2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools

An Improved Automotive Multiple Target Tracking System Design

Tobias Lange*, Naim Harb, Haisheng Liu, Smail Niar, Rabie Ben Atitallah
LAMIH
Université de Valenciennes et du Hainaut Cambrésis, France
{naim.harb,haisheng.liu,smail.niar,rabie.benatitallah}@univ-valenciennes.fr
*Faculty of Electrical Engineering
Technical University of Brunswick, Germany
tobias.lange@tu-bs.de

***Abstract*—Multiple Target Tracking (MTT) algorithms are widely used in various military and civilian applications but its use in automotive safety has little been investigated. In MTT algorithms, implemented in embedded systems, it is important to use the minimum required resources to allow the entire DAS system to be integrated on the same chip (data acquisition, MTT and alarm restitution). This allows the reduction of the System on Chip (SoC) complexity and cost. This paper presents an efficient Driver Assistance System (DAS) based on MTT application. To do so, we first identified the performance bottlenecks in the application. In this application, a set of optimizations were applied to reduce the MTT algorithm's complexity. Tuning in conjunction the hardware and the software yielded to optimize the final system and to meet the functional requirements. The result is a complete embedded MTT application running on an embedded system that fits in a contemporary medium sized FPGA device.**

I. Introduction

Driver Assistance Systems (DAS's) have become a widespread class of automotive applications in nowadays commercial vehicles. They lend even more confidence to driving and improve road safety in stressful driving conditions (e.g. at night or in bad weather conditions). Adaptive cruise control, radar-aided automatic proximity control, and navigation system are examples of well-known range of high-tech DAS's. Recently, we started working on a Multiple Target Tracking (MTT) DAS feature which aims for an early warning and collision avoidance system onboard a vehicle. The purpose of target tracking is to collect data from the sensor's field of view (FOV), containing one or more potential obstacles of interest, and to partition that data into sets of observations, or tracks. In our application, we use radar as sensor because it has the advantages of longer range as compared to camera based systems. It performs better in bad visibility conditions and has lower computational requirements. Moreover, radars help to detect obstacles at longer distances and hence ensures longer reaction time for vehicle drivers.

Traditionally, automotive systems have been designed using 8- and 16-bit microcontrollers. However, increasing levels of complexity and computational demands in automotive applications are forcing a move to more powerful processors, DSPs, and even ASICs. In recent years, field programmable gate arrays (FPGAs) also have been used to implement various automotive subsystems. With their inherent support for parallelism, high logic densities, and rich sets of embedded hardware components, FPGAs are very well suited for implementing computationally demanding applications. Their flexibility, programmability, and fast design turnaround times also enable system designers to quickly introduce new features or update exisiting ones in response to changing requirements or new standards.

This paper focusses on the optimization and improvement of the MTT system as an application specific System on Chip (SoC) implemented on FPGA. Taking into account automotive embedded system constraints in terms of performance, low cost and reliability, we adjust the hardware features in order to meet the realtime requirements. In this context, we applied three optimizations: an optimized filtering block, a new assignment algorithm and a utilization of fixed point instead of floating point numbers. With these optimizations, the execution time was reduced by a factor of 100 while maintaining a good accuracy.

This paper is organized as follows. After Section II which presents the related work, Section III exposes an overview of the MTT system. In Section IV we are identifying the performance bottlenecks of the application. Section V details a set of optimizations applied to the MTT software in order to improve the performance. In order to evaluate the efficiency of our redesigned MTT, results are presented in Section VI.

II. Related Work

Different types of DAS's have been proposed in the last few years. Most of the existing DAS systems have either limited functionalities or are too costly for a large-scale automotive utilization. Adaptive cruise control, lane keep assistance systems, parking assistance aystems, obstacle detection and avoidance systems are the most frequent functionalities in existing DAS systems[1][2].

Recent research activities concentrate on the use of DAS in complex environments and scenarios, such as detecting bikes, pedestrians, children, and under changing weather and lighting conditions. The proposed systems in these projects

978-0-7695-4171-6/10 $26.00 © 2010 IEEE
DOI 10.1109/DSD.2010.54

are implemented by different hardware and/or software architectures. From the hardware point of view, dedicated hardwired Application Specific Integrated Circuit (ASIC) to pure programmable processors are used. To offer a good performance/flexibility/cost trade-off designers multi-processor system-on-chips (MPSoC) and/or hardwired FPGA-based circuits are more and more used in DAS systems [3]. Our contribution is to design an optimized MTT software that can be mapped on a low cost FPGA.

