

REAL-TIME MODELING OF WHEEL-RAIL CONTACT LAWS WITH SYSTEM-ON-CHIP

Y. Zhou, T.X. Mei, and S. Freear

Abstract—This paper presents the development and implementation of a multiprocessor system-on-chip solution for fast and real time simulations of complex and nonlinear wheel-rail contact mechanics. There are two main significances in this paper. Firstly, the wheel-rail contact laws (including Hertz and Fastsim algorithms), which are widely used in the study of railway vehicle dynamics, are restructured for improved suitability that can take advantage of the rapid developing multiprocessor technology. Secondly, the complex algorithms for the contact laws are successfully implemented on a medium-sized FPGA (Field-Programmable-Gate-Array) device using six NiosII processors, where the executions of the Hertz and Fastsim parts are pipelined to achieve further enhancement in doing multiple contacts and the operation scheduling is optimized. In the Fastsim part the floating point units with buffering mechanism are efficiently shared by five processors connected in a token ring topology. The FPGA design shows good flexibility in utilizing logic element and on-chip memory resource on the device and scalability for a significant speed-up on a larger device in future work.

Index Terms—Multiprocessor system, Token rings, Sequencing and scheduling, Real-time and embedded systems

1 INTRODUCTION

Computer simulation has long been established as an essential method in both research and development of railway vehicles, and there are many commercially available software packages such as Adams Rail, Simpact and Vampire [1] for offline applications. The demand for computer power is normally very high to manage the computation time of the complex algorithms involved, and a bottleneck is the computation of the wheel-rail contact laws. Furthermore, the requirement of real-time simulation is becoming increasingly significant as it offers a realistic and cheaper alternative to carry out experimental studies of vehicle dynamics and active control technologies through the use of hardware-in-the-loop, without the need of real track experiments which are both economically and logistically difficult. In a simulation of vehicle dynamics of a two-axle railway vehicle where the contact patches of the 4 wheels with the rail need to be dealt with, the contact laws take 71% of the whole dynamic system computation or 21.5ms per step for calculating the forces at the 4 contact patches - running in Simulink on an Intel Pentium4 3.0GHz 1GB-memory PC platform [22]. Given that simulation step size for rail vehicle dynamics is typically less than 2ms, a much enhanced performance of the contact laws computation, at least less than 2ms latency per step, is clearly needed so that the models can be simulated in or better than real-time.

For modern embedded systems in the realm of high

throughput signal processing, the emerging multiprocessor based platforms are increasingly becoming popular especially in the applications where the performance requirements can no longer be supported by embedded system architectures based on a single processor [2]. Nevertheless, it is found that the required performance is difficult to achieve by simply upgrading the PC platform portfolio, for example, an Intel Core2 Duo P7350 2.0GHz (dual-core) 3GB-memory one only gives a 39% enhancement at 13.1ms per step. On the other hand, the latest embedded system development with multiprocessor technology in the area of FPGA, DSP, etc., makes it feasible to design custom accelerators for complex and computationally demanding simulation models. For example an MPEG-4 video encoder is implemented on an Altera Stratix 1S40 FPGA device with four synthesized processors with fair scalability [8]. A molecular dynamics simulation which requires floating point arithmetic is accelerated with FPGA-equipped reconfigurable computers [4]. A complete road traffic simulation application is implemented on an FPGA [3].

In this paper, a restructured version of the contact laws with good parallelization efficiency is presented. Based on it, an FPGA multiprocessor design is discussed and successfully implemented on a medium-sized FPGA device. The FPGA design utilizes a hardware/software approach in which the most computationally intensive floating point tasks of the contact laws are executed on hardware FPU (Floating Point Unit), whereas the remaining tasks are executed in software in general processors. This Hardware/software approach is used in [3][4] and discussed in [5][20][21]. Software is responsible to distribute floating point tasks to hardware, which requires a good schedule of distribution tasks to achieve a minimized completion time. A hybrid task scheduling approach is developed and a heuristic task scheduling algorithm

- Y. Zhou is with the School of Electronic and Electrical Engineering, the University of Leeds, LS2 9JT, UK. E-mail: eenyz@leeds.ac.uk.
- T.X. Mei is with the School of Computing, Science and Engineering, Salford University, Salford, M5 4WT, UK. E-mail: t.x.mei@salford.ac.uk
- S. Freear is with the School of Electronic and Electrical Engineering, the University of Leeds, LS2 9JT, UK. E-mail: s.freear@leeds.ac.uk.

Manuscript received November 18, 2008; revised May 20, 2009; accepted June 15, 2009.

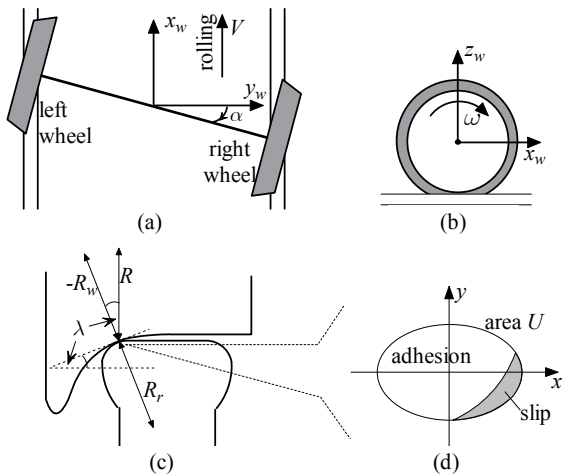


Fig. 1. The wheelset standing on the rails. (a) plan view (b) side view of the right wheel (c) profile of the right wheel (d) contact of right wheel

HLFET for parallel processing [6][7] is used. The computer network token-ring scheme [17] is introduced and adapted to allow the sharing of FPUs between processors in the design.

The paper is organized as follows. An introduction is given in section 1. Section 2 presents the wheel-rail contact laws and a restructured version for FPGA implementation. The FPGA development based on the revised contact laws is discussed in section 3. Section 4 presents the implementation and tests results of the design. Finally, a conclusion is given and further work discussed in section 5.

2 CONTACT LAWS INVESTIGATION

2.1 Original Method

A rail wheelset consists of two wheels mounted on an axle. Fig. 1(a)(b) schematically show the wheelset rolling on the rails at a velocity V and introduce a coordinate system in which the origin lies in the centre of the wheelset, the z_w -axis points vertically upwards, the x_w -axis points along the rails in the rolling direction, and the y_w -axis points to the right if one is facing the rolling direction. α is the small angle between the y_w -axis and the centre line of the wheelset. The profiles of the right wheel and rail and their contacting position are shown in Fig. 1(c) with the angle λ between the contact normal line and the vertical line, the rail curvature radii R_r , the wheel curvature radii R_w , and the wheel radii R which equals to R_0 when the wheelset lateral displacement y_w is zero. The relative velocity of a wheel with respect to the rail divided by V is known as 'creepage' and normally exits in three directions - longitudinal, lateral and spin. The creepages on the right wheel are defined in (1)-(4), and a similar set of equations can be given for the left wheel.

$$\text{creepage: } \boldsymbol{\gamma} = (\gamma_{xw}, \gamma_{yw}, \gamma_s) \quad (1)$$

where

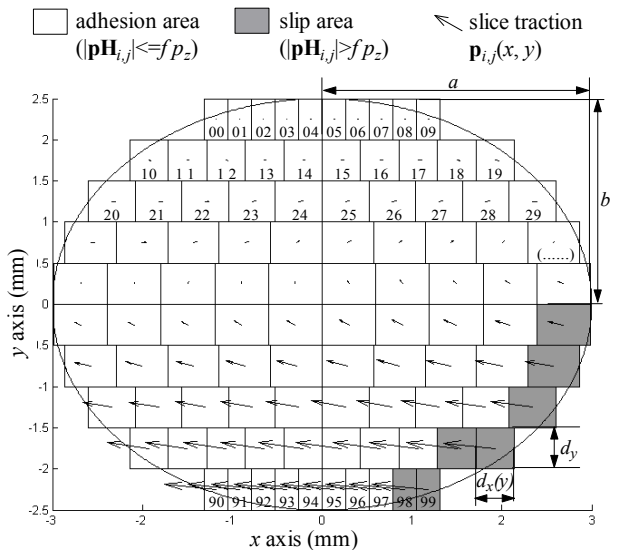


Fig. 2. Contact area and Fastsim calculation (contact semi-axes a and b have been achieved by Hertz; $m_\sigma=10$, $n_\sigma=10$; segments are indexed by ij ($i=0, 1, \dots, m_\sigma-1$; $j=0, 1, \dots, n_\sigma-1$))

$$\text{lateral creepage: } \gamma_{yw} = \dot{y}_w / V - \alpha \quad (2)$$

$$\text{longitudinal creepage: } \gamma_{xw} = (R_0 - R) / R_0 - L_g \cdot \dot{\alpha} / V \quad (3)$$

$$\text{spin creepage: } \gamma_s = \dot{\alpha} / V - \lambda / R_0 \quad (4)$$

A non-zero contact area for each of the wheel-rail contact patches is formed by the normal load force N acting on the two elastic bodies, as shown in Fig. 1(d). An x - y coordinate system is introduced for the definition of the contact area. According to the Hertz theory [9] which assumes that the pressure distribution over the contact area is semi-elliptical, the area is shaped as an ellipse with semi-axes a and b , as determined in (5)~(6).

$$a = m_c \cdot (3 \cdot N \cdot (1 - \nu^2) / (E \cdot (1/R - 1/R_w + 1/R_r)))^{1/3} \quad (5)$$

$$b = n_c \cdot (3 \cdot N \cdot (1 - \nu^2) / (E \cdot (1/R - 1/R_w + 1/R_r)))^{1/3} \quad (6)$$

in which m_c and n_c are tabulated non-dimensional coefficients along with a ratio variable r_{mn} as in (7)

$$r_{mn} = (1/R + 1/R_w - 1/R_r) / (1/R - 1/R_w + 1/R_r) \quad (7)$$

ν is Poisson's coefficient and E is Young's modulus, all connected with the contacting materials. A contact ellipse U is determined from the values of a and b .

