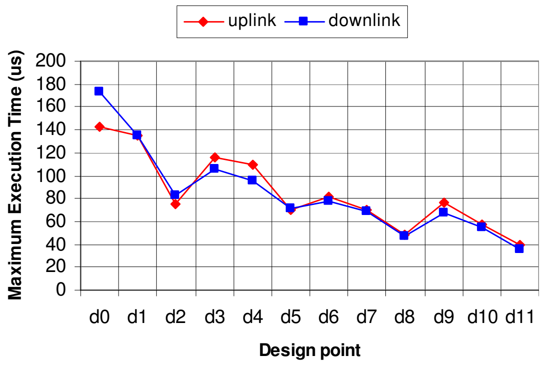
As a second step, we apply different variations of threaded SDL models into design points $d_6 - d_8$. As Fig. 9 shows, the downlink execution time increases by 45 % with configuration of two threads, and much larger with 6 and 17 threads. It is obvious that we will not gain performance with higher number of threads due to thread management overhead without possibility for parallelism since the protocol stack is running on one core. However, we realize that the impact of thread management is quite huge in a single core and this has to do with the overhead of thread management coming from the SDL kernel as well as the L4/Fiasco based operating system. This architectural exploration and analysis allows us to highlight this issue, which should be taken into consideration for further optimization especially when executing the stack on several cores.



(a)



(b)



(c)

Fig. 8. Evaluation of several design points through variation of hardware parameters. Average execution time is shown in subplots (a) and (b), whereas plot (c) shows the maximum execution time of both uplink and downlink processing.

Finally, we select the design point d_7, where the processor with a 32 kB cache operates at 350 MHz and the memory (read,write) latency is (16,12) cycles at 100 MHz. In addition, the single threaded SDL model is selected. This configuration achieves an average execution time which meets the timing constraints and is on average 80 % faster than other design configurations (see Fig. 8). Moreover, it provides a good balance between area and power costs.

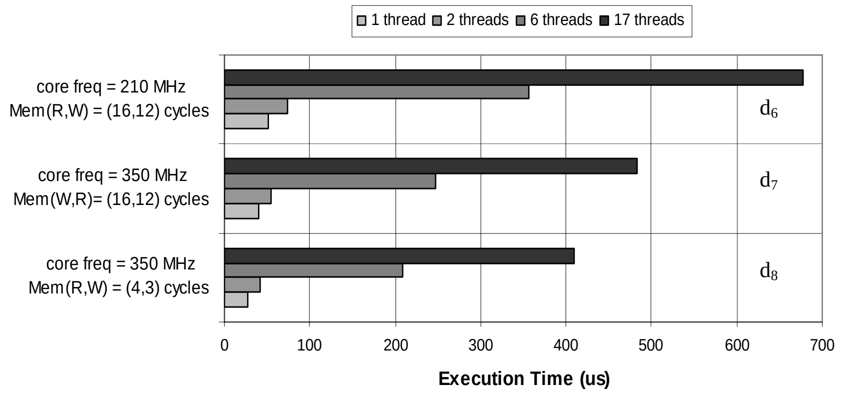


Fig. 9. Evaluation of different SDL threaded models at 32 kB cache size.

7 Conclusion

In this paper we propose a hardware/software co-design flow for embedded systems. Within this flow, software components are abstractly modelled using SDL, while hardware components are emulated in software using the concept of virtual prototyping. This approach allows for fast and early investigations of several design options for both hardware and software due to the low effort and high speed associated with such modelling techniques. To prove our methodology, an architectural exploration of a mobile phone platform is considered. We demonstrate an SDL-modelled LTE protocol stack, an L4/Fiasco based RTOS, and a designed virtual prototype of a mobile terminal. We customize the processor and memory subsystem of the platform by rapidly obtaining a suitable design configuration which meets the required timing constraints and provides 80 % packet processing speedup compared to other unoptimized implementations. In addition, the achieved design parameters provide a balanced power/area consumption tradeoff. For further study, different Hw/Sw partitioning configurations can be applied and evaluated in a similar way. As a future work, we will adapt the software part to make efficient utilization of multi-core architectures and make further investigations about the performance gain, power consumption and scalability of such a multi-core based communication system.