III. General Overview of the MTT Application

In our DAS application, a radar sensor is used to detect other traffic participants in front of the own vehicle. Because the data delivered by the radar sensor is affected by measurement noise and other forms of interference, there is a need for an algorithm which filters and smoothen the incoming measurements. For handling multiple targets in the FOV, incoming measurements also need to be associated with objects detected previously. This, tracking and filtering, is done by the MTT application. The structure of the application can be devided into five functional blocks as shown in Figure 1. The next subsections will describe the use of each block in details.

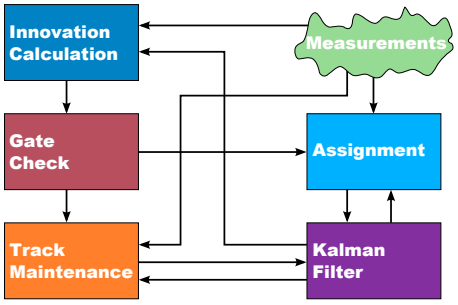

Fig. 1. Structure of the revised MTT application

1) Innovation Calculation: The innovation calculation computs the differences between the predicted values for each track and the actual measurements. These measurements are delivered by a radar sensor. The predicted values of each track were computed in a previous iteration. The results of this step are needed by the following gate check block. Later, these statistically differences are used in the assignment process.

2) Gate Check: The gate check calculates a matrix of binary values which indicates possible assignments between measurements and tracks. Only the measurements inside a specific window around each track are assigned (Figure 2). The result of this block is later used by the track maintenance block to detect appearing and disappearing objects as can be shown in Figure 3.

3) Track Maintenance: On the road, vehicles enter or leave the field of view of the radar sensor all the time. As the radar is able to detect only one object per angle, it is also possible that a car is hidden behind another car. In order to get a fully working tracking application, we implemented the detection

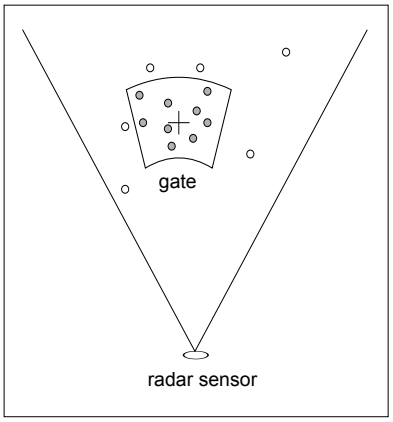

Fig. 2. Gating process: All measurements outside a specific window are not considered

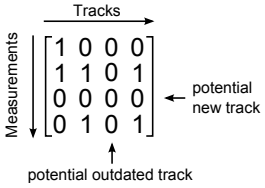

Fig. 3. Gate mask: Detection of new and outdated tracks

of appearing and disappearing obstacles. This is done by the track maintenance functional block. If the gate mask contains a line or a column of zeros, there might be either an appearing or a disappearing object (Figure 3). After an object is detected in at least 9 iterations, this upcoming track is initialized. If an object is missed for more than 9 iterations, the outdated track is deleted (The number of iterations was obtained empirically). The block also manages the possible fusion and split-up of tracks.

4) Assignment: After a list of tracks is initialized and measurements are filtered out during the gate check process, tracks must be paired with the residual measurements. As there are probably more possible measurements per track, only the one with the shortest statistical distance is associated (Figure 4). In literature, there are multiple solutions to solve this assignment, for example the Munkres algorithm or the Auction algorithm. As will be discussed later, in our case a simplification is possible.

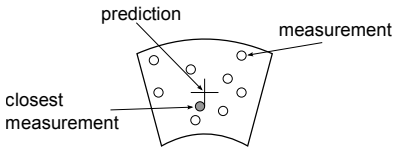

Fig. 4. Assignment process: The statistically nearest measurements are assigned to each track

5) Kalman Filter: A radar sensor, like every other sensing element, is affected by measurement noise. To smooth the measurements obtained from the radar we are using a

Kalman filter suitable for Gaussian noise environment like ours. Figure 5 shortly explains the main idea behind the Kalman filter. From an actual state (including position and speed) a prediction of the next position is calculated. In the next radar scan an estimation is computed as a weighted average between the previous calculated prediction and the incoming new measurement.

