Design of Multiple-Target Tracking System on Heterogeneous System-on-Chip Devices

Guanwen Zhong, Smail Niar, *Senior Member, IEEE*, Alok Prakash, *Member, IEEE*, and Tulika Mitra, *Member, IEEE*

***Abstract*—Advanced driver-assistance systems (ADAS) generally embrace heterogeneous platforms consisting of central processing units and field-programmable gate arrays (FPGAs) to achieve higher performance and energy efficiency. The multiple-target tracking (MTT) system is an important component in most ADAS and is particularly suited for heterogeneous implementation to improve responsiveness. However, the platform heterogeneity necessitates numerous design decisions to obtain the optimal application partitioning between the processor and the FPGA. In this paper, multiple configurations of the MTT application have been investigated on the Xilinx Zynq commercial heterogeneous platform. An extensive design space exploration was performed to recommend the optimal configuration with high performance and energy efficiency. A reduction of more than 65%, both in execution time and energy consumption, has been obtained by the utilization of the heterogeneous architecture. Finally, an analytical model is proposed to estimate execution time and energy consumption to enable a rapid exploration of the different configurations and predict the performance that can be expected with future system-on-chip (SoC) platforms and radar sensors in ADAS.**

***Index Terms*—Hardware accelerators, heterogeneous architecture, multiple-target tracking (MTT), Zynq.**

I. Introduction

THE automotive industry continues to look at ways to reduce the fatalities and the severity of road accidents. To achieve this, various innovations have been proposed in the past decades, such as automatic airbag deployment, antilock braking systems, lane-departure warning systems, etc. However, the ever-rising number of cars plying the roads necessitates additional improvements and additions to the existing control systems to reduce road accidents and resulting loss of life and property. The last few years have seen the introduction of more sophisticated safety systems, which are collectively termed advanced driver-assistance systems (ADAS), that integrate sensors such as radar or camera to track objects, such as other vehicles, cyclists, or pedestrians, around a moving car to better gauge and maintain a safe distance at all times [12].

However, this also leads to significantly increased electrical power consumption as well as higher processing requirements to ensure the stringent real-time constraints expected in such systems. Power consumption in embedded systems for automotive is important for several reasons. First, reducing energy consumption minimizes hardware complexity and enables to fit the system in a smaller field-programmable gate array (FPGA), which plays an important role in reducing the cost of the system. Second, higher power consumption leads to higher temperature, which, in turn, reduces the mean time between failure of the ADAS. With the ever-reducing geometries of embedded systems, higher power consumption, in the presence of important electromagnetic interferences from the mechanical parts, increases probability of failure.

System designers are constantly morphing new functionalities and/or new algorithms for robustness, which makes an application-specific-integrated-circuit-based solution prohibitively expensive. In addition, the number of sensors in next-generation transportation systems will increase to enable driverless and autonomous cars to further improve safety and reliability of future transport systems. An ADAS for urban scenarios can be connected to several short-distance radars, such as the TRW AC1000 [13] or the short-distance radar used in [1]. Each of these radars is capable of sending to the computing part of the ADAS data relative to more than 100 objects, such as cars, pedestrians, motorcycles, etc., every millisecond (ms). All these objects must be tracked and identified to generate the appropriate alarm to the driver when needed. For this reason, it will become crucial to process a large number of tracks in a short execution time. To meet such conflicting constraints, FPGA-based heterogeneous platforms have recently gained popularity for systems that demand the programmability offered by a processor and performance achieved by the reconfigurable FPGA fabric. The Zynq platform from Xilinx [14] and system-on-chip (SoC)-FPGA platform from Altera [15] are examples of such heterogeneous platforms in the current embedded market. Such platforms typically integrate an application processor such as the dual-core Cortex A-9 from ARM [16], with a highly reconfigurable FPGA fabric. These heterogeneous hardware (HW) platforms are becoming increasingly complex and will integrate more and more processors and logic elements. The new Xilinx Zynq UltraScale+ multiprocessor system-on-chip (MPSoC) [17] contains four ARM-Cortex-A53

Manuscript received June 10, 2015; revised October 26, 2015 and March 8, 2016; accepted March 18, 2016. Date of publication March 25, 2016; date of current version June 16, 2016. This work was supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 MOE2012-T2-1-115, by the French-Singaporean Merlion 2012 Ph.D. Project, and by the International Campus on Safety and Intermodality in Transportation (CISIT). The review of this paper was coordinated by Dr. D. Cao.

G. Zhong and T. Mitra are with the School of Computing, National University of Singapore, Singapore 117417 (e-mail: guanwen@comp.nus.edu.sg; tulika@comp.nus.edu.sg).

S. Niar is with the Laboratoire d'Automatique, de Mécanique, et d'Informatique Industrielles et Humaines (LAMIH), University of Valenciennes, 59313 Valenciennes, France (e-mail: smail.niar@univ-valenciennes.fr).

A. Prakash is with the School of Computing, National University of Singapore, Singapore 117417 (e-mail: alok.prakash@outlook.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVT.2016.2546957

cores running at up to 1.3 GHz, an ARM MALI embedded GPU, up to 1-M logic elements, and 3-K digital signal processors (DSPs). For the ADAS applications, the processors offer the opportunity to implement the applications with frequently changing specifications or standards, whereas the FPGA fabric allows to accelerate the critical components of the system for real-time responsiveness.

In this paper, we explore an important component of ADAS, namely, the multiple-target tracking (MTT) [18] system that uses radar-based sensors to track objects in its field of vision. Radar sensors are more robust and efficient than camera-based ADAS as they allow for detecting obstacles at longer distances. Moreover, the radar system behaves better in bad visibility conditions such as in foggy weather, rain, and snow. Finally, radar-based ADAS has lower computational requirements when compared with camera-based ADAS.

The MTT application has also been shown to be amenable to hardware acceleration and presents an opportune application for harnessing the potential of an FPGA-based heterogeneous platform. We chose the new Zynq platform from Xilinx [14] as the heterogeneous platform for implementation and performed extensive investigations on various hardware–software partitions for the MTT application on such a platform. Our work is the first that explores MTT implementation for heterogeneous FPGA-based MPSoC. These architectures will be more and more powerful and will be widely used in future transportation systems. For this reason, in this paper, we develop analytical models for a rapid design space exploration in the realization of next-generation radar-based MTT for ADAS.

The rest of this paper is organized as follows. Section II details the existing work in this area. Section III describes the MTT application, whereas Section IV introduces the target platform. We discuss the design space exploration of the MTT on the heterogeneous Zynq platform in Section V. In Section VI, we present our analytical models for execution time and energy consumption estimation, and we conclude this paper in Section VII.

II. Related Work

Many researchers have focused and developed diverse ADAS in the past decades. Adaptive cruise control, antilock braking systems, lane keep assistance, parking assistance systems, obstacle detection/avoidance systems, and traffic sign recognition are among the most common ADAS functionalities [19]. To achieve both high performance and low cost, researchers have proposed their ADAS based on three kinds of HW platforms: simple homogeneous SoC such as microcontrollers, FPGA-based platforms, and heterogeneous SoC platforms. However, existing ADAS systems have either limited functionalities or are too costly to support such complex applications. In contrast, our architecture is able to process a large amount of data coming from the different sensors in real time.

