

# FPGA IMPLEMENTATION OF WHEEL-RAIL CONTACT LAWS

Y.J. Zhou<sup>1</sup>, T.X. Mei<sup>2</sup>, S. Freear<sup>3</sup>

<sup>1,3</sup> *The University of Leeds, School of Electronic and Electrical Engineering, UK*

<sup>2</sup> *Salford University, School of Computing, Science and Engineering, UK*

<sup>1</sup> *(eenyz@leeds.ac.uk)*

<sup>2</sup> *(t.x.mei@theiet.org) – contact author*

<sup>3</sup> *(s.freear@leeds.ac.uk)*

---

**Abstract:** This paper presents the development of an FPGA (Field Programmable Gate Array) based accelerator that is highly suited to the implementation of complex and computationally demanding control or real-time simulation tasks. The application studied in this paper is for the real-time simulation of wheel-rail contact laws, which may be used for hardware-in-the-loop experimental studies of the latest active control technology for wheelset stabilization and steering. The complex wheel-rail contact laws are implemented using a single FPGA chip that outperforms substantially modern general-purpose CPU or DSP in the aspects of processing time, configuration flexibility and cost. The algorithms are restructured to utilise the FPGA's parallel processing capability. Reusable IP core are used for the floating point operations. The scheduling of the operations is optimised to ensure effective and efficient use of the FPGA's resources.

---

## 1. INTRODUCTION

The implementation of complex control laws or computationally demanding simulation algorithms for real-time applications often requires the use of high performance processors and sometimes complex multi-processor architectures. In particular, hardware-in-the-loop approaches are now becoming increasingly popular in the development of control solutions for large and complex systems where full scale testing can be time consuming and/or expensive to carry out. Real-time simulations for such systems necessitate great computational power as well as hardware complexity. One such application is in a simulation of an arc electric drive, where the drive motor and inverter are simulated in real-time using Hypersim which is based on a Dec Alpha processor for the calculations and a Sharc DSP for the communications [6]. In another example, three Texas Instruments TMS320C40 DSPs were used in order to achieve the desired performance for the real-time simulation of an electrical machine and power converters [7]. The complexity of the models has been identified as the main cause of the high computational demand, but the nature of the largely sequential processing of software algorithms also contributes significantly to the problem.

In the railway industry, there have recently been much increased research and development activities in the area of active controls, especially in the primary suspensions for active steering and stability control to overcome the fundamental design conflict between the high speed stability and deteriorated low speed performance on curves in rail vehicle design [11]. Simulation of the behaviour of railway vehicles is regarded as an essential design method in order to verify the improvement in dynamic performance offered by the emerging control technologies. Many modern simulation software packages with detailed modelling elements offer convenience but require more computation power. Furthermore, the requirement of simulation in real-time is increasing significantly because it offers a realistic means to test the control strategies, using hardware-in-the-loop, without the need of real track experiments which are both economically and logistically difficult. One example is a study in [2] which requires the simulation of the interaction between the wheel and the rail in real-time. In this case, parallel processing via the use of multiple CPUs is applied for the intensive computation, resulting in complex architectures and also expensive costs. On the other hand, the latest embedded multi-processing system technology makes it viable to design custom accelerators for complex and computationally demanding simulation models. Contemporary platforms/devices for this purpose include, multi-core PC processors, e.g. [22][23], multi-core GPUs (graphics processing units), e.g. [24], multi-core DSP devices, e.g. [25][26], and some general multi-core processors, [27][28][29]. However, the custom accelerator designs based on this fixed hardware would be platform-specific, with little scalability and limited flexibility. An FPGA device, with its high hardware/software configuration scalability and flexibility and outstanding execution parallelism, are finding increasing usage in control applications. For example, in a real-time digital power system simulator, FPGA devices are used as pre-processors, co-processors and post-processors connected to a main DSP processor to achieve a time step reduction [10]. Moreover, with the density of FPGAs steadily increasing, they have reached the point where they are capable of implementing complex floating-point applications [12], which will drive more applications into the FPGA field.

For the simulation of railway vehicle dynamics, the main part that demands the most computation power is the complex and non-linear wheel-rail contact laws. Therefore the great potential in reducing the overall simulation time is to find solutions to remove the bottleneck, and the use of a dedicated accelerator is clearly a way forward. This paper presents the development of an effective and flexible accelerator to reduce the processing time and eventually to help achieve simulation in real-time.

The paper is structured as follows: An introduction is given in section 1. Section 2 presents the overall requirement for the proposed accelerator and also provides an initial assessment on the performance potential with different approaches. Various aspects of FPGA design are discussed in section 3. The FPGA implementation is discussed in section 4 and key results are also provided. Finally conclusions are given in section 5.

## 2. COMPUTATIONAL REQUIREMENT

Modern railway vehicles are typically comprised of a body frame, two additional masses (known as bogies) and four solid-axle wheelsets [5]. The vehicle frame is connected to the bogies via secondary suspensions and each of the bogies is connected to two wheelsets via primary suspensions. However, railway vehicles of the future are expected to be mechanically much simpler, enabled by the advances in active control technologies, e.g. the two-axle vehicles [4, 5]. A two axle vehicle consisting of a body frame and two wheelsets (or four wheels), as shown in Figure 1, is similar to the conventional bogies. In this paper a two axle vehicle is used as an example for the development of the proposed accelerator, without the loss of generality, as it may be considered as an independent system or a part of a complete rail vehicle (in the form of a bogie).

The equations governing the dynamic motions of the body frame and the wheelsets are normally standard second order differential equations and do not require excessive computation power. Furthermore, many applications may not necessarily need all degrees of freedom to be modelled, e.g. studies of wheelset steering control and wheel-rail interactions typically involve the plan-view motions only, as in equations (1)-(6) [11]. The meaning of the equation symbols can be found in the LIST OF PARAMETERS at the end of the paper.

Front wheelset's lateral and yaw motions:

$$m_w \times \ddot{y}_{1w} = T_{1Lx} + T_{1Rx} + F_{1wc} - F_{1wm} + (y_{1w} - y_b) \times K_s + (\dot{y}_{1w} - \dot{y}_b) \times C_s \quad (1)$$

$$I_w \times \ddot{\psi}_{1w} = T_{1Ly} \times L_g - T_{1Ry} \times L_g - k \times \psi_w \quad (2)$$

Rear wheelset's lateral and yaw motions:

$$m_w \times \ddot{y}_{2w} = T_{2Lx} + T_{2Rx} + F_{2wc} - F_{1wm} + (y_{1w} - y_b) \times K_s + (\dot{y}_{1w} - \dot{y}_b) \times C_s \quad (3)$$

$$I_w \times \ddot{\psi}_{2w} = T_{2Ly} \times L_g - T_{2Ry} \times L_g - k \times \psi_w \quad (4)$$

Body frame's lateral and yaw motions:

$$m_b \times \ddot{y}_b = F_{bc} - F_{bm} - (y_{1w} + y_{2w} - 2y_b) \times K_s - (\dot{y}_{1w} + \dot{y}_{2w} - 2\dot{y}_b) \times C_s \quad (5)$$

$$I_b \times \ddot{\psi}_b = -F_{1wb} + F_{2wb} - (y_{1w} - y_{2w}) \times K_s - (\dot{y}_{1w} + \dot{y}_{2w}) \times C_s \quad (6)$$

