A Reconfigurable Platform Architecture for an Automotive Multiple-Target Tracking System

Naim Harb, Smail Niar, Jehangir Khan
LAMIH
Université de Valenciennes et du Hainaut Cambrésis
Valenciennes, France
{naim.harb,smail.niar,jehangir.khan}@univ-valenciennes.fr

Mazen A. R. Saghir
Department of Electrical and Computer Engineering
Texas A&M University at Qatar
Doha, Qatar
mazen.saghir@qatar.tamu.edu

ABSTRACT

This paper reports on our progress in developing a dynamically reconfigurable processing platform for an automotive multiple-target tracking (MTT) system. In addition to motivating our work, we present an overview of our design, some preliminary results, and report on the status of our work.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*real-time and embedded systems*

General Terms

Design, Experimentation, Performance

Keywords

Automotive multiple target tracking, FPGAs, Multiprocessor system-on-chip, Partial dynamic reconfiguration, System-level optimizations

1. INTRODUCTION

The past two decades have witnessed a proliferation of microelectronic devices in automotive systems. Today, such devices are commonly used in a wide range of automotive applications including engine and vehicle control, anti-lock brakes, navigation systems, and car entertainment units. While early automotive systems were designed using microcontrollers, DSPs, and ASICs, some of the newer systems use field-programmable gate arrays (FPGAs) [7]. In addition to providing high logic densities, integrated hardware components, and fast design turnaround times, FPGAs can be easily reconfigured to meet the needs of their operating environments. This makes FPGAs ideal platforms for new automotive applications.

An increasingly important class of automotive applications, particularly for commercial vehicles, is driver assistance systems. Such systems reduce a driver's workload and improve road safety in stressful driving conditions (e.g. at night or in bad weather). Driver assistance systems often require real-time monitoring of the driving environment and other vehicles on the road. A multi-target tracking (MTT) system uses an onboard radar to keep track of the speed, distance, and relative position of all vehicles within its field of view. Such information is crucial in applications such as collision avoidance or intelligent cruise control.

The computational needs of a MTT system increase with the number of targets that must be tracked. It is therefore very difficult for a single processor to handle these computational requirements alone. In this paper, we present an FPGA-based multiprocessor system-on-chip (MPSoC) architecture for implementing a MTT system. We describe the evolution of our system from a software-based implementation that distributes the computational workload among multiple soft processor cores to a hybrid implementation that combines soft processor cores with dedicated hardware blocks. We also discuss the system-level trade-offs we made and describe our plans to further evolve our architecture to support dynamic reconfiguration.

2. THE MTT APPLICATION

The MTT application tracks targets by processing data measured by the radar every 25 ms. This data is processed in three iterative stages. During the *observation* stage, the MTT application reads speed, distance, and azimuth angle data for each target. During the *prediction* stage, an adaptive filter, such as a Kalman filter, is used to predict the location of each target on the subsequent radar scan. This prediction is based on the actual location data measured during the observation stage and an estimate of the location data computed by the filter during the previous radar scan. Finally, during the *estimate* stage, new target location estimates are computed for use in the subsequent prediction stage. Figure 1 shows the structure of the MTT application. A more detailed description of the application and its mathematical formulation can be found in our earlier work [3].

