Hardware acceleration of multibody simulations for real-time embedded applications

Antonio J. Rodriguez[1] Roland Pastorino[2]
Angel Carro-Lagoa[3] Karl Janssens[2] Miguel A. Naya[1]

[1] Laboratorio de Ingeniería Mecánica, University of A Coruña, Mendizábal, 15403 Ferrol, Spain
antonio.rodriguez.gonzalez, minaya@udc.es
[2] Test Division, Siemens Digital Industries Software, Interleuvenlaan 68, B-3001 Leuven, Belgium
roland.pastorino, karl.janssens@siemens.com
[3] Department of Computer Engineering & CITIC Research Center, University of A Coruña, Campus de Elviña, 15071 A Coruña, Spain
angel.carro@udc.es

Abstract

New products in the automotive and aerospace industries must provide increased energy efficiency and exceed previous performance, safety and reliability. To meet these expectations, the role of simulation continues to grow.

Within this context, simulation models are used in real-time embedded applications such as advanced real-time control and real-time virtual sensing. Both applications require the execution of simulation models in real-time on embedded hardware. The limited computational power of this hardware is however a major challenge in the adoption of model-based embedded applications.

This research explores the use of multibody models for real-time embedded applications. It describes different techniques to accelerate parts or all of the multibody computations on ARM-based and/or FPGA-based hardware.

1 Introduction

Industrial products development benefits from real-time simulations. Today's systems are increasing in complexity due to market drivers such as energy efficiency, growing number of system variants, reliability and performance. Simulation solutions have matured to the extent they have transformed the development process of complex systems to become a process cornerstone. Highly accurate simulation models are used throughout the life-cycle of the product

to predict real-life performance, to understand the manufacturing process or to detect operational failures.

For instance, Model-based System Testing (MBST) [1] is a framework of engineering solutions that enable and optimally balance the combined use of test and simulation. This approach is of interest for testing and experimentally optimizing subsystem behavior in a virtual system integration context, which may even include the human user. Simulation models are also employed in virtual sensing, an emerging field in testing applications. Models and a limited set of experimental data are fused to estimate system variables that are difficult (inaccessible location, inaccurate direct measurements, inexistent sensor) or costly to measure. Virtual sensing enriches the measurement data-set, thus providing better insight into the system-under-test to engineers. Accurate models are required as a starting point. State estimators are then used to fuse test and simulation data and ensure optimal data quality. In the automotive field, advanced algorithms for chassis or suspension control benefits from this virtual measurements [2–5].

Many of these applications are executed on embedded systems and must accomplish real-time performance and low-power consumption [6]. Real-time is needed for synchronizing real world and simulation achieving determinism in the overall system [7]. Low-power consumption is required due to the strict power budget of embedded systems [6]. For example, in electric vehicles, the battery is responsible of supplying power not only to the powertrain, but to the auxiliary systems and electronic devices. Therefore, the energy consumption of additional embedded systems should be minimal in order to maximally extend the vehicle range.

Multiprocessor System on Chips (MPSoCs) represent an important trend in embedded systems. MPSoCs are systems-on-chip with more than one processor, which can satisfy the constraints of real-time and power consumption of new embedded applications [8]. Their computational power, however, remains modest relative to personal computers (PCs). MPSoCs are traditionally classified into homogeneous, where similar processing cores are interconnected to build a high-performance computer, and heterogeneous processors, built from cores of different architectures which can be used as accelerators, complementing the capabilities of the main processor.

This new trend gives an opportunity for simulating complex models, as multibody (MB), on embedded hardware in real time. MBST applications could be based on detailed models, instead of reduced sized models, while still maintaining real-time performance. Based on a low-end heterogeneous processor, where an Field Programmable Gate Array (FPGA) complements the main processor, this work explores the benefits that can be obtained from an FPGA for accelerating MB simulations.

This paper is organized as follows. Section 2 gives an overview on the modern hardware available for embedded applications, focusing on the FPGAs. In Section 3, through a simple example, the procedure to accelerate MB simulations with an FPGA is presented. The FPGA implementations obtained from the simple example are presented in Section 4. These implementations are scaled

and applied in a real use-case for automotive applications in Section 6. Finally, the conclusions of this work are presented in Section 7.

2 Modern Hardware Analysis

Computation capabilities of conventional processors have been growing exponentially due to sustained innovation in processors technology [9]. This evolution has led to more complex cores which provide higher performance at the expense of increasing the number of computing elements per microprocessor, with an increase of the chip footprint and power efficiency [10]. In embedded systems, however, the power consumption is critical and it has become a major aspect that limits the performance of processors for embedded applications [11].

In order to meet the high computational performance and low-power requirements, advanced multi-core architecture have been developed. Having more than one core in the system allows developing parallel processing on chips, reducing the computational cost of an application [12].

In a multiprocessor platform, however, there is not an ideal core for all the possible applications. Normally, in conventional processors, this issue is solved by designing a general purpose architecture. However, few applications actually need all those resources. Therefore, such an architecture is highly over-provisioned for any single application [10], leading to a higher power consumption.

Homogeneous designs assembly multiple cores of the same architecture and, in order to cover a high range of applications, the over-provisioned problem of conventional processors is repeated [10]. Conversely, heterogeneous processors can map each application to the best suited core to meet its performance demands.

In general, heterogeneous processors complement a main core with different type of secondary processors or co-processors. The main purpose of the co-processor is to supplement the functionality of the primary processor, and it is optimized for a single specific task. By offloading computations from the main processor to one or more co-processing units, the overall system performance can be accelerated [13]. Thus, having heterogeneous processor cores provides potentially greater power savings without dramatic losses in performance [14]. Due to the potential reduced energy consumption offered by heterogeneous multi-core platforms, their usage is appealing for hard real-time systems in embedded applications [11].

2.1 Heterogeneous processors for scientific computing

In the field of heterogeneous computing, traditional processors are commonly combined with unconventional cores such as Application-Specific Integrated Circuits (ASICs), Field Programmable Gate Arrays (FPGAs) or Graphic Processing Units (GPUs) [9, 15, 16]. Complex operations are offloaded to these co-processors obtaining hardware acceleration.

While ASIC devices are expensive and their development cycle spans several years, GPUs and FPGAs are cheaper and less time consuming in the designing phase in order to accelerate an application. GPUs and FPGAs are programmable and flexible, meaning that they can be used in any application without any extra cost of redesigning the hardware. Although an ASIC device can provide the highest performance for a particular application, FPGAs and GPUs are standing as the most popular co-processors in heterogeneous devices due to their flexibility and reduced price.

Graphic Processing Units (GPUs) were developed for parallel manipulation of tasks related to computer graphics [17]. However, their usage goes beyond graphics and GPUs are employed to increase the efficiency of expensive computations. The main advantage of the GPU as an accelerator stems from its high memory bandwidth and a large number of programmable cores to parallelize the execution of a program. They have been used for accelerating MB dynamics previously, as presented in [18–21].