However, the contact forces at the wheel-rail interfaces dominate the dynamic behaviour of railway vehicles and are fundamental in the provision of traction, braking and guidance forces [3]. The contact forces are caused by so-called "creepages" between the wheel and rail surfaces which are small relative velocities, resulting from elastic deformation of the steel at the point of contact. The dynamic properties of the contact forces (or creep forces) can vary significantly, and the

overall creep force (resultant of the forces in the longitudinal and lateral directions) at the contact point is a non-linear function of the creep, influenced by the non-linear contact geometry and limited by the adhesion available. The modelling and simulation of the complex contact mechanics with good accuracy for the study of vehicle dynamics and control was the subject of intensive research in 1970s-80s [9], and still remains a significant challenge for applications where fast simulations are desired. Even for simplified versions of the contact theory, e.g. Kalker and Hertz's theories [9] - the most widely used wheel-rail contact laws, the computations involved are still difficult to implement in real-time. The contact laws based on Hertz and Kalker algorithms are used in this study to demonstrate the potential performance gains offered by the proposed solution, but the design may be extended further if more sophisticated models are required. In Kalker's theory, the contact force at each wheel-rail interface is a result of double integration of traction forces over the corresponding contact patch as shown in equations (7)-(10) for the four wheels of the two wheelsets.

$$(T_{1Lx}, T_{1Ly}) = \iint_{U_{1L}} \mathbf{p}_{1L}(x, y) dx dy \quad (7)$$

$$(T_{1Rx}, T_{1Ry}) = \iint_{U_{1R}} \mathbf{p}_{1R}(x, y) dx dy \quad (8)$$

$$(T_{2Lx}, T_{2Ly}) = \iint_{U_{2L}} \mathbf{p}_{2L}(x, y) dx dy \quad (9)$$

$$(T_{2Rx}, T_{2Ry}) = \iint_{U_{2R}} \mathbf{p}_{2R}(x, y) dx dy \quad (10)$$

The calculation of the double integrations in equations (7)-(10) is computationally demanding. The contact areas for each of the contact patches  $U_{1L}$ ,  $U_{1R}$ ,  $U_{2L}$  and  $U_{2R}$  must first be determined. The contact positions of the wheel and rail surfaces are non-linear functions of the wheelset movements and wheel-rail profiles, which may be simplified using pre-determined look-up tables. The contact areas are then calculated using the Hertz theory which assumes that the pressure distribution is semi-elliptical at the contact patch and the contact area is elliptical with semi-axes  $a$ ,  $b$ . The semi-axes  $a$  and  $b$  of a contact patch may be determined using equations (11)-(12), where the parameters are related to the geometries of the wheel and rail surfaces at their respective contact points and the normal force applied for each of the contact patches.

$$a = m \times (3 \times N \times (1 - \nu^2) / (2 \times E \times (A + B)))^{1/3} \quad (11)$$

$$b = n \times (3 \times N \times (1 - \nu^2) / (2 \times E \times (A + B)))^{1/3} \quad (12)$$

Based on this contact size, the integration of traction forces ( $\mathbf{p}_{1L}$ ,  $\mathbf{p}_{1R}$ ,  $\mathbf{p}_{2L}$  and  $\mathbf{p}_{2R}$ ) is then carried out, normally in a numerical manner by dividing the contact patch into  $m_0 \times n_0$  small elements – an algorithm known as Fastsim [9][14].

Figure 2 is an illustration of the Fastsim calculation at the front left wheel contact patch where the contact area has been determined by Hertz and  $m_0=10$  and  $n_0=10$  are set. Within each small element, the traction  $\mathbf{p}_{1L}(x, y)$  is considered constant and is determined using equations (13)-(16). The tangential traction force  $\mathbf{p}_{1LH}(x, y)$  is dependent on the creepages in longitudinal and lateral directions  $(v_x, v_y)$ , and spin  $\varphi$ , and the flexibility  $(L_1, L_2, L_3)$  which are related to the material properties of the wheel and rail, as shown in equation (15)-(16). The traction force is also checked against the traction bound  $t_b$  (i.e. adhesion limit of the particular element).

$$\mathbf{p}_{1L}(x, y) = \begin{cases} \mathbf{p}_{1LH}, & \text{if } |\mathbf{p}_{1LH}| \leq t_b \text{ (at adhesion area)} \\ (t_b / |\mathbf{p}_{1L}|) \times \mathbf{p}_{1L}, & \text{if } |\mathbf{p}_{1LH}| > t_b \text{ (at slip area)} \end{cases} \quad (13)$$

where

$$t_b = 3 \times m_u \times N \times \sqrt{1 - x^2/a^2 - y^2/b^2} / (2 \times \pi \times a \times b) \quad (14)$$

$$\mathbf{p}_{1LH}(x, y) = \begin{cases} d_x \times \mathbf{c}(x + d_x/2, y) / (V \times L) & (j = 0, \text{ leading edge section of each row}) \\ \mathbf{p}_{1LH}(x - d_x, y) - d_x \times \mathbf{c}(x + d_x/2, y) / (V \times L) & (j \neq 0, \text{ other sections of each row}) \end{cases} \quad (15)$$

where  $\mathbf{c}(x, y) / (V \times L)$  is approximated by

$$\mathbf{c}(x, y) / (V \times L) = \begin{cases} v_x / L_1 - \varphi \times y / L_3 & (x \text{ direction component}) \\ v_y / L_2 + \varphi \times x / L_3 & (y \text{ direction component}) \end{cases} \quad (16)$$

The traction forces of the small elements are determined in a sequence as numbered in

Figure 2, and the total traction force is summed up at the end. A larger number of small elements would lead to higher integration accuracy, but there is an obvious trade-off with computation demand. The range of grid dimensions used for contact patch analysis varies in the literature and depends on the application. However for rail contact analysis typical values of grid size range from a minimum of 5x5 ( $m_0=5$  and  $n_0=5$ ) in [14], a medium of 14x14 in [30], to a maximum of 50x50 in [31]. For this application a grid dimension of 10x10 is found to provide sufficient accuracy and will form the basis for implementation and comparative studies across competing platforms including PC, DSP and FPGA. The design aim however, provides sufficient flexibility to implement higher order grid dimensions through simple software or hardware configuration.

In the simulation for the two axle vehicle model, the computation of equations (11)-(16) will have to be repeated 4 times to produce  $\mathbf{p}_{1L}$ ,  $\mathbf{p}_{1R}$ ,  $\mathbf{p}_{2L}$  and  $\mathbf{p}_{2R}$  for all 4 contact patches (8 times for a bogie vehicle) which will then be double integrated. The contact laws consume a large percentage of the computations. A MATLAB/SIMULINK simulation running on an Intel Pentium4 3.0GHz 1GB memory platform takes 303 seconds during which the Windows Task Manager shows on average 99% of non-idle CPU time is spent on MATLAB and 1% of non-idle CPU time on other OS tasks. The profiling command within

MATLAB shows that it takes 2% of the total computation time to find the contact areas and 69% to calculate the forces, whereas the computation to solve the differential equations for all the dynamic motions consumes the remaining 29%. The contact model computation time for 4 patches in each step is 21.5ms, which prevents the whole model from being simulated in real-time, which is typically required to be less than 1ms in step size.

It is proposed to assign a custom accelerator for the contact model in order to reduce its calculation time for 4 patches to be less than 1ms which would eventually allow the simulation to be run in real-time with an acceptable step size. The dynamic equations (1)-(6) can still be run on a PC, allowing for the necessary flexibility for different vehicle configurations. These will enable the use of hardware-in-the-loop simulation for testing a target controller. The overall arrangement for the simulation is shown in Figure 3.