Existing ADAS systems can be classified according to the following two criteria:

1) According to the used sensor(s) and the provided driver assistance functions, we have the following.

   Different types of sensors and data sources have been used, such as Lidar, camera, radar, vehicle-to-vehicle, and/or vehicle-to-infrastructure (V2I) communications. The European Project PREVENT [20] was aimed at developing preventive and corrective safety systems for automotive applications. This project was established to create electronic safety zones around vehicles by developing and demonstrating a set of complementary safety functions. It is based on laser scanners, video cameras, and bidirectional V2I communications. In [2]–[5], the authors proposed several ADAS systems for obstacle recognition and tracking. In these papers, the proposed architectures use a radar sensor and enables the ADAS, in addition to detecting obstacles, to recognize and categorize these obstacles by using their radar signatures [1]. Due to its large field of view, radar sensors allow for detecting obstacles at longer distances and, consequently, ensure longer reaction time for vehicle drivers. Moreover, the radar system behaves better in bad visibility conditions (foggy weather, rain, snow, etc.) and has lower computational requirements when compared with camera-based ADAS. For this reason, we develop our ADAS-MTT system based on radar technology.

2) According to their embedded system and hardware architectures, we have the following.

   ImapCAR [6] and EyeQ2 [7] systems are two examples of a fully programmable MPSoC that are dedicated to automotive security applications using a vision system. The ImapCar adopts a single-instruction multiple-data (SIMD) architecture of 128 processing elements and a four-way very long instruction word (VLIW) control processor. The EyeQ2 [7], which is a single chip with a monoscopic embedded vision system, consists of two 64-bit floating-point RISC processors for scheduling and controlling the concurrent tasks, five vision computing engines, three vector microcode processors, and eight processors for vision and vector processing. These two architectures provide support for a specific set of real-time data-intensive applications. Therefore, these systems are unable either to accommodate new applications or to adapt the hardware to different scenarios. The vision-based ADAS architectures used in [21] and [22] use the same approach. A multicore processor to provide adaptability to various applications is combined with accelerators to realize image processing/recognition tasks at high performance and low power consumption. The AutoVision processor [8] is a dynamically reconfigurable MPSoC prototype for video-specific pixel processing. Pixel processing engines offer functions such as object edge detection or luminance segmentation and are implemented as dedicated hardware accelerators to ensure real-time processing.

   In [3] and [4], the authors proposed an FPGA-based MPSoC architecture for their ADAS. The role of the MTT, in these designs and the present work, is to balance the inaccuracy of the low-cost radars by the utilization of data tracking and filtering processes. The architecture contains up to 23 Altera NIOS II soft cores, which makes

TABLE I
COMPARING EXISTING WORKS TO OUR PROJECT. RR STANDS FOR RECONFIGURABLE REGIONS

	Sensor	Architecture	ADAS function	Dynamic Reconfiguration	Exploration
Liu [1]	Radar	1 soft core + 1RR	Obstacle recogn.	No	No
Harb [2]	Radar	1 soft core + #RR	MTT	Yes	No
Khan [3][4]	Radar	1 to 22 soft cores	MTT	No	Cache archit.
Khan [5]	Radar	1 to 5 soft cores	MTT	No	Cache archit. and float. point instruct.
IMAPCAR [6]	Camera	SIMD, VLIW	Lane and sign recogn.	No	No
EyeQ [7]	Camera	MPSoC	lane and vehicle detect.	No	No
Claus [8]	Camera	2 soft Cores + #RR	Obstacle recogn.	No	No
Harb [9]	Radar	1 soft core + 1 RR	MTT	Yes	No
Shreejith [10]	Camera	1 soft core + 1 RR	Cruise Control, Park assist.	Yes	No
Lange[11]	Radar	1 soft core	MTT	No	Simplified KF and Munkres
Our work	Radar(s)	1 hard core + #RR	MTT	No	KF, GC, CG and Munkres

the architectures much more complex when compared with our architecture. In addition, the cumulative execution time of the whole MTT in their design exceeds the radar pulse repetition time (PRT), i.e., 25 ms in their implementation. In [5], the complexity of the architecture has been reduced by merging the 20 soft-core processors, implementing the Kalman filter (KF) function, into a single core. In [11], Lange *et al.* proposed to use precalculated values for the KF. This approach simplifies the calculation but reduces the system accuracy when the ADAS has to be used in different scenarios (urban, highway, etc.). In contrast to these solutions, our solution permits the processing of a large number of obstacles in a reduced time using one hard core and several HW accelerators, allowing to process 100 scans in 14 ms. In addition, it is the first time implementations of KF, Gate Checker (GC), Cost Generator (CG), and Munkres (MUN) components on HW are explored. In [2], [4], and [9], the MTT architectures have been extended along two dimensions. First, an HW accelerator for the KFs was proposed to reduce the complexity of the system; second, the dynamic and partial reconfiguration feature of Xilinx FPGA was used in the ADAS system to adapt the architecture when changing driving scenarios, for example, when moving from urban to suburban, or *vice versa*. Our work is complementary to the work in [2] and [9] as we explore different configurations. The partial dynamic reconfiguration for automotive applications has also been proposed in [8]. In this project, three different situations are considered: highway, tunnel entrance, and inside tunnel. For each situation, a given hardwired coprocessor must be loaded and mapped on the FPGA. In [10], Shreejith *et al.* proposed the utilization of the dynamic partial reconfiguration for implementing a cruise control application with an intelligent parking assistant system. At the opposite end from the ADAS in [2], mode switch is triggered by user commands in [10].

In comparison to the previous cited works, our work is the first that explores different configurations for the different MTT blocks. Our work differs also from existing works by the fact that our design space exploration is for a heterogeneous platform that contains a hard-core processor and programmable-logic elements. When the number of sensors increases, the MTT must process an increasing number of obstacle tracks in a short execution time. For this reason, our paper presents a

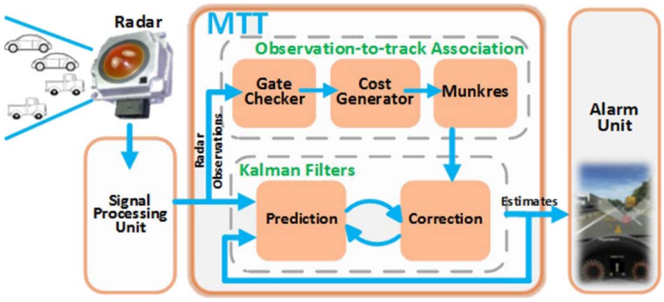

Fig. 1. Block diagram of ADAS.

Fig. 2. Car with AC10 radar.

methodology that is able to explore different MTT configurations and provides a scalable model to estimate the execution time and energy consumption for the MTT application with different numbers of hardware blocks. It is the first time that an analytical model for execution time and energy consumption is proposed for MTT applications. In our approach, we consider both existing low-cost radar-based MTT with a reduced number of tracks and next-generation radars allowing to track a large number of obstacles. Table I summarizes the previous projects and compares them to the work presented in this paper.

III. MULTIPLE-TARGET TRACKING ARCHITECTURE

Fig. 1 shows the block diagram of a typical ADAS. The ADAS consists of a radar, a signal processing unit (SPU), an MTT unit, and an alarm unit (AU).