Field Programmable Gate Arrays (FPGAs) are programmable hardware devices that offer the high performance of custom hardware, with some of the flexibility of software. An FPGA contains programmable logic elements which can be combined to exactly build the hardware required for a specific application, resulting in orders of magnitude speed-up over conventional processors for computationally-intensive tasks [22]. Traditionally, FPGAs where independent devices and the communication time between the FPGA and the main processor could affect the overall performance. New embedded systems incorporate a General Purpose Processor (GPP) with an FPGA in a MPSoC, yielding more efficient communication between both devices than two chip designs, resulting in improved performance [23].

The presence of FPGAs in industrial applications has been growing in last years. FPGAs are used for controlling industrial processes and motors, in machine vision to inspect manufacturing lines or in industrial networking [24, 25]. Being programmable devices provides the flexibility to adapt industrial designs to changing requirements and lower total solution costs [25]. In automotive applications, MPSoC solutions embedding an FPGA or GPU are starting to be used due to the high computational cost of Advanced Driver Assistance System (ADAS) [26].

In contrast to GPUs, there is no research on how to exploit FPGA for accelerating a MB simulation. Since the presence of FPGAs is increasing in industrial applications, they can be used to reduce the computational cost of MB simulations, thus providing accurate model-based solutions for real-time embedded applications.

Focusing on the automotive industry, the Xilinx® Zynq-7000 XC7Z020 is widely used in computer vision or control applications [24]. It is a low-end embedded platform based on an ARM Cortex-A9 combined with an Artix-7 FPGA as co-processor. Several works have been presented based on this device in automotive applications. It is used in computer vision applications such as signal recognition and detection of pedestrians or obstacles [27–32]. For control purposes and machine learning, the works presented in [17, 33] are also based

on the Zynq-7000 XC7Z020. Although there are more powerful devices on the market, their costs are also higher, which reduces their fields of applications. In addition, the automotive grade Zynq-7000 devices are based on the same processor [34]. Due to the broadly use of the Zynq-7000 XC7Z020 device and in order to provide results applicable to real use-cases, it is selected in this work. The properties of the device are summarized in Table 1.

Table 1 Features of the Zynq-7000 XC7Z020 processor [35]

Zynq-7000 (Device Code: XC7Z020-CLG484-1)		
Main processor	Processor Core	Dual ARM Cortex-A9
	Processor Extensions	NEON & Single\Double Precision Floating Point
	Max. Frequency	667 MHz
	L1 Cache	32KB Instruction, 32KB Data per processor
	L2 Cache	512KB
	On-Chip Memory	256KB
	External Memory	DDR3, DDR3L, DDR2, LPDDR2
Co-processor	FPGA	Artix-7
	Logic Cells	85K
	LUT	53,200
	FF	106,400
	BRAM	140 (4.9Mb)
	DSP	220

For clarity and better understanding of the FPGA elements presented in Table 1, it is worth detailing certain characteristics of the hardware elements. In general, an FPGA is made of wires connecting logic gates and registers [36]. The logic gates perform simple logic operations on signals. They represent the core functionality of digital circuits, which means they perform simple boolean logic on inputs. If several gates are combined, more complex operations can be performed. The registers are simple devices that store data for its use in future operations, which needs to be accessible quickly. Once the data is no longer needed, it is replaced by new information. Inside an FPGA, four main element types can be found [13]:

1. Look-Up Table (LUT): It is a flexible resource capable of implementing a logic function, small memory elements or registers.

2. Flip-Flop (FF): It is a sequential circuit element implementing a 1-bit register, whose purpose is to synchronize logic and save logical states during clock cycles.

3. Block Random Access Memory (BRAM): It is a memory block with the special purpose of satisfy dense memory requirements.

4. Digital Signal Processor (DSP): It is a sub-unit dedicated for high-speed

DSP arithmetic supporting operations such as multiply-add, multiply-accumulate (MACC), three-input add, etc.

2.2 FPGAs Considerations

FPGAs have different features that should be considered prior to implementing an application on these devices, such as the numerical data types, hardware capacity or programming language.

FPGAs are programmed in Hardware Description Languages (HDLs), such as Verilog or VHDL. Since HDL differs from the software programming languages, the use of FPGAs was limited to developers with a strong hardware design knowledge. With the popularization of these devices, manufacturers are releasing software for translating code from C-like languages to HDL, allowing software developers access to FPGA programming. However, hardware design knowledge is still required in order to guide this automatic-translation tools into the desired implementation.

Another factor to consider is the floating-point precision, which is critical for particular applications. Although FPGAs can support floating-point calculations, they are more performant for fixed-point operations. To operate with floating-point data, a large amount of resources is required. In the case of MB simulations, it is desired to work with floating-point numbers instead of fixed-point, since more precision and range of numbers is available. As the size of FPGAs is increasing, high-precision calculations are becoming more common as the resource footprint required for the implementation of floating-point calculations is reduced with respect to the size of the device [13].

Nevertheless, the main disadvantage of FPGAs is their limited hardware resources. The algorithm that needs to be accelerated might require more hardware resources than the ones available in the FPGA. To overcome this issue, the algorithm should be partitioned or its optimization level reduced. In both cases, the resulting FPGA implementation is not as efficient as it could be if more resources were available. This issue is being mitigated due to the continuous FPGA technology evolution. Nowadays, a large range of devices can be found. There are more powerful FPGAs with capabilities to implement designs which demand more resources or higher frequencies to achieve the desired acceleration.

3 FPGA acceleration in Multibody Simulation

From what has been presented in Section 2, the computational power of embedded hardware is low compared to conventional computers. This can result in MB simulations where real-time execution is not guaranteed, while it would largely be satisfied in a personal computer. Offloading the most time consuming operations to an FPGA could result into real-time execution in embedded platforms. For instance, in devices such as the Zynq XC7Z020, where the main

processor has a low frequency, the availability of a co-processor can make the difference.

However, the performance of a MB simulation in embedded hardware can not rely only on the FPGA acceleration. An efficient MB formulation should be selected for the modeling and, later, the FPGA can be used for reducing the computational cost if required.

In this section, the usage of an FPGA for accelerating a MB simulation is addressed. Through a simple example of an N-four-bar linkage, a proof of concept for accelerating a MB model whit an FPGA can be developed. The procedure followed in this section could be later extended to other models or formulations.

Since this work is in the context of automotive applications, the formulation presented in [37–39], named semi-recursive formulation, is selected. Compared with absolute methods, it offers high performance when large models are used, due to the reduction in the number of modeling coordinates. In small systems, like the suspension model presented in [37], there is no advantage in using the semi-recursive formulation. When applied to a full vehicle model, the method in relative coordinates is 10 times faster than the absolute.

The N-four-bar linkage mechanism employed is presented (Figure 1). It has a total of 9 moving bodies and the ground element, all connected by revolute joints. The size of the model is selected so that the size of the FPGA does not limit the optimization of the implementations addressed. The MB model is generated using the `MBScoder` presented in [40], which writes efficiently the minimal amount of code required for the simulation, in an automatic form.