The accelerator implementation was first attempted using a DSP platform. A TI's second generation floating point DSP C6713B was used to implement the contact model, resulting in a time duration of 1.6ms for 4 patches in TI's DSP simulation tool CCStudio. Although it is 12.4 times faster than the Pentium4 3.0GHz, the target requirement cannot be met on a single chip of this processor. Even for the TI's latest third generation floating point DSP C6727B that gives an enhancement of about 17% over C6713B (from 600 to 700 peak MMACS), it is still not sufficient for the requirement. It is possible to utilise an array of DSPs, e.g., four DSPs dealing with the four patches individually in parallel to get a 3-time enhancement, but this would not be the best approach not only because of the complexity in building arrays of DSPs [1] but also because of such platform's poor adaptability to a different number of contact patches. An alternative approach is considered in this study using FPGA. The target device is an Altera's CycloneII FPGA EP2C35, the cost of which is comparable to that of DSP devices such as TI C6713B. The study takes advantage of FPGA's flexible configurability, and optimises the implementation of the algorithms of contact laws to minimise the computation time within the available resource constraint. Parallel processing and pipelining capability are also exploited in dealing with all of the four contact patches, as discussed in the next section.

### 3. FPGA DEVELOPMENT

Five key aspects of the implementation of wheel-rail contact laws (Hertz and Fastsim) using an FPGA based approach will be discussed in this section.

#### 3.1 *Fastsim Algorithm Restructuring*

As described in section 2, the original Fastsim algorithm uses a numerical solving method. Its data flow is shown in Figure 4 (a). There is an inner  $x$ -loop and an outer  $y$ -loop. The  $x$ -loop will execute  $m_0$  times to tackle all  $m_0$  elements in one row, while  $y$ -loop will run for  $n_0$  times to tackle all  $n_0$  rows. In the typical case where  $m_0=10$  and  $n_0=10$ , the CPU would tackle the total 100 small elements ( $m_0 \times n_0$ ) one by one in the sequence as shown in

Figure 2.

Sequential processing inside  $x$ -loop is inevitable as in one particular row the tangential traction  $\mathbf{p}_H$  for a element is associated with the previous element's traction force. However results for one particular row can be worked out separately from other rows because all the computations in the  $n_0$  rows as well as the initiations of the  $x$ -loop are independent of any other rows. As long as there is the necessary computing capability and flexibility in the processor architecture for parallel executions which can be achieved in the FPGA designs, it is possible to split the sequential  $y$ -loop of the original algorithm into  $n_0$  parallel processes, as shown in Figure 4 (b).

In the restructured Fastsim algorithms, the  $x$ -loop operation remains the same to update tangential forces  $\mathbf{T}(1), \mathbf{T}(2) \dots \mathbf{T}(n_0)$  for each row. The results of all parallel processes will be summed up to achieve a total contact force  $\mathbf{T}$  at the final operation. Ideally all of these  $n_0$  parallel processes can be executed simultaneously by  $n_0$  processors embedded in a single FPGA, achieving approximately  $n_0$  times faster performance than the original one. With the resources available from the chosen FPGA device for the study, it is only found possible to compute 5  $y$ -loops in parallel with 5 separate processors on the device and hence the performance is expected to be 5 times faster than the original.

### 3.2 FPGA Overall Architecture

The overall FPGA hardware architecture is presented in Figure 5. There are a total of 6 CPUs implemented as an Altera NiosII, which is a 32-bit configurable processor with full 32-bit instruction set, data path and address space. NiosII/f is its fastest version employing a 6-stage pipeline. One NiosII processor (CPU<sub>0</sub>) is dedicated for processing the Hertz theory (equations (11)-(12)) of calculating the contact areas. Five NiosII processors (CPU<sub>1</sub>-CPU<sub>5</sub>) are used to implement the restructured Fastsim algorithms, each responsible for two  $y$ -loops of the algorithms. One set of FPU (floating point units) for floating point operations is dedicated to CPU<sub>0</sub> whereas another set of FPU is shared by CPU<sub>1</sub>-CPU<sub>5</sub> using bidirectional FIFO (first-in-first-out) memory. Processors are allocated for Hertz and for Fastsim separately because in this way the two parts can be pipelined, which improves the throughput significantly in calculating two or more contact patches in one step. Also the Hertz processor CPU<sub>0</sub> can handle other miscellaneous tasks such as communications with external devices and the host PC via the JTAG port.

The communication between the Hertz and the Fastsim processes is achieved via an internal dual-port memory, and the communication among CPU<sub>1</sub>-CPU<sub>5</sub> is done via unidirectional FIFO memories (TF1-TF5). An internal data bus is used to enable the processors to access the FPU. Each processor has dedicated memory for storing software program. And the contact laws' input and output data are stored in the external memory SDRAM.

The overall software architecture is presented in Figure 6 and Figure 7. Figure 6 shows the main routines of the software program run in each implemented processor (CPU<sub>0</sub>~CPU<sub>5</sub>). These processors perform memory-mapped I/O access. Both memories and FPUs are mapped into the address space of the processors. The code is programmed to tackle two contacts in one step. The corresponding execution sequence of block S1, S2 and S3 in the routines is shown in Figure 7. At the beginning block S1 code in CPU<sub>0</sub> takes approximately 3800 clock cycles to calculate the size data for contact 1 using the Hertz algorithms and write it into the dual-port memory, and then continued with the second execution for contact 2. Once CPU<sub>1</sub>~CPU<sub>5</sub> acknowledge the data validity in the dual-port memory, they run block S2 code to compute forces for contact 1 using the restructured Fastsim algorithms and write the result back to the dual-port memory which totally take about 16150 clock cycles. Note that in each of CPU<sub>1</sub>~CPU<sub>5</sub>, the appointed two y-loops are executed one after another. Then, CPU<sub>1</sub>~CPU<sub>5</sub> continue with the computation of contact 2 as its size data has already been prepared. At acknowledgement, CPU<sub>0</sub> runs block S3 code to perform the summation calculation for the final contact force result which takes about 950 cycles. The whole calculation for two contacts are carried out in a 3-stage-pipeline (S1, S2, S3) manner, taking about 37050 cycles in total, which is 11% faster than that of running in an un-pipelined or serial manner.

### *3.3 Floating Point Operation Implementation*

Floating point operations are required throughout the Fastsim and Hertz algorithms. The total floating point operation requirements for each contact patch equate to 1168 additions/subtractions, 1114 multiplications, 264 divisions and 212 square-roots, of which 97%, 98%, 95% and 99% respectively are required in the Fastsim algorithms.

In conventional digital signal processing practice, floating point operations are usually converted to fixed point arithmetic and implemented to minimise device resource usage and execution time under accuracy constraint. However, such conversion tends to require great manual effort in tuning the word-lengths and scaling factors for the variables according to their dynamic range and the required accuracy [15][16]. Although automatic conversion methodologies have become available for both hardware and software implementations, they tends to be either model-related [17][18] or processor/device-specific [19][20].

This study avoids such conversion and utilises reusable hardware IPs to implement floating point operations in order to reduce execution time. Also by using floating point directly, without the quantization errors produced by fixed point representation, the dynamic range and the accuracy requirements specified by varying application scenarios can easily be met [21].

Single precision provides sufficient accuracy in the model studied. A set of well developed open source floating point units (FPUs) [8] is used in the design, which offers the advantage that operations can be executed in parallel in different FPUs independently. The reusability of IPs makes it more flexible to allocate more resources to a particular type of operation, i.e. one more multiply, divide and square-root FPUs are implemented for the Fastsim part than that for the Hertz part, as shown in Figure 5, considering the difference of the floating point computation intensity between them.

The FPUs used are all 3-stage-pipelined and can run in single-task mode or bulk-task mode. In the Hertz part, the four FPUs run in single-task mode in which a particular FPU can only be accessed in an order as write-read-write-read..., i.e. CPU<sub>0</sub> has to read out the result from the FPU before sending a second task. In the Fastsim part, the seven FPUs are attached to the bidirectional FIFO memories before being connected to the data bus. These memories buffer both task requests from the processors (CPU<sub>1</sub>-CPU<sub>5</sub>) and results from the FPUs. Therefore the FPUs can run in bulk-task mode in which they can start carrying out a second task within the pipeline timescale and without the need to wait for the result to be read out by the processors. Bulk-task mode has higher throughput than single-task mode, as it produces a lower latency for subsequent tasks as shown in Table 1.

