



International Conference on Computational Science, ICCS 2012

Real-time Simulation of Dynamic Vehicle Models using a High-performance Reconfigurable Platform

Madhu Monga, Manoj Karkee¹, Song Sun, Lakshmi Kiran Tondehal, Brian Steward, Atul Kelkar, Joseph Zambreno²

Iowa State University, Ames, IA 50011 USA

Abstract

A purely software-based approach for Real-Time Simulation (RTS) may have difficulties in meeting real-time constraints for complex physical model simulations. In this paper, we present a methodology for the design and implementation of RTS algorithms, based on the use of Field-Programmable Gate Array (FPGA) technology to improve the response time of these models. Our methodology utilizes traditional hardware/software co-design approaches to generate a heterogeneous architecture for an FPGA-based simulator. The hardware design was optimized such that it efficiently utilizes the parallel nature of FPGAs and pipelines the independent operations. Further enhancement is obtained through the use of custom accelerators for common non-linear functions. Since the systems we examined had relatively low response time requirements, our approach greatly simplifies the software components by porting the computationally complex regions to hardware. We illustrate the partitioning of a hardware-based simulator design across dual FPGAs, initiate RTS using a system input from a Hardware-in-the-Loop (HIL) framework, and use these simulation results from our FPGA-based platform to perform response analysis. The total simulation time, which includes the time required to receive the system input over a socket (without HIL), software initialization, hardware computation, and transfer of simulation results back over a socket, shows a speedup of 2× as compared to a similar setup with no hardware acceleration. The correctness of the simulation output from the hardware has also been validated with the simulated results from the software-only design.

Keywords: FPGA, Real-Time Simulation, Hardware-in-the-Loop, Non-linear functions, Hardware acceleration

1. Introduction

Real-time simulation (RTS) is often a component of virtual prototyping used to study the dynamics of a physical system prior to actual hardware development. It has been utilized by engineers in various industries such as aviation [1], power systems [2], networking [3], automotive [4], traffic management [5], and medicine [6]. The physical systems encountered in these areas are mathematically modeled by deriving the ordinary differential equations (ODEs)

Email addresses: madhum@iastate.edu (Madhu Monga), manoj.karkee@wsu.edu (Manoj Karkee), sunsong@iastate.edu (Song Sun), kirantl@iastate.edu (Lakshmi Kiran Tondehal), bsteward@iastate.edu (Brian Steward), akelkar@iastate.edu (Atul Kelkar), zambreno@iastate.edu (Joseph Zambreno)

¹Manoj Karkee is with the Department of Biological Systems Engineering, Washington State University.

²Corresponding author

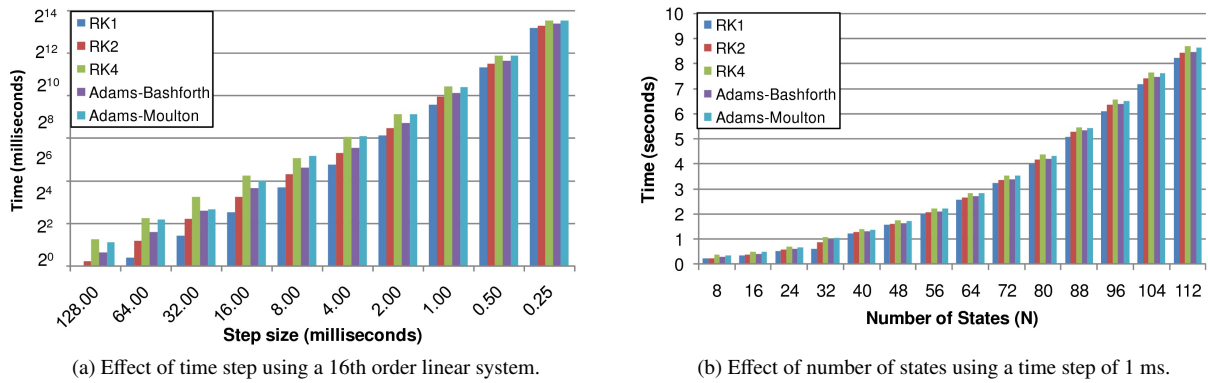


Figure 1: Effect of step size and number of states on the CPU computation time for a linear system integrator