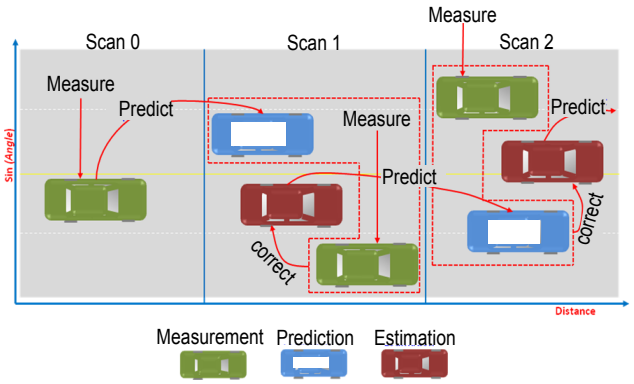

Fig. 5. A basic principle Kalman filter [4]

IV. System Profiling

In order to reduce the complexity and the hardware resource consumption of the base system, we had to identify the performance bottlenecks of the MTT application. In this subsection, a hardware-based architecture is presented in which the profiling was implemented. Following this, profiling results are presented as well and conclusions are deducted.

A. Profiling Platform

We considered the ML403 Virtex-4 Xilinx board as the hardware platform in which the architecture is implemented. The profiling architecture is based on the Xilinx Microblaze (MB) soft processor. The Xilinx embedded development tools are consequently used for our profiling experimentations. In our baseline system (Figure 6), all the functional blocks are executed on a single micoblaze processor running at 100 MHz. The execution time is derived by using an embedded timer IP that counts the number of cycles required by the blocks.

Figure 6 depicts a microblaze-based architecture. The Microblaze processor (MB) communicates with the local BRAM memory (Block RAM), where the application's local data and instructions are stored. Both the DLMB and ILMB (Data Local Memory Bus and Instruction Local Memory Bus) buses connect the MB to the BRAM via local memory controllers, respectively.

B. Profiling Results and Bottlenecks

The software was implemented using floating-point values. Table II summarizes the execution time consumed by each task. In our case, 1000 iterations and 20 targets are considered. Several remarks can be drawn: First, the total execution time is bigger than the system's requirement (25 ms). Second, the

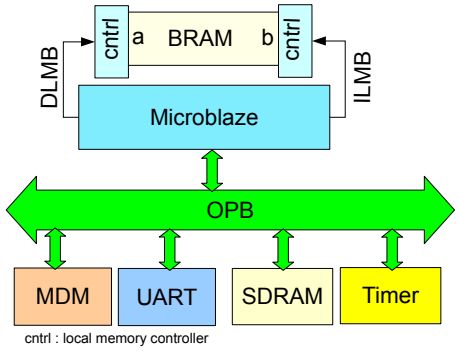

Fig. 6. A microblaze-based architecture

Kalman Filter and the Gating Module are the two performance bottlenecks in the system consuming 44% and 52% of the total execution time. Hence, the main optimizations are concerning these two tasks.

Task	Exec. time (ms)
Kalman Filter	134
Track Maintenance	(not implemented)
Assignment	9.7
Gating Module	157.6
Total	301.3

TABLE I
Profiling results of the base MTT.

V. Optimization of the MTT software

After profiling and identifying the performance bottlenecks of the application, we applied some optimizations on the implementation of several functions. These functional changes and optimizations are desribed in the next sections.

A. Optimized Kalman Filter

The filtering in the Kalman block is based on parameters which are normally calculated automatically during the filtering process. These parameters stabilize to fixed values after some iterations and can be precalculated [5]. By using the precalculated values directly, a lot of computing time can be saved. Also the results of the first iterations (scans) do achieve more precision. There were special implemented functions used to handle these calculations. For example, computing a multiplication of two 3×3 matrices needs at least 27 multiplications and 18 additions. Because in the MTT application these matrices contains some ones, zeros and other constant values, the filtering can be simplified by using scalar calculations.

By applying these two optimizations, the number of additions and multiplications per filter cycle is reduced by 90%.

B. Alternative of the Munkres Algorithm

The first version of our application was developed under the assumption that there is exactly one object measured by the radar for each real target. But in real scenarios and real

data, it has been noticed that the radar detects multiple objects for each real target caused by reflections. After the described gating process, there rests a cloud of measurements around each track.