APRES 2009 Grenoble, France

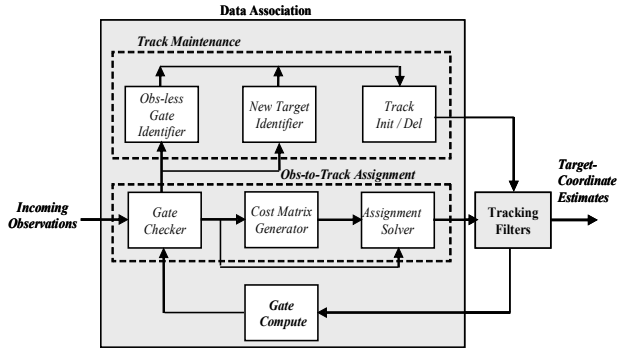

Figure 1: The MTT Application

3. BASELINE MTT SYSTEM

Figure 2 shows our baseline MTT system implementation, which uses multiple Nios-II soft processor cores [5] implemented in an Altera Stratix-II FPGA. To map the MTT application onto multiple processors, we divided our code into functions that interact in a producer-consumer fashion. We then distributed the functions among the processors to meet the 25 ms real-time constraint imposed by the radar scan window. To help us profile the run-time characteristics of our application and guide the allocation of functions to processors, we used dedicated hardware profiling counters to measure the latency and execution frequency of individual functions.

Our baseline system consists of 23 heterogeneous processor cores arranged in a multiprocessor-system-on-chip (MPSoC) configuration. Due to the time-critical nature and floating-point requirements of the Kalman filter, and the need to track up to 20 targets simultaneously, our baseline system includes 20 dedicated processors configured with single-precision floating-point units that execute the Kalman filtering function. Moreover, the assignment solver, gating, and track maintenance functions [3] are each allocated to a different processor to minimize execution time and communication overhead. Where needed, processors are configured with appropriately sized instruction and data caches and local memories. Processors with interacting functions are also interconnected using appropriately sized FIFO buffers.

To meet the 25 ms real-time constraint, we applied a number of optimizations to reduce the execution times of different functions. These included sizing the on-chip instruction and data cache memories for different processors to reduce off-chip memory access times; introducing on-chip private data memory (PDM) banks to store the system stack and the heap to reduce the cost of dynamic memory allocation in various functions; adding floating-point hardware support to some processors to execute floating-point operations more efficiently; and transforming some functions to only use integer data types and reduce overall execution time. Table 1 shows the different optimizations applied to various functions along with their final, corresponding, execution times.

4. HYBRID MTT SYSTEM

Our baseline MTT system demonstrated the feasibility of using software-programmable processor cores to execute a complex application and still meet its real-time constraints. It also demonstrated the importance of optimizing the system implementation to meet these performance constraints. However, it also demonstrated the high cost, in area and resource utilization, associated with a software-only implementation. This is particularly evident with the Kalman filtering block where dedicated processors, configured with floating-point units, cache memories, and private data memories, are used to execute the Kalman filtering code for each target. Even when multiple targets could be tracked by a single Kalman filtering block, the cost of implementing an entire processor in the logic fabric of an FPGA to only support a single, dedicated function may be too high. A more cost-effective solution would be to use a hybrid system that integrates dedicated Kalman filtering hardware blocks with software-programmable processor cores.

To better understand the functional and numerical characteristics of the Kalman filtering block, we developed a MATLAB model using single-precision floating-point arithmetic. Details of the mathematical model are described in [3] and [2]. We then tested our model using two data sets: a random set of target data from the radar's operational range (i.e. distances from 0 to 150 meters and azimuth angles from -6 to +6 degrees); and a more realistic data set that emulates a target being overtaken by the vehicle with the radar. Gaussian noise was also added to both data sets to model the error introduced by the radar sensors.

Function	I$	D$	PDM	FPU	Int	Latency
Kalman	4KB	–	2KB	Yes	–	3 ms
Gating	16KB	2KB	3KB	Yes	–	23 ms
Solver	8KB	16KB	–	–	Yes	24 ms
Track	–	–	–	–	–	8 ms

Table 1: Optimizations applied to various functions and resulting execution latencies

