FPGA IMPLEMENTATION OF WHEEL-RAIL CONTACT LAWS

Y.J. Zhou*, T.X. Mei, S. Freear*****

School of Electronic and Electrical Engineering, University of Leeds
**(eenyz@leeds.ac.uk)*
***(T.X.Mei@leeds.ac.uk)*
****(S.Freear@leeds.ac.uk)*

Abstract: This paper presents the development of an accelerator for the real-time simulation of wheel-rail contact laws (Hertz and Fastsim), which would enable the use of hardware-in-the-loop for 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 modern general-purpose CPU or DSP in the aspects of processing time, configuration flexibility and cost. The Fastsim algorithm is 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 allocation of the FPGA's resources. Copyright © UKACC2008

1. INTRODUCTION

There have recently been research and development activities in the railway industry in the area of active controls, especially in the primary suspensions for active steering and stability control [Mei and Goodall, 2000]. Simulation of the behaviour of railway vehicles is regarded as an essential designing method in order to verify the improvement in dynamic performance offered by the emerging technology. 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 without the need of real track experiments which are both economically and logistically difficult. One of the examples is a study in [Bonaventura, et al., 2000] 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 system technology makes it viable to design custom accelerators for complex and computationally demanding simulation models. One of such applications 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 [Champagne, et al., 2003]. FPGA with its high configuration flexibility and outstanding execution parallelism is applied in many specific cases. 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 time step reduction [Lavoie and Dessaint, 1997]. Moreover, the peak FPGA floating-point performance is now growing significantly faster than the peak floating-point performance for CPU [Underwood, 2004], which will certainly help and drive more applications to the FPGA field.
For the simulation of railway vehicle dynamics, the main part that demands the most computation power is that for 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 a cost effective approach with high flexibility to reduce the processing time and eventually to help in achieving simulation in real-time.

The paper is structured as follows: An introduction is given in section 1; Section 2 states the overall requirement for the proposed accelerator and also compares the performance using different approaches; Four aspects of FPGA implementation are discussed in section 3; The overall architecture on the FPGA is proposed in section 4; Finally a conclusion is made.

2. OVERALL REQUIREMENT

A model of a two-axle vehicle consisting of a body frame and two wheelsets (or four wheels), as shown in Figure 1, is used in the study. The equations of motions are given in equations (1)~(6) [Mei and Goodall, 2000]. The model can be easily modified to represent a conventional bogie.

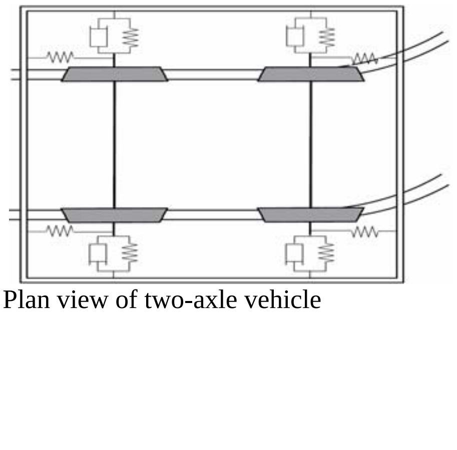


Figure 1. Plan view of two-axle vehicle

Front wheelset's lateral and yaw motions:

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

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

Rear wheelset's lateral and yaw motions:

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

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

Body frame's lateral and yaw motions:

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

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

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 [Brickle, 1986]. In many complex dynamic railway vehicle designs, e.g. [Busturia, et al., 2004], non-linear wheel-rail contact geometry is required for high simulation accuracy. Kalker's simplified theory [Kalker, 2000], one of the most widely used wheel-rail contact laws, is used for calculating contact forces at this type of contacts as shown in equations (7)-(10).

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

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

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

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

The contact areas for each of the contact patches U_{1L} U_{1R}, U_{2L} and U_{2R} are typically 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 geometry at the interface and normal force applied for each of the contact patches.

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

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

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 $m_0\times n_0$ slices. Figure 2 is an illustration of the Fastsim calculation at the front left wheel contact patch where contact area has been determined by Hertz and m_0=10 and n_0=10 are set. The traction at each slice is denoted by $\mathbf{p}_{1L}(x, y)$ and is determined by checking if the pre-calculated traction $\mathbf{p}_{1LH}(x, y)$ exceeds the traction bound t_b, as shown in equations (13)~(15).