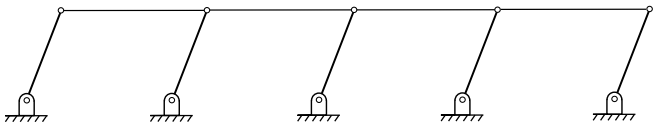

Fig. 1 MB model of a N-four-bar linkage which is employed as example for presenting the approaches of using FPGAs as co-processors in a MB simulation

It must me remarked that the semi-recursive formulation could not be the most efficient for this model size. As stated above, the interest of this example is to show how to implement certain tasks of a MB simulation on an FPGA. Using the semi-recursive formulation allows to later apply the resulting implementations on a real automotive use-case, presented in Section 6.

Following the approach presented in [37], a double set of coordinates is defined for the modeling: the relative coordinates of the hole mechanism and six coordinates (three translations plus three rotations) for each body, which are the so-called body coordinates presented below,

$$\mathbf{Z} = \begin{bmatrix} \dot{\mathbf{s}} \\ \boldsymbol{\omega} \end{bmatrix} \tag{1}$$

where $\dot{\mathbf{s}}$ is the velocity of the point of the body which is coincident with the

fixed frame origin in a particular time step, thus representing the translations of the body, and $\boldsymbol{\omega}$ is the angular velocity of the body, which represents the rotations of the body.

Selecting the point of each elements coincident with the fixed frame origin of coordinates instead of the center of mass as usually, the recursive kinematic relations between bodies are simpler [39]. Applying the kinematic relations of two neighbor bodies, denoted by *i-1* and *i*, a general expression for obtaining the value of each body coordinate and its derivative in a recursive form can be derived,

$$\mathbf{Z}_i = \mathbf{Z}_{i-1} + \mathbf{b}_i \dot{\mathbf{z}}_i \tag{2}$$

$$\dot{\mathbf{Z}}_i = \dot{\mathbf{Z}}_{i-1} + \mathbf{b}_i \ddot{\mathbf{z}}_i + \mathbf{d}_i \tag{3}$$

where the form of the terms $\mathbf{b}_i$ and $\mathbf{d}_i$ depends on the type of joint that connects the bodies *i-1* and *i*. In [39], expressions of $\mathbf{b}_i$ and $\mathbf{d}_i$ are presented for different type of joints.

The dynamic equations which describe the motion of the mechanism are expressed initially in relative coordinates. In this method, the equations are stated according to the Penalty Index-3 Augmented Lagrangian (ALI3P) formulation [41] in the form,

$$\mathbf{M}\ddot{\mathbf{z}} + {\boldsymbol{\Phi}_{\mathbf{z}}}^t \boldsymbol{\alpha} \boldsymbol{\Phi} + {\boldsymbol{\Phi}_{\mathbf{z}}}^t \boldsymbol{\lambda}^* = \mathbf{Q} \tag{4}$$

where $\mathbf{z}$ are the relative coordinates, $\mathbf{M}$ is the mass matrix of the mechanism expressed in terms of the relative coordinates, $\boldsymbol{\Phi}$ is the constraint vector due to the closure conditions of the loops, $\boldsymbol{\Phi}_{\mathbf{z}}$ is the Jacobian matrix of the constraints, $\boldsymbol{\alpha}$ is the penalty factor, $\mathbf{Q}$ is the vector of applied and velocity-dependent forces, and $\boldsymbol{\lambda}^*$ is the vector of Lagrange multipliers obtained from the following iteration process [37]:

$$\boldsymbol{\lambda}^*_{i+1} = \boldsymbol{\lambda}^*_i + \boldsymbol{\alpha} \boldsymbol{\Phi}_{i+1}, \qquad i = 0, 1, 2... \tag{5}$$

However, if the relative coordinates $\mathbf{z}$ are used, the calculation of the dynamic terms presented in (4) is complex. Here is where the body coordinates (1) become relevant, since they lead to simpler expressions for obtaining the dynamic terms in a recursive form, as is shown in [37–39]. To introduce these terms in the dynamic equations (4), a matrix $\mathbf{R}$ can be defined from kinematic relations in such a form that,

$$\mathbf{Z} = \mathbf{R}\dot{\mathbf{z}} \tag{6}$$

$$\dot{\mathbf{Z}} = \mathbf{R}\ddot{\mathbf{z}} + \dot{\mathbf{R}}\dot{\mathbf{z}} \tag{7}$$

being $\mathbf{R}$ a matrix which depends on the topology of the mechanism and on the $\mathbf{b}_i$ terms from equation (2), and $\dot{\mathbf{R}}\dot{\mathbf{z}}$ a vector obtained from the $\mathbf{d}_i$ terms of (3), as presented in [38]. It must be noted that, from the structure of the matrices,

each element of the mechanism is only influenced by its previous bodies on the kinematic chain, as expected.

After establishing the kinematic relation between coordinates, expressions for the the mass matrix and the generalized forces applied of each element in body coordinates can be derived yielding,

$$\bar{\mathbf{M}}_i = \begin{bmatrix} m\mathbf{I} & -m\tilde{\mathbf{g}} \\ m\tilde{\mathbf{g}} & \mathbf{J} - m\tilde{\mathbf{g}}\tilde{\mathbf{g}} \end{bmatrix} \tag{8}$$

$$\bar{\mathbf{Q}}_i = \begin{bmatrix} \mathbf{f} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times m\mathbf{g}) \\ \mathbf{n} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{g} \times (\mathbf{f} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times m\mathbf{g})) \end{bmatrix} \tag{9}$$

where m is the body mass, $\tilde{\mathbf{g}}$ is the dual anti-symmetric matrix of the global position of the mass center of the body ($\mathbf{g}$), $\mathbf{J}$ is the inertia tensor of the body, $\mathbf{f}$ and $\mathbf{n}$ are the external forces and torques applied to the body respectively.

From the expressions presented in (8) and (9), the dynamic matrices for the whole mechanism in body coordinates can be assembled. Thus, the mass term $\bar{\mathbf{M}}$ is a diagonal matrix whose diagonal elements are the sub-matrices $\bar{\mathbf{M}}_i$, and the vector of forces $\bar{\mathbf{Q}}$ contains each individual term $\bar{\mathbf{Q}}_i$.

In order to obtain the equivalent dynamic matrices in relative coordinates for their use in (4), the equation of motion in body coordinates is obtained. From the virtual power principle,

$$\mathbf{Z}^{*T}(\bar{\mathbf{M}}\dot{\mathbf{Z}} - \bar{\mathbf{Q}}) = \mathbf{0} \tag{10}$$

being $\mathbf{Z}^{*T}$ the virtual velocities on the body coordinates.