Acknowledgement. The authors acknowledge the excellent cooperation with all partners within the ICT-eMuCo project and the support by the European Commission. Further information is available on the project web-site: http://www.emuco.eu.

References

1. Grötker, T., Liao, S., Martin, G., Swan, S.: System Design with SystemC. Kluwer Academic Publishers, Boston (2002)
2. Thiele, L., Wandeler, E.: Performance Analysis of Distributed Embedded Systems. In: Zurawski, R. (ed.), Embedded Systems Handbook. CRC Press (2005)
3. Ernst, R., Henkel, J., Benner, T.: Hardware-Software Cosynthesis for Microcontrollers. IEEE Design & Test of Computers, 10(4), 64–75.(1993)
4. Thomas, D.E., Adams, J.K., Schmit, H.: A Model and Methodology for Hardware-Software Codesign. IEEE Design & Test of Computers, 10(3), 6–15 (1993)
5. Chiodo, M., Giusto, P., Jurecska, A., Hsieh, H.C., Vincentelli, A.S., Lavagno, L.: Hardware-Software Codesign of Embedded Systems. IEEE Micro, 14(4). 26–36 (1994)
6. Gajski, D.D., Vahid, F.: Specification and Design of Embedded Hardware-Software Systems, IEEE Design & Test of Computers, 12(1), 53–67 (1995)
7. Gong, J., Gajski, D.D., Narayan, S.: Software Estimation using a Generic-Processor Model. In: Proceedings of the 1995 European Conference on Design and Test, pp. 498. IEEE Computer Society, Washington, DC (1995)
8. Vahid, F., Gajski, D.D.: Specification Partitioning for System Design. In: Proceedings of the 29th ACM/IEEE Design Automation Conference, pp 219–224. IEEE Computer Society Press, Los Alamitos (1992)
9. Cai, L., Gajski, D.D.: Transaction Level Modeling: An Overview. http://www.cecs.uci.edu/conference_proceedings/isss_2003/cai_transaction.pdf
10. Donlin, A.: Transaction Level Modeling: Flows and Use Models. In: Hardware/Software Codesign and System Synthesis, 2004,. CODES + ISSS 2004, pp. 75–80. ACM, New York (2004)
11. Wild, T., Herkersdorf, A., Ohlendorf, R.: Performance Evaluation for System-on-Chip Architectures using Trace-based Transaction Level Simulation. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 248–253. European Design and Automation Association Leuven, Belgium (2006)
12. Buck, J., Ha, S., Lee, E.A., Messerschmitt, D.G.:Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. http://ptolemy.eecs.berkeley.edu/publications/papers/94/JEurSim/JEurSim.pdf
13. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, F.M.: The Click Modular Router. ACM Trans. on Computer Systems, 18(3), 263–297 (2000)
14. Palesi, M.: Multi-Objective Design Space Exploration using Genetic Algorithms. In: Proceedings of the Tenth International Symposium on Hardware/Software Codesign, CODES '02. ACM, New York (2002)
15. The VaST Systems Technology Corporation. http://www.vastsystems.com
16. RealView Platform Baseboard for the ARM11 MPCore. http://www.arm.com/products/DevTools/PB11MPCore.html

17. Silven, O., Jyrkkä, K.: Observations On Power-Efficiency Trends in Mobile Communication Devices. EURASIP Journal on Embedded Systems (2007)
18. Hessel, S., Bruns, F., Bilgic, A., Lackorzynski, A., Härtig, H., Hausner, J.: Acceleration of the L4/Fiasco Microkernel Using Scratchpad Memory, International Workshop on Virtualization in Mobile Computing, MobiVirt 2008. ACM, New York (2008)
19. Evolved Universal Terrestrial Radio Access (E-UTRA), 3GPP Specifications: Rel8, Dec. 2008. http://www.3gpp.org
20. Szczesny, D., Showk, A., Hessel, S., Hildebrand, U., Frascolla, V., Bilgic, A.: Performance Analysis of LTE Protocol Processing on an ARM based Mobile Platform, accepted for 11th International Symposium on System-on-Chip (SoC 2009), Tampere, Finland, Oct. 2009
21. IBM® Rational® SDL Suite^{TM},
http://www.ibm.com/software/awdtools/sdlsuite/
22. The Fiasco Microkernel, http://os.inf.tu-dresden.de/fiasco