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

where

$$\mathbf{p}_{1LH}(x,y)=\mathbf{p}_{1LH}(x-d_x,y)-d_x\cdot\gamma(x+d_x/2,y)/L \quad (14)$$

$$t_b=3\cdot m_u\cdot N\cdot\sqrt{1-x^2/a^2-y^2/b^2}\,/(2\cdot\pi\cdot a\cdot b) \quad (15)$$

Every slices' tractions are determined in a sequence as numbered in Figure 2, and the total traction force is summed up at the end. Larger number of slices can be used to achieve higher integration accuracy, but m_0=10 and n_0=10 is sufficient in most cases and is used in the design.

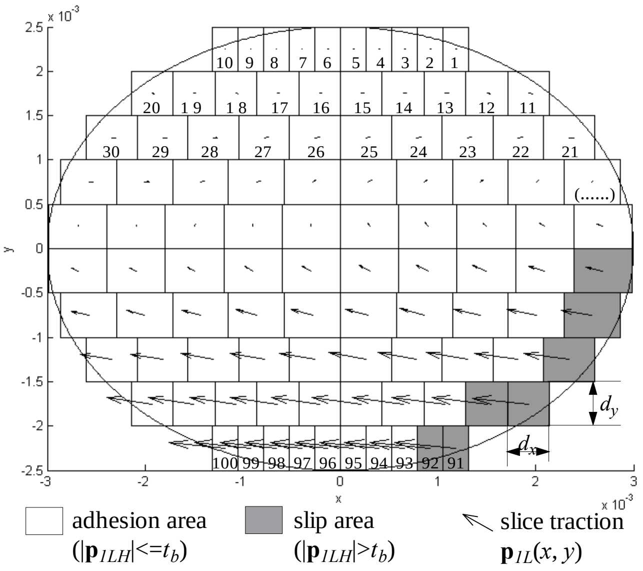


Figure 2. Contact area and Fastsim calculation at the front left wheel contact patch (contact semi-axes a=3.0mm, b=2.5mm; number of x-intervals, m_0=10; number of y-intervals, n_0=10)

The equations (11)-(15) are repeated 3 more times to produce $\mathbf{p}_{1R}$, $\mathbf{p}_{2L}$ and $\mathbf{p}_{2R}$ for other 3 contact patches. Together with the integrations in equations (7)-(10), the contact laws consume a large part of computation in the simulations. Simulations run on an Intel Pentium4 3.0GHz 1GB memory platform show that the Fastsim takes 69% of the total computation time and Hertz 2% making the contact model taking up 71% of the total time, whereas the dynamic model part takes only 29%. The contact model computation time for 4 patches in each step is 5.3ms, which prevents the whole model to be simulated in real-time, typically 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 (or 0.5ms preferably) which would eventually allow the simulation to be run in real-time with 1ms step size. The dynamic equations (1)-(6) can still be run on a PC, allowing for the necessary flexibility for different vehicle configurations. The overall arrangement for the simulation is shown in Figure 3.

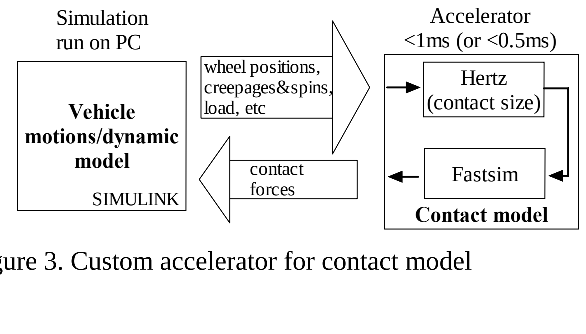


Figure 3. Custom accelerator for contact model

The accelerator implementation was first attempted by using DSP devices. A TI's second generation floating point DSP C6713B was used to implement the contact model, yielding profiling result of 1.6ms for 4 patches in TI's DSP simulation tool CCStudio. Although it is 2.3 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 a best approach not only because of the complexity in building arrays of DSPs [Ai and Hu, 2001] but also because of such platform's poor adaptability to a different number of 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 algorithm 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.