Substituting now the result of (2) and (3) in the equation (10) yields,

$$\dot{\mathbf{z}}^{*T}\left\{\mathbf{R}^{\top}\bar{\mathbf{M}}\mathbf{R}\ddot{\mathbf{z}} - \mathbf{R}^{\top}(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\dot{\mathbf{R}}\dot{\mathbf{z}})\right\} = \mathbf{0} \tag{11}$$

or, in a more compact form,

$$\dot{\mathbf{z}}^{*T}\left\{\mathbf{M}\ddot{\mathbf{z}} - \mathbf{Q}\right\} = \mathbf{0} \tag{12}$$

Therefore, comparing the equations (4) and (12), the mass matrix and force vector in relative coordinates can be obtained from their equivalent in body coordinates as follows,

$$\mathbf{M} = \mathbf{R}^{\top}\bar{\mathbf{M}}\mathbf{R} \tag{13}$$

$$\mathbf{Q} = \mathbf{R}^{\top}(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\dot{\mathbf{R}}\dot{\mathbf{z}}) \tag{14}$$

Due to the form of the matrix $\mathbf{R}$, the mass matrix and force vector of the system are suitable to be assembled in a recursive form, as presented in [38], leading to an increment of computational efficiency.

Once that the mass matrix and the vector of forces is obtained, the integration of the dynamic equations presented in (4) can be addressed. For this

purpose, the implicit single-step trapezoidal rule combined with the Newton-Raphson iterator has been adopted, yielding,

$$\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} = -\mathbf{f}(\mathbf{z}) \tag{15}$$

where the residual vector is

$$\mathbf{f}(\mathbf{z}) = \frac{\Delta t^2}{4}\left(\mathbf{M}\ddot{\mathbf{z}} + \boldsymbol{\Phi}_{\mathbf{z}}{}^{\top}\boldsymbol{\alpha}\boldsymbol{\Phi} + \boldsymbol{\Phi}_{\mathbf{z}}{}^{\top}\boldsymbol{\lambda}^{*} - \mathbf{Q}\right) \tag{16}$$

and the approximated tangent matrix is:

$$\frac{\partial \mathbf{f}(\mathbf{z})}{\partial \mathbf{z}} \simeq \mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\left(\boldsymbol{\Phi}_{\mathbf{z}}{}^{\top}\boldsymbol{\alpha}\boldsymbol{\Phi}_{\mathbf{z}} + \mathbf{K}\right) \tag{17}$$

being $\mathbf{C}$ and $\mathbf{K}$ the damping and stiffness matrices, respectively, whose full expressions can be found in [39].

Although this formulation offers high efficiency, it presents some limitations. The mass matrix of the system must be computed each iteration, since it is not constant, and a post-process must be performed at the end of each time step in order to derive the absolute motion of the bodies and joints coherently with the new value of the relative coordinates. Both operations are the most time demanding tasks during the simulation. Furthermore, although the system of equations presented in (15) has a reduced size with respect to other methods, it is still a expensive computation. These bottlenecks of the MB simulation can be lighted through their FPGA implementation.

4 Hardware Implementations

In order to accelerate the MB simulation with an FPGA, the computation bottlenecks of the MB formulation have to be offloaded to the FPGA. The rest of the simulation is performed on the main processor. Comparing the results of each implementation allows evaluating which strategy offers more improvement in terms of execution time.

Due to the different data dependencies and structure of each operation, an independent analysis is required. The software employed to program each function in the FPGA is `Vivado HLS` from Xilinx®. The procedure followed during the implementations presented on this work can be extended to any other task of a MB simulation, either modeled in global or relative coordinates.

4.1 Mass matrix calculation

The process for obtaining the mass matrix of the system can be divided in two steps: the calculation of the body mass matrices, following (8), and the assembly of each body mass matrix into the global matrix in a recursive form, as stated in (13). The individual mass matrix can be computed in parallel for each body, since each matrix only depends on the variables of its body, as can be seen in

(8). However, to compute the global mass matrix, the recursive procedure has a sequential nature due to the dependencies between bodies. This is where the advantages of an FPGA can be exploited.

Following a pipeline strategy, the calculation of all the individual mass matrices and their assembly on the global mass matrix can be executed in a concurrent manner: before the calculation of one mass matrix is finished, the calculations for another body can start. This concurrent computation is represented in Figure 2, where the operations performed during a set of clock cycles is represented. It must be noted that each operation is performed at a matrix element level.

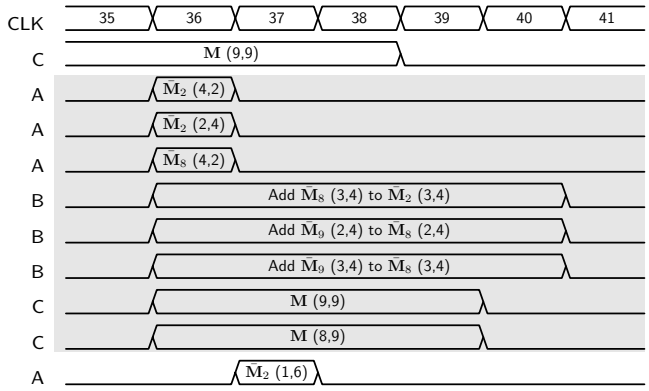

Fig. 2 Schedule of a stage (gray background) of the mass matrix calculation (13) in the FPGA. Operation A is referred to the evaluation of the matrix elements from (8). Operation B accumulates the individual mass matrices. Operation C performs the multiplication of (13)

The sequence of Figure 2 is one of the stages repeated through the total computing process. By means of the pipeline directive, `Vivado HLS` analyses the code to implement and achieves an optimal scheduling for the algorithm, deciding which matrix element should be calculated at each clock cycle. However, since it is automatic, it is also necessary to adapt the code in order to guide the program to the desired solution. In this particular case, the three operations where clearly divided, and intermediate copies of variables are introduced in order to allow more operations over the same data.

Since the storing-data arrays are defined with a single port memory resource by default, the read and write ports of the array are limited. If the evaluation of the mass matrix is performed in parallel, the number of read and write operations will increase, and the limited number of read and write ports of the arrays will lead to an inefficient implementation. As a solution to this problem, the arrays can be partitioned into smaller arrays (implemented as multiple memories) increasing the number of load and store ports.

In addition, due to the topology of the mechanism, some terms involved in the calculations are always null and, therefore, the operations can be simplified with the consequent saving of resources. In Table 2, the results in terms of re-

sources and latency under the optimizations explained and based on a pipeline strategy are presented. The maximum FPGA frequency for executing this implementation is 125 MHz.

Table 2 Summary report of the FPGA implementation for computing the mass matrix of the MB system of Figure 1 following a pipeline approach

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	508	1	5	9	24
Read Input data	56	0	0	≈0	1
Mass Matrix calculation	266	1	5	9	22
Write Output data	179	0	0	≈0	1

4.2 MB Post-Processing stage

The post-processing required to obtain the absolute motion of the mechanism is based on kinematic relations between bodies and joints. This process is recursive, since the position of a body depends directly on the position of its previous body in the kinematic chain. Due to this dependency between bodies, the structure of this function is sequential and, as in the mass matrix calculation, FPGAs are suitable devices to accelerate the post-processing stage of the MB model.