The ADAS used in our experiments utilizes AC10 radars [23] that were installed in front of our host vehicle, as shown in Fig. 2. Each AC10 can detect a maximum of ten targets (obstacles) in each radar scan. The radar PRT is the time

TABLE II
AC10 RADAR CHARACTERISTICS

Variable	Range
Distance	0-200 m
Speed	30 to 100 kmph
Angle	$\pm$ 12^0
Angular Speed	0 to 2 (rad/s)

duration between two successive radar scans. The PRT for AC10 is 20 ms and corresponds to the time window within which the tracking system must complete the processing of the information received during a scan. For a TRW AC10, one scan of ten obstacles is done every PRT at a maximum of 200 m ahead and within the coverage angle of $-/+12^{\circ}$. Radar characteristics are shown in Table II. In next radar generation, such as the TRW AC100, up to 100 data from 100 different obstacles can be collected every 20 ms. Our data have been collected in real urban scenarios. For each AC10 radar, ten obstacle positions (distance and angle) have been recorded in the span of about 4 min and correspond to a total of 10 K scans. When the number of obstacles exceeds ten in a scan, only the ten closest obstacles to the radar are recorded. These ten obstacles are considered as the most dangerous. In our MTT-based ADAS, the embedded system is able to support additional short-range radars on the sides of the vehicle.

When the host vehicle moves, the radar detects obstacles around it and sends the radar raw data (distance and angle) to the SPU (sampling, analog-to-digital conversion, data preprocessing, etc.). After processing the radar raw data, the SPU sends measurement data (observations) to the MTT unit. In this step, MTT tracks multiple obstacles based on their previous observations and produces obstacles' estimated distance, speed, angle, and angular speed to the AU. The AU shows the states of the obstacles to the users and raises an alarm if any dangerous traffic situations arise based on the estimates provided by MTT. The three units SPU, MTT, and AU work in a pipelined manner. While the SPU is preparing measurement data (observations) of scan $(i+1)$, the MTT processes the data for scan (i), and the AU informs the driver (if needed) about the results of scan $(i-1)$.

A. MTT Architecture

The MTT system is a key component of ADAS above, which processes multiple targets' observations obtained from the radar during each scan. In each scan, the MTT system goes through four major functions shown in Fig. 1: 1) **Kalman Filters (KFs)**, 2) **Gate Checker (GC)**, 3) **Cost Generator (CG)**, 4) **Munkres (MUN)**.

In each radar scan, MTT first obtains ten targets' observations from the SPU and sends them to the observation-to-track association block in Fig. 1. The GC and the CG in the observation-to-track association block together determine which observation-to-prediction pairings are probable. A single observation may be paired with several predictions, and *vice versa*. After these two functions, it will output a matrix, i.e., *cost matrix*, describing all possible association relationships with weights/cost. In the subsequent step, the MUN function takes the *cost matrix* as input and solves the assignment problem to identify the best associations for each observation to the appropriate KF.

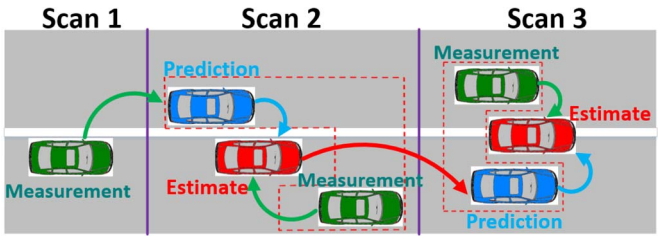

Fig. 3. Brief explanation for the KF.

According to the associations previously obtained, each KF takes its previous estimates from the last radar scan and the current observation as inputs. The filter output is an estimated result and predicted error covariance that is used by the observation-to-track association block for the subsequent radar scan. Finally, after ten KFs finish in sequence, the alarm system obtains ten estimates for ten obstacles detected by the radar and reacts according to the estimation. For example, the AU raises an alarm if it detects that traffic collision may happen. The following sections describe the key components of MTT in further detail.

B. Kalman Filters (KFs)

A radar sensor is typically affected by Gaussian noise, and therefore, we employ the KF that is suitable for such noisy environment. The mathematical computations of KF are shown in the following:

$$\begin{cases} Y_k^p = AY_{k-1}^e & (1) \\ E_k^p = AE_{k-1}^e A^T + Q & (2) \\ K = E_k^p H^T \left(HE_k^p H^T + R\right)^{-1} & (3) \\ Y_k^e = Y_k^p + K\left(Z_k - HY_k^p\right) & (4) \\ E_k^e = \left(I - KH\right) E_k^p. & (5) \end{cases}$$

Here, Y_k^p and Y_k^e are a prediction state matrix and an estimation state matrix including (distance, speed, angle, and angular speed) at radar scan k, respectively; similarly, E_k^p is a prediction error covariance matrix at radar scan k, whereas E_k^e is an estimation error covariance matrix; A is an $n \times n$ state transition matrix, which relates the state at scan $k-1$ to the state at scan k; Q is the process noise covariance matrix; R is the measurement noise covariance matrix, which depends on the characteristics of a radar; Z_k is a measurement state matrix consisting of measured distance d and angle θ; H is a measurement relation matrix that relates the current estimation state matrix Y_k^e to the measurement (observation) matrix Z_k; K is the Kalman gain matrix; and I is an identity matrix. More details can be found in [3], [4], and [24].

Fig. 3 in [3] shows an example of the KF. A KF takes an actual (measured) state (*distance*, *speed*, *angle*, and *angular speed*) and a previous estimate as its input in scan 2 and provides a new weighted estimate in the next radar scan, i.e., scan 3.

The number of KFs employed in MTT is related to the feature of radar used in ADAS. That is, the maximum number of targets that a radar can detect determines the number of KFs in MTT.

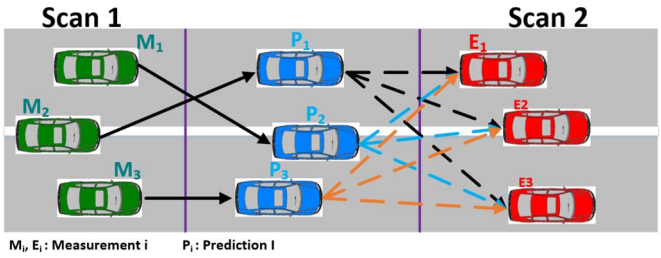

Fig. 4. Association problem.

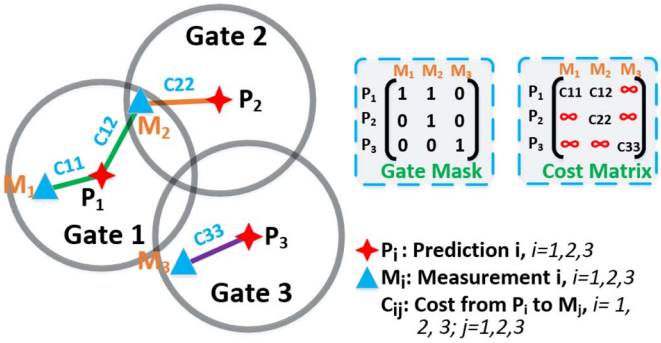

Fig. 5. Example for conflict situation in data association.