that represent the underlying physics which dictates system behavior. To simulate these systems and estimate their state trajectories across a time duration, these ODEs are solved numerically using integration algorithms such as Runge-Kutta methods, Adams-Bashforth, or Adams-Moulton. The algorithms employ either fixed or variable integration time steps. The response generated by the simulation after each time step is considered useful for RTS only if the computation time for each time step remains below or equal to the actual time being simulated. However, the general-purpose CPU-based simulation of these systems continues to pose a major limitation on the smallest time-step with which RTS can be achieved. The reduced time-step required to simulate complex and fast systems imposes a tighter constraint on the time within which the computations have to be performed. The sequential execution of these computations thus fail to cope with the real-time constraints which further restrict the usefulness of RTS in a Virtual Reality (VR) environment.

In this paper, we focus on acceleration of real-time Hardware-in-the-Loop (HIL) simulation of vehicle systems. In our target system, an operator provides an input via a physical steering wheel, and the steering input is then presented to the vehicle model, after which a graphical engine takes the output of the model and renders graphics showing the movement of the vehicle in a virtual world [7].

The following general form represents a continuous-time state-space model form of dynamic systems:

$$\frac{dy_i}{dt} = f(u_i, y_i) = A * y_i + B * u_i \tag{1}$$

where y_i is an $N \times 1$ size vector representing the N states of the system at the present time, u_i is the system input vector of size $M \times 1$, A is an $N \times N$ state transition matrix that defines the coupling between various states of the system, and B is an $N \times M$ input matrix that relates the system inputs to the system states.

For an initial experiment, a simple and generic linear order model was implemented in MATLAB using different methods, where it was observed that the computation time was negatively affected by the increase in the number of computations involved in solving the ODEs, which varies with the choice of numerical integration method, the size of the time step t , and the number of physical states N being modeled. It is important to note that though an equivalent C code may be faster than a MATLAB implementation, we would still observe similar trends, although a more complex method and model and a smaller time-step would be needed.

Fig. 1 compares the CPU computation time with varying time step and number of states using different numerical integration methods. The computation time does not include the time required to receive the steering wheel angle, time to compute the position coordinates, and time to send these coordinates to the graphical engine. Fig. 1a shows the effect of reducing the time step on a 16th order linear system solved using RK1, RK2, RK4, Adams-Bashforth, and Adams-Moulton algorithms. It was observed that as the time step is reduced, the time taken to simulate for five seconds increases, for a fixed set of design parameters. When the time step is reduced to 0.2 ms the simulation fails to meet the real-time constraints for all the algorithms. Fig. 1b shows the effect of increasing the number of states with a fixed time step of size 1 ms. When the number of states are increased to 88, all of the integration algorithms fail to

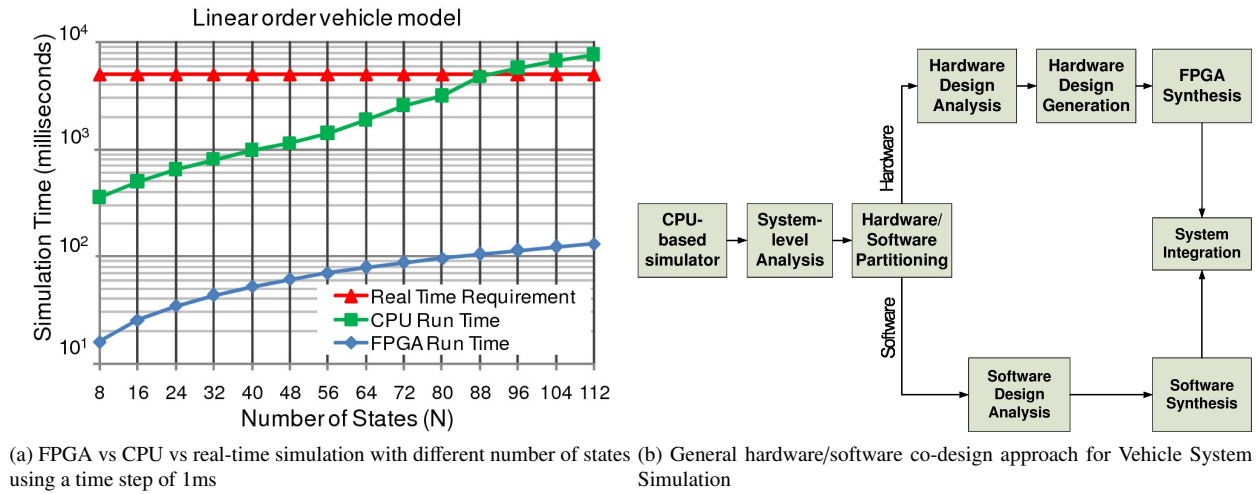


Figure 2: Motivation and Approach

meet the constraints as the overall computation time surpasses the real time of 5 s. These results drive our research into alternate platforms for simulating more complex vehicle dynamics in real-time.