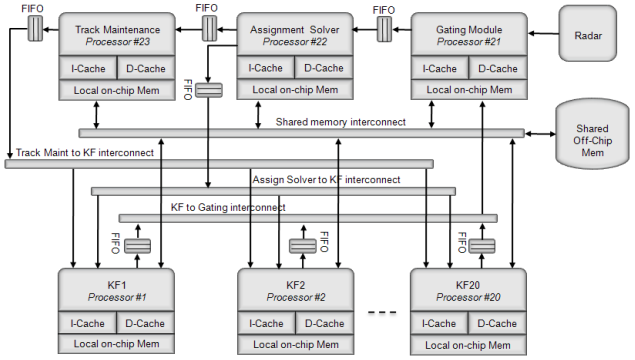

Figure 2: Baseline MTT System

4.1 Filter Coefficients Stability

Figure 3 shows the output response curves of different Kalman filter implementations for the distance measurements of the realistic data set. After a transient training interval, the filter output stabilizes and begins to track the input data (represented on the graph by the solid line). When analyzing our results, we noticed that for all filter implementations, and for both data sets, the elements of the

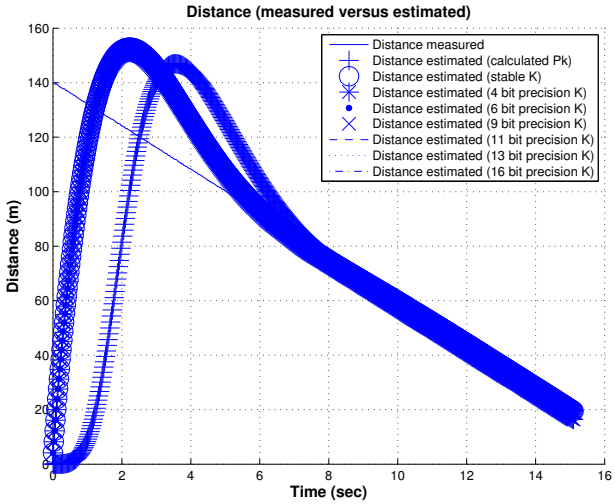

Figure 3: Kalman filter response for distance measurements.

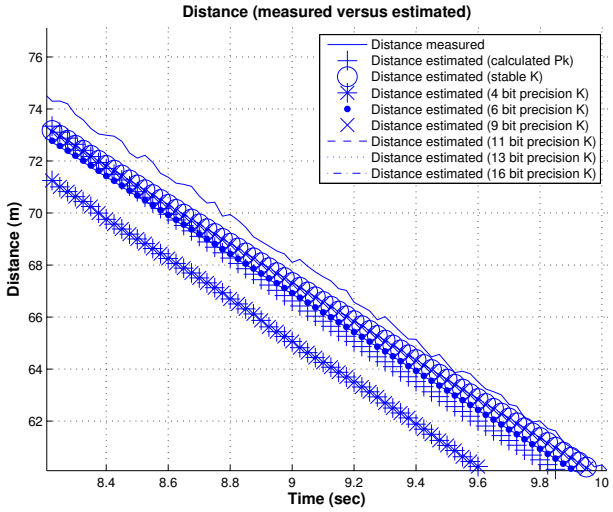

Figure 4: Filter response errors for different precision levels.

estimation covariance matrix [3][2] converge to optimal values and stabilize. This suggests that these elements, which are computed iteratively and depend on the characteristics of the radar, can be fixed at their optimal values. By setting these elements to constant values, the elements of other matrices (e.g. the *Kalman gain matrix* [3][2]) also become constants. This significantly reduces both the computational complexity and execution time of the filter, and leads to a more efficient implementation of the filter block. It also leads to a faster filter response time (c.f. the curves labeled **calculated Pk** and **stable K** in Figure 3). Our hardware implementation of the Kalman filter block was therefore designed using the optimal constant values of various filter coefficients.

4.2 Fixed-Point Data Precision

The software implementation of the Kalman filter block uses single-precision floating-point arithmetic and requires a processor with a floating-point unit to execute. However, the dynamic range of the data obtained from the radar's sensors (150-meter distances and -6 to +6 degree angles, respectively) is significantly smaller than the range supported by a floating-point unit. Furthermore, an automotive target can still be tracked effectively even when the level of numerical precision is low. For example, the MTT system will likely behave the same way whether a target is determined to be at 29.8 or 29.7889993 meters away. This suggests that a fixed-point Kalman filter block that provides acceptable levels of numerical accuracy and precision is a more cost-effective alternative to a floating-point implementation.