From the contact ellipse, Fastsim algorithms [9] are often used to calculate the contact tangential traction forces developed as a result of the existence of the normal load and creepages. The elliptical contact area U is divided evenly into n_0 rows along the y -axis, and each row is then divided evenly into m_0 segments along the x -axis, totaling $N_U = m_0 \times n_0$ segments. Fig. 2 shows an example of a divided contact area in which both m_0 and n_0 are set to 10.

Each segment's area is denoted as $\Delta S_{i,j}$ and the center point tractions as $\mathbf{p}_{i,j}$ ($i=0, 1, \dots, m_0-1$; $j=0, 1, \dots, n_0-1$) which is a vector with two components in x and y directions. The leading edge segment (facing the rolling direc-

tion) of each row has j set at 0. The tangential force at each segment is $\mathbf{p}_{i,j} \Delta S_{i,j}$. Fastsim algorithms use a double numerical integration approach to achieve an approximation of the total tangential force \mathbf{T} as in (8).

$$\mathbf{T} = \sum_{i=0}^{m_0} \sum_{j=0}^{n_0} \mathbf{p}_{i,j} \Delta S_{i,j} \quad (8)$$

The area $\Delta S_{i,j}$ of each segment is dictated by that particular row's center point position on the y -axis, as in (9)-(11)

$$\Delta S_{i,j}(y) = d_x(y) \cdot d_y \quad (9)$$

where

$$d_y = 2 \cdot b / n_0 \quad (10)$$

$$d_x(y) = 2 \cdot a(y) / m_0 = 2 \cdot a \sqrt{1 - (y/b)^2} / m_0 \quad (11)$$

In each segment, tractions at different positions are considered constant and the same as the center point traction $\mathbf{p}_{i,j}(x, y)$ which is determined by (12)-(18). According to Coulomb's Law, the traction doesn't exceed a traction bond $f \cdot p_z(x, y)$ where f is an empirical coefficient of the contacting materials and $p_z(x, y)$ is the normal pressure at that point. $\mathbf{pH}_{i,j}(x, y)$ is a pre-calculated traction which has an initialized value for the leading edge segments of each row and is related to previously known segment traction $\mathbf{p}_{i,j-1}$ for other segments of the row. $\mathbf{p}_{i,j}(x, y)$ at each segment is assigned with the value of $\mathbf{pH}_{i,j}(x, y)$ if it is within (or equal to) the traction bound. $\mathbf{c}(x, y)$ is the rigid slip. L is the flexibility parameter. In (15), L 's feature is approximated by (L_{xw}, L_{yw}, L_s) which correspond to the three directions of creepage and are related to the contact size, the combined modulus of rigidity G and the creepage coefficients (C_{xw}, C_{yw}, C_s) which are typically tabulated along with the ratio of a to b .

$$\mathbf{p}_{i,j}(x, y) = \begin{cases} \mathbf{pH}_{i,j}(x, y), & \text{if } |\mathbf{pH}_{i,j}(x, y)| \leq f \cdot p_z(x, y) \text{ (adhesion)} \\ f \cdot p_z(x, y) \cdot (\mathbf{pH}_{i,j} / |\mathbf{pH}_{i,j}|), & \text{if } |\mathbf{pH}_{i,j}(x, y)| > f \cdot p_z(x, y) \text{ (slip)} \end{cases} \quad (12)$$

where

$$p_z(x, y) = 3 \cdot N \cdot \sqrt{1 - x^2/a^2 - y^2/b^2} / (2 \cdot \pi \cdot a \cdot b) \quad (13)$$

$$\mathbf{pH}_{i,j}(x, y) = \begin{cases} d_x(y) \cdot \frac{\mathbf{c}(x + d_x(y)/2, y)}{V \cdot L} & (j = 0, \text{ leading edge slice of each row}) \\ \mathbf{p}_{i,j-1} - d_x(y) \cdot \frac{\mathbf{c}(x + d_x(y)/2, y)}{V \cdot L} & (j \neq 0, \text{ other slices of each row}) \end{cases} \quad (14)$$

where the two components in x and y directions of $\mathbf{c}(x, y)/(V \cdot L)$ are approximated by

$$\frac{\mathbf{c}(x, y)}{V \cdot L} = \begin{cases} \gamma_{xw} / L_{xw} - \gamma_s \cdot y / L_s & (x \text{ direction component}) \\ \gamma_{yw} / L_{yw} + \gamma_s \cdot x / L_s & (y \text{ direction component}) \end{cases} \quad (15)$$

where

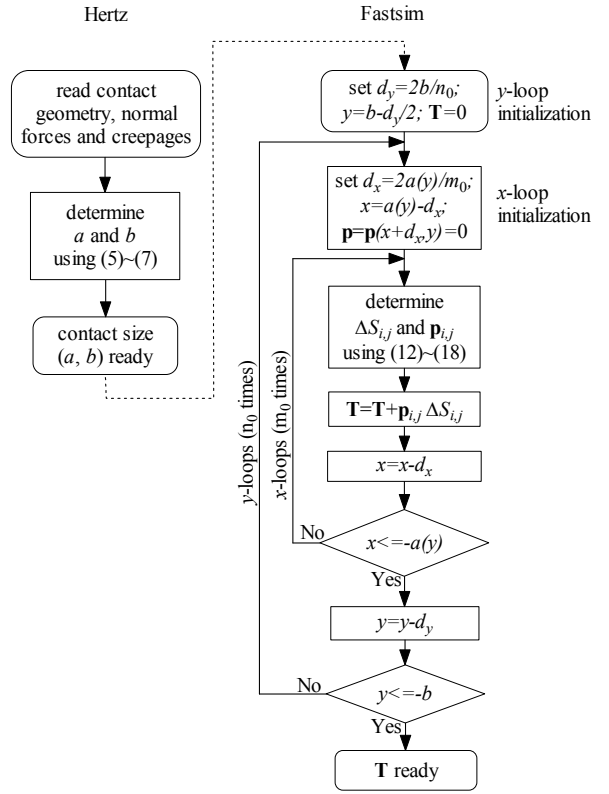


Fig. 3. Original contact laws flow chart.

$$L_{xw} = 8 \cdot a / (3 \cdot G \cdot C_{xw}) \quad (16)$$

$$L_{yw} = 8 \cdot a / (3 \cdot G \cdot C_{yw}) \quad (17)$$

$$L_s = \pi \cdot a^2 / (4 \cdot G \cdot \sqrt{ab} \cdot C_s) \quad (18)$$

It is obvious that the greater m_0 and n_0 in (8) then the calculated creep forces would more closely approximate the continuous form of the contact laws as in (19)

$$\mathbf{T}_c = \int_{-b}^b \int_{-a(y)}^{a(y)} \mathbf{p}(x, y) dx dy \quad (19)$$

where

$$a(y) = a \sqrt{1 - (y/b)^2} \quad (20)$$

However, in normal practice where the computation resources are limited, m_0 and n_0 need to be set to reasonable values so that the approximation with acceptable accuracy is achievable within a restricted calculation time. In most cases a grid size setting of $m_0=10$ and $n_0=10$ in (8) is sufficient and is used in the design. When only a single processor is available, the traction forces of all the segments will have to be determined sequentially and integrated one by one as numbered in Fig. 2, 00, 01, 02, ..., 99.

The data flow of the discussed contact laws is summarized in Fig. 3. The computationally intensive portion is found at the Fastsim part where there is an inner x -loop and an outer y -loop. x -loop will execute m_0 times to tackle all m_0 segments in one row, while y -loop will run for n_0 times to tackle all n_0 rows.