In this work, we explore the reconfigurable capabilities of FPGAs for accelerating this class of algorithms and design a methodology to generate a reconfigurable architecture for different vehicle systems. FPGAs provide a platform to parallelize the independent computations and a custom pipelined architecture provides an opportunity to improve the throughput, though at the expense of an initial latency. More work per clock cycle thus results in a significant improvement in the computation time and the reconfiguration capability allows the platform to be used for RTS of different vehicle systems without having to develop a custom ASIC for the same. We aim to improve end-to-end computation time for vehicle system simulation. This is triggered when a system input is sent from the user-control to the simulation model and ends when hardware sends the simulation results back to the display monitor.

The vehicle system targeted for hardware implementation in this work consists of two subsystems. The first component is a steering valve subsystem which contains the dynamics of a hydraulic system which receives the steering input and relates that input to an output of the steering angle of the front wheels of the vehicle. The second component is the vehicle subsystem which relates the steering angle to the trajectory of the vehicle as it is propelled at a constant forward velocity. To simulate the system in real-time, an integration time step of 10 μ s is required for the valve subsystem simulation and a time step of 2 ms for the vehicle subsystem. A CPU-based (MATLAB) simulator was first used to simulate the whole vehicle system. However, it was observed that the computation time taken to run the simulation was 13 μ s per integration time step. Fig. 2a further describes the main motivation behind our use of FPGA technology to implement the RTS of the vehicle system. It compares the computation time of the vehicle system for a simulation period of five seconds on an FPGA running at 55 MHz and MATLAB on an Intel Core 2 Quad CPU running at 2.83 GHz using the RK4 integration method. The computation time increases with an increase in the number of states of the system for both the implementations. However, for the CPU-based simulator the computation time exceeds the real time when the number of states being modeled for the system is greater than 88. On the other hand, for the FPGA-based simulator, the computation time remains well below the real-time constraint. We can intuitively say that a more complex system with additional subsystems and forces will further add to the time taken per iteration and result in violation of constraint even with a lesser number of states.

We propose a hardware/software co-design approach to accelerate the RTS using a heterogeneous parallel architecture. Fig. 2b provides an overview of our approach. Using this approach we claim the following contributions to the state-of-art in simulation of vehicle system dynamics which otherwise fail to meet the real-time constraints using software (CPU-based) simulator:

- A co-design approach for RTS by partitioning the tasks between a hardware and a software platform.

- A methodology based on heuristic approach to generate an FPGA-based simulator. The approach uses a hardware component library which contains fast hardware implementations of non-linear functions and timing information of these components.
- Application of our methodology to generate the FPGA-based simulator for the vehicle system and various design strategies explored based on our methodology.
- Proof-of-concept of RTS using a simulator with both hardware and software components.

2. Design Methodology

In the system-level analysis of Fig. 2b we determine the partitions based on two factors: the computation time of different components of the simulation model, and the frequency of communication between different components. To efficiently utilize both the hardware and software resources, we obtain an initial partition such that the computation-intensive part of the simulation model and modules which can benefit the most by the parallel architecture are implemented on the hardware, with the rest to be implemented in software. If there is continuous exchange of data between the two partitions the increased communication delay between hardware and software will negatively affect the overall computation time. Based on the components selected for hardware and software implementation we first discuss the hardware partitioning followed by software partitioning.

2.1. Factors Affecting Hardware Partitioning

Hardware partitioning is governed by three factors: accuracy/precision in the simulation results, space occupied on the hardware, and the time required to complete the computations of a single time step. The accuracy/precision and hardware resource utilization (RU) are affected by the manner in which the data is represented on the hardware. For this work we use fixed-point representation [8] which consists of a fixed number of integer and fractional bits before and after the fixed-point. The accuracy is determined by the number of integer (I) bits available whereas the precision is governed by the number of fractional (F) bits selected for fixed-point representation. For hardware implementation, as we increase the number of bits to achieve better accuracy/precision in the simulation results, the space required increases.