In our case, since ADAS employs TRW's AC10 radar [23], which can detect a maximum of ten targets in each scan, the KF block in Fig. 1 uses ten independent KFs. It should also be noted that if the MTT application is executed on a single processor core, then the ten KFs have to be executed in a sequential order. On the other hand, in the presence of multicore processors or multiple KF hardware accelerators, the KF block can be executed in parallel for each obstacle. In Section V, we exploit this feature of the KF blocks by using multiple hardware accelerators to speed up the execution time.

Fig. 4 shows that each KF tracks a car, i.e., *Green Car* M_i, and predicts its position, i.e., *Blue Car* P_i, for the next scan. In the subsequent scan, the predicted positions are associated to the observed positions, i.e., *Red Car* E_i, obtained from the radar. To solve this association problem, the MTT system incorporates the GC, CG, and MUN functions. The following sections describe these components in MTT.

C. Gate Checker (GC)

The GC is a small functional block, but it is the most frequently executed component in the MTT application. It determines which measurement-to-prediction pairings are probable and produces the gate mask matrix G. Fig. 5 shows the basic principle for GC with an example of tracking three targets. The red stars in Fig. 5 are predictions, i.e., P_i, that the GC receives from KFs in the last radar scan, whereas the blue triangles, i.e., M_i, are measurements in the current radar scan. First, the GC calculates the corresponding gate windows (Gates 1–3) for each prediction. Radius r of each gate window is obtained via the same method in [24]. Second, it checks whether measurements fall inside gates. If measurement "i" resides in gate "j" of prediction "j," then an edge will be added to connect measurement "i" and prediction "j," which is shown in Fig. 5, and the GC sets the element g_{ij} of G to 1. Otherwise, the GC sets g_{ij} to 0. After these two steps, the GC produces a gate_mask matrix G, where each row represents the predictions obtained from a KF, and each column represents potential associations between measurements and predictions.

It is possible that after GC, multiple measurements are associated to a prediction. This situation is shown in Fig. 5 for prediction P_1. Both M_1 and M_2 are possibly associated with P_1. However, in a real MTT system, we only allow one measurement associated to a prediction. To avoid such conflicts, we need to convert the problem into an assignment problem with weights added to each edge.

D. Cost Generator (CG)

The CG is also a relatively small application segment that is called frequently during the execution of MTT application. It converts the association problem in the GC into an assignment problem via assigning weight/cost to edges and produces a cost matrix C. In the CG step, it first takes the gate mask matrix G in the previous GC step as an input and calculates each element c_{ij} of C via the following equations:

$$c_{ij} = \begin{cases} \infty, & \text{if } g_{ij} \text{ is } 0 \quad (6a) \\ d_{ij}^2, & \text{if } g_{ij} \text{ is } 1 \quad (6b) \end{cases}$$

where d_{ij}^2 is an assignment cost, which is the statistical distance between measurement i and prediction j, and $g_{ij} \in G$. Detailed calculation of d_{ij}^2 could be found in [24].

Fig. 5 shows an example of the CG. It assigns weight/cost $C11$, $C12$, $C22$, and $C33$ to those edges that have g_{ij} equal to 1. Then, the CG outputs a cost matrix that indicates the cost for all possible associations to the assignment solver, i.e., MUN block, as an assignment problem.

E. Munkres (MUN)

The MUN algorithm is employed in MTT application as an assignment solver that determines the final one-to-one pairings of measurements and predictions. It takes the cost matrix C obtained from the CG as an input and provides the optimal associations between m measurements and n predictions with the lowest total cost sum. Equations are shown as follows:

$$\text{Minimize} \sum = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \quad (7)$$

$$\text{subject to} \begin{cases} \sum_{i=1} x_{ij} = 1 \quad \forall j & (8) \\ \sum_{j=1} x_{ij} = 1 \quad \forall i & (9) \\ c_{ij} \in C & (10) \\ x_{ij} \in X. & (11) \end{cases}$$

The output will be a one-to-one assignment mask matrix X indicating one measurement to one existing predicted target mapping. The MUN block is a control-intensive block. More details on MUN algorithms can be found in [3] and [24].

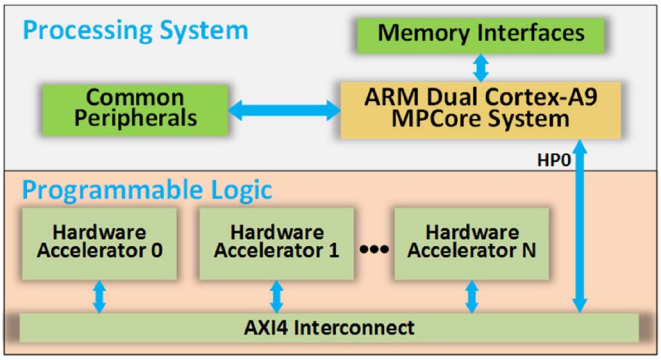

Fig. 6. Simplified block diagram of ZYNQ architecture.

IV. Implementation Platform

To explore the various hardware accelerators that can be efficiently used for the MTT application, a heterogeneous platform is used as it can provide for efficient hardware–software implementation. Such platforms typically contain a processor for executing the software segments of the application and other processing elements that can be used to accelerate the critical components in the application. FPGAs have been widely used in the past to implement the hardware accelerators for applications. The Zynq-7000 SoC [14] is such a heterogeneous platform consisting of a processing system (PS) and a programmable logic (PL), as shown in Fig. 6.

A. ZYNQ Architecture

The target Zynq platform for implementing the MTT application is a commercial off-the-shelf Xilinx ZC702 Evaluation Kit that contains a Zynq-7000 all-programmable SoC [25]. The PS contains an ARM Dual Cortex-A9 MPCore system, memory interfaces (DDR3 controller), and common peripherals (GPIO, UART, USB, Network, SD, etc.), whereas the PL is a Xilinx Artix-7 fabric containing 220 DSPs, 280 block RAMs (BRAMs), 13 300 slices, 53 200 LUTs, and 106 400 registers. The communication between PS and PL is achieved by Advanced eXtensible Interface (AXI) interconnection. In our MTT implementation, hardware accelerators are attached to the AXI4 interconnection via the AXI master interface and connect to PS by its high-performance AXI ports (HP0–HP3).

B. MTT System Setup

Based on initial profiling results, we partition MTT into software and hardware segments. The software segment runs on a single ARM Cortex A9 processor at 666.67 MHz on the Zynq platform. The hardware accelerators are generated by Xilinx Vivado HLS [26]. Vivado HLS is a high-level synthesis tool provided by Xilinx, which accepts C/C++ and SystemC languages and automatically generates corresponding Verilog/VHDL codes. In this paper, we manually inserted various pragma settings such as loop unrolling, loop pipelining, array partitioning, etc., in the MTT application to form a design space. The different pragma settings are used to explore the design space extensively and, therefore, provide more design choices for the designer. The maximum frequency achieved for the hardware accelerators as generated by Vivado HLS lies between 71 and 75 MHz, and we fix the frequency at 70 MHz for all the hardware components utilized in our system for simplicity.

The interface for all hardware accelerators attached to AXI4 interconnection is an AXI4 master interface that is capable of burst read/write operations. The required memory bandwidth for MTT application was measured to be 13.3 MB/s, which is much smaller than the maximum memory bandwidth of DDR3 Xilinx ZC702 (4.264 GB/s). This ensures that sufficient memory bandwidth is available for data transfer between the hardware and software sections of the application.