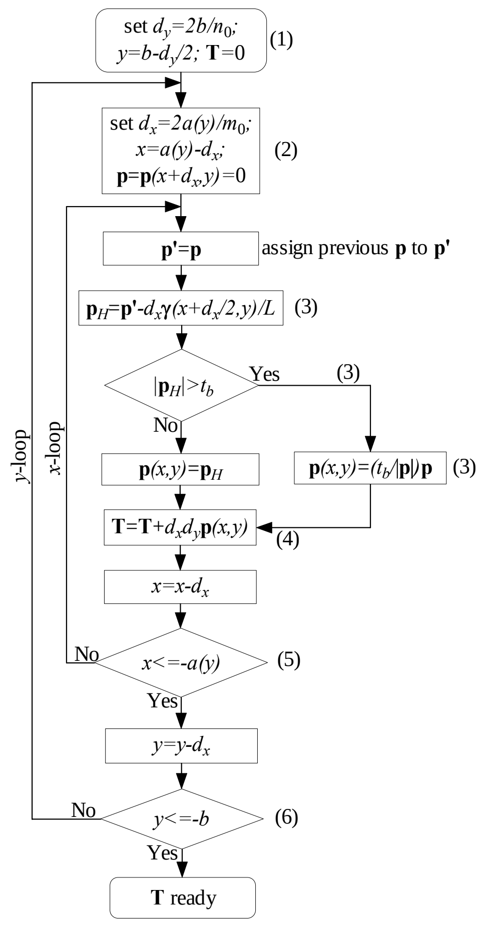
3. FPGA IMPLEMENTATION

Four 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. There is an inner *x*-loop and an outer *y*-loop. *x*-loop will execute m_0 times to tackle all m_0 slices 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 slices ($m_0 \times n_0$) one by one in the sequence as shown in Figure 2.

Since in one particular row the tangential traction $\mathbf{p}_H$ for the current slice is associated with the previous slice's traction **p'**, such "one by one" sequential process inside *x*-loop is inevitable. The original Fastsim also does computations for all rows sequentially, but it is found to be dispensable. Because all n_0 rows' computations as well as the initiations of *x*-loop are independent of any other rows, viz. calculation results for one particular row can be worked out without knowing other rows' results. To make use of FPGA's parallel execution capability, it is proposed to split the sequential *y*-loop of the original algorithm into n_0 parallel processes, as shown in Figure 5.



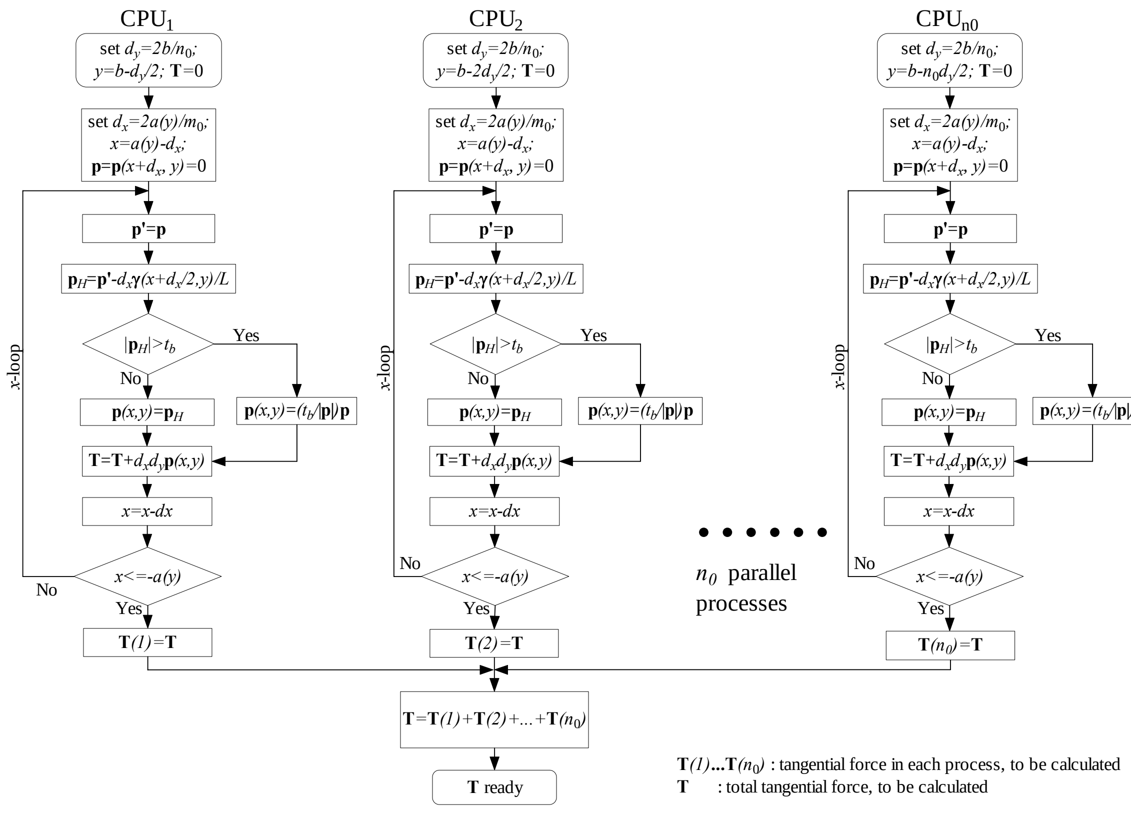
set $d_y=2b/n_0$; $y=b-d_y/2$; $\mathbf{T}=0$
(1)
set $d_x=2a(y)/m_0$; $x=a(y)-d_x$; $\mathbf{p}=\mathbf{p}(x+d_x,y)=0$
(2)
$\mathbf{p'}=\mathbf{p}$
assign previous **p** to **p'**
$\mathbf{p}_H=\mathbf{p'}-d_x\gamma(x+d_x/2,y)/L$
(3)
$|\mathbf{p}_H|>t_b$
Yes
(3)
No
$\mathbf{p}(x,y)=\mathbf{p}_H$
$\mathbf{p}(x,y)=(t_b/|\mathbf{p}|)\mathbf{p}$
(3)
$\mathbf{T}=\mathbf{T}+d_xd_y\mathbf{p}(x,y)$
(4)
$x=x-d_x$
No
$x<=-a(y)$
(5)
Yes
$y=y-d_x$
No
$y<=-b$
(6)
Yes
T ready
y-loop
x-loop