IP core	Symbol	without FIFO Single-task mode	with FIFO Bulk-task mode
Add/Subtract	Add	7/7	7/3
Multiply	Mul	12/12	12/8
Divide	Div	35/35	35/31
Square root	Sqrt	33/33	33/29

Table 1. Floating point unit performance (initial latency/subsequent latency, unit: clock cycles)

Due to the memory resource limit on the device and the much smaller floating point computation demand in the Hertz part, no bidirectional FIFOs are used for the FPUs in the Hertz part.

### 3.4 Multi-CPU sharing FPUs

As multiple NiosII processors are required for the parallel processes in the restructured Fastsim algorithm, it would seem to be desirable to use dedicated FPU sets to each of the processors so that they run independently. However, this would substantially increase the resources required of the FPGA device and has been found to be inefficient [13].

A more efficient technique for sharing the FPUs with multi-NiosIIs is proposed. As in the Fastsim part in Figure 5, seven FPUs are shared by five processors. A token ring scheme is introduced to the design. Five unidirectional FIFOs, TF1-TF5, for buffering tokens, are put in between the processors to form a ring topology. A token, specified for accessing a particular FPU, circles through the ring and a processor is programmed to acquire the token to have the privilege to access that FPU. In the main routines in CPU1-CPU5 shown in Figure 6, writing operands to and obtaining results from a FPU are by calling subroutines FPUWR() and FPURD() respectively, defined in Figure 8. FPUWR() is executed in 3 steps - in CPU1 for example, read token from FIFO TF1 (step W1), write task to FPU (step W2), and write token to FIFO TF2 (step W3). FPURD() is executed in 3 steps similarly, represented by R1, R2 and R3. The execution of these steps in the token ring scheme is illustrated by Figure 9 (a). Each of the processors is programmed to run FPUWR->FPURD->FPUWR->FPURD to perform two consecutive divide operations using one of the two divide FPUs. The five processors start running simultaneously. It takes a few cycles for a processor to call the subroutines FPUWR()/FPURD(), depicted by the striped blocks. A processor stalls when it is waiting for tokens or FPU results to be ready. Two tokens are initialized at TF1 at different addresses, one for writing the divide FPU and the other for reading the divide FPU, which firstly allows CPU<sub>1</sub> to acquire the token and start accessing the FPU, followed by CPU<sub>2</sub>, CPU<sub>3</sub>, CPU<sub>4</sub> and CPU<sub>5</sub>. The token transfer paths are denoted by single-line arrows. The FPU run in bulk-task mode and some of the floating-point data/task transfer paths are denoted by double-line arrows in Figure 9 (a).

### *3.5 Task Scheduling and Optimization*

The goal of optimization is to minimize the whole complete time which should incorporate appropriate design configurations of both hardware and software. On the hardware side, the Fastsim part has a higher priority than the Hertz part in allocating device resources to processors and FPUs at these two parts, as the former has much higher computational demand than the latter. Hence 5 processors and 7 FPUs with buffering FIFOs are implemented at the Fastsim part whereas only 1 processor and 4 FPUs without buffering FIFOs at the Hertz part. On the software side, the order of the processor tasks can be scheduled to eliminate processor stalls as much as possible so that a shorter complete time is achieved. The key concept is to schedule other tasks to the time slot where a particular task is stalled such that the dependence constraints between tasks are preserved. Figure

9 (b) shows the execution of an optimized solution of performing two consecutive divide operations, by using one more divide FPU, the hardware configuration, and by rescheduling the order of the processor tasks as FPUWR->FPUWR->FPURD->FPURD, the software configuration. It is shown that except for the beginning stalls which are mandatory due to the token ring scheme, other processor stalls have been removed, and hence the complete time is reduced from 328 cycles to 274 cycles. The discussed task scheduling approach is applied to the whole processor program for the Fastsim algorithms as well as the Hertz to minimize the whole complete time.

#### 4. FPGA IMPLEMENTATION AND RESULTS

The discussed design is implemented on an Altera's Cyclone II FPGA EP2C35 which is integrated on an evaluation board and configured to run at a frequency of 60MHz by the design. The whole design consumes 96.8% of total 33216 logic elements (LEs) and 100% of total 59.06KB on-chip memory available on the FPGA device, as shown in the resource utilization diagram in Figure 10. The Fastsim part is allocated 3.7 times more LEs and 4.7 times more on-chip memory than the Hertz part. Note that the token FIFO is implemented by LEs due to insufficient on-chip memory on the device. Based on the statistics data, it can be worked out that by using FPU sharing mechanism 38424 LEs (116% of availability) and 22.52KB memory (38% of availability) have been saved, compared to a non-sharing approach.

Two simulation runs are carried out to compare the calculation result by the contact laws model in Matlab (PC) and then by the FPGA, in order to verify the design. The scenario of the test is configured as - a two axle vehicle running at a speed of 50m/s brakes on a straight railway track with irregularities; the simulation time is 10s and the time step is 1ms; the vehicle starts braking at 2s by gradually increasing and then decreasing the longitudinal contact force by 10% of the normal force on each contact, and ceases braking from 9s. The contact laws' input data are firstly loaded to the SDRAM memory on the FPGA evaluation board offline. When the FPGA starts running, it reads the input data from the SDRAM, execute contact laws computation and places the contact force result data back to the SDRAM. The FPGA takes 0.625ms on average for computation of each two contact patches, and 1.25ms for the four contact patches in one step. The result data are then downloaded from the SDRAM offline and compared with the known results calculated by Matlab in floating point double precision. The result data of one of the four contacts with the relative errors in percentage are shown in Figure 11. The relative errors in both longitudinal and lateral directions are within the order of  $10^{-7}$ % which is considered more than acceptable.

It is conceivable that on a larger FPGA device, two copies of the design can be fitted and run in parallel to achieve a performance of 0.625ms execution time for the four contact patches in one step, which can meet the 1ms-step real-time requirement. Furthermore, as the embedded CPUs read inputs at each step, the parameters of the algorithms can be changed at run-time through register access, enabling real-time re-configurability for part of the system.

## 5. CONCLUSION AND FUTURE WORK

In this paper, an FPGA based multiprocessor design for accelerating the execution of the complex and computationally demanding wheel-rail contact laws for real time control implementations has been presented. The Fastsim algorithms have been restructured for utilizing the FPGA's parallel capability. Hardware floating point units are used and shared by several NiosII processors in a token ring scheme to deliver a fast solution with efficient use of resources. The task schedules in the software programmes of each processor are also optimized to reduce the execution time. The FPGA computation results have been verified and the FPGA design has been shown to be suitable for real-time simulation.

The design also has great potential to be used in the implementation of complex control algorithms. For example, many of newly developed control technologies, contain matrix-matrix or matrix-vector multiplications, e.g. [11], resulting in time consuming sequential computation loops and these loops can be accelerated by using the multiprocessing methodology discussed in this study.

The developed multiprocessor design can be easily extended in hardware/software to host additional applications and implemented on the same FPGA device given sufficient unused on-chip resources, or alternatively on a larger FPGA device. The planned further work includes the development of real-time communications between the host PC and the FPGA to allow real hardware-in-the-loop system, and the investigation for the use of a larger FPGA device to improve the real-time performance.