TABLE III
Precision Error of 32-bit/16-bit MTT Implementations

	32-bit MTT		16-bit MTT	
	angle	distance	angle	distance
precision error(%)	0.96758	0.17398	1.78447	0.28767

C. Performance and Power Measurement

For performance measurement, we employ a global timer inside the ARM Cortex-A9 processor to measure the performance of MTT designs. The global timer is a 64-bit incrementing counter with an autoincrementing feature running at half of the CPU's clock frequency [14]. In our system, the timer is clocked at 333.33 MHz.

For power measurement, we utilize four digital powertrain modules from Texas Instruments provided by the Xilinx ZC702 platform to measure voltage, current, power, and temperature at runtime [25]. In our system, the power consumption of PL, BRAM, PS, DDR3, and other peripherals such as USB, SD, etc., has been measured using the different rails that are available in the Xilinx ZC702 platform [25].

V. Design Space Exploration

Here, we explore the MTT partitioning design space to find the various design points with tradeoffs in terms of performance, energy efficiency, and FPGA resource utilization. The execution time and energy results of the following sections are based on 100 consecutive radar scans.

A. Application Profiling

As a first step in the design space exploration process, the application is profiled on the target processor to identify its hotspots. To efficiently utilize the available resources, we converted the MTT application into a 32-bit fixed-point (14-bit integer and 17-bit fraction) format. The precision error in moving from a floating point to a 32-bit fixed-point format was calculated and shown in Table III. It is evident that the loss of precision is not significant and, hence, can be safely ignored.

The fixed-point implementation of the MTT application was then mapped onto a single ARM Cortex A9 processor core of the Zynq platform. The CPU core runs at 666.67 MHz, whereas the application execution time is measured via the global timer

TABLE IV
EXECUTION TIME BREAKDOWN OF THE MTT APPLICATION

Breakdown	KFs	GC	CG	Munkres	Others
Percentage (%)	71	12	6	7	4

running at 333.33 MHz (half CPU clock frequency), as discussed in the previous section [14].

As can be seen from Table IV, the KF block is the most time-consuming function of the MTT application, taking up to 71% of the total execution time and, hence, a suitable candidate for hardware acceleration. A closer inspection of this block reveals extensive matrix operations that can take advantage of the DSP blocks in the FPGA fabric to accelerate computations.

On the other hand, while the GC and CG functions are relatively smaller components in the application, the total time required by these blocks account for 18% of the total execution time. This behavior is explained by studying the frequency profile of the MTT application that shows these functions as most frequently executed blocks.

The last block with a significant execution time is a control-intensive MUN function, requiring 7% of the total time spent in MTT application.

In the subsequent step, we explore accelerating various combinations of these blocks in hardware to obtain different performance, energy efficiency, and resource utilization. Different bit widths were also tried to achieve maximum performance without sacrificing the quality of results.

B. 32-bit Fixed-Point Implementations

Guided by the profile results previously discussed, we first attempted to accelerate the KF function in a 32-bit fixed-point (Q14.17) format. As discussed in Section III, the number of invocation of the KF block depends on the number of targets tracked by the radar. A typical radar, such as that used for our experiments, tracks between 10 and 20 targets during each scan. Currently, we focus on a radar that tracks ten targets per scan, thereby requiring ten calls, once for each obstacle, to the KF function per radar scan. This processing can be performed independently for every obstacle since there is no data dependence between the KFs for different obstacles. In other words, if there are ten hardware blocks available for the KF function, then all the ten iterations can be completed in parallel by exploiting the loop-level parallelism, and total time taken would theoretically be equal to time taken for a single KF processing. Hence, loop-level parallelism can be effectively exploited here to maximize the advantage of hardware implementation.

Fig. 7 shows the execution time of MTT application with different numbers of hardware blocks implementing the KF. As expected, the MTT configuration with more number of hardware KF blocks achieves the highest performance with resource utilization of 71% LUTs, 82% slices, and 80% DSP48E1s in the target device. However, the high resource utilization for three KF blocks limits the ability of implementing more than three 32-bit KF hardware accelerators. In addition, the speedups obtained from the three configurations are not significant. This is caused by the frequency gap between the ARM processor (666.67 MHz) and hardware components (70 MHz).

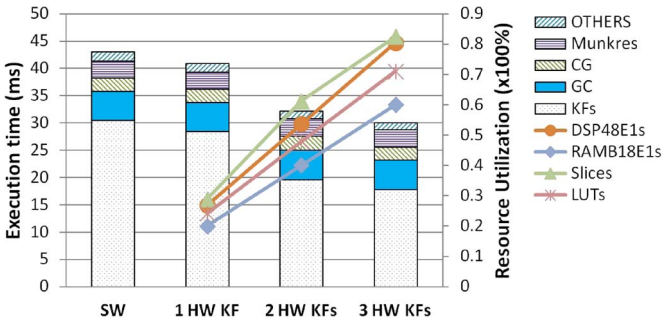

Fig. 7. Execution time and resource utilization for 32-bit KF hardware implementations.

To further accelerate the MTT application, we reduce the resource utilization of hardware accelerators by reducing their bit width. Hence, we employ the 16-bit fixed-point (Q8.7 format) KF hardware in the following section instead of 32-bit KFs and show that the resulting loss of precision is low enough to be acceptable.

C. 16-bit Fixed-Point Implementations

Table III shows the average precision error introduced by reducing the bit width from 32-bit fixed to 16-bit fixed point. The acceptably low precision error allows us to reduce the bit width from 32 bits to 16 bits, freeing up resources for hardware acceleration of more parts of the MTT application.

Fig. 8(a) shows the execution time and resource utilization for different 16-bit implementations. As is evident in the figure, MTT with one 16-bit KF can achieve performance close to that of MTT with three 32-bit KFs with less than 20% utilization of all resources. The result is due to the fact that less bit-width implementations utilize smaller hardware components that can run faster and consume less hardware resources compared with that with higher bit width.

It is also notable that the execution time of KFs significantly drops from MTT with one 16-bit KF to MTT with five 16-bit KFs; however, the descending trend slows down particularly from five KFs to seven KFs. This behavior can be explained as follows: In the case of a single KF hardware block, to process ten iterations of the KF in each radar scan, MTT needs to run all the ten iterations sequentially on the single KF hardware block. This improves the execution time of the KF with respect to the software-only execution due to acceleration in hardware. However, loop-level parallelism as explained in the previous section has not been exploited in this case. In a different scenario with two KF hardware blocks, two iterations of KF can be performed in parallel. In essence, this configuration with two KF hardware blocks can finish the ten iterations of the KF blocks in a time equivalent to five iterations, thereby theoretically reducing the runtime by almost 50%. In practice, however, the reduction in execution time is close 30% for this hardware block due to additional overhead of data communication, synchronization, etc. Moreover, when the number of KF hardware blocks is increased beyond five, the advantage of loop-level parallelism starts to shrink for implementing ten iterations of the KF function. The reduction in execution time of the application also