The relation between time and space is based on the parallelism that can be explored in the FPGA-based simulator. If all the independent computations of the CPU-based design are implemented concurrently, then the resulting FPGA-based simulator would complete a single iteration in as minimal a time as possible. However, the parallelism comes at the expense of hardware resources. The serialized computation such that when one component completes the execution only then the next one is executed would be the slowest. In this case, the hardware RU will be equivalent to that of the single largest component. A pipelined implementation of the design will result in increased throughput and the hardware RU will be somewhere between that of the earlier two implementations.

The methodology to generate the FPGA-based simulator, based on the factors discussed above, is divided into three phases - hardware design analysis, hardware design generation and verification and software design analysis. Fig. 3 shows the heuristic approach for the hardware design analysis phase where we analyze the requirements i.e. the required bit combination, the time taken to complete a single iteration, and the hardware RU. The hardware design generation phase uses this information to generate the actual hardware design.

2.2. Hardware Design Analysis

The input to the hardware design analysis phase is the CPU-based simulator which uses the RK4 integrator, Permissible Relative Error (PRE) in the simulation output, the Real-Time Constraint (RTC), and the available hardware resources (AR). The input is based on the assumption that the platform is pre-decided with an aggressively parallelized design and maximum number of bits for fixed-point representation. Ideally, the PRE value should be set by the engineers who design the simulation model. They should be able to determine the acceptable relative error in the simulation results.