## REFERENCES

- [1] Khandelia, M. Bambha, N.K. Bhattacharyya, S.S. (2006) Contention-conscious transaction ordering in multiprocessor DSP systems, *IEEE Transactions on Signal Processing*, Vol. 54, NO. 2
- [2] Bonaventura, Clifford S., Palese, Joseph W., Zarembski, Allan M.(2000). Intelligent system for real-time prediction of railway vehicle response to the interaction with track geometry, *Proceedings of the IEEE/ASME Joint Railroad Conference*
- [3] Brickle, B.V.(1986). Railway Vehicle Dynamics, Phys Technol 17

- [4] Bruni, S, Goodall, R.M. Mei, T.X. and Tsunashima (2007). Control and Monitoring of railway vehicle dynamics, *Vehicle System Dynamics*, **Vol. 45** - Special Issue: the state of art, 2007, pp743-779
- [5] Busturia, J.M.; Mei, T.X.; Vinolas, J.(2004). Combined active steering and traction for mechatronic bogie vehicles with independently rotating wheels, *Annual Reviews in Control*, **Vol. 28**, pt.2
- [6] Champagne, R., L.-A. Dessaint, H. Fortin-Blanchette (2003). Real-time simulation of electric drives, *Mathematics and computers in simulation*, **Vol. 63**, n 3-5
- [7] Jack, A G; Atkinson, D J and Slater H J. (1998) “Real-time emulation for power equipment development. Part 1: Real-time simulation”, *IEE Proceedings – Electric Power Applications*, **Vol. 145**, No 2, pp. 92-97
- [8] Jidan AI-Eryani (2006) Floating Point Unit, available via [www.opencores.org/projects.cgi/web/fpu100/overview](http://www.opencores.org/projects.cgi/web/fpu100/overview)
- [9] Kalker, J.J.(2000). Rolling contact phenomena – linear elasticity, CISM Courses And Lectures No.411 ‘Rolling contact phenomena’
- [10] Lavoie M. and L.A. Dessaint (1997). FPGAs as accelerators for real-time digital power system simulators, ICDS'97. Second International Conference on Digital Power System Simulators
- [11] Mei T.X. and R.M. Goodall (2000). LQG and GA solutions for active steering of railway vehicles, *IEE Proc.-Control Theory Appl.* **Vol. 147**.
- [12] Beauchamp, M.J. Hauck, S. Underwood, K.D. and Hemmert, K.S. (2008) Architectural Modifications to Enhance the Floating-Point Performance of FPGAs, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **Vol. 16**, No. 2, pp.177-187
- [13] Zhou Y.; Mei, T.X.; Freear S. (2008). FPGA Implementation of Wheel-rail Contact Laws, *UKACC Control 2008*
- [14] J. J. KALKER (1982). A Fast Algorithm for the Simplified Theory of Rolling Contact, *Vehicle System Dynamics*, **Vol. 11** (1982). pp.1-13.
- [15] Zhou Y.; Mei, T.X. (2005). FPGA based real time simulation for electrical machines. *IFAC World Congress 2005*.
- [16] T. Gr’otker, E. Multhaupt, and O. Mauss (1996) Evaluation of HW/SW tradeoffs using behavioral synthesis. *Proceeding of 7th International Conference on Signal Processing Applications and Technology (ICSPAT '96)*, pp. 781–785
- [17] Fiore, P.D. (2008). Efficient Approximate Wordlength Optimization. *IEEE Transactions on Computers*, vol.57, no.11, pp.1561-1570, Nov. 2008
- [18] D. Menard, D. Chillet, and O. Sentieys. (2006). Floating-to-fixedpoint conversion for digital signal processors. *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 96421, 19 pages, 2006.
- [19] Herve, N.; Menard, D.; Sentieys, O. (2005). Data wordlength optimization for FPGA synthesis. *IEEE Workshop on Signal Processing Systems Design and Implementation, 2005.*, vol., no., pp. 623-628, 2-4 Nov. 2005
- [20] Seehyun Kim; Wonyong Sung. (1994) A floating-point to fixed-point assembly program translator for the TMS 320C25. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol.41, no.11, pp.730-739, Nov 1994
- [21] G. Frantz, R. Simar. (2004). Fixed vs. floating point white paper. *Texas Instruments Incorporated* (<http://focus.ti.com/lit/wp/spry061/spry061.pdf>, as of 13 April 2009)
- [22] Intel Corporation. (2008). Intel Core i7 Processor Extreme Edition Series and Intel Core i7 Processor Datasheet Volume 1. <http://download.intel.com/design/processor/datashts/320834.pdf>, as of 14 April 2009.
- [23] Advanced Micro Devices, Inc. (AMD). (2009). Family 10h AMD Phenom II Processor Product Data Sheet. [http://support.amd.com/us/Processor\\_TechDocs/46878\\_Phenom\\_II\\_PDS\\_3.04\\_PUB.pdf](http://support.amd.com/us/Processor_TechDocs/46878_Phenom_II_PDS_3.04_PUB.pdf), as of 14 April 2009.
- [24] NVIDIA. (2008). NVIDIA Tesla C1060 Datasheet. [http://www.nvidia.co.uk/content/PDF/Tesla\\_product\\_literature/NV\\_DS\\_Tesla\\_C1060\\_US\\_Jun08\\_FINAL\\_LowRes.pdf](http://www.nvidia.co.uk/content/PDF/Tesla_product_literature/NV_DS_Tesla_C1060_US_Jun08_FINAL_LowRes.pdf), as of 14 April 2009
- [25] Freescale Semiconductor, Inc. (2009). MSC8144 Quad Core Digital Signal Processor Data Sheet.
- [26] Texas Instruments. (2009). TMS320C6474 Multicore Digital Signal Processor.
- [27] Tiler Corporation, (2008). Tiler TILEPro64TM Processor Product Brief. [http://www.tiler.com/pdf/ProductBrief\\_TILEPro64\\_Web\\_v2.pdf](http://www.tiler.com/pdf/ProductBrief_TILEPro64_Web_v2.pdf), as of 14 April 2009
- [28] Sun Microsystems, Inc. (2007). OpenSPARC™ T2 Core Microarchitecture Specification.
- [29] Sun Microsystems, Inc. (2006). OpenSPARC™ T1 Microarchitecture Specification.
- [30] J. J. Kalker, J. Piotrowski. (1989). Some New Results in Rolling Contact. *Vehicle System Dynamics*, Volume 18, Issue 4 1989, pages 223 – 242
- [31] J. J. Kalker. (1990). Three-dimensional elastic bodies in rolling contact. Kluwer, Dordrecht.

#### LIST OF PARAMETERS

- $A, B$  : Contact patch mean curvatures  
 $a, b$  : semi-axes of contact ellipse  
 $a(y), -a(y)$  : leading edge, trailing edge of contact ellipse  
 $C_s$  : Damping per wheelset (37,000 N s/m)  
 $d_x, d_y$  : length of x-elements, length of y-elements  
 $E, \nu$  : Young’s modulus, Poisson’s coefficient

- $F_{1wc}, F_{2wc}, F_{bc}$ : Centrifugal forces  
 $F_{1wm}, F_{2wm}, F_{bm}$ : Component of the gravity force in lateral direction  
 $F_{1wb}, F_{2wb}$ : Reaction forces between wheelsets and body frame  
 $I_b$ : Wheelset Moment of inertia (558,800 kg·m<sup>2</sup>)  
 $I_w$ : Wheelset Moment of inertia (750 kg·m<sup>2</sup>)  
 $k$ : Spring coefficient (5,000,000 N·m/rad)  
 $K_s$ : Stiffness per wheelset (511,000 N/m)  
 $L$ : flexibility  
 $L_1, L_2, L_3$ : flexibility (for approximation)  
 $L_g$ : Half wheelset axle length (0.75 m)  
 $m, n$ : tabulated nondimensional coefficients  
 $m_w$ : Wheelset mass (1,500 kg)  
 $m_b$ : body frame mass (30,000 kg)  
 $m_u$ : wheel-rail material coefficient (0.3)  
 $m_0, n_0$ : number of x-intervals, number of y-intervals  
 $N$ : Normal force at the contact point  
 $\mathbf{p}_H$ : tangential traction  
 $t_b$ : traction bound at  $(x, y) \in$  contact area  
 $\mathbf{T}$ : total tangential force(contact force)  
 $T_{1Lx}, T_{1Ly}, T_{1Rx}, T_{1Ry}, T_{2Lx}, T_{2Ly}, T_{2Rx}, T_{2Ry}$ :  
 Tangential forces in lateral and longitudinal directions (Front left wheel, front right wheel, rear left wheel, rear right wheel)  
 $y_{1w}, y_{2w}, y_b$ : Wheelsets, body frame's lateral displacement  
 $\psi_{1w}, \psi_{2w}, \psi_b$ : Wheelsets, body frame's yaw displacement  
 $u_x, u_y, \varphi$ : lateral creepage, longitudinal creepage, and spin