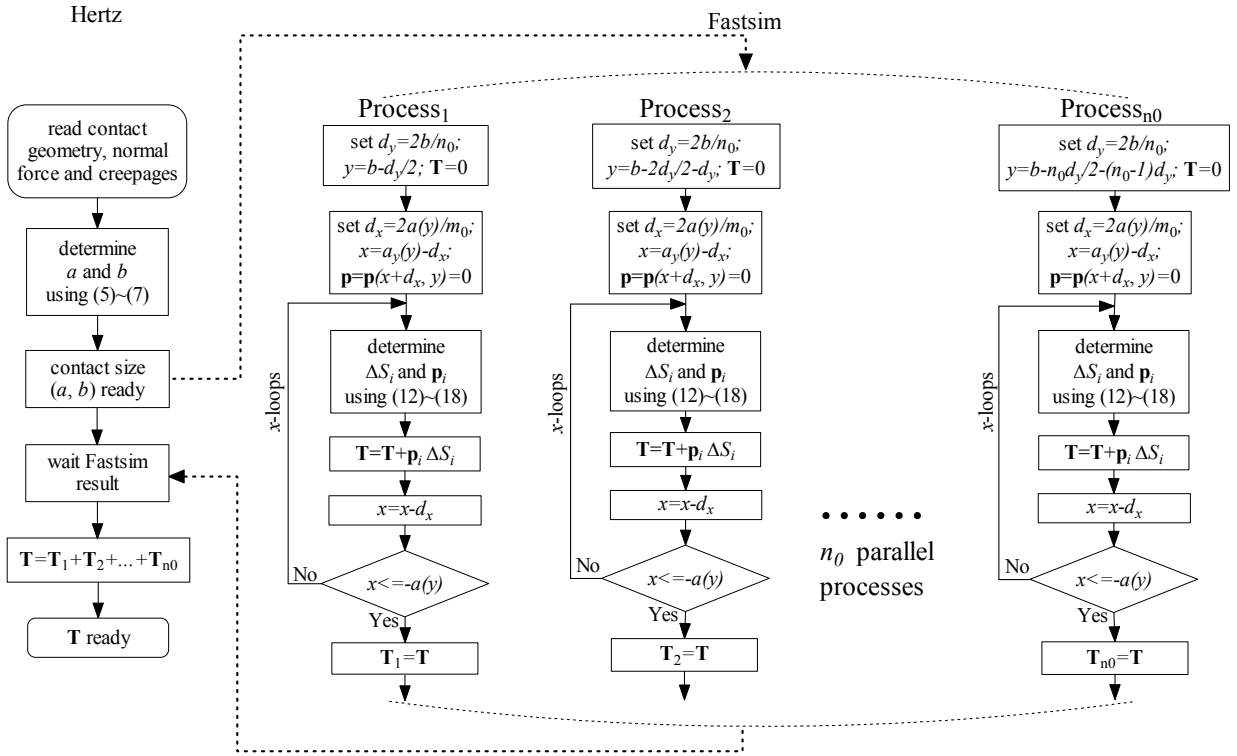


Fig. 4. Restructured contact laws flow chart

2.2 Revised Method

As in the original method, in one particular row the pre-calculated tangential traction $\mathbf{pH}_{i,j}$ for the current segment is associated with the previous segment's traction $\mathbf{p}_{i,j-1}$, such "one by one" sequential process inside the x -loops is inevitable. However, computations for all rows sequentially are found to be dispensable for any multi-processor solution. Because all n_0 rows' computations as well as the initiations of x -loop are independent of any other rows, calculation results for one particular row can be worked out without knowing the other rows' results. To make use of the parallel execution capability of multi-processor technology, it is proposed to split the sequential y -loop in the original algorithm into n_0 parallel processes, as shown in Fig. 3.

In the restructured contact laws, the x -loop operation in the Fastsim part remains the same to update tangential forces $\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_{n_0}$ for each row in all n_0 parallel processes which will then be summed in the Hertz part to achieve a total contact force \mathbf{T} at the final operation.

The speed enhancement can be theoretically estimated using (21)~(22), based on the assumption that the communication between Hertz and Fastsim parts takes a neglectable time and the runtime of the Hertz part is expected not to change significantly although a few more summing calculations are added. The Hertz consumption percentage $H\%$ and the Fastsim consumption percentage $F\%$ are found to be 3% and 97% respectively. Ideally all of the $n_0=10$ parallel processes can be executed simultaneously by $n_p=10$ processors, which would give a 7.9 times speedup than that of the original contact laws. If with $n_p=5$ processors, the 10 processes can still be executed

(repeated twice) to give a 4.5 times speedup. Even with only $n_p=2$ processors, a 1.9 times speedup can be expected.

$$\text{speed enhancement} = \frac{100\%}{F\%/n_p + H\%} - 1 \quad (21)$$

where

$$\text{mod}(n_0, n_p) = 0 \quad (22)$$

3 FPGA DEVELOPMENT

A medium-sized Altera's CycloneII FPGA chipset EP2C35 is targeted and NiosII processors are used in the design.

3.1 Contact Area (Hertz Part) Design

The Hertz algorithms are used for contact area calculations in the design.

3.1.1 Architecture

In this part, a soft-core processor NiosII/f (fast) [10] is used. NiosII is a 32-bit CPU with full 32-bit instruction set, data path and address space. NiosII/f is its fastest version which employs a 6-stage pipeline.

Fig. 5 depicts the configuration of the processor used for the Hertz implementation. The processor has separate instruction and data caches implemented by on-chip RAM with low-latency access. Their sizes are set at 4KB and 1KB respectively, which allows the whole repetitive part of the program to reside at the instruction cache and all the frequently used variables to be allocated at the data cache in order to reduce access to high-latency off-chip memory and to accelerate the program execution. An

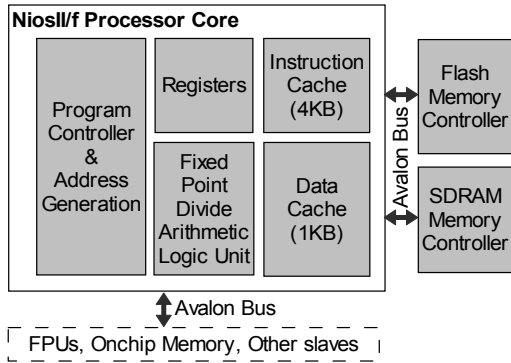


Fig. 5. Processor configuration in the Hertz part.

arithmetic logic unit is implemented for the fixed-point divide operation as this is required in a small portion in the calculation of the Hertz algorithms. The processor commands its slaves or peripherals via the Avalon bus [11].

Most of the operations/computations in the Hertz and Fastsim algorithms are found to be floating point operations, the quantities of which are shown in Table 1. Single precision operations have been found to provide sufficient accuracy for the application. The ways to implement single-precision floating point operations in an FPGA can be categorized as ‘software’ approach and ‘hardware’ approach. The former uses generic processors (eg. NiosII) to execute floating point operations and it is suitable for designs which only contains a small proportion of floating point tasks and have low speed requirements. In this design where floating point operations are intensive and high calculation speed is demanded a ‘hardware’ approach is used. A set of well developed open source FPUs (floating point units) are adapted [15] such that floating point operations can be carried out in parallel in different FPUs independently, which leads to a more efficient usage of logic resources. The reusability of IP (intellectual property) cores makes it more flexible to allocate more resources to a particular type of operation, e.g. to implement two or more divider FPUs for floating point divisions if required. Table 2 shows the properties of the IP

TABLE 1
NUMBERS OF FLOATING POINT OPERATIONS

	Add/Sub	Mul	Div	Sqrt
Hertz	31	26	13	1
Fastsim	1137	1088	251	211

TABLE 2
FPU IP CORE PROPERTY

IP Core	Pipeline Stages	Initial Latency (cycles)	Subsequent Latency (cycles)
Add	3	7	3
Mul	3	12	8
Div	3	35	31
Sqrt	3	33	29

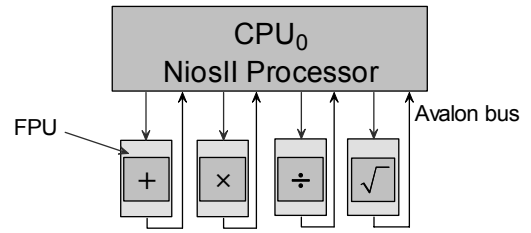


Fig. 6. FPGA architecture of the Hertz part.

cores used in the design. They are all 3-stage-pipelined and have higher throughput for subsequent operations.

In the Hertz part, 1 add/subtract, 1 multiply, 1 divide and 1 square-root FPUs are used. The determination of this amount of FPUs in the Hertz part is based on the consideration of its relatively low computation requirement compared to the Fastsim part (the second part of the contact laws) and the target device resource limits. The processor and the FPUs are connected in a way as shown in Fig. 6, so that the processor can write data to the FPUs and read results back.

3.1.2 Task Scheduling

The software running on the processor is responsible for managing the distribution of the floating point operations to the FPUs. An example is given to show the scheduling of the operations.

Fig. 7 shows an objective equation sampled from part of the Hertz algorithms written in C language, in which ‘xx1’, ‘xx2’, ‘xx3’ and ‘xx4’ are known variables and ‘yy’ is the variable needed to be calculated. The equation contains 4 individual floating point operation tasks (or FPU tasks), numbered as 1, 2, 3, and 4, which need to be assigned to the FPUs by the processor. Each FPU task corresponds to two processor tasks IOWR (operand-write) and IORD (result-read). And 1w, 1r, 2w, 2r, 3w, 3r, 4w and 4r are used to index the eight processor tasks respectively. The execution order is as follows. Processor task 1w (IOWR(xx1, xx2, div_in)), for instance, instructs the processor to prepare and write the operands ‘xx1’ and ‘xx2’ sequentially to the address ‘div_in’ which represent the input port of the divide FPU. As soon as the processor finishes task 1w, the divide FPU starts task 1r (IORD(&x_1, div_out)) which is started at some point after 1w, instructs the processor to prepare and read the result from the divide FPU output port address ‘div_out’ to variable ‘x_1’. This read operation cannot be done until the FPU

Objective Equation	FPU Task	ID	Processor Task	ID
$yy=(xx1/xx2)*sqrt(xx3)+xx4$	$x_1=xx1/xx2$	1	IOWR (xx1, xx2, div_in); IORD (&x_1, div_out);	1w 1r
	$x_2=sqrt(xx3)$	2	IOWR (xx3, n/a, sqrt_in); IORD (&x_2, sqrt_out);	2w 2r
	$x_3=x_1*x_2$	3	IOWR (x_1, x_2, mul_in); IORD (&x_3, mul_out);	3w 3r
	$yy=x_3+xx4$	4	IOWR (x_3, xx4, add_in); IORD (&yy, add_out);	4w 4r

Fig. 7. Objective equation, the correspondent FPU tasks and processor tasks.