The values of each body that must be computed are the position and velocity of its center of gravity (CoG), the rotation matrix, and its angular velocity. Regarding the joints, since they are of a revolute type, their position and the direction of the axis of revolution must be calculated. There exists also a data dependency between the operations performed for each body and joint. For example, in order to evaluate the CoG of a body, the rotation matrix of the same body is required to transform the local position of the CoG to absolute.

As opposite to the mass matrix calculation, the post-processing phase for the mechanism of Figure 1 can be made in a loop which performs the same operations over different bodies. However, as explained above, the data dependency between loop iterations does not allow to compute all the bodies in parallel at the same time.

Using a pipeline approach, the schedule of the function is as depicted in Figure 3. As in the mass matrix computation, arrays which store data have been partitioned in order to increase the number of operations that can be executed over the stored data, and intermediate copies of variables are created. In Figure 3, only a loop iteration is presented, since it is repeated for each body. As can be seen, the different operations inside a body are also executed in a concurrent manner.

However, the main advantage of the pipeline appears when parallelizing the operations within different bodies. After analyzing the data dependencies

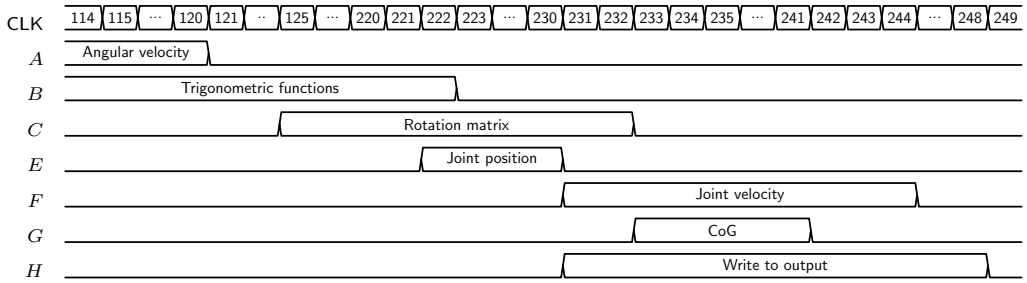

Fig. 3 Schedule of a loop iteration of the FPGA implementation for updating the motion of one body of the system presented in Figure 1. The clock cycles are shortened in order to make the diagram more readable

between loop iterations, `Vivado HLS` achieves a solution in which the evaluation of a new body can start 10 clock cycles after the previous body. This means that the computations of two consecutive bodies can also be executed in a concurrent manner. This leads to a high reduction in latency, from 7701 to 494 clock cycles. The results of the implementation are presented in Table 3. The FPGA runs this implementation at a frequency of 125 MHz.

Table 3 Summary report of the FPGA implementation for updating the motion of the MB system of Figure 1 following a pipeline approach

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	494	11	36	33	69
Read Input data	17	0	0	1	1
Post-Process	355	3	36	31	65
Write Output data	116	0	0	≈0	3

In this implementation, it is interesting to underline the effects of the trigonometric functions required for the evaluation of the rotation matrix. The cost of these functions in terms of hardware is high, leading to an increment of the required resources.

4.3 Solver of a System of Equations

Solving systems of equations as the one presented in (15) is a common problem in more fields than MB simulations. Therefore, several studies are available in the literature showing which method is more efficient to be implemented on an FPGA. In this work, the algorithm for FPGAs proposed in [42] is employed. It offers an efficient implementation in terms of resources and latency of a solver based on the Gauss-Jordan algorithm.

The MB model of Figure 1 leads to a system of 9 equations and 9 unknowns. The report of the implementation achieved for the system size of this work through `Vivado HLS`, and under the guidelines presented in [42] is shown in Table 4. This implementation achieves a frequency of 125 MHz in the FPGA.

Table 4 Summary report of the FPGA implementation for the Gauss-Jordan algorithm employed for solving Equation 15 following the approach presented in [42]

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	1018	≈0	36	14	39
Read Input data	90	0	0	≈0	≈0
Gauss-Jordan	913	≈0	36	13	39
Write Output data	9	0	0	≈0	≈0

5 Use-case: N-four-bar linkage

In order to test the implementations presented in Section 4, several simulations of the N-four-bar linkage mechanism employed during this work are launched on the Zynq-7000 XC7Z020, modifying the solver time step and the solver tolerance. The simulation of the mechanism in the ARM processor is taken as reference. Later, simulations offloading each of the tasks implemented in Section 4 are launched. Comparing the elapsed time for each simulation gives a measure of the acceleration achieved through the use of the FPGA.

For all the simulations, the ARM processor runs at its maximum frequency (667 MHz) and in a bare-metal system, the Xilinx Stand-alone (bare-metal) Environment. Hence, there are no any other processes running at the same time. Only one core is used and the code is compiled under an optimization level 2, meaning that the compiler performs nearly all supported optimizations. In addition, although the ARM supports double-floating-point data (Table 1), the simulation is executed with single precision for a fair comparison with the FPGA implementations. The Eigen library [43] is used for the linear algebra operations in the ARM processor. The summary of the results is shown in Table 5 and Table 6.

The results show that only the evaluation of the global mass matrix of the system in the FPGA offers an speed-up of approximately 15% with respect to the simulation on the ARM processor. The achieved acceleration increases with the reduction of the time step, since more mass matrix evaluations are necessary. In fact, for an integration time step of 1 millisecond, only real-time performance can be achieved offloading the mass matrix calculation to the FPGA. Due to the parallelization of the FPGA, it is expected that the effect of increasing

Table 5 Summary report of the comparison between the ARM execution of the MB model and each of the FPGA implementation proposed in Section 4. The simulation is executed with a time step of 4 milliseconds and with a tolerance for the integration of 10^{-6}

Summary Report					
Version	Simulation Time (s)	Time Step (s)	Elapsed Time (s)	Average of Iterations	Tolerance
Reference	10	0.004	8.467	18.62	10^{-6}
Gauss-Jordan FPGA	10	0.004	8.389	18.58	10^{-6}
Mass Matrix FPGA	10	0.004	7.065	18.67	10^{-6}
Post-Process FPGA	10	0.004	8.396	18.75	10^{-6}

Table 6 Summary report of the comparison between the ARM execution of the MB model and each of the FPGA implementation proposed in Section 4. The simulation is executed with a time step of 1 milliseconds and with a tolerance for the integration of $2 \cdot 10^{-6}$

Summary Report					
Version	Simulation Time (s)	Time Step (s)	Elapsed Time (s)	Average of Iterations	Tolerance
Reference	10	0.001	11.012	4.88	$2 \cdot 10^{-6}$
Gauss-Jordan FPGA	10	0.001	10.954	4.87	$2 \cdot 10^{-6}$
Mass Matrix FPGA	10	0.001	9.291	4.89	$2 \cdot 10^{-6}$
Post-Process FPGA	10	0.001	10.879	4.89	$2 \cdot 10^{-6}$

the size of the mass matrix of the system has lower impact on the hardware implementation than in the ARM processor, leading to a higher acceleration.