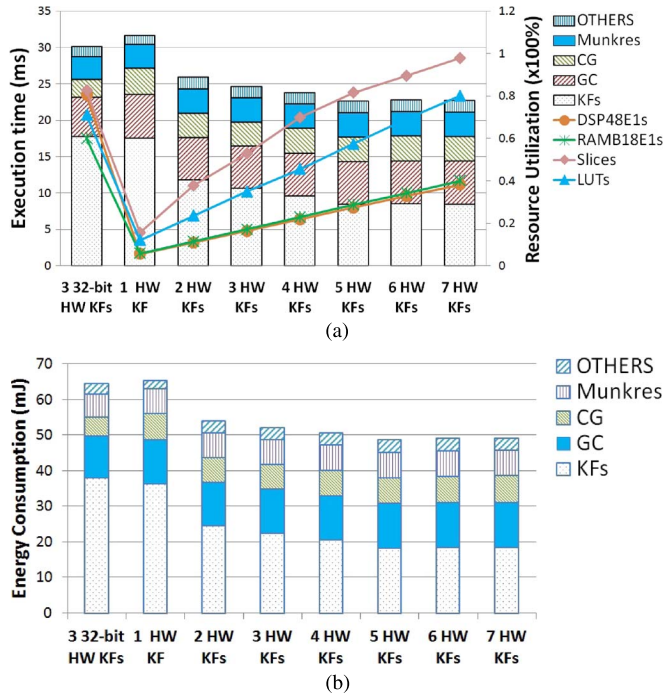

Fig. 8. Execution time, resource utilization, and energy consumption comparison for 16-bit KF hardware implementations. (a) Execution time and resource utilization. (b) Energy consumption.

translates to the reduction in energy consumption of different 16-bit implementations, as shown in Fig. 8(b). The reduction in energy consumption of the KF hardware component alone, going from one KF block to five, is close to 50%. Moreover, the MTT application with five 16-bit KFs can save over 24% energy when compared with the MTT implementation with three 32-bit KFs due to the reduction in execution time of the application. This leads us to conclude that while tracking ten targets, requiring ten iterations of Kalman filtering, MTT with five 16-bit KF hardware blocks achieves the best efficiency in terms of performance and energy.

In the following section, we explore the acceleration of other functions in the MTT application such as GC, CG, and MUN, which together account for nearly 25% of runtime. It is necessary to reduce the runtime of these parts to obtain higher performance.

D. Optimization for Other Components

We accelerate the GC, CG, and MUN components of the MTT application to identify their contribution in the entire system. The output of GC is fed directly as the input to CG, resulting in strong data dependence and good candidates for simultaneous acceleration as a single hardware block. Prakash *et al.* in [27] showed the advantages of accelerating code segments with mutual data dependence relations simultaneously in hardware to reduce the communication cost between CPU and hardware accelerators.

Fig. 9 shows the execution time and energy consumption comparison among pure-software MTT, MTT with one GC+CG accelerator, and MTT with one HW accelerator for the

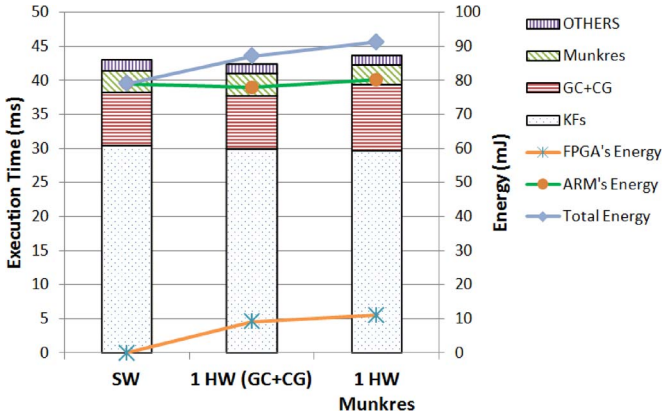

Fig. 9. Execution time and energy consumption comparison.

MUN component. As shown in this figure, the reduction in the execution time, with GC+CG components accelerated in hardware, is not significant while the energy consumption increases due to the inclusion of FPGA logic. The less-than-expected speedup achieved by accelerating the GC+CG component can be attributed to the high data dependence between this component and the rest of the MTT application. Therefore, the high communication cost of transferring data between the CPU and FPGA overshoots the advantage of accelerating GC+CG in hardware.

Similar to the behavior of GC+CG hardware block, the reduction in the execution time with the MUN function accelerated in hardware is also not significant. Even so, the energy consumption significantly rises compared with its software implementation. The control-intensive MUN function does not benefit sufficiently from hardware acceleration, particularly since the hardware accelerated MUN function runs at 70 MHz in our platform (the maximum frequency of hardware MUN is 74 MHz), whereas the highly efficient ARM CPU has a clock frequency of 666.67 MHz. This indicates that the MUN function is not suitable for hardware acceleration in the MTT application, particularly on a heterogeneous platform such as Zynq with a high-performance CPU.

E. Communication Cost Reduction

In the previous section, we briefly discussed the implications of data dependence relations between the various sections of the application code and the resulting communication overheads during hardware acceleration. To reduce this communication cost, it is evident that all the functions with mutual data dependence relations should be simultaneously executed in hardware [27]. Hence, we integrated KFs and GC+CG functions into a single hardware block to reduce the communication cost between KF and GC+CG blocks. Fig. 10(a) shows the execution time for 100 radar scans of the entire MTT application and the resource utilization for implementing hardware blocks with combined KF, GC, and CG.

It is evident in this figure that the total execution time of KF+GC+CG functions of MTT with one (KF+GC+CG) hardware block is 25 ms, whereas MTT with five KF hardware blocks consumes only 23 ms. However, it should be observed

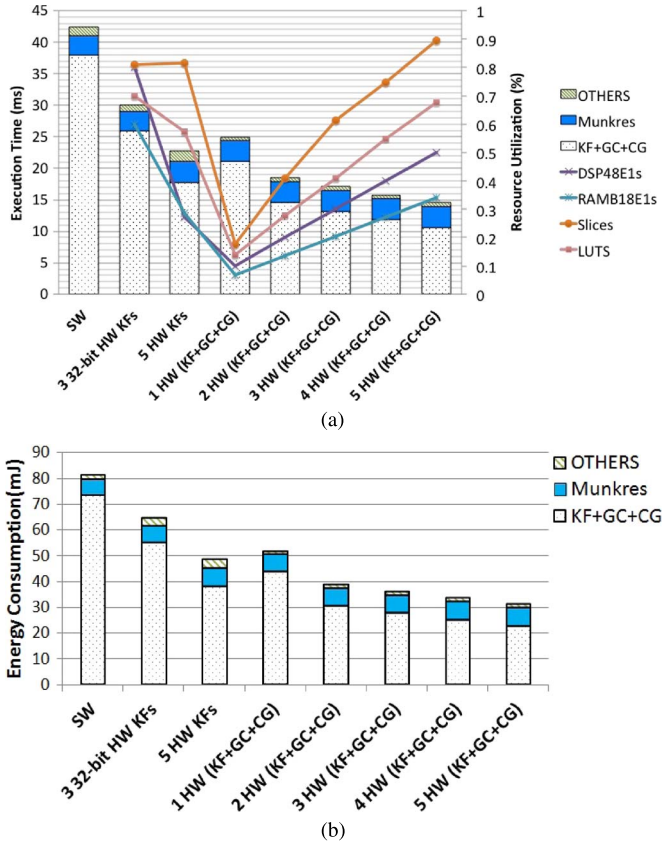

Fig. 10. Execution time, resource utilization, and energy consumption comparison for 16-bit KF+GC+CG hardware implementations. (a) Execution time and resource utilization. (b) Energy consumption.

that the resource utilization for MTT with one KF+GC+CG is significantly reduced compared with that of MTT with five KFs. This implies that MTT can employ more (KF+GC+CG) hardware components to further reduce its execution time. The trend is shown in Fig. 10(a). The implementation, MTT with five KF+GC+CG, achieves the highest performance with execution time of less than 15 ms for 100 radar scans.