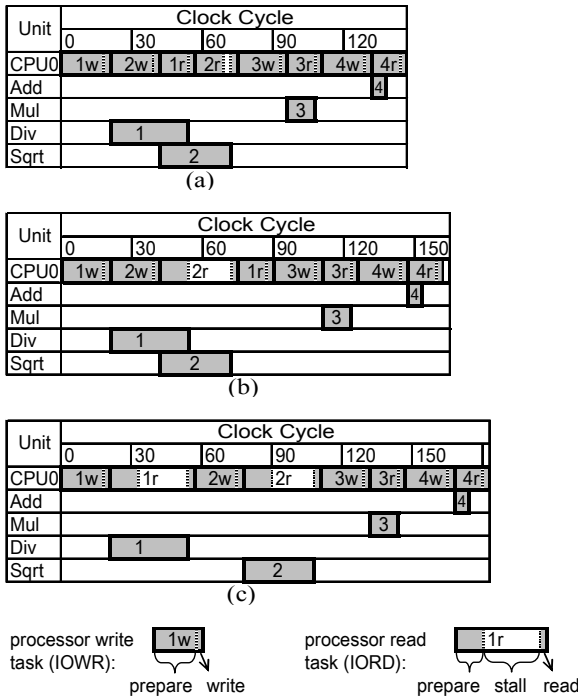


Fig. 8. (a) Optimal schedule (shortest run time); (b) Another schedule (longer run time); (c) Another schedule (even longer run time)

task has finished. A stall period will be added if the processor needs to wait for FPU results after the preparation. An IOWR operation approximately takes 21 cycles, and an IORD operation 15 cycles in the case without stall. Note that intermedia variables 'x_1', 'x_2' and 'x_3' are introduced.

The goal of scheduling is to find a running order of the processor tasks such that the whole completion time is minimized such that the dependence constraints are preserved. The known optimal schedule of this example is 1w, 2w, 1r, 2r, 3w, 3r, 4w and 4r, whose execution, derived from Modelsim simulation results, is depicted by the scheduling chart in Fig. 8(a). Solid lines are used to separate between two consecutive tasks, e.g. tasks 1w and 1r, indicating the finish of former task and the start of the later. During each processor task, the preparation and the FPU accessing (write/read) is indicated by a dashed line. Comparison with two other schedules with longer complete times is shown in Fig. 8(b)&(c). The optimal solution effectively eliminates unnecessary processor stalls by scheduling other tasks into the time slots in which the divide/square-root FPU results are pending. Hence, an appropriate approach needs to be developed to achieve this optimal schedule.

Weighted directed acyclic graphs (DAG) are used in the task scheduling, as shown in Fig. 9. Nodes are used to denote the tasks. The weight on a node is the computation cost of the corresponding task while the graph also uses directed edges to represent the dependencies among the tasks, e.g. $\boxed{1} \rightarrow \boxed{3}$ implies that node 3 is a child which cannot start until its parent node 1 finishes. The weight on an edge is the communication cost of two nodes. For the target system, the communication costs are negligible. HLFET (Highest Level First with Estimated

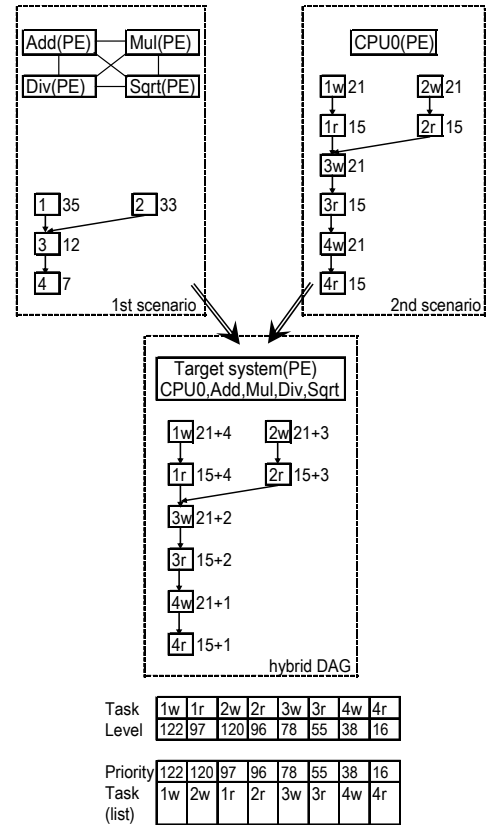


Fig. 9. Hybrid DAG representation and its list scheduling by HLFET

Times) [6] is used as it is one of the simplest DAG scheduling algorithms and it has been shown to produce good quality results when the communication costs of the edges are ignored [7]. In HLFET, the length of a directed path is defined to be the sum of the weights of all nodes along the path including the initial and final nodes. The level of an exit node is the weight of itself and the level of a non-exit node is defined to be the length of the longest path from that node to the exit node. The HLFET algorithm calculates each node's level which is the same as the node's priority, according to which the algorithm then schedules the nodes on the list one by one.

A hybrid DAG representation is developed for the purpose of building a clear and effective corresponding relationship between the tasks for scheduling and the nodes. As shown in Fig. 9, this hybrid approach incorporates two scenarios and overcomes their drawbacks. The first scenario assumes that the processor ran fast enough, viz. the processor IOWR/IORD task times were neglectable compared with the FPU task times, so the FPUs can be regarded as 4 individual processing elements (PEs) in full-connection and each node in a DAG represents one FPU task (instead of processor tasks) with its latency as the node's weight. The second scenario assumes that the FPUs ran fast enough, viz. the FPU task times were neglectable compared with the processor IOWR/IORD task times, so only one PE is considered and each node in a DAG represents one processor IOWR/IORD task with its latency as the node's weight. Both of these two scenarios fail to take into account the interference between the FPU

tasks and the processors tasks which is proved to be vital in producing an optimal schedule. In the hybrid DAG approach, one PE is considered in the target system and a DAG is achieved basing on the second scenario. The difference is that the weight on a node t'_i is not only determined by the IOWR/IORD task latency t_i but also supplemented by a compensator tc_j reflecting the latency of the corresponding FPU task

$$t'_i = t_i + tc_j, \quad \text{where } tc_j = [tc_{add}, tc_{mul}, tc_{div}, tc_{sqr}] \geq 1$$

The basic context is that: a longer FPU task should be started earlier so the corresponding processor write task (IOWR) should be given higher priorities; and to avoid the corresponding processor read task (IORD) to be started immediately after by allocating other irrelevant processor tasks instead, the maximum compensator should be smaller than the difference of the IOWR and IORD task latencies

$$\max[tc_{add}, tc_{mul}, tc_{div}, tc_{sqr}] < t_w - t_r = 21 - 15 = 6$$

Based on these constraints, the compensators tc_{add} , tc_{mul} , tc_{sqr} and tc_{div} are set to values 1, 2, 3, 4 respectively according to the order in ranking the corresponding task latencies of different type of FPUs from the shortest to the longest. Hence the node weights are

$$\begin{aligned} t'_{1w} &= t_{1w} + t_{div} = 21 + 4, & t'_{1r} &= t_{1r} + t_{div} = 15 + 4 \\ t'_{2w} &= t_{2w} + t_{sqr} = 21 + 3, & t'_{2r} &= t_{2r} + t_{sqr} = 15 + 3 \\ t'_{3w} &= t_{3w} + t_{mul} = 21 + 2, & t'_{3r} &= t_{3r} + t_{mul} = 15 + 2 \\ t'_{4w} &= t_{4w} + t_{add} = 21 + 1, & t'_{4r} &= t_{4r} + t_{add} = 15 + 1 \end{aligned}$$

The weights are decided in such a tentative way with the hope to combine both the processor task latencies and the FPU task latencies into leveling consideration. The HLFET algorithm is employed to produce a leveled task list, as shown in Fig. 9. It shows that all tasks are assigned unique priorities and the determined schedule 1w, 2w, 1r, 2r, 3w, 3r, 4w and 4r agrees with the known optimal one. Hence, the dependence constraints are maintained and meanwhile a minimized completion time is achieved by this approach.

The discussed hybrid approach is then tested on two other equations from the Hertz algorithms, one with eight processor tasks and one with ten processor tasks, and the optimal schedules can be achieved in both cases. This approach is then applied to the whole Hertz algorithms. The implemented Hertz algorithms take about 3800 cycles to calculate the contact size data and about 950 cycles to perform the final summation for the total contact force.

3.2 Contact Force (Fastsim Part) Design

The revised Fastsim algorithm is used for contact force calculation in the design.

3.2.1 Architecture

NiosII/f processors are also used for the second part.

Fig. 10 shows the processor configuration.

Ideally, 10 parallel processes are required to perform the computation loops in parallel for the restructured contact laws with $m_0=10$ and $n_0=10$. However, because the resources in the targeted medium-sized FPGA device are limited and increased number of processors lead to increased complexity and overhead, only 5 NiosII/f processors are implemented for the Fastsim part, each responsible for dealing with 2 processes.

For floating point operations, a straightforward solution is to allocate sufficient FPUs exclusively to each individual processor. The advantage of this approach is that all of the 5 processors can run simultaneously without any interference and the shortest run time can be achieved. A practical application can be found in Sun's OpenSPARC T2 [16] architecture in which 8 SPARC processors with proprietary sets of floating point units are implemented. However, this type of approach is clearly inefficient in utilizing the resources especially for small- and medium-sized FPGA devices. As already described in section 3.1 of the Hertz part FPGA design which uses a processor with exclusive FPUs, the processor's operand distribution and result acquisition count up to a high percentage of the overall runtime cycle and even with an appropriate operation schedule the executions in each of the FPUs are scattered sparsely which reflects inefficient usage at the FPU circuit. But because the Hertz part and the Fastsim part are required to perform in a pipeline mode (to be discussed in section 3.3), each of these two parts needs a separate set of hardware resources. And because the arithmetic in the Hertz algorithms is much less intensive than that in the Fastsim algorithms (refer to Table 1), the Hertz part has a lower priority in the hardware resource allocating and hence using a processor with exclusive FPUs in the Hertz part is inevitable. An approach of utilizing FPUs more efficiently by sharing them in a multi-processors system is introduced for the Fastsim algorithms.