Figure 4. Original Fastsim algorithm flow chart

In the restructured Fastsim, the *x*-loop operation remains the same to update tangential forces $\mathbf{T}(1)$, $\mathbf{T}(2)\ldots\mathbf{T}(n_0)$ for each row. The results of all parallel processes will be summed up to achieve a total contact force **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_0-1 times faster performance than the original one. In the case where m0=10 and n0=10 performance is now 9 times faster than the original one, under the ideal condition and assuming that 10 processors are available for the 10 rows' calculation. Even with only 5 processors, the 10 rows' calculation can still be executed in 2 times and give time enhancement.

3.2 Floating Point Operation implementation

Floating point operations are required through out the Fastsim and Hertz algorithms. The total floating point operation requirements equate to 1168 additions/subtractions, 1114 multiplications, 264 divisions and 212 square-roots for each contact patch. Single precision provides sufficient accuracy in the model studied. There are many practical ways to implement single-precision floating point operations in an FPGA. Altera NiosII's complier environment supports floating point function in the code, giving a software implementation solution. As NiosII is not developed specifically for floating point operations, this "soft" approach is not suitable for applications with intensive floating point



CPU$_1$
CPU$_2$
CPU$_{n0}$
set $d_y=2b/n_0$; $y=b-d_y/2$; **T**=0
set $d_y=2b/n_0$; $y=b-2d_y/2$; **T**=0
set $d_y=2b/n_0$; $y=b-n_0d_y/2$; **T**=0
set $d_x=2a(y)/m_0$; $x=a(y)-d_x$; **p**=**p**$(x+d_x, y)$=0
p'=**p**
p$_H$=**p'**-$d_x\gamma(x+d_x/2,y)/L$
|**p**$_H$|>t_b
Yes
No
p(x,y)=**p**$_H$
p$(x,y)=(t_b/|$**p**$|)$**p**
T=**T**+d_xd_y**p**(x,y)
$x=x-dx$
$x<=-a(y)$
x-loop
T(1)=**T**
T(2)=**T**
T(n_0)=**T**
n_0 parallel processes
T=**T**(1)+**T**(2)+…+**T**(n_0)
T ready
T(1)...**T**(n_0) : tangential force in each process, to be calculated
T : total tangential force, to be calculated

Figure 5. Restructured Fastsim algorithm flow chart

computations. Altera also offers a hardware solution called “Floating-Point Custom Instructions” which attaches a floating point coprocessor to NiosII to accelerate the operation by hardware. But this multi-functional coprocessor can only do floating point operations sequentially, viz. the current operation needs to be finished before the later one can be started, as shown in Figure 6. This implies that different parts of the coprocessor circuit are functioning successively rather than simultaneously, which is not a cost efficient method. Another approach is to apply featured IPs for this purpose. A set of well developed floating point units (FPUs) from open source [Jidan, 2006] is used in the design. Although floating point operations are initialised from NiosII sequentially likewise, they can be executed in parallel in different FPUs independently, as shown in Figure 6. The reusability of IPs makes it more flexible to allocate more resource to a particular type of operation, e.g. to implement one more divider for floating point divisions if required.

