Abstract—This paper describes the design of an automotive traffic sign recognition application. All stages of the design process, starting on system-level with an abstract, pure functional model down to final hardware/software implementations on an FPGA, are shown. The proposed design flow tackles existing bottlenecks of today’s system-level design processes, following an early model-based performance evaluation and analysis strategy, which takes into account hardware, software and real-time operating system aspects. The experiments with the traffic sign recognition application show, that the developed mechanisms are able to identify appropriate system configurations and to provide a seamless link into the underlying implementation flows.

I. INTRODUCTION

The requirements for modern electronic systems are manifold: Designed with complex functionality, these products have to be comfortable, safe, energy-saving, and reliable. Economical requirements like low manufacturing costs, short time-to-market cycles, and high configuration potentials, complement the challenges. Design methodologies for such system-on-chip (SoC) products are mainly characterised by a high level of abstraction for design entry, early model-based verification and performance analysis, an integrated view on the entire system, and a concertated view on hardware and software components [1], [2]. The building blocks for the system’s functionality are primarily standardised intellectual properties (IP). Additionally, these IP components are supplemented by application-specific in-house developed function blocks, usually representing the internal core knowledge of the application. This is true for hardware, but also for software, based on standardised real-time operating systems (RTOS) and in-house developed libraries. The result is a platform that fulfils the requirements mentioned before and supports a mapping of functionality to architecture that fulfils the application’s needs and minimises the implementation and production costs.

In this paper, the design and implementation process of a traffic sign recognition (TSR) application (cf. Figure 1) is shown. This is a typical safety-critical application from the automotive domain with crucial requirements. To solve resulting design challenges, a seamless design flow is applied, enabling system-level design entry and modelling. The underlying methodology is extended to give the designer global overviews over the application as well as deeper insights into crucial characteristics, which are relevant to the topology of a target platform implementation. This encloses a seamless and flexible integration of hardware, application software, and RTOS aspects. These insights are an essential basis for an early performance analysis and an early functional verification. A high-level synthesis stage are integrated in order to map function blocks to application-specific hardware.

The following section introduces the TSR application. This application is well suited for the validation and exploration of design methodologies for RTOS-based multi-core SoCs. Section III describes the TSR-specific performance requirements, crucial for such a safety-critical application. Section IV details the design methodology applied to the TSR application, covering system-level simulation and performance analysis, architecture design including hardware and software, application software development, and application-specific hardware implementation. In Section V functional validation and the results of the implementation on an FPGA target platform will be shown, including a comparison with simulation. Finally, Section VI concludes the work.
II. TRAFFIC SIGN RECOGNITION SYSTEM

A TSR application is part of a modern driver assistance system in cars. It allows to optically recognise traffic signs, e.g., speed limitation signs, while providing motoring information to the driver.

A. Functional Overview

Figure 1 shows the basic functioning of the TSR application. Relevant areas in the video image are recognised before those areas are classified in a second step. The grabbed image frame in Figure 1, taken from a video used as test sequence during the design, shows a traffic sign for a speed limitation of 50 km/h, which is recognised and classified correctly. Another traffic sign for pedestrian/biker way, which is not included in the classifiable coefficient subset yet, is preprocessed for an image enhancement and shape/edge detection, but not classified.

A TSR application generally consists of four main modules: Camera, recognition, classification, and display module. Figure 2 depicts the functional overview. The camera module grabs image frames and generates an image/video stream. Within the recognition module, image preprocessing and enhancement is performed and afterwards a shape/edge detection stage allows to detect circles. These circles can be found in image frames containing a speed limitation sign. These circles are subsequently classified using a template matching algorithm based on a support vector machine [3]. This classification stage allows to detect different speed limitations from “30 km/h”, ”50 km/h”, ”60 km/h”, ”70 km/h”, ”80 km/h”, ”100 km/h”, ”120 km/h”, as well as prohibition signs like ”No passing”, ”No passing for heavy goods vehicle”, ”End of all traffic prohibitions” and ”End of passing prohibition”. Finally, the recognised traffic sign is displayed to the driver.

The initial video/image stream data is based on byte-sized gray-level pixels. These pixels are scaled to floating-point data and the subsequent recognition and classification stages are performed using floating-point arithmetic. The support vector machine algorithms uses a coefficient field with about 800K data items for the classification procedure. The algorithms of the recognition and classification stage contain multiple loops and nested loops.