#### LIST OF FIGURES

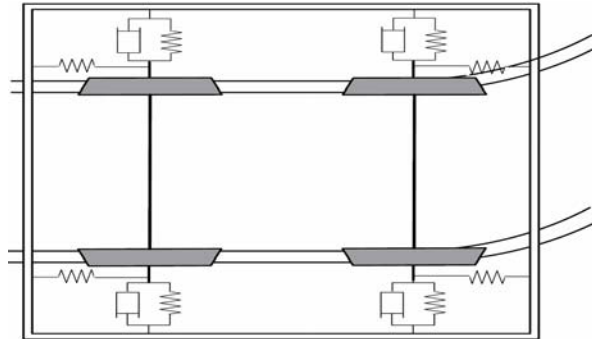


Figure 1. Plan view of a bogie (or a two-axle vehicle)

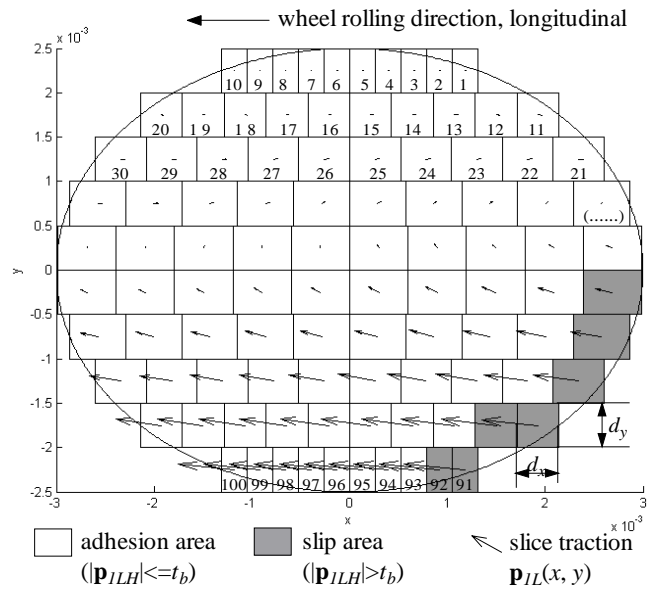


Figure 2. Contact area and Fastsim calculation at the front left wheel contact patch (contact semi-axes  $a=3.0\text{mm}$ ,  $b=2.5\text{mm}$ ; number of  $x$ -intervals,  $m_0=10$ ; number of  $y$ -intervals,  $n_0=10$ )