FPUs are shared when they are available for access by more than one processor. FPUs can be made shareable by simply connecting them to multiple NiosII processors' master ports in the connection matrix of SOPC Builder [12] and then an Avalon arbiter will be automatically built to route the signals. Nevertheless, in general, Altera's NiosII design environment does not natively support sharing non-memory peripherals between multiple processors

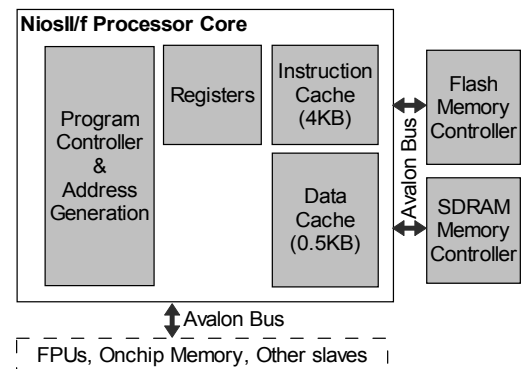


Fig. 10. Processor configuration in the Fastsim part.

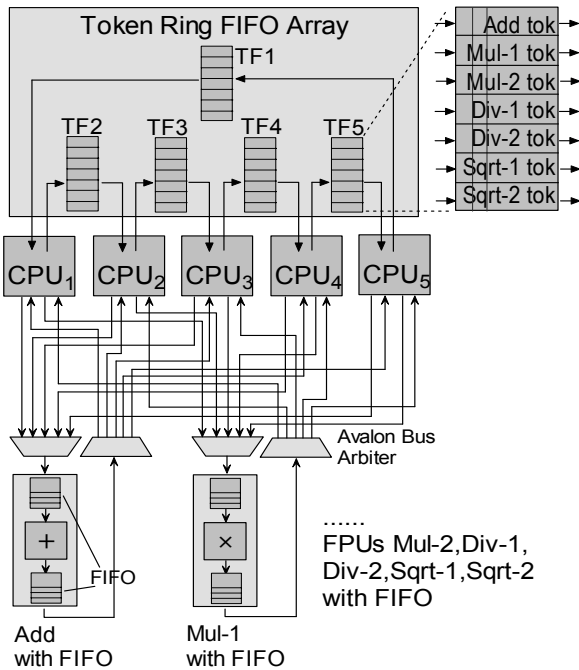


Fig. 11. FPGA architecture of the Fastsim part.

[13]. The difficulty is to determine which processor is supposed to access the shared FPUs preventing destructive accessing. It is conceivable that a handshaking mechanism could be created to handle these scenarios.

Good symmetry is found in the revised Fastsim between the parallel processes as most of their operations are identical although the operands are not. So, it is likely that all 5 processors request to access a particular shared FPU at the same time. The software running on each processor is responsible for coordinating access to shared resources with the system's other processors. The idea of token ring [17], a technology originally developed for local network system, is introduced into the design. The basic concept of token ring is that in a local area network (LAN) all computers are connected in a ring or star topology and a bit- or token-passing scheme is used in order to prevent the collision of data between two computers that want to send messages to others at the same time. This scheme is changed slightly in the design that each FPU has one token to be circulated through the ring topology of the processors which will be granted privilege to access that particular FPU in turn. This feature requires that each of the 5 processors has an identical task execution order/schedule. As the Avalon bus doesn't support direct connection between two masters (e.g. NiosII processors), FIFOs (First-In-First-Out) are used between adjacent processors to build the ring topology, denoted as TF1-TF5 in Fig. 11.

An FPU task buffering mechanism is used to synchronize tasks between the hardware and software. This introduces a minimal latency of 1 clock cycle for each transmitted operand or result. There are two FIFOs attached to each FPU, as shown in Fig. 11. One of them buffers write operands from different processors and dispenses next set of operands when the FPU core is ready. The other buffers calculation results and if results are not

Objective Equation	FPU Task	ID	Combined Processor Task	ID
Process a: $yya = (xx1a/xx2a) * \text{sqrt}(xx3a)$	$x_1a = xx1a/xx2a$	1a	IOWR (xx1a, xx2a, xx1b, xx2b, div_in);	1w
	$x_2a = \text{sqrt}(xx3a)$	2a	IORD (&x_1a, &x_1b, div_out);	1r
	$yya = x_1a * x_2a$	3a	IOWR (xx3a, n/a, xx3b, n/a, sqrt_in);	2w
Process b: $yyb = (xx1b/xx2b) * \text{sqrt}(xx3b)$	$x_1b = xx1b/xx2b$	1b	IORD (&x_2a, &x_2b, sqrt_out);	2r
	$x_2b = \text{sqrt}(xx3b)$	2b	IOWR (x_1a, x_2a, x_1b, x_2b, mul_in);	3w
	$yyb = x_1b * x_2b$	3b	IORD (&yya, &yyb, mul_out);	3r

Fig. 12. Objective equations, the correspondent FPU tasks, processor tasks and combined processor tasks.

available at the time when a processor requests to read, it halts the processor. With this buffering mechanism, the FPU could achieve a lower latency pipelined performance, Table 2. A reference design can be found in Sun's OpenSPARC T1 [18] architecture in which one set of floating point unit with built-in FIFO is shared by 8 SPARC processors.

In practice, the resources available on the targeted FPGA device limit the quantity of FPUs which can be implemented. In the final design, 1 add, 2 multiply, 2 divide and 2 square-root FPUs are implemented, taking into account a whole consideration of the floating point computation requirements in the Fastsim algorithms, the device logic resource usage of each type of FPU and the available resources.

3.2.2 Task Scheduling and Execution

As each of the 5 processors are responsible for 2 adjacent processes, a straightforward way is running them one by one in 2 times. Another tricky approach is introduced here in order to reduce run time. Good symmetry is found in the revised Fastsim among the parallel processes as mentioned before. This makes it possible to combine the 2 processes for one processor into one in which the number and order of the operations remain the same but the number of operands in each operations is doubled. And this approach can reduce run time by about 20%. A combined processor write (IOWR) task approximately takes 39 cycles, and a combined processor read (IORD) task 30 cycles if the processor doesn't stall.

Fig. 12 shows two objective equations sampled from two processes a and b in the Fastsim algorithms which are originally supposed to be run by a processor in 2 times. 'xx1a', 'xx2a', 'xx3a' and 'xx1b', 'xx2b', 'xx3b' are known variables while 'yya' and 'yyb' are the variables needed to be calculated. Each equation consists of 3 FPU tasks denoted as 1a, 2a, 3a and 1b, 2b, 3b. A set of combined processor tasks denoted by 1w, 1r, 2w, 2r, 3w, 3r can be formed, whose correspondences are depicted by solid arrowed lines in Fig. 12. A combined processor task incorporates the operands from both origins. For example, IOWR(xx1a, xx2a, xx1b, xx2b, div_in) instructs the processor to write operands xx1a and xx2a first, followed by operands xx1b and xx2b, to the divide FPU; IORD(&x_1a, &x_1b, div_out) instructs the processor to read two re-

sults from the divide FPU in order to variables x_{1a} and x_{1b} respectively; in the case that two or more duplicate FPUs of the same type are available, a combined IOWR can be programmed to write the sets of operands to different duplicate FPUs and a combined IORD can read results from different sources as well.

Now the combined processors tasks 1w, 1r, 2w, 2r, 3w and 3r need to be scheduled to achieve minimized completion time. It is conceivable that different amount of FPUs implemented would lead to different optimal schedule of the combined processor tasks. But for the purpose of simplicity, this factor will not be discussed in the paper. The scheduling approach assumes that each of the 5 processor owned one proprietary set of FPUs, viz. 1 add, 1 multiply, 1 divide and 1 square-root, although in practice a different number of FPUs are implemented and shared between the processors. Hence the hybrid scheduling method described in section 3.1.2 can be used here again to tackle the scheduling problem. This yields the optimal schedule as 1w, 2w, 1r, 2r, 3w and 3r which is identical for all the 5 processors.

The 5 processors CPU1-CPU5 are programmed with code prior to run time. The scheduling of the code has been previously optimized for the target algorithm. The NiosII's hardware abstraction layer (HAL) single-threaded runtime environment is employed rather than using an operating system as it reduces the overall program size and running time.