Fig. 10(b) shows the energy consumption for different MTT implementations. After reducing the communication cost between KFs and GC+CG, MTT with five (KF+GC+CG) can save more than 65% energy compared with pure-software MTT. Hence, the design point, MTT with five KF+GC+CG hardware blocks, can achieve not only high performance but also high energy efficiency.

Here, we explored various design options for partitioning the MTT application between the high-performance ARM processor core and FPGA fabric on a modern heterogeneous platform. Various parameters such as frequency, code size, data dependence relations, etc., were considered during design space exploration, to produce multiple design points with a tradeoff between performance, area, and energy efficiency. In particular, it was observed that it is essential to reduce the communication cost between the application partitions being executed on CPU and FPGA to maximally exploit such heterogeneous systems. In case of the MTT application, the optimum partition was achieved by mapping the combination of KF, GC, and CG as a single hardware block to reduce the communication cost between the hardware and the rest of the application running on the ARM processor. Moreover, multiple instances of the hardware block were implemented in hardware to exploit the loop-level parallelism in the MTT application.

VI. Performance—Energy Estimation Model

In this paper, we performed an extensive case study of the MTT application on the heterogeneous Zynq platform, to identify the optimum hardware configuration that minimizes execution time and energy consumption. Based on this study, we propose a scalable model to estimate the execution time and energy consumption of implementing the MTT application with different numbers of hardware blocks. Additionally, a model to estimate the execution time and energy consumption of implementing the MTT application with varying numbers of targets being tracked has also been derived.

1) Varying Numbers of Hardware Accelerators: In the previous section, it was established that to reduce the communication cost between CPU and FPGA, the KF, GC, and CG functions should be simultaneously accelerated as a single hardware component. Therefore, we assume this combination to be the optimum hardware accelerator for this application. To estimate the execution time with varying numbers of hardware accelerators, a one-time characterization step was performed. In this step, we implemented different numbers of hardware accelerators (KF, GC, and CG together) functions and measured their execution time and energy requirement on the Zynq platform, as shown in Fig. 10.

Based on these observations, curve fitting was used to generate an estimation model for the execution time of the MTT application with varying numbers of hardware accelerators comprising KF, GC, and CG functions. Equation (12), shown below, estimates the time for 100 scans containing ten obstacles/scan as

$$\text{Total Time (us)} = 25066 - 1384 * \left(10 - \left\lceil \left(\frac{10}{N}\right) \right\rceil\right) \quad (12)$$

where N is the number of (KF+GC+CG) hardware accelerators for a system tracking ten obstacles/scan. As discussed in Section V-C, the rate of reduction in execution time of the MTT application reduces with increasing number of hardware accelerators. This reduction was attributed to the characteristics of the MTT application, which requires ten calls to the KF+GC+CG hardware accelerator to track ten targets. Equation (10) captures this by effectively using the *ceil* function to obtain the number of calls required, when the number of hardware accelerators varies from 1 to 10.

Similarly, extrapolating for energy consumption, (13) provides an estimation model for energy consumption with N number of KF+GC+CG hardware accelerators, i.e.,

$$\text{Total Energy (uJ)} = \text{Total Time} * (2.075 + (N-1) * 0.018). \quad (13)$$

The models presented in this section were verified against the data obtained from real implementation and were seen to be within 5% error margin for all design points.

TABLE V
ACCURACY OF EXECUTION TIME AND ENERGY ESTIMATION MODEL

	N=1, M=20		N=5, M=20	
	Observed	Estimated	Observed	Estimated
Execution Time (ms)	61.61	62.54	38.34	40.40
Energy (mJ)	127.84075	129.78	82.2393	86.74

2) Varying Number of Objects Tracked: The number of obstacles being tracked in the current MTT system has been considered to be ten. This number is based on the features of the radar currently used for our experiments. However, in the future, the number of obstacles tracked by the radar will increase to enable more robust analysis of the surroundings. The new TRW AC1000 short-distance radar allows a 360° environmental sensing. Multiple TRW AC1000 can be mounted on the car to detect a high number of obstacles. This functionality is very useful in an urban situation [13]. Therefore, it will be useful to rapidly estimate the performance and energy requirements of implementing the MTT application that tracks varying number of targets while using different numbers of hardware accelerators.

To cater for such scenario, we extended (12) to estimate the execution time of the MTT application with varying number of targets. Due to the different scalability factors for the KF, GC+CG, and MUN, the execution time with varying number of tracked targets cannot be directly scaled from (12) by multiplying with the number of tracked targets. Instead, the execution time has been calculated independently for the different kernels.

To obtain the new execution time for the individual kernels for a given number of tracked targets, we computed the average execution time of each kernel for one target, in the first step. This was calculated from the extensive experimental results obtained during this work. In the next step, we analyzed these kernels to understand their behavior and obtain their scalability factor. The presence of 2-D matrix operations in the GC+CG, as well as the MUN kernels, necessitate the scalability factor to be M^2, where M is the number of obstacles tracked by the MTT application. On the other hand, the execution time of the KF kernel scales linearly with the number of obstacles. Additionally, the communication cost of transferring data between the hardware and software sections after the partitioning step was also considered by analyzing the amount of data transfer required per obstacle and the time taken to transfer this data. Equation (14), shown below, can be used to rapidly estimate the execution time of the MTT application that tracks M obstacles per scan with N KF+GC+CG hardware accelerators, i.e.,

$$\text{Total Time (us)} = \underbrace{\left[1137 - 138.4 * \left(10 - \left\lceil \left(\frac{M}{N}\right) \right\rceil\right)\right] * M}_{\text{KF}} + \underbrace{48.12 * M^2}_{\text{GC+CG}} + \underbrace{(62 + 3 * M) * 60}_{\text{Communication Cost}} + \underbrace{\left(\frac{M}{10}\right)^2 * 3308}_{\text{MUN Cost}}. \quad (14)$$

The total energy consumption can still be calculated by using (13) substituting for the new execution time. Table V shows the execution time and energy estimation results for 20 obstacles with different numbers of hardware blocks.

VII. CONCLUSION

In this paper, we have performed an extensive design space exploration of the MTT system, which is used widely in ADAS, on a modern heterogeneous SoC platform integrating a high-performance processor and a configurable FPGA fabric. Numerous hardware/software partitioning decisions were explored to recommend the optimal configuration with high performance and energy efficiency. Data dependence between the various functions was taken into consideration to decide on the optimal hardware–software partition, whereas loop-level parallelism was exploited to achieve high performance, as well as energy efficiency of more than 65%.

In addition, we also proposed a scalable model to estimate the execution time and energy consumption of implementing the MTT application with different numbers of hardware blocks. To cater for future devices capable of tracking more number of obstacles, a model to estimate the execution time and energy consumption of implementing the MTT application, with varying number of targets being tracked, has also been derived. In the future, we plan to extend this work by exploring and comparing the design space of the MTT application as well as the other ADAS applications on other high-performance yet low-power embedded heterogeneous SoCs such as those used in modern smartphones and tablets. Such SoCs integrate multicore CPUs as well as GPUs to enable energy-efficient implementation of various ADAS applications of the future.