B. Target System Platform

The target hardware platform consists of an Aeroflex Gaisler/Pender Electronics FPGA board, GR-CPCI-XC4V [4], and a Leutron Vision PicSight-Smart video camera (GigE Vision) [5]. The FPGA board is in Compact PCI plug-in format. It comes with on-board memory interfaces for Flash PROM, SDRAM, and SRAM via memory expansion. Additional 152 user I/O connectors, Ethernet PHY 10/100 transceiver, on-board oscillators, and serial interfaces complement the board. The Xilinx Virtex-4 LX100 FPGA on this board is well suited and configurable for LEON core implementations. In this paper a LEON3-based processor architecture [6] is used as a multi-core platform consisting of up to 4 cores. Figure 3 shows a basic configuration of the target architecture consisting of 2 LEON3 CPU cores, 2 application-specific classification modules, memories, and interfaces. The base of the software platform is built by an eCos RTOS layer [7] supporting a subset of the IEEE Portable Operating System Interface standard (POSIX 1003.1-1996) [8].

III. TSR-SPECIFIC PERFORMANCE REQUIREMENTS

The TSR application has several crucial design requirements. At first, traffic signs have to be detected and classified before the car reaches the validity area of the sign. Assuming a velocity of 180 km/h and a recording distance of the camera of 30 m the latency must not exceed 600 ms. This real-time requirement is important for the image preprocessing and shape detection and for the traffic sign classification stage, in cases where traffic signs have been detected. Additionally, a TSR application is usually integrated with other driver assistance functions like lane departure control or driver vision enhancement systems. Therefore the resources on the target platform, i.e. chip area and clock frequency, are heavily
constraint. This results in a limited number of CPU cores, limited clock frequencies, a limited memory space, and a limited chip space for application specific blocks. Additionally, the LEON3 CPUs don’t have a floating-point processing unit (FPU) due to a cost limitation. For the mapping described in this paper, the latency for the processing of an entire image frame must be smaller than 600 ms, the number of CPU cores is limited to 4 cores without FPU, and the system’s clock speed is set to 40 MHz. These requirements must be seen against the background of the TSR application structure, like the large number of loop iterations, floating-point arithmetic, and a relatively large coefficient field.

IV. APPLICATION OF THE DESIGN FLOW

In this section the system-level exploration methodology and the continuous design flow are applied to the TSR application introduced above. Figure 4 gives an overview over the entire flow. It starts with an algorithmic specification independent from a final target mapping. On this level of abstraction, the designer can get first hints for the further system refinement by an initial simulation-based performance profile of the application. During hardware/software partitioning the designer gets detailed insights into performance numbers, like latency constraints for function blocks mapped to hardware or to the RTOS software layer. The described design flow integrates standard implementation flows for hardware, like high-level synthesis, and an RTOS design environment for the software. Finally, both parts are integrated on the FPGA target platform.

A. System-Level Architectural Exploration

The first step in the design flow is to build a Kahn Process Network (KPN) model of the TSR application using SystemC as modelling language [9], [10]. As KPN models are highly abstract system level models, they provide the advantage of easy model implementation and modification on one hand and high simulation performance on the other. Due to these characteristics KPN models are predestined as test environment for algorithm design and validation. In the design flow described here, the KPN model of the TSR application is used to compare different traffic sign classification algorithms in terms of accuracy and to test the functional correctness of the implementation.

As stated in Section III the TSR application has crucial requirements in regard to real-time behaviour and performance. As KPN models don’t have a concept of time, it is neither possible to get information about the temporal behaviour of the TSR application from simulating its KPN model nor to optimise its performance. Therefore the untimed functional model has to be instrumented with timing information. At this early stage the designer is often faced with the challenge to predict the temporal behaviour of hardware and software parts and instrument the untimed model with the predicted runtime values. Unfortunately, timing behaviour highly depends on many parameters such as the hardware/software partitioning, the number of available CPU cores as well as their architecture(s), configuration of the real-time operating system(s), algorithm implementation, compiler optimisations etc. This multitude of parameters paired with the fact that these parameters are highly mutable during design space exploration make timing instrumentation by hand very impracticable. To avoid this, the design flow presented in this paper allows to extend abstract KPN models with a concrete system configuration and annotate the resulting model with software runtime information automatically. Below both parts are outlined briefly. Details can be found in [11].

a) KPN model extension: To accomplish the extension of the abstract KPN model with a generic system configuration, the abstract KPN model has to be recompiled using adopted compiler parameters. The modification of the model by hand is not necessary. The resulting model can be configured using description scripts in XML to generate a specific system configuration avoiding to recompile the model for every different configuration. This gives many degrees of freedom to the designer, comprehending the partitioning of functionality in hardware and software, the number and configuration of CPU cores, the runtime of the different application-specific hardware components, the availability and configuration of one or more RTOS instances etc. Listing 1 shows an extract of a corresponding XML description. Currently there are two different RTOS models integrated into the extension. One generic model not annotated with timing information just providing RTOS functionality like thread scheduling, interrupt handling, and parts of the POSIX interface. The other one is not an RTOS model itself but an interface to eCos which allows to connect the extended model with multiple eCos instances and use the functionality as implemented by eCos.