The execution of the scheduled tasks of the design, derived from Modelsim simulation results, is depicted in

Fig. 13(a). First of all, this illustrates how the architecture with token-ring topology works. The 5 processors are assumed to start running simultaneously. In doing a task, a processor needs to acquire a specific token from the previous processor before accessing (write/read) the required FPU, and passes the token to the next processor after FPU accessing, which is denoted by the solid arrowed lines in the figure (denoted only for tasks 1w in the figure). The FPU tokens are initialized at TF1 (the FIFO between the processor CPU1 and processor CPU5 as shown in Fig. 11), which allows CPU1 firstly to acquire the token and access the correspondent FPU, followed by CPU2, CPU3, CPU4 and CPU5. Note that the execution at the token FIFO side is not shown in the figure. Secondly, Fig. 13(a) also illustrates the interaction between the hardware and software. In each operation for the hardware multiply, divide and square-root FPUs, the two sets of operands are distributed to the two duplicate FPUs of the same type by the programmed software, so that the two calculations can be processed in parallel. The dotted arrowed lines in Fig. 13(a) denote such operation for processor tasks 2w and 2r on CPU1. Thirdly, a processor will stall if the token is not available for it to read or if FPU results are pending. The initial stalls (IS) are inevitably produced due to the token-ring feature. They are of fixed duration and are ignorable if considered in a long execution procedure. The subsequent stalls (SS) are produced due to the overly long latency of the FPUs. Both types of stall can postpone the overall complete time. The delay is accumulated from preceded processors' stalls and previ-

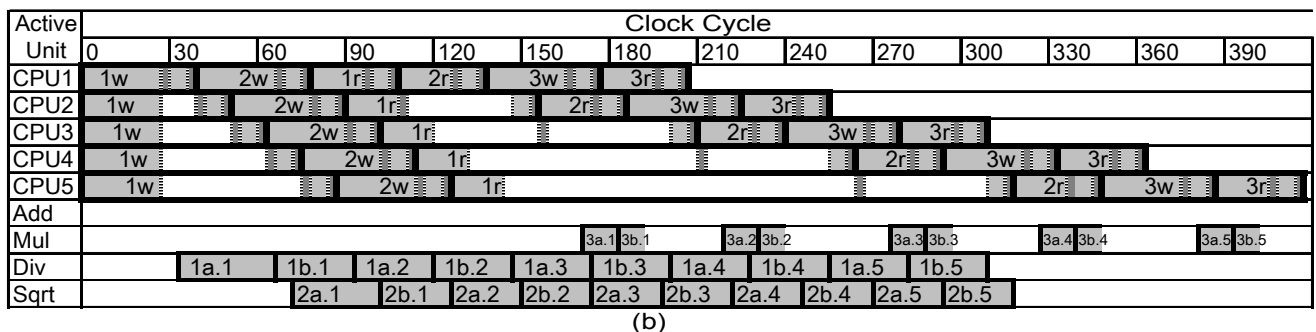
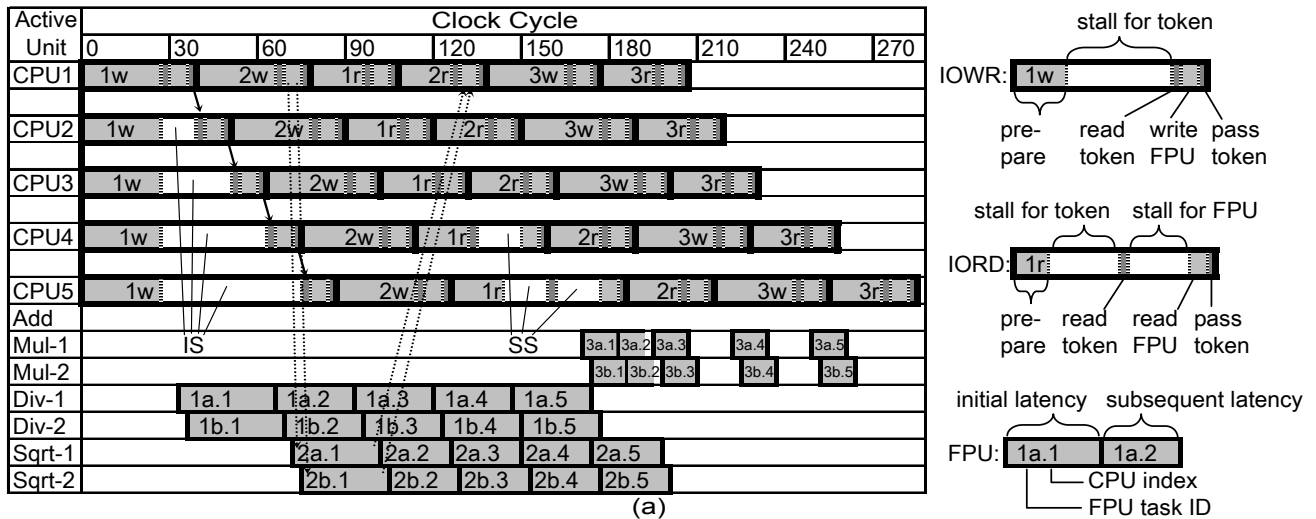


Fig. 13. (a) Task execution in the final design. (b) Task execution in another design with less FPUs

ous tasks' stalls within the same processor. Obviously processor CPU5 has the most delay and hence finishes execution the latest, taking 288 cycles.

Fig. 13(b) shows the execution of the tasks in the same order by another design which uses only 1 add, 1 multiply, 1 divide and 1 square-root. It can be observed that the subsequent stalls (SS) at the processor circuit are increased, due to the slower floating point calculation at the FPU circuit.

It is conceivable that on a device which allows sufficient quantity of FPUs or fast enough FPUs to be implemented, most of the subsequent stalls (SS) can be eliminated. And this infers that the processor minimum time overhead introduced by the token-ring scheme only consists of the token read/pass time and the processor initial stalls (IS) which are trivial in the whole Fastsim process.

3.2.3 Summary

The advantage of using the proposed architecture for the Fastsim part with 10 parallel processes has been demonstrated. Implementing even one more FPU can give enhancement to all processors, and the slowest processor get the most enhancement. While in a common FPU-non-share multiprocessor architecture, eg. the OpenSPARC T2 [16], implementing one more FPU can only enhance the performance of the objective processor. The proposed architecture with token-ring topology also has an inherited protocol to convey the access privilege with low overhead. Comparably, in a common FPU-share multiprocessor architecture, eg. the OpenSPARC T1 [18], access privilege are often gained by polling/interrupting which are found to be inefficient.

Finally, 5 NiosII/f processors are connected in a token ring manner via FIFOs sharing 1 add, 2 multiply, 2 divide and 2 square-root buffered FPUs, are implemented for the contact force calculation (Fastsim part). The demonstrated task scheduling approach for the sampled objective equations is applied to the whole algorithm and its execution takes about 16150 clock cycles to finish.

3.3 Overall Design

Fig. 14. shows the overall hardware design on the FPGA and its peripherals on the evaluation board. The design comprises one processor and four FPUs for the Hertz algorithms and five processors and seven FPUs for the Fastsim algorithms. The diagram omits the connection between the CPU0-CPU5 and the Avalon external bridge which communicates with the JTAG port peripheral, loads program and updates data from external SDRAM memory, and connects host PC via Ethernet. Program data is pre-programmed into SDRAM via JTAG. During run-time, simulation data is communicated between the host computer and the FPGA accelerator via Ethernet under UDP protocol, with an introduced latency typically <0.35ms. The communication between the Hertz part and the Fastsim part, as required in the restructured contact laws, is done via a dual-port memory on which two separate regions are defined storing Hertz-to-Fastsim data and Fastsim-to-Hertz data respectively to prevent collision.

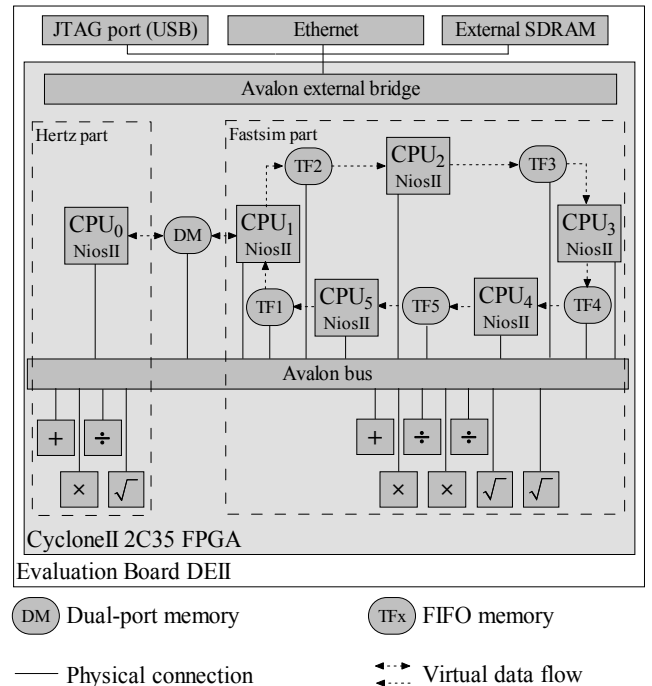


Fig. 14. Overall hardware architecture

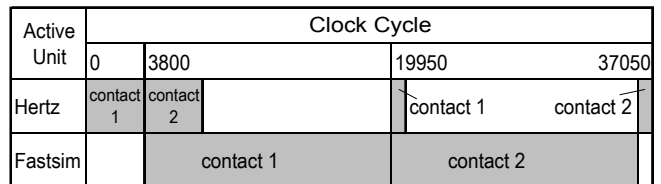


Fig. 15. Pipelining between the Hertz and the Fastsim

The processors in the two parts are programmed to perform in a pipelined mode to tackle two or more contacts' calculation. Fig. 15 shows the case when tackling two contacts. Firstly the Hertz part takes about 3800 cycles to calculate the size data for contact 1 and put it in the dual-port memory, and then continued with the calculation for contact 2. Once the Fastsim part acquires contact 1's size data from the memory, it starts computation for forces which will take 16150 cycles. The contact 1's force results are sent via the memory back to the Hertz part for the final summation calculation which takes 950 cycles. The Fastsim part right away continues with the contact 2 as its size data has already been prepared. The whole calculation for the two contacts takes 37050 cycles, which is 11% faster than that of running in an un-pipelined/serial manner. Note that the period for memory write/read is too short to be shown explicitly in the Fig. 15.

4 IMPLEMENTATION AND RESULTS

4.1 Implementation

The whole design is implemented on an Altera's CycloneII FPGA device EP2C35 on an evaluation board DEII.