With respect to the Gauss-Jordan implementation, the achieved acceleration is minimal. As explained in [42], the acceleration level of the Gauss-Jordan algorithm increases with the size of the system to solve. It is expected therefore that, with MB systems of higher number of variables, the achieved acceleration will be more reliable.

Regarding the post-processing tasks, the results are similar to the Gauss-Jordan version. There is almost no benefit to offload this task to the FPGA. As in the other versions, it is expected that the speed-up achieved would increase with the complexity of the MB model: more bodies evaluated in parallel will lead to a higher speed-up.

6 Use-case: Automotive virtual sensing

In Section 1, several applications of real-time simulation were introduced. Out of them, the virtual sensing approach for automotive applications is taken as a real

use-case. In this use-case, the objective is to extend the test measurement dataset with virtual sensors, during vehicle maneuver execution. The MB model has to be executed in real-time on the embedded hardware located in the vehicle. For correcting the drift due to uncertainty between model and reality, real sensor measurements are fed to a state estimator for correcting the behavior of the model [2, 5].

One of the main limitations of virtual sensing for in-vehicle applications is related to the low computational power of the on-board embedded systems. New generation Electronic Control Units (ECUs) and automotive on-board data-acquisition systems are based on heterogeneous devices, as stated in Section 2. As a consequence, the computational power of the new available hardware paves the way to real-time embedded virtual sensing.

Following the guidelines presented in Section 4 for the three implementations, the use of the FPGA with a complete MB model of a vehicle is explored. The vehicle (Figure 4) is modeled in relative coordinates, with the semi-recursive formulation presented in [37]. A summary of the MB model is presented in Table 7.

Fig. 4 Modelled vehicle

Table 7 MB parameters

Vehicle MB model	
DOFs	14
MB coordinates	Relative
MB formulation	ALI3P
Nº of coordinates	60
Nº of bodies	29
Nº of integrable variables	42

6.1 Hardware Implementations

The MB model of the studied vehicle has the following differences with respect to the N-four-bar linkage employed in this work. First, the size of the model is larger. Second, the complexity of the model is higher. For the N-four-bar linkage, all the joints involved are of the same type, revolute. Instead, the vehicle model includes more type of joints, which affect the amount of code that must be flashed on the FPGA.

The effects of increasing the size of the MB model can be seen by comparing the vehicle model presented to the N-four-bar linkage of Section 3. First, for the implementation of the mass matrix evaluation, it has been proven that in the Artix-7 of the Zynq-7000 XC7Z020 device, there are not enough hardware resources. Therefore, the mass matrix of the system cannot be computed efficiently on the FPGA. As an alternative, the evaluation of the individual mass matrices of the bodies (8) is addressed. In this case, the results of the implementation are shown in the Table 8. This implementation can run in the FPGA

at a frequency of 125 MHz.

Table 8 Summary report of the FPGA implementation for computing the mass matrices of each body with a pipeline of the mass matrices calculation

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	1478	20	45	38	74
Read Input data	377	0	0	≈0	1
Indiv. Mass Matrices calculation	52	0	45	36	72
Write Output data	1045	0	0	≈0	1

The implementation presented in Table 8 leads to a minimal latency for the individual mass matrix calculation. Due to the independence among different mass matrices, they can be computed in parallel. It is similar to using one core for each individual mass calculation in the FPGA. The data of each body is partitioned so that all the matrices are evaluated at the same time. Therefore, the total latency for computing all the individual mass matrices corresponds to the latency of one individual mass matrix evaluation.

Regarding the post-processing phase, the available resources on the FPGA are not sufficient to follow the implementation of Section 4. In order to accelerate this part of the simulation, the process can be divided into sub-tasks. At this point, it is interesting to implement a subsystem that is repeated in the model, in order to use the implementation programmed on the FPGA more than once and achieve a higher acceleration level. Hence, the suspension system is a suitable candidate, since the topology of the mechanism is almost identical for the four suspensions in the modeled vehicle. The trigonometric functions required are computed in the ARM processor and given as inputs to the FPGA. Otherwise, the implementation cannot fit on the FPGA. The summary for the implementation of a suspension system are shown in Table 9. This implementation is executed at a frequency of 100 MHz in the FPGA.

Table 9 Summary report of the FPGA implementation for the post-processing of each suspension system

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	407	1	55	15	58
Read Input data	66	0	0	≈0	2
Suspension Post-Process	101	1	55	14	49
Write Output data	240	0	0	1	7

The last operation implemented is the resolution of the equation presented in (15), involving 42 variables, following the implementation presented by J.P. David in [42]. Due to resources limitations, the optimization level of the algorithm is reduced compared with the implementation presented in Section 4. In Table 10, the final FPGA implementation is presented. The frequency achieved in the FPGA is 100 MHz.

Table 10 Summary report of the FPGA implementation for the Gauss-Jordan algorithm employed for solving the equation presented in (15) following the approach presented in [42]

Summary Report					
Function	Latency (clock cycles)	Resources (%)			
		BRAM	DSP	FF	LUT
Total function	20120	29	53	18	96
Read Input data	1806	0	0	≈0	1
Gauss-Jordan	18266	14	53	18	93
Write Output data	42	0	0	≈0	1

6.2 Results

To evaluate the benefits of the proposed FPGA implementations, a maneuver of 10 seconds is simulated. Similar to obtaining the results for the N-four-bar linkage presented in Section 5, the simulation executed on the main processor is considered as reference. The ARM implementation follows the same directives as for the N-four-bar linkage. Later, simulations offloading each of the tasks presented in Section 6.1 are executed.

Since the MB model of the vehicle is intended to be used for state estimation, the tolerance of the simulation is less stringent than when the objective is to accurately forward simulate the vehicle dynamics. The results in terms of time consumed are presented in Table 11.

The results show that with the same time step, a higher computational efficiency is achieved with the FPGA implementation of the Gauss-Jordan algorithm due to the increment of the MB size, as expected with the results presented in [42]. The simulation is accelerated close to 25% with respect to the ARM execution.

Regarding the individual mass matrices evaluation and the post-processing of each suspension system, their FPGA implementations do not offer any speed-up. Accordingly to Section 5, only offloading to the FPGA the evaluation of the complete mass matrix and the full post-processing phase is of interest for accelerating MB simulations on the hardware used in this research.

Table 11 Summary report of the comparison within the ARM execution of the vehicle MB model and each of the FPGA implementation proposed in Section 6.1. The simulation is executed with a time step of 4 millisecond and with a tolerance for the integration of $5 \cdot 10^{-5}$

Summary Report					
Version	Simulation Time (s)	Time Step (s)	Elapsed Time (s)	Average of Iterations	Tolerance
Reference	10	0.008	11.888	2.062	$5 \cdot 10^{-5}$
Gauss-Jordan FPGA	10	0.008	8.982	2.009	$5 \cdot 10^{-5}$
Partial Mass Matrix FPGA	10	0.008	11.968	2.043	$5 \cdot 10^{-5}$
Partial Post-Process FPGA	10	0.008	12.344	2.054	$5 \cdot 10^{-5}$

7 Conclusions

In industry, model-based applications are steadily gaining importance in modern development processes. For a subset of these use-cases, real-time is a mandatory pre-requisite for synchronizing real and virtual environments. A main limiting factor for these applications when using MB models is the high computational load with respect to the available hardware resources.