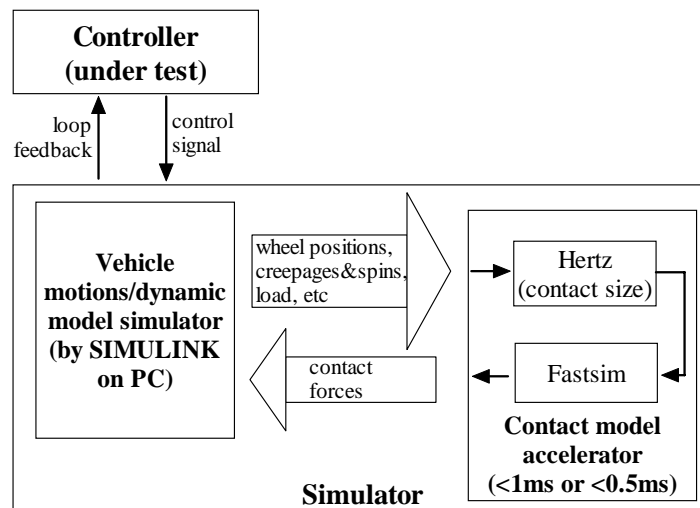


Figure 3. Hardware-in-the-loop simulation and custom contact model accelerator

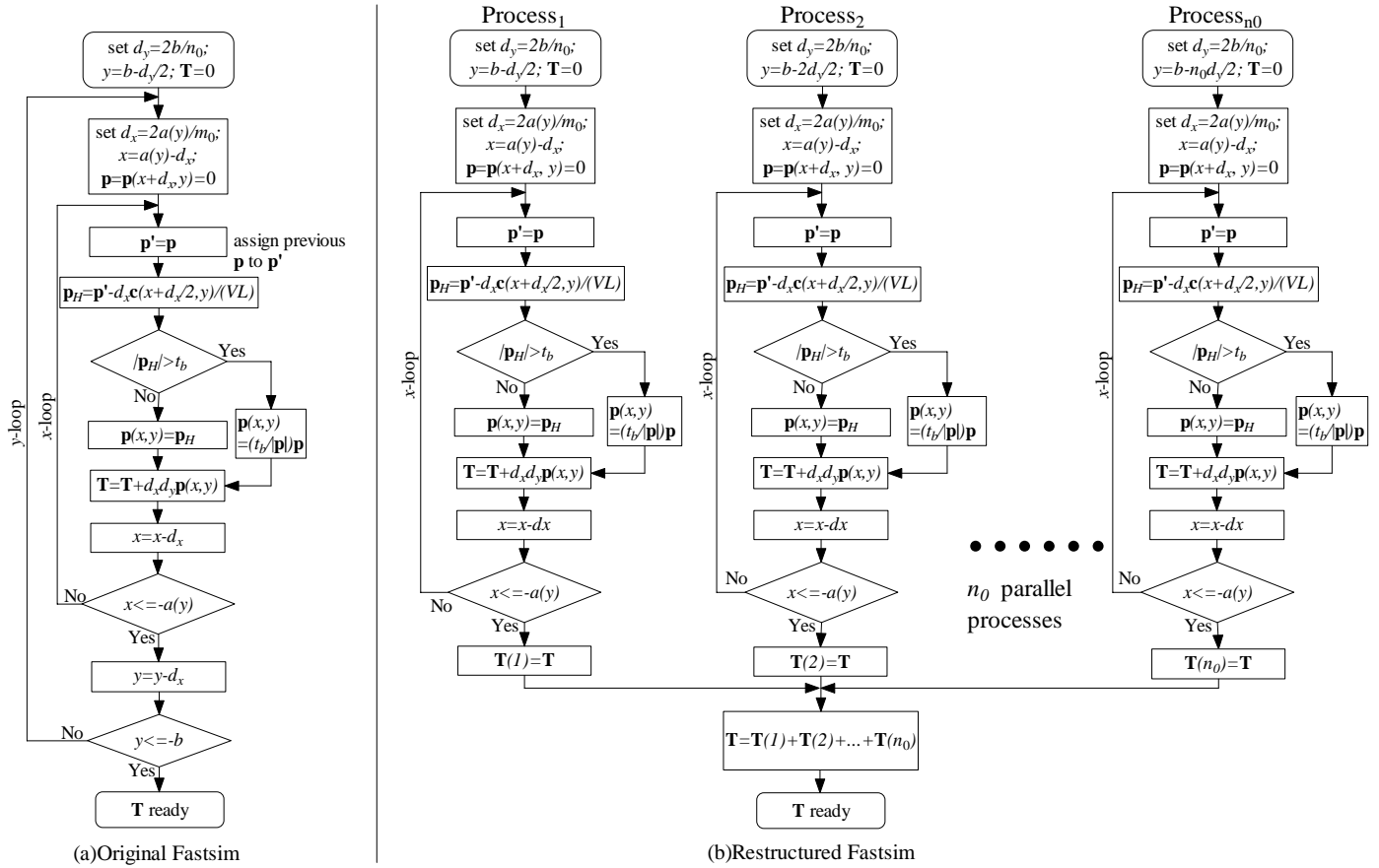


Figure 4. Algorithm flow of (a) original Fastsim and (b) restructured Fastsim

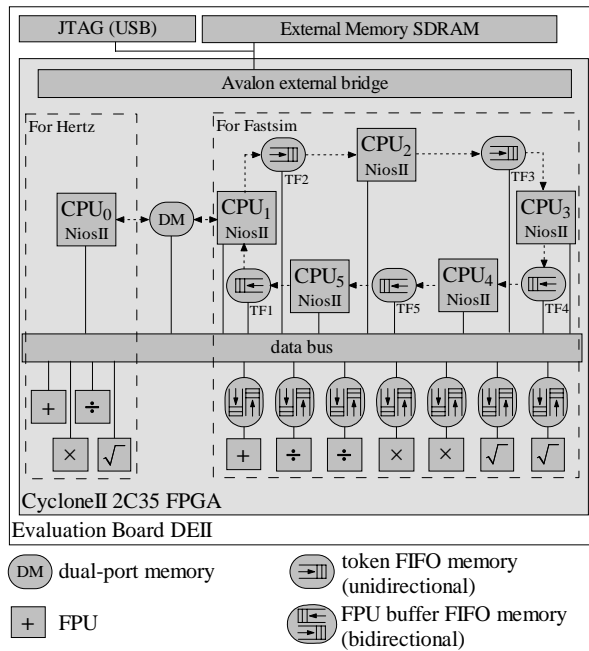


Figure 5. FPGA System-level diagram

main routine in CPU<sub>0</sub>

```

main() {
...
REPEAT 2 times
-----
  READ inputs R1,R2,R3,gammal,gamm2,gamma3,N,... FROM
  external

  //compute contact semi-axes a and b by Hertz
  WRITE v1,R1 TO FPU mul
  ...
  OBTAIN b FROM FPU add

  WRITE Hertz results a,b and other parameters gammal,
  gamma2,... TO DM memory (dual-port ) S1
-----

REPEAT 2 times
  WHILE data invalid on DM memory
  WAIT
-----
  READ forces data T FROM DM memory,and accumulate to
  achieve the final result T S3
-----
...
}

```

main routine in CPU<sub>1</sub>-CPU<sub>5</sub>

```

main() {
...
REPEAT 2 times
  WHILE data invalid on DM memory
  WAIT
-----
  READ data a,b,gammal,... from CPU0 via DM memory

  //compute T_I in 1st y-loop of restructured Fastsim
  CALL loop_initialization_I()
  REPEAT 10 times
  WRITE v1,a TO FPU mul1
  ...
  WRITE old T_I,acc TO FPU add1
  OBTAIN new T_I FROM FPU add1

  //compute T_II in 2nd y-loop of restructured Fastsim
  CALL loop_initialization_II()
  REPEAT 10 times
  WRITE v1, a TO FPU mul1
  ...
  WRITE old T_II,acc TO FPU add1
  OBTAIN new T_II FROM FPU add1

  //sum up T_I and T_II and write result to DM memory
  WRITE T_I,T_II TO FPU add1
  OBTAIN result T FROM FPU add1
  WRITE result T to DM memory S2
-----
...
}

```

Figure 6. Main routines in the processors

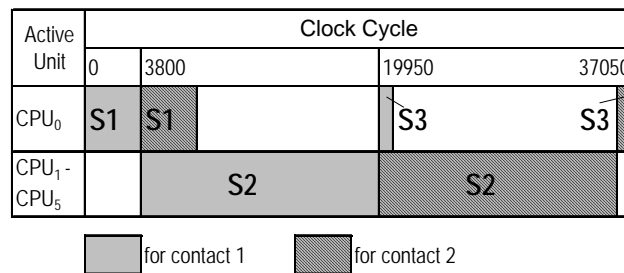


Figure 7. FPGA overall execution

### subroutines CPU<sub>1</sub>-CPU<sub>5</sub>

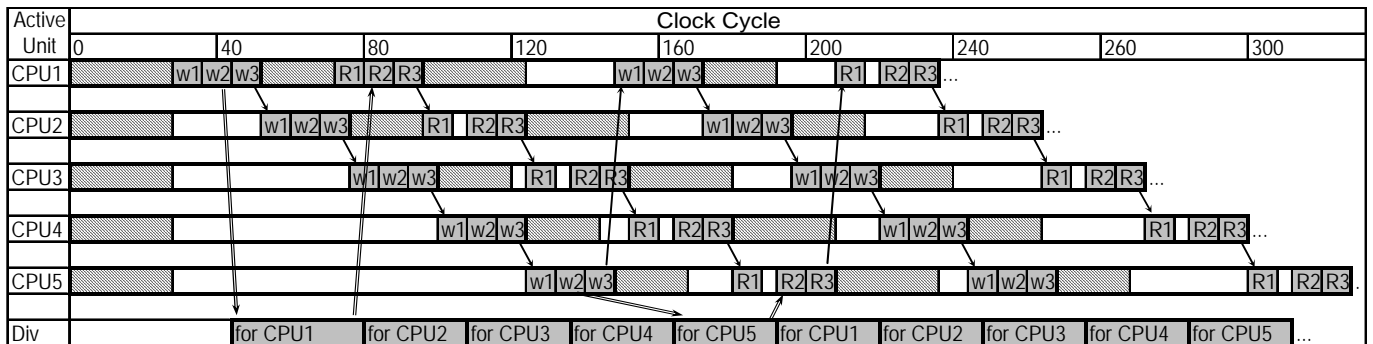
```

//definition of FPUWR()
void FPUWR(int FPU_in,int op1,int op2,FIFO_offset) {
    //read token from FIFO
    WHILE token invalid in TF1 memory //on CPU1, use TF1
    WAIT //on CPU2, use TF2
    READ token from TF1 //...etc
    //send operands to FPU
    WRITE op1,op2 TO FPU
    //write token to FIFO
    WRITE token TO TF2 memory //on CPU1, use TF2
    //on CPU2, use TF3
    //...etc
}

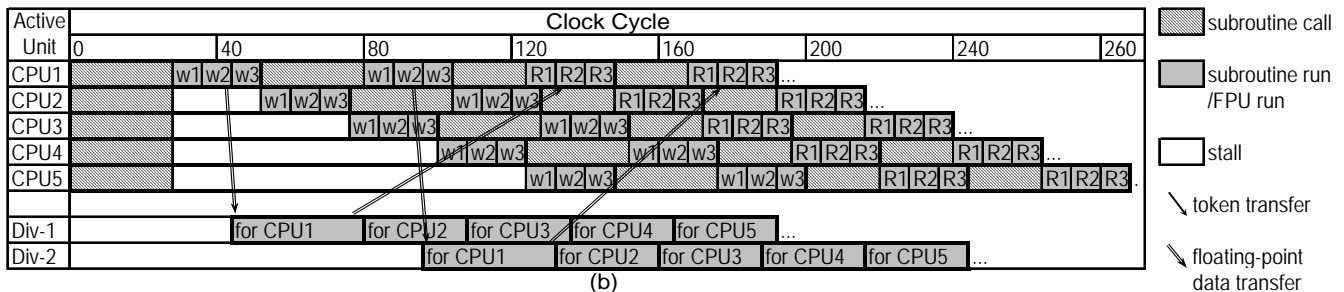
//definition of FPU RD()
int FPU RD(int FPU_out) {
    //read token from FIFO
    WHILE token invalid in TF1 memory //on CPU1, use TF1
    WAIT //on CPU2, use TF2
    READ token from TF1 //...etc
    //get result from FPU
    OBTAIN res FROM FPU
    //write token to FIFO
    WRITE token TO TF2 memory //on CPU1, use TF2
    //on CPU2, use TF3
    //...etc
    RETURN res;
}