Table 3 shows the FPGA utilization of the implementation. The area consumption is measured in terms of Logic Elements (LE) and Memory usage is the utilization of the on-chip RAM. The statistics have been obtained by syn-

TABLE 3
FPGA UTILIZATION STATISTICS

Hardware Module	Unit count	Unit Mem [KB]	Total Mem [KB]	% of Total Mem	Unit Area [LE]	Total Area [LE]	% of Total LE
CPU0	1	10.13	10.13	17.1%	2905	2905	8.7%
CPU1~CPU5	5	8.44	42.19	71.4%	2202	11012	33.2%
Add FPU (Hertz)	1	0	0	0	835	835	2.5%
Mul FPU (Hertz)	1	0	0	0	1271	1271	3.8%
Div FPU (Hertz)	1	0	0	0	908	908	2.7%
Sqrt FPU (Hertz)	1	0	0	0	977	977	2.9%
Add FPU (Fastsim)	1	1.13	1.13	1.9%	1108	1108	3.3%
Mul FPU (Fastsim)	2	1.13	2.25	3.8%	1514	3028	9.1%
Div FPU (Fastsim)	2	1.13	2.25	3.8%	1190	2379	7.2%
Sqrt FPU (Fastsim)	2	0	0	0	1545	3089	9.3%
Token FIFO	35	0	0	0	15	510	1.5%
Dual-port Memory	1	1.13	1.13	1.9%	214	214	0.6%
Others		0	0	0	3907	3907	11.8%
Utilization			59.06	100.0%		32143	96.8%

thesizing the design with Quartus 7.1 into the targeted device. With more data cache embedded, CPU0 consumes more on-chip memory than CPU1~CPU5 do, and also takes more LEs due to its additional built-in JTAG module and fixed-point divide unit. Each type of FPUs for the Fastsim utilizes more Logic Elements than that for the Hertz, as a result of their higher complexity with the buffering FIFOs. All buffering FIFOs are implemented with on-chip RAM, except that in the Square-root FPU which is implemented by LEs, because of on-chip RAM insufficiency on the device. The token FIFOs use LEs as well. The implementation of the dual-port memory consumes on-chip RAM and a few LEs. ‘Others’, including a Ethernet controller, a SDRAM controller and other miscellaneous connections, consumes 11.8% of total LE. All on-chip RAM are used. The statistics also shows that the design has a good capability to balance the usage between on-chip RAM and LE. Frequency 60 MHz is achieved on this device.

4.2 Experiments and Results

Two experiments are conducted to verify the design, one on a curved railway track and the other on a straight railway track with irregularities. In each experiment, 5 steps are carried out as shown in Fig. 16. Firstly, a two-axle railway vehicle model used in [19] with different track inputs consisting of the vehicle dynamic model and the contact laws model is built and run in the Simulink environment in double-precision. At the leading wheel-set’s two contacts, input data to the contact laws model and the output data from it are recorded at 1ms step-size which is also the running step-size of the model. The simulation time is 10s and hence 20000 sets of input and output data are recorded. In step 2, the recorded input data are converted to single-precision format (with loss of precision) and uploaded to the SDRAM memory. In step 3, the FPGA device runs online reading both program code and input data from the SDRAM, executing the implemented contact laws and writing results back to the SDRAM. The whole online run takes 0.625s, yielding 0.625ms (37500 cycles @60MHz) on average for two con-

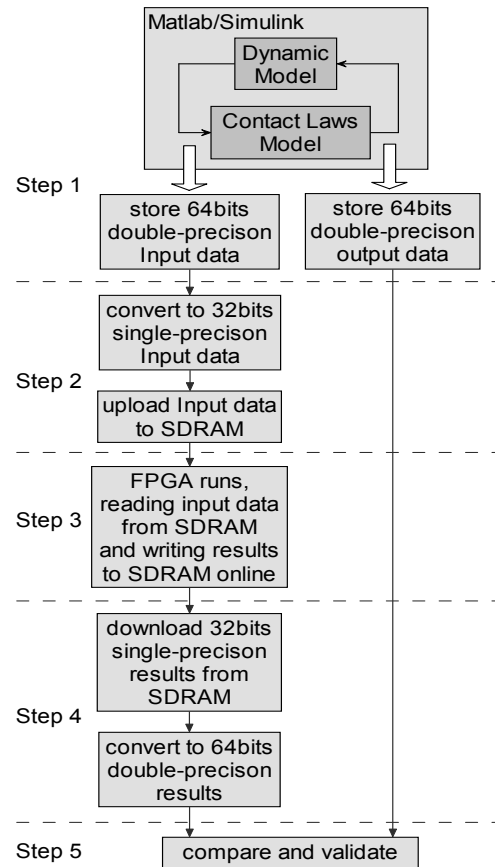


Fig. 16. Experiment and validation flow.

tacts’ calculation per iteration, which approximately agrees with the theoretical value (37050 cycles) stated in section 3.3. In step 4, the single-precision results are downloaded from the SDRAM and converted to double-precision format. In the final step, these results are compared with the previously recorded output data from the contact laws model in Simulink. The longitudinal and lateral contact force of the FPGA results on one of the contacts and the relative errors of the comparison are shown in Fig. 17 and Fig. 18, for the curved track’s experiment

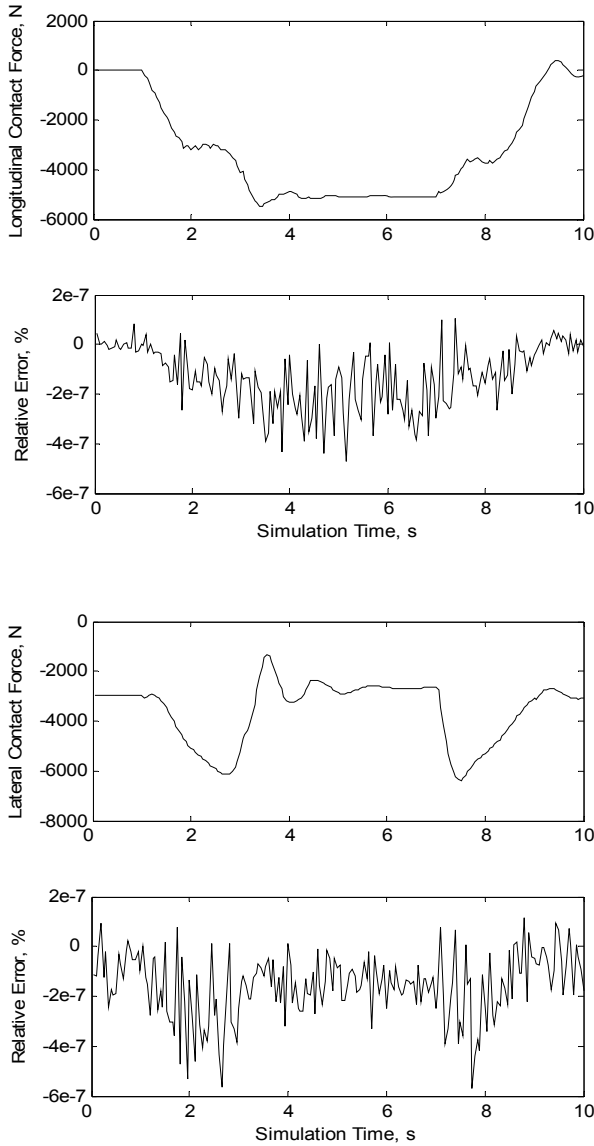


Fig. 17. FPGA calculation results and errors (on curved track)

and for the irregularized straight track's experiment respectively. Both of the relative errors in percentage are within the order of 10^{-7} which is very small considering that the Fastsim algorithm itself contains a 10%-15% relative error based on the comparison with the exact theory

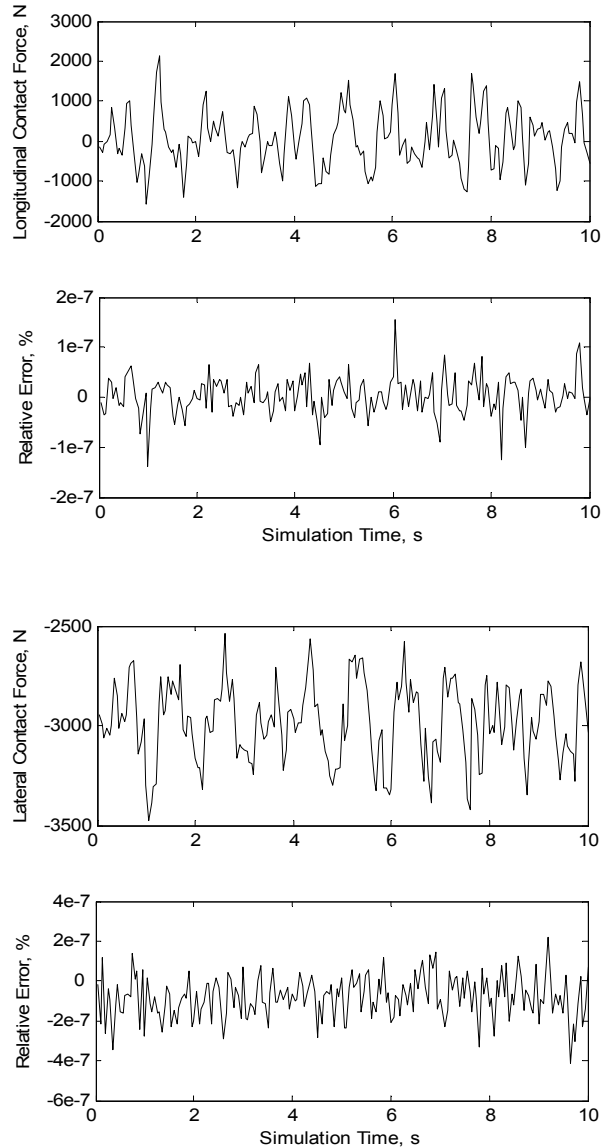


Fig. 18. FPGA calculation results and errors (on irregularized straight track)

(CONTACT) [9]. The relative errors of the results for the other contact have the same property. Thus it is concluded that the accuracy by using single-precision calculation in the FPGA design is much better than acceptable.