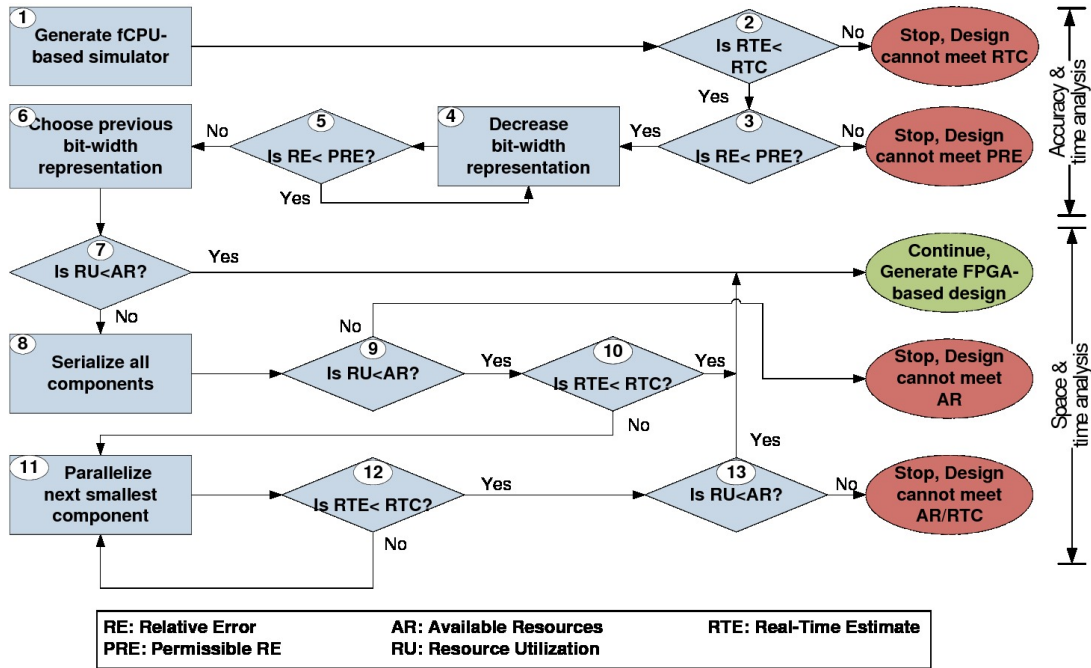


Figure 3: Heuristic approach for hardware partitioning

2.2.1. Accuracy/Precision and Time Analysis

Step 1: To implement the methodology described in Fig. 3 we need a model which can give us an estimate of the required bit combination, time taken to complete a single iteration and the hardware RU. These estimates can be obtained by having a model which can emulate the FPGA computation process and we call this a fixed-point CPU-based (fCPU-based) simulator. We first design a software component library which contains equivalent software (MATLAB) representation of all the components in the hardware component library. Each function is implemented using the same techniques that are used in the hardware design. The functions in the CPU-based simulator are then replaced with their modified implementation from the software component library. For example, instead of using the MATLAB built-in ode45 function for integration, we implement the algorithm for RK4 in MATLAB and use the same for the hardware implementation. In addition, the arithmetic operations in the modified implementation are also done using fixed-point notation and are parameterized for different bit combinations. Since the algorithms used are the same as those used for hardware implementation, the architecture is close to that of the FPGA-based simulator. The computation process also emulates the FPGA computation thus making the fCPU-based simulator an appropriate model to estimate the required bit combination that affects the accuracy/precision and the hardware RU.

Step 2: Before we actually generate the design, we estimate whether the hardware is capable of meeting the RTC even with the completely parallelized design. A parallelized design assumes that all the independent computations are implemented in parallel, optimizing for time. Thus, the time obtained from such a design is the estimate of the minimum possible time which the hardware will take to compute the output of a single iteration.

The fCPU-based simulator gives us the hardware components required for FPGA-based simulator. For each of the independent components in the hardware component library we use the cycle information to compute the number of cycles taken by the whole design to generate the output, taking into consideration the parallelism employed. Assuming different clock frequencies for the hardware, we can determine RTE of this design for these clock frequencies. If the RTE is more than the input RTC, the hardware will not be able to meet the RTC. This is because the RTE is being compared against the minimum possible time that an FPGA-based simulator will take by exploring all the parallelism in the model. If the time taken remains within the RTC, we check for the RE constraint in the next step.

Step 3: In Section 2.1 we discussed that the number of bits affect the accuracy/precision with which the data values

and computation results are represented on the hardware. To obtain an estimate of the required bit combination without generating the FPGA-based simulator, we emulate the hardware computation process in the fCPU-based simulator. This is achieved by converting all the data values at each computation step in the fixed-point format of length I+F bits. The converted values are equal or close to the true values if the bits are sufficient. The local truncation error due to each conversion and global propagation error due to previous conversions thus results in an error in the final simulation output after every iteration. Since the fCPU-based simulator is generated using algorithms used for the FPGA-based simulator, the conversion accurately models the hardware computation process and the error generated from this process can be considered as a close estimate of the RE that will be generated from the FPGA-based simulator. For a fixed range of PRE and given the maximum number of bits for representation if RE of the design fails to meet the PRE constraint we cannot proceed to the next step.

Step 4, 5 and 6: If the constraints are met, we further optimize the design by reducing the bit-width combination such that the RE remains within the PRE. We first reduce the number of I bits while keeping F=64. After obtaining the number of sufficient I bits we reduce the number of F bits until it fails the constraint. However, at this point we would like to mention that the process of estimating the bit-combination using fCPU based simulator is highly dependent on the number of iterations we run the simulation for. As we increase the number of iterations, the number of sufficient bits that satisfy the PRE criteria may increase. So, the selected bit-width combination may not be the final estimate that would represent the values close to the required values on the hardware.

2.2.2. Space and Time Analysis

Initially the timing analysis is performed assuming an aggressively parallelized model, which if implemented on hardware would utilize the maximum resources available. If the optimized design meets the RTC, it can then be optimized for space to determine if the model meets the AR constraints.

On the hardware, as the RU increases, the area covered by the design increases and so does the path traversed by the clock. This in turn lowers the overall frequency at which the design can run. During space analysis, we thus optimize the design for space by serializing or pipelining the components. However, optimizing for space in turn increases the time taken to run a single iteration.

Before we proceed to the next step, we present our approach to estimate the hardware RU of the design. The components present in the hardware component library are independent entities that can be plugged into any design as long as the input and output ports are correctly mapped. We ran the hardware synthesis for all the components in the library and obtained their hardware RU for different bit combination. Due to space constraints we skip the graphs for resource utilization. The synthesis was run on Altera's Stratix III board so the RU is in terms of Altera's Adaptive Logic Modules (ALMs). An equivalent number of 6-input LUTs on Virtex-5 FPGAs of Xilinx can be obtained using the relation given in [9].