b) Software runtime annotation: The automatic software runtime annotation step takes place during the recompilation using an adapted LLVM compiler [12]. In a first step it generates a low-level intermediate representation (LLVM-IR) of the KPN model sources. Secondly, this LLVM-IR is used as basis for calculating the runtime of software basic blocks

Fig. 4. Design Flow Overview.
for the SPARC v8 compliant LEON3 processor pipeline, and
back-annotating the LLVM-IR with information about basic
block runtime, memory accesses (instruction and data) and
other relevant events of the software execution. Static runtime
analysis can not take into account dynamic system influ-
ences [13] as for example cache behaviour or the occurrence
of interrupts. Therefore the extended KPN model calculates
the effective software runtime during simulation, based on
the information provided by the annotation and the current
system state. The approach described here not only allows the
annotation of KPN model code, but can be applied to eCos
code, too. By connecting such an annotated eCos library to
the extended KPN model, as described above, it is possible
to take into account the runtime overhead introduced by eCos
during simulation and consider its influence on the system’s
temporal behaviour.

c) Design Space Exploration: Using this automatic ex-
tension and annotation approach the designer can generate
system-level models of different system configurations without
the need to modify the KPN model sources or predict runtime
values whenever changing some aspect of the design. The
subsequent simulation provides a global view on the functional
and temporal behaviour of the application in context of the
corresponding system configuration but also delivers insights
into important details such as the cache behaviour or the
effect of interrupts. These extensive options for analysing
a design globally as well as in detail are essential for an
efficient and guided system-level design space exploration,
which supports the decision-making process necessary in early
design phases. This helps avoiding time-consuming and cost-
intensive redesign cycles at later dates during the design,
resulting from wrong design decisions.

As mentioned before the latency requirement for the TSR
application is to detect and classify a single traffic sign
in less than 600 ms. So the objective of the design space
exploration using the extended and annotated model of the
TSR application is to optimise the application’s latency. Sim-
ulation of the TSR application with all threads configured as
software threads executing on different numbers of LEON3
cores provide end-to-end latencies given in Table I. As can
be seen, only the system configuration with 4 cores provides
enough performance to meet the latency requirement. As the
simulation is able to deliver detailed insights into the activities
of the TSR application, it is possible to determine the runtime
of functions and to extract a profile of the application. This
allows the identification of the most time consuming com-
ponent: the appliance of the non-linear SVM kernel function
[14]. Guided by this knowledge the extended TSR application
model can be configured to remove the SVM kernel function
from software and shift it to hardware by simply modifying
the XML system configuration description. The simulation of
hardware supported system configurations shows that such
a system can outperform the pure software solutions. By
simulating varying hardware module latencies, it was possible
to extract, that a hardware module for calculating the SVM
kernel function must complete its task within a period of less
than 530 ms (cf. Table I). This extracted upper limit for the
hardware module latency is used as an input constraint for
hardware synthesis later in the design flow.

B. Software Design Flow
The KPN model of the TSR application together with the
system configuration determined during design space explo-
ration serve as input for the software design flow, as can be
seen in Figure 4.

As the model is written in SystemC—being a C++ library—
this source code can be used as starting point for software
implementation. To simplify the transition from SystemC to
C/C++, a set of rules was developed to map SystemC specific
parts of the model onto POSIX compliant system calls usable
with eCos and other POSIX compliant RTOS. As KPN models
only make use of a subset of features provided by SystemC—
among others sc_fifo<T>, SC_MODULE and SC_THREAD
[10]—only those features have to be mapped to corresponding
C++ and POSIX structures. Table II shows the developed
mapping rules.
In combined hardware/software systems, hardware and software communicate via drivers and interrupts. The development of drivers and interrupt handling routines highly depends on the hardware module to be handled as well as the underlying target platform. To simplify this, a driver development approach based on templates is proposed. The templates provide the common framework for driver development for eCos on a LEON3/AMBA-based target platform including the initialisation of the hardware module, interrupt handling routines, and methods for software access.

By applying the mapping rules and the driver development approach to the TSR model, it was possible to derive POSIX compliant C/C++ code with small effort. This code was used directly as input to the software development toolchain, but it could also be used as starting point for code enhancements. The toolchain itself is based on LLVM GCC used as cross compiler for SPARC v8 to build TSR object files. The eCos library is built from eCos sources parametrised by the configuration script derived from system-level design space exploration. Finally, the application object files are linked together with the eCos library to build an executable for LEON3-based systems.