Figure 4 shows a close-up view of the Kalman filter's response curves for the distance measurements after the output has stabilized. The solid line corresponds to the floating-point implementation while the other curves correspond to fixed-point implementations at different levels of precision. Since the radar has a range of 150 meters, only 9 bits are needed to represent the (signed) integer portion of the distance measurement. The curves in Figure 4 therefore correspond to the filter response as we increase the *fractional* portion of the distance measurement from 4 to 16 bits.

Figure 4 clearly shows that the error between the floating-point reference and the various curves decreases as the fractional precision level increases. However, even when designing with the fast and area-efficient on-chip hardware multipliers found in most high-density FPGAs, increasing precision levels typically result in larger and slower hardware. To optimize the utilization of 18×18 hardware multipliers, we implemented our filter block using an 18-bit fixed-point data format. For distance computations we used a 9-bit, signed, integer component and a 9-bit fractional component, and for angle computations, we used a 4-bit, signed, integer component and a 14-bit fractional component.

4.3 Software vs. Hardware Implementation

Table 2 shows the FPGA resources used by both the software and hardware implementations, which, unlike our baseline system, targeted the Xilinx XC4VFX12 Virtex-4 FPGA and Xilinx MicroBlaze soft processor core, respectively [4]. The reason for this change is that, currently, only Xilinx FPGAs and design tools support dynamic partial reconfiguration [1]. Since we are interested in exploring dynamically reconfigurable MPSoC architectures, we are limited to using Xilinx devices and tools. However, this does not diminish the results of our baseline implementation, which demonstrated the feasibility of implementing a heterogeneous MPSoC architecture using customized soft processor cores.

Our results show that the hardware implementation uses almost 80% fewer logic resources (LUTs and slices) and no slice flip-flops or BRAMs compared to the software implementation. On the other hand, the hardware implementation uses four times as many DSP48 blocks as the software implementation. These results clearly demonstrate the architectural characteristics of the hardware implementation, which makes use of the constant filter coefficient and fixed-point arithmetic optimizations described earlier, in addition to making use of the fast and area-efficient DSP48 hardware

FPGA Resources	Available	Used (SW)	Used (HW)
Slices	5,472	1,265 (23%)	265 (4%)
Slice Flip Flops	10,944	1,347 (12%)	0 (0%)
4 input LUT	10,944	1,902 (17%)	403 (3%)
DSP48s	32	3 (9%)	12 (37%)
RAMBs	36	8 (22%)	0 (0%)

Table 2: Hardware resources used by the software and hardware Kalman implementations.

blocks. Conversely, the software implementation requires enough logic and storage resources to implement a pipelined processor datapath.

We also compared the software and hardware implementations in terms of latency. For both implementations we used a 100 MHz system clock. To measure the latency of the Kalman software function we used a hardware timer to measure the number of CPU clock cycles spent inside the function. We then multiplied the clock cycle count by the period of the system clock, which showed the latency of the software implementation to be 268.0325 μsec. To measure the latency of the Kalman hardware block we used the results of the post-placement and routing timing report, which showed the latency to be 0.033 μsec. Our results therefore show that the hardware implementation is 8,122 times faster than our software implementation.

5. DYNAMICALLY RECONFIGURABLE MTT SYSTEM

One of the disadvantages of a hybrid MTT system is that the hardware blocks used in the system may become obsolete if the application or the algorithms around which they are based change. For example, different types of filters may be needed for tracking targets in different driving environments (e.g. urban, suburban, rural, etc...). One way around this problem is to implement a system with *all* the necessary hardware blocks, and to select and use the appropriate blocks at run-time. However, this is not a very area-efficient solution, and it does not safeguard against hardware block obsolescence. Another solution, particularly for systems implemented in FPGAs, is to reconfigure the entire FPGA to implement a new system with new hardware blocks. However, this also is not very efficient since only a small portion of the hardware implementation typically needs to be reconfigured. A better solution would be to build a *dynamically reconfigurable* system that enables an application to replace obsolete or inadequate hardware blocks with new ones on the fly. However, such a solution requires FPGA devices and tools capable of supporting partial dynamic reconfiguration.