Step 7: For space analysis, we first check if with the selected bit-width combination from the previous step, the design meets the AR constraint. We use the fCPU-based simulator to determine the components that make up the FPGA-based simulator and use the graphs for resource utilization (for some example components) to determine their RU for the selected combination. We compare the RU for the whole design with the AR input for the selected platform. If the constraint is met, we use the automated scripts to generate the VHDL-design for the selected components with the selected bit-width combination. If it does not, we perform the space optimization and start with a completely serialized design in Step 8.

Steps 8, 9 and 10: In Step 1 we started with an aggressively parallelized design which was optimized for time to check for the RTC and obtain an estimate of the speed-up that can be achieved. To optimize for space we serialize/pipeline the components such that new computation starts either after completion of the previous computation or a cycle delayed. This process reduces the RU since the number of computations being done in parallel has been reduced. However, to obtain a lower limit on the hardware RU of the design, we start with a design that is completely serialized for which we again check whether with the selected bit-width combination, the design that has been optimized completely for space is able to meet the AR constraint (Step 9). Since we started with a serialized design, the estimate of RU is the minimum resources that a design is expected to consume and if the constraint fails, it is not possible to proceed further. If the constraint is met we then check if the new design meets the RTC (Step 10). As mentioned earlier, the serialization affects the RTE and if the completely serialized design meets RTC we use automated scripts to generate the VHDL design for the selected components with the selected bit-width combination. If the RTC is not met, we still have the option to parallelize components in Step 11.

Step 11, 12 and 13: In Step 8, we made an assumption of completely serialized design and obtained the minimum resources that a design would consume. Since the RTC is not met, while still optimizing for space, we parallelize the component which consumes the minimum resources and thus results in minimum increase in the overall RU. We check for RTC in Step 12 and iterate through Step 11 and Step 12 until we meet the RTC. As discussed earlier as we parallelize components RTE reduces but the RU increases. So having met the RTC, we check for the RU constraint. If the constraint still fails, we cannot proceed further to generate the FPGA-based simulator.

To meet the real-time constraints the simulation results should be available at the host, for further processing, within or even less than the RTC. This is necessary because the time taken to complete an iteration includes the computation as well as communication delay. So the RTE is actually computation and communication delay where the latter varies with the amount of data being transferred. We thus need an interface, which provides sufficient bandwidth and minimizes the latency in sending the data back and forth between the host and the hardware. Since the hardware/software partitions have already been decided, we know the amount of information that needs to be exchanged between the two partitions. We use this information to determine the required bandwidth of the interface and compare it with the bandwidth of the selected platform.

2.3. Hardware Design Generation and Verification

An important aspect of the design methodology is automatic generation of the design models based on different design decisions. The design decisions in this case include the selection of appropriate bit combination that meets the accuracy, time and space criteria. The advantage of having this automation is that this allows the designer to focus on making the best design decisions without having to devote much time in creating the designs every time a change is required.

2.3.1. Design Generation

The VHDL design for each component is highly parameterized and pipelined. The parameters for each component, and those specific to vehicle system simulation, are saved as constants in the parameters file in fixed-point format based on the bit combination. To use these constants, the components need to include the parameters file while implementing the design. However, the selection of appropriate bit combination is an iterative process during which the parameters and the VHDL design have to be regenerated. For a complex system, with numerous components this step would require the designer to create the design for all the components every time the bit combination changes. Thus, to automate the process of design generation, we designed the MATLAB scripts which take the bit combination and order of the system (if required) as input to generate the parameters file and VHDL design for components based on serialization or pipelining involved.

2.3.2. Design Verification

In this step we compare simulation output from Modelsim, with that from the CPU-based simulator. Since there are different components connected together it is essential to validate that data from these components is represented correctly. Assuming the design meets the functionality criteria, an insufficient number of bits may result in a mismatch of the final simulation output if the output from any component is incorrect. After analyzing whether the mismatch is due to insufficient number of I or F bits, we increase the bits accordingly and go back to Step 7 of the hardware design analysis phase. In addition to data validation, simulation is an important phase to check the speedup that might be expected from the present implementation. Once the Modelsim simulation shows a perfect match with the results from CPU-based simulator we generate the programming file and integrate it with the software design to run RTS.