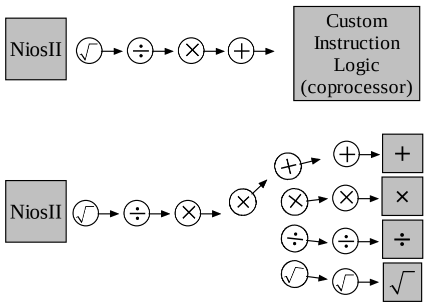


Figure 6. NiosII with “Floating-Point Custom Instructions” coprocessor (upper) VS NiosII with Floating Point Units (lower)

Table 1 summarizes the floating point operation performance and resource usage by these three approaches. The comparison between the NiosII with “Floating-Point Custom Instructions” coprocessor and the NiosII with Floating Point Units also shows that the later one has overall faster performance and meanwhile consumes less resources.

	FP SW	FP CI	FP FPU**
Add/Sub	307	14	7 (3)
Multiply	325	12	12 (8)
Divide	482	32	35 (31)
Square root	>482*	>32*	35 (31)
Total cycles	**>950058**	**>44952**	**38204 (27172)**
Resource	**0 LEs**	**≈5400 LEs**	**≈5000 LEs**

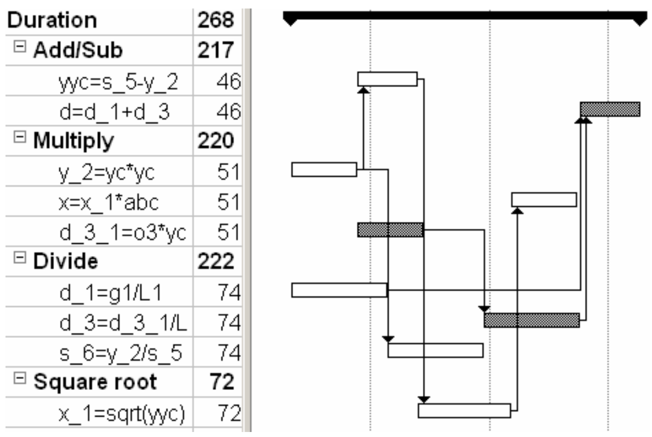
*Estimated figures, not tested

**Figures in parenthesis is results in pipelined operation

Table 1. Floating point operation performance and resource usage in different approaches (unit: cycles)

3.3 Operation Scheduling and Optimization

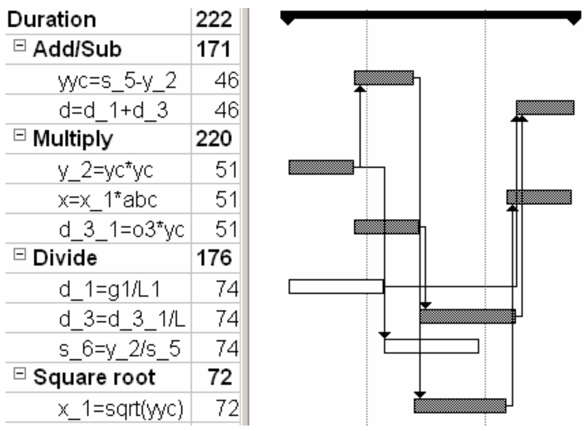
Following the FPU approach in the previous subsection, each floating point operations in the contact laws studied are regarded as individual elements having dependency and independency relationships which requires scheduling and mapping. Figure 7 is an example of a scheduled part of the entire algorithm, under the FPU resource constraint of 1 adder/subtracter, 1 multiplier, 1 divider and 1 square root. The operation dependencies and independencies are clearly shown. For example, operation “d_3=d_3_1/L” depends on the result of “d_3_1=o3*yc” and is the predecessor of “d=d_1+d_3”. “d_3=d_3_1/L” and “s_6=y_2/s_5” are independent with each other, yet are executed sequentially due to the 1 divider resource constraint. The critical path (grey boxes) on which tasks (operations) and the resources assigned to them can be paid close attention to in order to reduce the duration. In this example, it is considerable to add another divider to do "d_3=d_3_1/L" so that it doesn't need to wait until "s_6=y_2/s_5" is finished. Thereupon after assigning a second divider as extra resource, the operations can be rescheduled, as depicted in Figure 8, showing a 17% reduction of the whole duration (from 268 cycles to 222 cycles). Note that the given durations for each operations compromise the NiosII accessing time and the FPU calculation time.