4.3 Improvement Space

It is obvious in Fig. 15 that a reduction of the Fastsim runtime will significantly speed up the whole pipelined process. As discussed in section 3.2.2, the current design of using 1 add, 2 multiply, 2 divide and 2 square-root FPU's fails to eliminate all avoidable processor stall periods in the Fastsim part. A benchmark on the implementation is carried out to measure the stall period (stall-for-token and stall-for-FPU-results) of each processor in the Fastsim part, as shown in Fig. 19, in order to estimate the design's improvement space on a larger FPGA device. The data shows that about 50% of the processor cycles are wasted on stalling. This proposes a viable enhancement solution on a larger device by simply implementing more

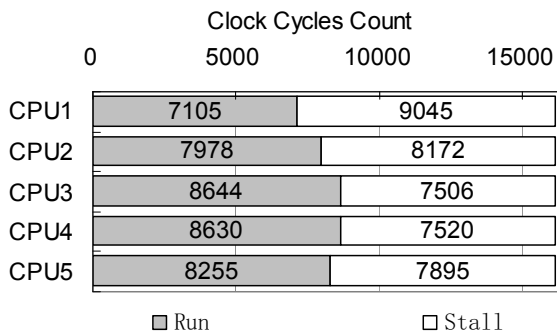


Fig. 19. Fastsim processors clock cycles count

FPU and having tasks scheduled well using the approach discussed. A preliminary trial using 2 add, 3 multiply, 4 divide and 2 square-root FPUs in the Fastsim part can have most of processor stall periods eliminated and yields 25% reduction in execution clock cycles, from 37500 to 28125 clock cycles, for the whole pipelined process of a pair of contacts.

5 CONCLUSION

The original version of the contact laws (Hertz and Fastsim) has been properly investigated to produce a restructured version which is more suitable for implementation on a multiprocessor platform. An implementation of the restructured contact laws has been successfully made on a medium-sized FPGA device. A hardware/software approach is utilized in the FPGA design. The hardware is well allocated to efficiently and effectively coordinate the 6 processors and 11 floating point units, by using token ring theory and buffering mechanism. A hybrid DAG approach using HLFET is developed to aid the software design. The implementation results show high FPGA utilization efficiency of the design.

The performance of the design has also been validated to meet the real-time requirement. The latency of the FPGA contact law accelerator for the 4 contact patches is 1.25ms (0.625ms per pair). An additional overhead is created through the communication of simulation data to the host PC which typically adds a further 0.35ms latency for point-point UDP traffic. Therefore the overall latency meets the real-time requirement of less than 2ms.

The developed multiprocessing design for the Fastsim part has great scalability in terms of varying the number of processors and varying the grid size of the contact patch, e.g. 10 processors with grid size $m_0=10$ and $n_0=10$, or 10 processors with grid size $m_0=20$ and $n_0=20$. In the future, a larger device will be used for implementation for validation purposes. In addition more floating point unit resources can be put on the device to reduce the processor stalls. Furthermore, scalability will be evaluated by connecting several such designs to tackle several pairs of contacts in parallel.

REFERENCES

[1] S. Iwnicki, "The Manchester Benchmarks for Rail Vehicle Simulation", Sweets & Zeitlinger Publishers, 1999

[2] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and automated multiprocessor system design, programming, and implementation," IEEE Transactions on Computer-Aided Design of Integrated Circuits, v 27, n 3, 542-55, March 2008

[3] J.L. Tripp, H.S. Mortveit, A.A. Hansson, and M. Gokhale, "Metropolitan Road Traffic Simulation on FPGAs," Proc.13th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM), 2005

[4] R. Scrofano, M.B. Gokhale, F. Trouw, and V.K. Prasanna, 'Accelerating Molecular Dynamics Simulations with Reconfigurable Computers', IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 6, June 2008

[5] R. Scrofano, M. Gokhale, F. Trouw, and V.K. Prasanna, "A Hardware/Software Approach to Molecular Dynamics on Re-

configurable Computers," Proc. 14th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM '06), pp. 23-34, Apr. 2006

[6] T. L. Adam, K. M. Chandy, and J. Dickson, "A comparison of list scheduling for parallel processing systems", Comm. ACM 17, No. 12, pp. 685-690, Dec. 1974

[7] Y.-K. Kwok and I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms," J. Parallel & Distributed Computing, vol. 59, no. 3, pp. 381-422, Dec. 1999.

[8] A. Kulmala, O. Lehtoranta, T.D. Hämäläinen, and M. Hännikäinen, "Scalable MPEG-4 Encoder on FPGA Multiprocessor SOC", EURASIP Journal on Embedded Systems, vol. 2006, Article ID 38494, p.p. 1-15, 2006

[9] J.J. Kalker, Rolling contact phenomena - linear elasticity, CISM Courses And Lectures No.411 "Rolling contact phenomena", 2000, 1-84

[10] Altera Corporation, "Nios II Processor Reference Handbook," Ver. 7.1.0, May 2007

[11] Altera Corporation, "Avalon Memory-Mapped Interface Specification," Ver. 3.3, May 2007

[12] Altera Corporation, "Quartus II Version 7.1 Handbook Volume 4: SOPC Builder," Ver. 7.1, May 2007

[13] Altera Corporation, "Creating Multiprocessor Nios II Systems Tutorial," Ver. 7.1, May 2007

[14] Terasic, DE2 Development and Education Board User Manual, available at http://www.terasic.com.tw/attachment/archive/30/DE2_UserManual_1.42.pdf, version 1.42, Dec. 2006 (Site visited on 10-Sep-2008)

[15] Jidan AI-Eryani, Floating Point Unit, OPENCORE project, available at <http://www.opencores.org/?do=project&who=fpu100> (as of 14-May-2008)

[16] Sun Microsystems, Inc., "OpenSPARC™ T2 Core Microarchitecture Specification," December 2007

[17] IEEE, "802.5, 1998 Edition (ISO/IEC 8802-5:1998) IEEE Standard for Information technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Specific requirements--Part 5: Token Ring Access Method and Physical Layer Specification," 1998

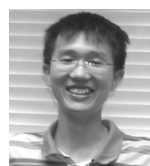
[18] Sun Microsystems, Inc., "OpenSPARC™ T1 Microarchitecture Specification," August 2006

[19] T.X. Mei, R.M. Goodall, LQG and GA solutions for active steering of railway vehicles, IEE Proc.-Control Theory Appl. Vol. 147. No.1, 2000, 111-117

[20] G. De Micheli, R. Ernst, and W. Wolf, "Readings in Hardware/Software Co-design". Morgan Kaufmann Publishers Inc., San Francisco, CA, 2001.

[21] W. Wolf, "A decade of hardware/software codesign," Computer, vol. 36, no. 4, pp. 38-43, Apr. 2003.

[22] Y. Zhou, T.X. Mei, S. Freear, FPGA Implementation of Wheel-rail Contact Laws, UKACC Control 2008



Yongji Zhou received his BEng degree from the Xidian University, Xi'an, China in 2003, and his MSc by research degree from the University of Leeds, UK, in 2005. He worked as an R&D Engineer in Midea Group, China, in 2005, performing electronic designs for Air Conditioners. He is pursuing his PhD degree funded by the ORSAS awards in the School of Electronic and Electrical Engineering, the University of Leeds. He is an IET student member. His research interest includes FPGA embedded system, real-time hardware-in-loop simulation. He published one journal paper during his BEng study, two conference papers during his MSc study and one journal and two conference papers (so far) in his PhD study.



Prof. T.X. Mei's educational background includes B.Sc. (1982) and M.Sc. (1985) both from Shanghai Tiedao University, M.Sc (by research). (1991) from Manchester University and Ph.D. (1994) from Loughborough University. He was previously an academic (Lecturer 2001, Senior Lecturer 2003, and Reader 2008) at the University of Leeds, UK. He holds the position of Chair in Control and

Mechatronics at the School of Computing, Science and Engineering, Salford University, UK. Professor Mei is a Fellow of IET (UK) and Chartered Engineer. His main research is concerned with the advanced control and mechatronic solutions for industrial problems especially applied to railway and automotive systems including fault tolerance, condition monitoring, traction control, wheel-rail contact mechanics, vehicle dynamics, intelligent sensing and data fusion, and system integration. Professor Mei is one of the most active researchers world-wide in the latest fundamental research into active steering and system integrations for railway vehicles. He has given a number of invited seminars and state-of-art reviews at the international level and has published about 90 papers in the last 10 years in leading academic journal and international conferences.



Dr. Steven Freear Dr. Steven Freear is currently Senior Lecturer in the School of Electronic and Electrical Engineering at the University of Leeds. His main research interest is concerned with advanced analogue and digital signal processing and instrumentation for a number of application areas including control systems, communication systems and ultrasonics. A core strand concerns

real-time operation and efficient implementation of signal processing onto VLSI architectures. He teaches digital signal processing, micro-controllers/microprocessors, VLSI and embedded systems design, hardware description languages at both undergraduate and post-graduate level.