Processor technology evolution has led to new processors which increase the computational capabilities of embedded systems. Heterogeneous processors, specially based on an ARM as main processor and an FPGA or GPUs as co-processor stand out among other alternatives. In these devices, the co-processor behaves as an accelerator for the most computationally expensive tasks of the simulation, thus reducing the overall computing time. Due to the increasing presence of FPGAs in industrial applications, this research explored their use for accelerating MB simulations.

In this work, the use of FPGAs in MB simulations is first studied with a model of an N-four-bar linkage. The most time consuming tasks of the simulation are identified, and the procedure to optimally program the FPGA is presented. Three implementations are tested: the mass matrix evaluation of the MB model, the post-processing calculations performed to obtain the motion of the model, and a solver of equations for computing the increments of the MB variables at each iteration. From the studied implementations, it was shown that the mass matrix computation in the FPGA offers a speed-up of a 15% approximately on the Zynq-7000 XC7Z020. In the case of the post-processing and the solver, it was concluded that for the size of the tested MB model, there are almost no benefits using the aforementioned hardware.

The next use-case studied in this paper is a real application for the automotive industry. For in-vehicle virtual sensing, the use of an FPGA can be the differentiating element for achieving real-time. Following the approaches of the N-four-bar linkage, the same tasks of the simulation are offloaded to the FPGA. However, the increment of elements in the MB model leads to a scalability prob-

lem: the tasks implemented do not fit on the FPGA and should be shortened. As a result, only the solver of equations accelerates enough the simulation for achieving real-time performance.

Results show that an FPGA can be employed to accelerate a MB simulation. This work also establishes guidelines to select and to implement any operation of a MB simulation in an FPGA. It should be noted that the achieved acceleration will depend on the size of the model and the FPGA resources. Thus, the acceleration required and the model size are important factors to determine the size and type of the FPGA required for a given application. To conclude, as the list of processors including FPGAs is increasing, it is important to learn how to take advantage of them. In model-based applications where the simulation time has to be reduced to meet execution time requirements, an efficient FPGA implementation can be decisive in making an implementation successful.

8 Acknowledgements

The support of the Spanish Ministry of Economy and Competitiveness (MINECO) under project TRA 2014-59435-P and through the grant BES-2015-071372, co-financed by the European Union through the ERDF program, is greatly acknowledged.

References

[1] Model-based system testing: Efficiently combining test and simulation for model-based development. https://www.plm.automation.siemens.com/global/en/topic/virtual-sensing-techniques-and-their-applications/67985 (2019). Accessed: 2020-03-02

[2] Cuadrado, J., Dopico, D., Perez, J.A., Pastorino, R.: Automotive observers based on multibody models and the extended Kalman filter. Multibody Syst Dyn **27**(1), 3–19 (2011). https://doi.org/10.1007/s11044-011-9251-1

[3] Pastorino, R., Richiedei, D., Cuadrado, J., Trevisani, A.: State estimation using multibody models and non-linear Kalman filter. International Journal of Non-Linear Mechanics **53**, 83–90 (2013)

[4] Sanjurjo, E., Naya, M.Á., Blanco-Claraco, J.L., Torres-Moreno, J.L., Giménez-Fernández, A.: Accuracy and efficiency comparison of various nonlinear Kalman filters applied to multibody models. Nonlinear Dynamics (2017). DOI 10.1007/s11071-017-3354-z

[5] Sanjurjo, E., Dopico, D., Luaces, A., Naya, M.A.: State and force observers based on multibody models and the indirect Kalman filter. Mechanical

Systems and Signal Processing **106**, 210–228 (2018). https://doi.org/10.1016/j.ymssp.2017.12.041

[6] Wolf, W., Jerraya, A., Martin, G.: Multiprocessor System-on-Chip (MPSoC) Technology. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **27**(10), 1701–1713 (2008). https://doi.org/10.1109/TCAD.2008.923415

[7] Bélanger, J., Venne, P., Paquin, J.: The What, Where and Why of Real-Time Simulation. Tech. rep., OPAL-RT p. 13 (2010)

[8] Dorta, T., Jiménez, J., Martín, J.L., Bidarte, U., Astarloa, A.: Reconfigurable Multiprocessor Systems: A Review. Int. J. Reconfigurable Comput. **2010**, 1–10 (2010). https://doi.org/10.1155/2010/570279

[9] Chung, E.S., Milder, P.A., Hoe, J.C., Mai, K.: Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs? In: 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 225–236. IEEE, Atlanta, GA, USA (2010). https://doi.org/10.1109/MICRO.2010.36

[10] Kumar, R., Tullsen, D.M., Jouppi, N.P., Ranganathan, P.: Heterogeneous chip multiprocessors. Computer **38**(11), 32–38 (2005). https://doi.org/10.1109/MC.2005.379

[11] Valentin, E., de Freitas, R., Barreto, R.: Towards optimal solutions for the low power hard real-time task allocation on multiple heterogeneous processors. Sci. Comput. Program. **165**, 38–53 (2018). https://doi.org/10.1016/j.scico.2017.08.005

[12] Radhamani, A.S.: Performance Analysis of Homogeneous and Heterogeneous Multicore Processor Using Static and Dynamic Schedulers. Asian J. Inf. Technol. **15**, 533–541 (2016). http://docsdrive.com/pdfs/medwelljournals/ajit/2016/533-541.pdf

[13] Crockett, L.H., Elliot, R.A., Enderwitz, M.A., Stewart, R.W.: The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC, first edn. Strathclyde Academic Media (2014)

[14] Kumar, R., Farkas, K., Jouppi, N., Ranganathan, P., Tullsen, D.: Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In: 22nd Digital Avionics Systems Conference. Proceedings (Cat. No.03CH37449), pp. 81–92. IEEE Comput. Soc, San Diego, CA, USA (2003). https://doi.org/10.1109/MICRO.2003.1253185

[15] Choi, Y.K., Cong, J., Fang, Z., Hao, Y., Reinman, G., Wei, P.: In-Depth Analysis on Microarchitectures of Modern Heterogeneous CPU-FPGA Platforms. ACM Trans. Reconfigurable Technol. Syst. **12**(1), 1–20 (2019). https://doi.org/10.1145/3294054

[16] Nunez-Yanez, J., Amiri, S., Hosseinabady, M., Rodríguez, A., Asenjo, R., Navarro, A., Suarez, D., Gran, R.: Simultaneous multiprocessing in a software-defined heterogeneous FPGA. J. Supercomput. (2018). https://doi.org/10.1007/s11227-018-2367-9