```

Figure 8. Subroutines in the processors CPU<sub>1</sub>-CPU<sub>5</sub>



(a)



(b)

- subroutine call
- subroutine run /FPU run
- stall
- token transfer
- floating-point data transfer

Figure 9. (a) Execution of two divide operations. (b) Execution of two divide operations with more FPUs and optimized

scheduled

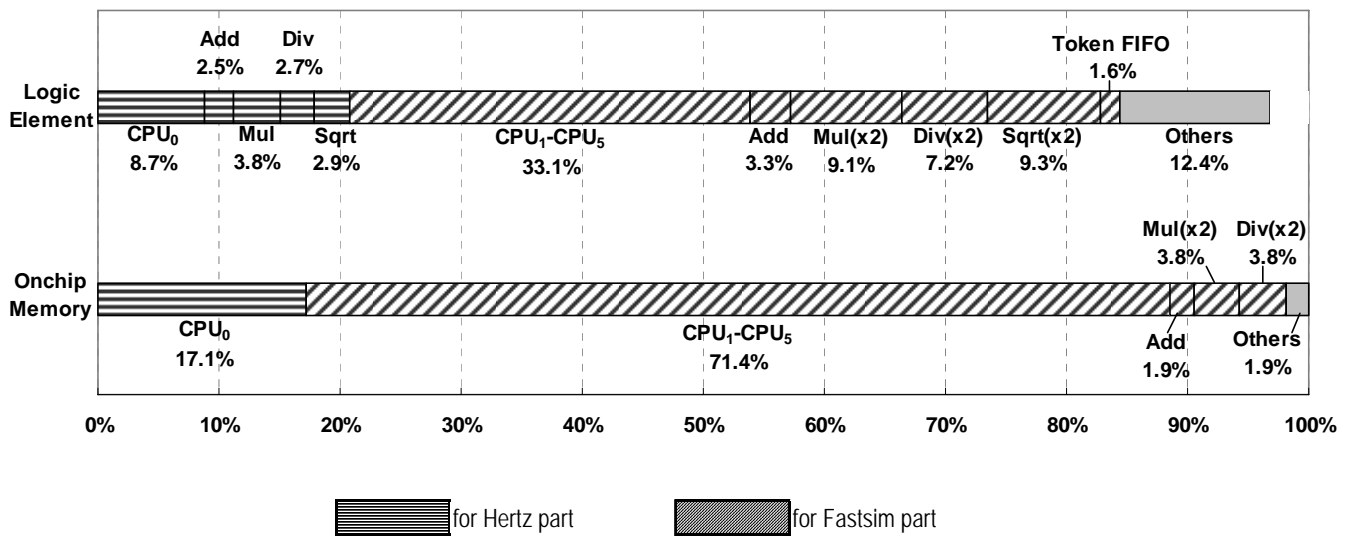
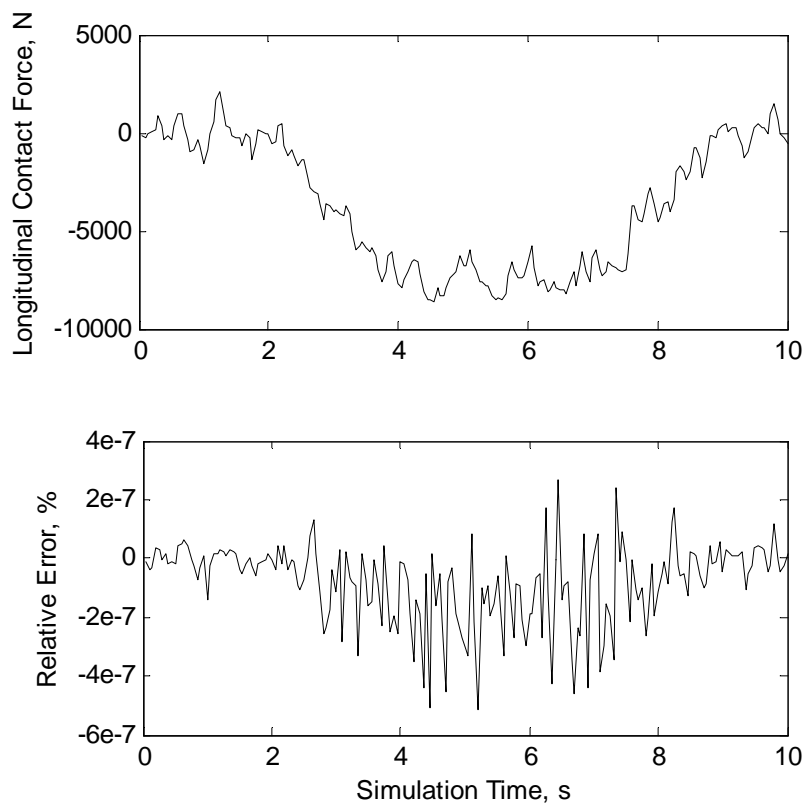
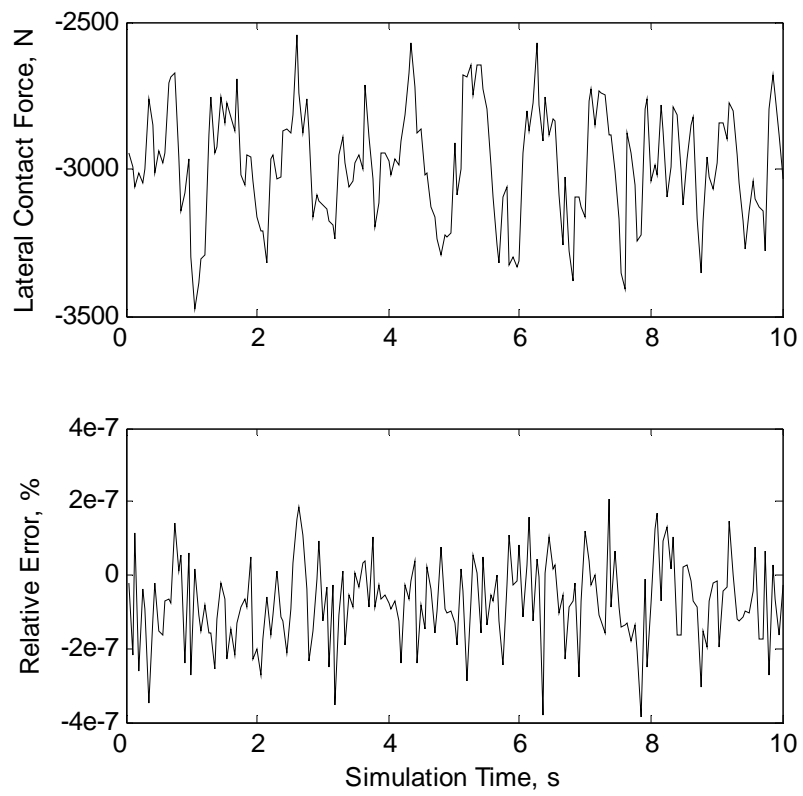


Figure 10. FPGA device utilization



(a)



(b)

Figure 11. FPGA calculation results and errors of (a) longitudinal contact force and (b) lateral contact force