REFERENCES

[1] H. Liu and S. Niar, "Radar signature in multiple target tracking system for driver assistant application," in *Proc. DATE Conf. Exhib.*, Mar. 2013, pp. 887–892.
[2] N. Harb, S. Niar, M. Saghir, Y. Hillali, and R. Atitallah, "Dynamically reconfigurable architecture for a driver assistant system," in *Proc. IEEE 9th SASP*, Jun. 2011, pp. 62–65.
[3] J. Khan, "Embedded multiprocessor architectures for automotive driver assistance systems," Ph.D. dissertation, Dept. Comput. Sci. (LAMIH), Univ. Valenciennes, Valenciennes, France, 2009.
[4] J. Khan, S. Niar, A. Menhaj, Y. Elhillali, and J. L. Dekeyser, "An MPSoC architecture for the multiple target tracking application in driver assistant system," in *Proc. Int. Conf. Appl.-Spec. Syst., Architect. Process.*, 2008, pp. 126–131.
[5] J. Khan, S. Niar, A. Rivenq, and Y. El-Hillali, "Radar based collision avoidance system implementation in a reconfigurable MPSOC," in *Proc. Int. Conf. ITST*, 2009, pp. 586–591.
[6] H. Hiroto, "Automotive image recognition processor IMAPCAR," *NEC Tech. J.*, vol. 6, no. 5, pp. 24–28, Dec. 2010.
[7] G. Stein, E. Rushinek, G. Hayun, and A. Shashua, "A computer vision system on a chip: A case study from the automotive domain," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2005, p. 130.
[8] C. Claus, W. Stechele, and A. Herkersdorf, "Autovision: A run-time reconfigurable MPSoC architecture for future driver assistance systems," *IT—Inf. Technol.*, vol. 12, pp. 1624–1639, May 2007.
[9] N. Harb, S. Niar, and M. Saghir, "Dynamically reconfigurable embedded architectures for safe transportations systems," in *The Handbook of Research on Embedded Systems Design*, A. Bagnato, L. Soares, I. R. Quadri, and M. Rossi, Eds. Hershey, PA, USA: IGI Global, 2014, ch. 14.
[10] S. Shreejith, S. Fahmy, and M. Lukasiewycz, "Reconfigurable computing in next-generation automotive networks," *IEEE Embedded Syst. Lett.*, vol. 5, no. 1, pp. 12–15, Mar. 2013.
[11] T. Lange, N. Harb, H. Liu, S. Niar, and R. B. Atitallah, "An improved automotive multiple target tracking system design," in *Proc. 13th Euromicro Conf. DSD, Architect., Methods Tools*, 2010, pp. 255–258.
[12] J. Zhang *et al.*, "Data-driven intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1624–1639, Dec. 2011.
[13] TRW, Livonia, MI, USA, AC1000 Short Range Radar. [Online]. Available: http://www.trw.com/sites/default/files/TRW_ge_ac1000_en.pdf

[14] Zynq-7000 All Programmable SoC TRM, Xilinx Inc., San Jose, CA, USA, 2013.
[15] Cyclone V FPGAs, Altera Inc., San Jose, CA, USA, 2012.
[16] ARM Ltd. [Online]. Available: http://www.arm.com
[17] Xilinx Inc., San Jose, CA, USA, Zynq UltraScale+ MPSoC System Features. [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc
[18] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Norwood, MA, USA: Artech House, 1999.
[19] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Trans. Intell. Transp. Syst.*, vol. 4, no. 3, pp. 143–153, Sep. 2003.
[20] Prevent, "Prevent: European Project." [Online]. Available: http://www.prevent-ip.org/
[21] J. Tanabe *et al.*, "18.2 a 1.9TOPS and 564GOPS/W heterogeneous multi-core SoC with color-based object classification accelerator for image-recognition applications," in *Proc. IEEE ISSCC*, Feb. 2015, pp. 1–3.
[22] N. Ozaki *et al.*, "Implementation and evaluation of image recognition algorithm for an intelligent vehicle using heterogeneous multi-core SoC," in *Proc. 20th ASP-DAC*, Jan. 2015, pp. 410–415.
[23] TRW, Livonia, MI, USA. [Online]. Available: http://www.trw.com
[24] J. Khan, S. Niar, M. Saghir, Y. El-Hillali, and A. Rivencq-Menhaj, "Trade-off exploration for target tracking application in a customized multiprocessor architecture," *EURASIP J. Embedded Syst.*, vol. 2009, no. 1, Mar. 2010, Art. no. 175043.
[25] ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide, Xilinx Inc., San Jose, CA, USA, 2013.
[26] Vivado Design Suite User Guide: High-Level Synthesis, Xilinx Inc., San Jose, CA, USA, 2013.
[27] A. Prakash, S.-K. Lam, T. Srikanthan, and C. Clarke, "Modelling communication overhead for accessing local memories in hardware accelerators," in *Proc. IEEE 24th Int. Conf. ASAP*, Jun. 2013, pp. 31–34.

Guanwen Zhong received the Bachelor's degree in microelectronics from the School of Physics and Engineering, Sun Yat-sen University, Guangzhou, China, in 2012. He is currently working toward the Ph.D. degree with the School of Computing, National University of Singapore (NUS), Singapore.

After receiving the Bachelor's degree, he joined NUS. His major research focuses on decision models for mapping application kernels on embedded graphics processing unit and field-programmable gate array (FPGA)-based heterogeneous system-on-chip architectures. In addition, he is also working on design space exploration for mapping application kernels on FPGA using high-level synthesis.

Smail Niar (SM'15) received the Ph.D. degree in computer engineering from the University of Lille, Lille, France, in 1990.

Since then, he has been a Professor with the University of Valenciennes, Valenciennes, France, and a member of the Optimization and Mobility research group with the Laboratory of Automation, Mechanical and Computer Engineering, which is a joint research unit between the Centre National de la Recerche Scientifique and the University of Valenciennes. He is a member of the European Network of Excellence on High Performance and Embedded Architectures and Compilation. His research interests include multiprocessor system-on-chip (MPSoC) architectures, power/energy consumption optimization, dynamically reconfigurable embedded systems (field-programmable gate array), simulation acceleration techniques for MPSoC design space exploration, and reliability and security issues for intelligent transportation systems.

Alok Prakash (M'15) received the Ph.D. degree from Nanyang Technological University (NTU), Singapore, in 2014.

He is currently a Research Fellow with the School of Computer Engineering, NTU. Before his current assignment, he was a Research Fellow with the School of Computing, National University of Singapore, Singapore. Prior to that, he was an Application Programmer with IBM, Bangalore, India. His current research interests include power and thermal management techniques for mobile system-on-chip, customized application-specific platforms, custom instructions, and processor selection and customization.

Tulika Mitra (M'03) received the Ph.D. degree in computer science from the State University of New York at Stony Brook, Stony Brook, NY, USA, in 2000.

She is a Professor of computer science with the School of Computing, National University of Singapore, Singapore. She has authored more than 100 scientific publications. Her research interests include embedded real-time systems, energy-efficient computing, and heterogeneous computing systems.

Dr. Mitra serves on the Editorial Board and the Program Committee of several leading journals and conferences in her areas of interest.