Duration	268
⊟ Add/Sub	217
yyc=s_5-y_2	46
d=d_1+d_3	46
⊟ Multiply	220
y_2=yc*yc	51
x=x_1*abc	51
d_3_1=o3*yc	51
⊟ Divide	222
d_1=g1/L1	74
d_3=d_3_1/L	74
s_6=y_2/s_5	74
⊟ Square root	72
x_1=sqrt(yyc)	72

▨ Critical path

Figure 7. Operations scheduled with FPU resources of 1 adder/subtracter, 1 multiplier, 1 divider and 1 square root.



Duration	222
⊟ Add/Sub	171
yyc=s_5-y_2	46
d=d_1+d_3	46
⊟ Multiply	220
y_2=yc*yc	51
x=x_1*abc	51
d_3_1=o3*yc	51
⊟ Divide	176
d_1=g1/L1	74
d_3=d_3_1/L	74
s_6=y_2/s_5	74
⊟ Square root	72
x_1=sqrt(yyc)	72

▨ Critical path

Figure 8. Operations scheduled with FPU resources of 1 adder/subtracter, 1 multiplier, 2 dividers and 1 square root.

In this way the operations through out the algorithm can be scheduled and optimized for different resource constraints.

3.4 Multi-NiosIIs sharing FPU

Following the method suggested in section 3.1, it is proposed to assign individual NiosII processors for the parallel processes in the restructured Fastsim algorithm. It is effortless to attach dedicated sets of FPU to each NiosIIs so that they run independently. However, it is found to be an inefficient implementation approach. The timing diagram of an example of connecting one FPU(adder) to one NiosII is shown in Figure 9. NiosII takes 28 cycles to prepare the operands (pre) before accessing the FPU (a) which includes writing operands and reading results and takes 6 cycles. The FPU adder is regarded as busy (e) while it is being accessed. NiosII needs about 10 more cycles to deal with the results (o). This yields that the FPU adder is working only in 13% of the whole period producing one outcome per 46 cycles.

Active Unit	clock cycle																											
	0			21	22			43	44			65	66			87	88			109	110			131	132			153
NiosII	pre					a	o		pre					a	o		pre					a	o					
FPU						e								e								e						

Figure 9. Timing diagram of one FPU(adder) connected to one NiosII

A more efficient way by sharing the FPU with multi-NiosIIs is proposed. The timing diagram of an example of connecting one FPU(adder) to three NiosIIs is shown in Figure 10. NiosII *a* takes about 6 cycles to inform (i) NiosII *b* after it has finished accessing FPU so that the later can start to access FPU. Then likewise, NiosII *b* will inform NiosII *c*, and NiosII *c* will inform NiosII *a*. NiosII *c* is postponed by about 22 cycles at the beginning and won't get further delay in subsequent operations. This makes the FPU adder to give one outcome per 18 cycles, about 60% more efficient than the previous approach. Resources on FPGA are also utilized more efficiently.

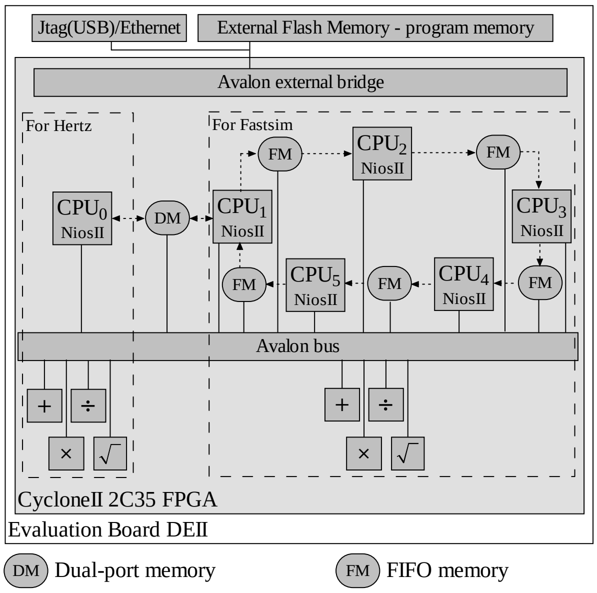
Active Unit	clock cycle																											
	0			21	22			43	44			65	66			87	88			109	110			131	132			153
NiosII a	pre					a	i	o		pre					a	i	o		pre					a	i	o		
NiosII b			pre					a	i	o		pre					a	i	o		pre					a	i	o
NiosII c					pre					a	i	o		pre					a	i	o		pre					a
FPU						e		e		e					e		e		e					e		e		e