### C. Hardware Design Flow

A manual implementation of application-specific, algorithmic function blocks is time-consuming, error-prone, and limited in terms of optimisation alternatives. Behavioural synthesis gives the designer an efficient way from an algorithmic specification to powerful hardware modules. This allows the evaluation of different implementation topologies and optimisation approaches like pipelining, loop unrolling, etc. compared to manual RTL coding. Therefore application-specific functionality is specified according to the design flow introduced above in SystemC as algorithmic behavioural model. Forte Cynthesizer [15] is used to perform high-level synthesis and transform these clock-free specifications down to RTL. Subsequently, these RTL specifications are fed into the logic synthesis flow (cf. Figure 4). The core IP components for the LEON3 platform like CPU cores, interfaces, memories, etc. are provided on RTL as VHDL specifications, including all required configuration scripts and constraint files. The logic synthesis of these components is performed using the Synplicity SynplifyPremierDP tool [16], the place and route configuration of the FPGA is performed using the Xilinx ISE tool suite [17]. According to the performance analysis results and the given design constraints, the traffic sign classification module was implemented directly as an application-specific hardware component. In a first step the floating-point data types and arithmetic have been replaced by a synthesisable class of fixed-point data types. The final bit layout of 12 bit integer and 12 bit fractional part has been determined by simulation and evaluation runs in order to find a suitable trade-off between algorithmic precision and area. An exponential function (software math library) has been replaced by a synthesisable algorithm version based on look-up tables. Additionally, two division operations within the image scale function have also been replaced by table lookups to speed up the design and meet the hardware latency requirement of 530 ms derived from system-level architectural exploration. This specification is the basis for the high-level synthesis. To ensure the correct functionality, a test bench has been developed using SystemC. On all levels of abstraction—beginning from behavioural level down to gate level—this test bench was used for verification. The adaption between the test bench and the different design levels is performed automatically. Mentor Graphics ModelSim [18] is used to simulate the designs, the simulation results are cross-checked by the Cynthesizer tool against the results of a golden model.

### D. System Integration

All hardware components—IP components for the LEON3 platform and application-specific blocks for the traffic sign classification stage—are loaded as configuration on the FPGA. Finally, the compiled and linked application software and eCos components are written to the memory space using the debug monitor for LEON3 processors, grmon-eval, provided by Aeroflex Gaisler. From this point in time the system is ready to start traffic sign recognition either based on test data streams coming from a file or live image streams coming from the video camera.

### V. Validation and Results

The final result of the system implementation has been validated using real-world test data. A recorded video sequence of a car ride of about 20 minutes has been used as input for the TSR system model and the implementations on the target


TABLE III
WORST CASE END-TO-END LATENCIES OF IMPLEMENTATION ALTERNATIVES.

<table>
<thead>
<tr>
<th>System configuration</th>
<th>Latency Simul.</th>
<th>Latency Impl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pure SW, 4 LEON3</td>
<td>593 ms</td>
<td>596 ms</td>
</tr>
<tr>
<td>HW SVM kernel (439 ms latency), 2 LEON3</td>
<td>508 ms</td>
<td>503 ms</td>
</tr>
<tr>
<td>HW SVM kernel (369 ms latency), 2 LEON3</td>
<td>438 ms</td>
<td>433 ms</td>
</tr>
</tbody>
</table>

resulting implementation alternatives on the FPGA target have been validated against the model of the TSR application, showing that the applied design flow methodology gave rise to the avoidance of time-consuming and cost-intensive redesign cycles.

ACKNOWLEDGMENT

This work was partially supported by the BMBF project SANITAS under grant 01M3088.

REFERENCES


platform. The comparison of the classification results shows that the realisations on the target platform and the KPN model of the TSR application are functionally equivalent.

Additionally the fulfilment of the application-requirements has been inspected. Table III shows the measured results for three different system configurations, combined with the corresponding latencies resulting from simulating the extended and annotated KPN model of the TSR application. The real-time requirement of latencies less than 600 ms is fulfilled by a highly-parallelised software solution, running on 4 LEON3 CPUs clocked with 40 MHz. As an implementation alternative, the TSR application is mapped to a software part running on 2 LEON3 CPUs and a synthesised hardware block for the classification stage of the system (equally clocked with 40 MHz). The latency is additionally reduced by a second optimised version of the hardware classification block.

For assessing the system-level part of the design flow described in Section IV-A, Table III additionally shows the latencies determined by simulating the extended and annotated system-level model. Those are very close to the latencies measured on the target platform. This approves that the approach is well suited for early performance analysis and design space exploration on abstract system-level.

VI. CONCLUSION AND OUTLOOK

In this paper the design and implementation of an automotive traffic sign recognition application has been shown. An early simulation-based performance evaluation allows to explore and narrow the design space. This gave valuable hints to limit the implementation phase to two design alternatives that achieve the design requirements: A mapping to a multicore-based and RTOS-based software solution and a combined solution, that includes application-specific hardware blocks. High-level synthesis has been used to implement and optimise time-critical function blocks seamlessly in hardware.