[17] Dendaluce Jahnke, M., Cosco, F., Novickis, R., Pérez Rastelli, J., Gomez-Garay, V.: Efficient Neural Network Implementations on Parallel Embedded Platforms Applied to Real-Time Torque-Vectoring Optimization Using Predictions for Multi-Motor Electric Vehicles. Electronics **8**(2), 250 (2019). https://doi.org/10.3390/electronics8020250

[18] Mazhar, H., Heyn, T., Negrut, D.: A scalable parallel method for large collision detection problems. Multibody Syst Dyn **26**(1), 37–55 (2011). https://doi.org/10.1007/s11044-011-9246-y

[19] Negrut, D., Tasora, A., Mazhar, H., Heyn, T., Hahn, P.: Leveraging parallel computing in multibody dynamics. Multibody Syst Dyn **27**(1), 95–117 (2012). https://doi.org/10.1007/s11044-011-9262-y

[20] Serban, R., Melanz, D., Li, A., Stanciulescu, I., Jayakumar, P., Negrut, D.: A GPU-based preconditioned Newton-Krylov solver for flexible multibody dynamics. International Journal for Numerical Methods in Engineering **102**(9), 1585–1604 (2015). https://doi.org/10.1002/nme.4876

[21] Cano, J.C., Cuenca, J., Giménez, D., Saura-Sánchez, M., Segado-Cabezos, P.: A parallel simulator for multibody systems based on group equations. J Supercomput **75**(3), 1368–1381 (2019). https://doi.org/10.1007/s11227-018-2602-4

[22] Chakraborty, S., Lukasiewycz, M., Buckl, C., Fahmy, S., Naehyuck Chang, Sangyoung Park, Younghyun Kim, Leteinturier, P., Adlkofer, H.: Embedded systems and software challenges in electric vehicles. In: 2012 Design, Automation & Test in Europe Conference & Exhibition (2012), pp. 424–429. IEEE, Dresden (2012). https://doi.org/10.1109/DATE.2012.6176508

[23] Stitt, G., Lysecky, R., Vahid, F.: Dynamic hardware/software partitioning: a first approach. In: Proceedings of the 40th conference on Design automation - DAC '03, p. 250. ACM Press, Anaheim, CA, USA (2003). https://doi.org/10.1145/775832.775896

[24] Lin, Y.: Using FPGAs to Solve Challenges in Industrial Applications. Tech. rep., Xilinx (2011)

[25] Chiang, J., Zammattio, S.: Intel® MAX® 10 and Cyclone® FPGAs help industrial designs adapt to changing requirements, and lower total solution costs. Tech. rep., Intel

[26] Mata-Carballeira, O., Gutiérrez-Zaballa, J., del Campo, I., Martínez, V.: An FPGA-Based Neuro-Fuzzy Sensor for Personalized Driving Assistance. Sensors **19**(18), 4011 (2019). https://doi.org/10.3390/s19184011

[27] Han, Y., Oruklu, E.: Real-time traffic sign recognition based on Zynq FPGA and ARM SoCs. In: IEEE International Conference on Electro/Information Technology, pp. 373–376 (2014). https://doi.org/10.1109/EIT.2014.6871793

[28] Brenot, F., Fillatreau, P., Piat, J.: FPGA based accelerator for visual features detection. In: 2015 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), pp. 1–6 (2015). https://doi.org/10.1109/ECMSM.2015.7208697

[29] Zhou, Y., Chen, Z., Huang, X.: A system-on-chip FPGA design for real-time traffic signal recognition system. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1778–1781 (2016). https://doi.org/10.1109/ISCAS.2016.7538913

[30] Sahlbach, H., Thiele, D., Ernst, R.: A system-level FPGA design methodology for video applications with weakly-programmable hardware components. J Real-Time Image Proc **13**(2), 291–309 (2017). https://doi.org/10.1007/s11554-014-0403-4

[31] Saponara, S.: Hardware accelerator IP cores for real time Radar and camera-based ADAS. J Real-Time Image Proc **16**(5), 1493–1510 (2019). https://doi.org/10.1007/s11554-016-0657-0

[32] Helali, A., Ameur, H., Górriz, J.M., Ramírez, J., Maaref, H.: Hardware implementation of real-time pedestrian detection system. Neural Comput & Applic (2020). https://doi.org/10.1007/s00521-020-04731-y

[33] del Campo, I., Martínez, V., Echanobe, J., Asua, E., Finker, R., Basterretxea, K.: A versatile hardware/software platform for personalized driver assistance based on online sequential extreme learning machines. Neural Comput & Applic **31**(12), 8871–8886 (2019). https://doi.org/10.1007/s00521-019-04386-4

[34] XA Zynq-7000 SoC Data Sheet: Overview. DS188 (v1.3.2). https://www.xilinx.com/support/documentation/data_sheets/ds188-XA-Zynq-7000-Overview.pdf (2018). Accessed: 2020-02-21

[35] Zynq-7000 SoC Data Sheet: Overview. DS190 (v1.11.1). https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (2018). Accessed: 2020-02-21

[36] Moore, A.: FPGAs For Dummies®, 2nd Intel® Special Edition. John Wiley and Sons, Inc. (2017)

[37] Cuadrado, J., Dopico, D., Gonzalez, M., Naya, M.A.: A Combined Penalty and Recursive Real-Time Formulation for Multibody Dynamics. J. Mech. Des. **126**(4), 602 (2004). https://doi.org/10.1115/1.1758257

[38] Cuadrado, J., Dopico, D., Naya, M.A., Gonzalez, M.: Real-Time Multibody Dynamics and Applications. In: G. Maier, J. Salençon, W. Schneider, B. Schrefler, P. Serafini, M. Arnold, W. Schiehlen (eds.) Simulation Techniques for Applied Dynamics, vol. 507, pp. 247–311. Springer Vienna, Vienna (2008). https://doi.org/10.1007/978-3-211-89548-1_6

[39] Dopico, D.: Formulaciones semi-recursivas y de penalización para la dinámica en tiempo real de sistemas multicuerpo. Ph.D. thesis, Universidade da Coruña (2004)

[40] Pastorino, R., Cosco, F., Naets, F., Desmet, W., Cuadrado, J.: Hard real-time multibody simulations using ARM-based embedded systems. Multibody Syst. Dyn. **37**(1), 127–143 (2016). https://doi.org/10.1007/s11044-016-9504-0

[41] Cuadrado, J., Gutierrez, R., Naya, M.A., Morer, P.: A comparison in terms of accuracy and efficiency between a MBS dynamic formulation with stress analysis and a non-linear FEA code. International Journal for Numerical Methods in Engineering **51**(9), 1033–1052 (2001). https://doi.org/10.1002/nme.191

[42] David, J.P.: Low latency and division free Gauss–Jordan solver in floating point arithmetic. J. Parallel Distr. Com. **106**, 185–193 (2017). https://doi.org/10.1016/j.jpdc.2016.12.013

[43] Guennebaud, G., Jacob, B., et al.: Eigen v3. http://eigen.tuxfamily.org (2010)