Figure 10. Timing diagram of one FPU(adder) connected to three NiosIIs

4. FPGA OVERALL ARCHITECTURE

The overall FPGA architecture is presented in Figure 11. It consists of one NiosII (CPU_0) for Hertz and five NiosIIs (CPU_1~CPU_5) for Fastsim. A set of FPUs is dedicated to CPU_0 while another set of FPUs is shared by CPU_1~CPU_5. A particular processor is allocated for Hertz because in this way the two parts, Hertz and Fastsim, can be pipelined which improves the throughput significantly as four contact patches need to be calculated. Also the Hertz processor CPU_0 can handle other miscellaneous tasks such as communicating with PC. The Hertz part and the Fastsim part communicate via a dual-port memory, while the correspondence among CPU_1~CPU_5 is done via FIFO memories. Processors' programs are loaded from external Flash memory.

This design is fitted on the EP2C35 FPGA consuming 80% of total logic elements and 90% of total RAM block bits and runs at a frequency 60MHz.



The diagram omits the connection between the CPU_0~CPU_5 and the Avalon external bridge for loading program from external Flash memory. It also omits the FIFO memory inside of each FPU.

Figure 11. FPGA System-level diagram

5. CONCLUSION

In this study, the complex models for the simulation of wheel-rail contact mechanics at four contact patches have been implemented using a single medium performance FPGA chipset. Fastsim algorithms have been restructured and optimised. Hardware floating point units of addition, multiplication, division, square root which are integrated, and shared by several NiosII processors in the overall architecture to deliver a fast solution with efficient use of resources. The developed solution meets the requirement for real time simulations of vehicle dynamics and can be used in experimental studies of active control technologies in railway applications.

REFERENCES

Ai, H.J. and Hu, R.M.(2001). Implementing an audio multipoint processor on DSP array, *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing. ISIMP 2001 (IEEE Cat. No.01EX489)*

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*

Brickle, B.V.(1986). Railway Vehicle Dynamics, Phys Technol 17

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*, **v 28**, pt.2

Champagne, R., L.-A. Dessaint, H. Fortin-Blanchette(2003). Real-time simulation of electric drives, Mathematics and computers in simulation, **v 63**, n 3-5

Jidan AI-Eryani (2006) Floating Point Unit, available via www.opencores.org/projects.cgi/web/fpu100 /overview

Kalker, J.J.(2000). Rolling contact phenomena – linear elasticity, CISM Courses And Lectures No.411 'Rolling contact phenomena'

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

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**.

Stribersky, A., Moser, F.; Rulka, W.(2000). Structural dynamics of rail vehicle systems: a virtual systems approach, *Fifth International Conference on Computational Structures Technology and the Second International Conference on Engineering Computational Technology*

Underwood, K.(2004). FPGAs vs. CPUs: Trends in peak floating-point performance, *ACM/SIGDA International Symposium on Field Programmable Gate Arrays - FPGA*, **v 12**

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-slices, length of y-slices
E, v	: Young's modulus, Poisson's coefficient
F_{1wc}, F_{2wc}, F_{1bc}	: Centrifugal forces
F_{1wm}, F_{2wm}, F_{2bm}	: Component of the gravity force
I_b	: Wheelset Moment of inertia (558,800 $kg \cdot m^2$)
I_w	: Wheelset Moment of inertia (750 $kg \cdot m^2$)
k	: Spring coefficient (5,000,000 N·m/rad)
K_s	: Stiffness per wheelset (511,000 N/m)
L	: flexibility
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	: load in normal direction
$\mathbf{p}_H$	: 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
ψ_{1w}, ψ_{2w}, ψ_b	: Wheelsets, body frame's yaw displacement
$\gamma(x, y)$	: creepage vector at (x, y) $\in$contact area (including longitudinal creepage, lateral creepage and spin, in x and y directions)