5.1 Partial Dynamic Reconfiguration

Xilinx is currently the only FPGA vendor that provides devices and tools that support partial dynamic reconfiguration [1]. This technology allows designers to map presynthesized, implemented, and routed blocks onto specific regions of an FPGA device. The bit streams needed to reconfigure a portion of an FPGA can be stored either on- or off-chip. An internal configuration access port (ICAP) controller [6] can be added to the system to manage the reconfiguration process, and dedicated bus macros [1] can be used to interface the reconfigurable portions of the architecture with the rest of the system.

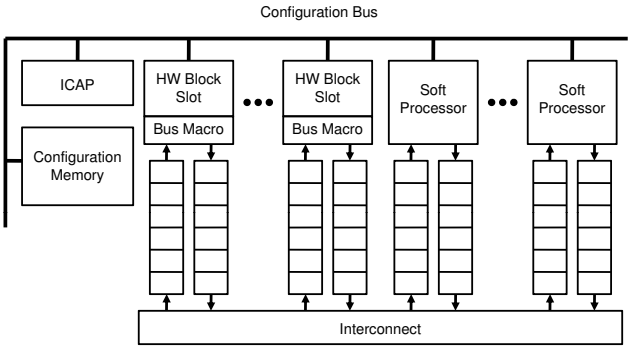

Figure 5: Proposed dynamically reconfigurable system architecture.

5.2 Proposed System Architecture

Figure 5 shows a high-level view of our proposed system architecture. Like the hybrid MTT system, it includes a number of soft processor cores for executing the control-intensive portions of the MTT application. It also includes a number of slots that can be configured with pre-designed hardware blocks for accelerating the performance-critical portions of the application. The soft processors and hardware blocks would be able to exchange data through interconnected FIFO buffers. The system also includes an ICAP controller for managing the configuration of hardware blocks on the fly. The configuration bit streams could be stored on- or off-chip, while the loading, removal, and swapping of hardware blocks would be controlled by the main MTT application software using various performance-enhancing heuristics.

6. STATUS AND REMAINING WORK

We have implemented the baseline and hybrid MTT systems and evaluated both in terms of performance and FPGA resource utilization. We are currently implementing the dynamically reconfigurable MTT system using the Xilinx ICAP and bus macro technologies. Once we complete developing and validating our new hardware platform, we will focus on developing the software mechanisms for managing the dynamic reconfiguration of hardware accelerators and the heuristics for maximizing run-time performance. Our long-term goal is to develop a dynamically reconfigurable platform architecture that would be suitable for a wide range of automotive applications.

7. REFERENCES

[1] J. Becker et. al. "Dynamic and Partial FPGA Exploitation," *Proceedings of the IEEE*, pp. 438-452, Vol. 95, No. 2, IEEE, 2007.

[2] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*, Artech House Publishers, 1999.

[3] J. Khan et. al. "An MPSoC Architecture for the Multiple Target Tracking Application in Driver Assistance System," *Proceedings of the 19th*

International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 126-131, IEEE, 2008.

[4] *MicroBlaze Processor Reference Guide*. UG081 (v9.0). `http://www.xilinx.com`.

[5] *NIOS-II Processor Reference Handbook*. `http://www.altera.com`.

[6] *OPB HWICAP Data Sheet*. DS 280. `http://www.xilinx.com`.

[7] M. Ullmann et. al. "On-Demand FPGA Run-Time System for Dynamic Reconfiguration with Adaptive Priorities," *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 454-463, Springer-Verlag LNCS Vol. 0302, 2004.