In the first approach of our system we used the Munkres algorithm to find the statistically nearest measurement for assigning it to a specific track and feed it into the Kalman filter. By doing this, one measurement can be assigned to only one track and vice versa (Figure 7a). By having a cloud of measurements, the algorithm can be simplified to allow one measurement to be assigned to multiple tracks as shown in Figure 7b. Possible errors are smoothed by the Kalman filter in a later stage of the MTT application. The Munkres algorithm is very computing intensive with a runtime in $O(n^3)$, where n represents the maximum number of trackable objects in the system. The result of using this simple algorithm is a gain in runtime.

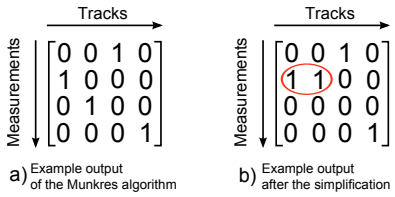

Fig. 7. Simplification of the assignment: Difference between original Munkres algorithm and simplified approach

C. Calculation with Fixed-Point Numbers

The last optimization is achieved by using a fixed-point representation instead of a floating-point. The error introduced by the use of fixed-point numbers, which can be described as noise, is much smaller than the noise of the radar sensor. After this optimization we obtained a speed improvement of up to a factor of 18 in execution time. Also the need for floating-point units in the embedded processors is now unnecessary which help in conserving hardware resources.

After implementing these optimizations, we tested the whole application using sample data generated in MATLAB. For testing the influence of using a fixed-point representation instead of a floating-point one, we compared the resulting distance and angle for one track. The results is an estimation error up to 1% for angle and distance.

VI. Profiling of the Revised MTT

The local data and instructions are at first saved into the local memory, and then loaded through the respective bus while executing the application.

One measurement comprises a couple of a distance and an angle value. Considering the optimizations, these data are thus coded with 32 bits (integers). Thus, it is necessary to assign 160,000 bytes memory for storing the measurement data. Since there is only 32 kB of local memory available in our case, thus these measurement data are mapped into the DDR SDRAM memory, as well as the heap and the stack sections. The stack and heap sizes are equal to 0x1400 bytes and 0x400 bytes respectively.

It is important to mention that the time profiling results might vary according to the Microblaze configuration. Tab. II shows the profiling results on three different configurations (A, B and C) for individual tasks as well as the entire MTT system. Configuration A has no caches while configuration B and C have a 2 kB instruction cache and 4 kB of data cache. However, configuration B has all the code sections mapped to external memory.

	Configurations		
Function Block	**A**	**B**	**C**
Innovation Calculation	27%	14%	30%
Gate Check	19%	30%	18%
Track Maintenance	40%	44%	36%
Assignment	10%	9%	8%
Kalman Filter	4%	3%	5%
Total	9.3 ms	15.4 ms	2.1 ms

TABLE II
The timing profiling results.

In general, the time consumed for a local memory access is less than that of an external one. Thus, this access time becomes important in case the data and instructions need to be transferred from external memory to local memory. This justifies the greater execution time for configuration B. However, this time does not exceed the imposed time constraint, 25 ms. Using the local memory for data and instruction storage, our system needs merely 2.1 ms to accomplish one iteration.

The time profiling demonstrates that the track maintenance and Kalman filter functions are the most and least time-consuming components, respectively.

VII. Conclusion

In this paper we presented an optimized MTT application in DAS systems. We showed some profiling results of the application's functional blocks and determined the bottlenecks on this application. It was obvious, judging by the execution time of the base application, the violation of the realtime constraint of the system (25 msec). Mainly for that reason, a search for some optimizations was a necessity in order to meet the constraint of such a real time system.

References

[1] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," in *IEEE Transactions on Intelligent Transportation Systems*, 2003.

[2] "The european fp7, prevent-intersafe project," in *http://www.prevent-ip.org/en/prevent_subprojects/intersection_safety/intersafe/*.

[3] J. Saad and F. Bodereau, "Fpga-based radar signal processing for automotive driver assistance system," in *IEEE International Workshop on Rapid System Prototyping RSP*, 2008.

[4] J. Khan et. al., "An mpsoc architecture for the multiple target tracking application in driver assistance system," in *Proc. of the 19th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2008.

[5] N. Harb et. al., "A reconfigurable platform architecture for an automotive multiple-target tracking system," in *ACM SIGBED Review*, 2009.