2.4. Software Design Analysis

The software design analysis phase is based on the platform selected because the width of the interface governs the alignment and data format in which the system input should be sent to the hardware and the simulation output should be received back from the hardware. With focus on vehicle system simulation, the software design should be able to perform the following tasks for the complete HIL RTS.

- Receive the system input from the HIL
- Send the system input to the hardware

- Receive the simulation output from the hardware
- Convert the hexadecimal format of the output to the decimal format
- Perform software computation if any
- Send the simulation output to the VR display

If the system input is assumed to change every T ms, the software should be able to perform the above tasks which includes the network and communication delay and computation time on the hardware and software within this real-time. The fast computations on the hardware can cause the simulation to run faster than the real-time. Thus, to emulate the real-time scenario we start the timer in the software just before it receives the system input. The hardware runs the simulation for T ms and sends the output back to the software and stalls until it receives the new system input. On the software side, timer stops after sending the simulation output for further processing. At this point if the difference between stop and start timer is less than T ms, we invoke a sleep command to stall the software for the remaining amount of time i.e. $T - (\text{stop} - \text{start})$.

As the complexity of the physical system being simulated increases, the amount of work load for either hardware or software partition also increases. Considering the RTC, it thus becomes essential to develop an efficient software design that minimizes the time spent between the start and stop timers, apart from the hardware computation involved.

3. Hardware Implementation

We applied the methodology discussed in Section 2 to generate an FPGA-based simulator for an 8th order steering valve subsystem of a vehicle. The steering valve dynamics, described in detail in [10], simulates the dynamics between the rotation of the steering wheel and the rotational motion of the front steer wheel about the king pin. The dynamics of this system are quite stiff due to small time constants associated with internal volumes in the steering valve. Thus with RK4, the system is numerically unstable for integration steps larger than time step h_{valve} of $10 \mu s$. The vehicle subsystem dynamics[7] that describe the vehicle motion based on steering angle inputs can be simulated with a time step $h_{vehicle}$, of the order of few milliseconds. With such a small time step requirement for the steering valve model, when the vehicle system simulation is implemented on MATLAB, simulation output failed to meet the real-time constraints. The vehicle subsystem, however, when simulated by itself, met the real-time constraints.

Fig. 4a shows the architecture of the vehicle system. However, we skip the details of these models due to space constraints. The parallelism in the computations involved in RK4 make it an ideal candidate for FPGA implementation whereas the method to compute the position co-ordinates involves relatively simpler execution and can be implemented in software. We implement both the computationally-intensive models, which can be efficiently parallelized and pipelined, in hardware while keeping the computation of the position coordinates in software.

For hardware implementation of the FPGA-based simulator we chose XtremeData's XD2000i development system [11]. The system consists of a development PC with a Xeon dual-processor system, running the Linux CentOS operating system, and an XD2000i FPGA in-socket accelerator that plugs directly into one of the CPU sockets. The XD2000i module features three Stratix III EP3SE260 Altera FPGAs, one bridge and two application (FPGA A and FPGA B), each with 254,400 logic elements (101,760 ALMs), a 1067M front-side bus (FSB) interface that provides a bandwidth of 8.5 GB/s, two QDRII+ 350 MHz SRAM each of 8 MB, connected with two application FPGAs through an interface that provides a bandwidth of 2.8GB/s. The bridge FPGA is dedicated to implementing the FSB protocol that connects the bridge FPGA to the Northbridge on one side and to the two application FPGAs on the other side. It is not modifiable by the user and only the application FPGAs are used for implementation. The bus connecting the bridge and the two application FPGAs is a 64-bit wide unidirectional bus running at 200 MHz. The data between the two application FPGAs is transferred through a 256-bit wide bus running at 100 MHz.

The software partition runs on the Xeon processor and implements the software controller that uses blocking send and receive functions to communicate with the application FPGAs. The blocking functionality implies that for every send command to the FPGA, the next send cannot be executed until the response to the previous send has been received. An additional requirement for this communication process is that, before the controller can initiate the transfer it should have the information about the number of bits that should be sent and the number of bits that are expected from the FPGA. It then prepares a send and a receive buffer of the required size. Any mismatch between the size of the buffer and the number of bits sent or received will cause the controller and the hardware to stall.

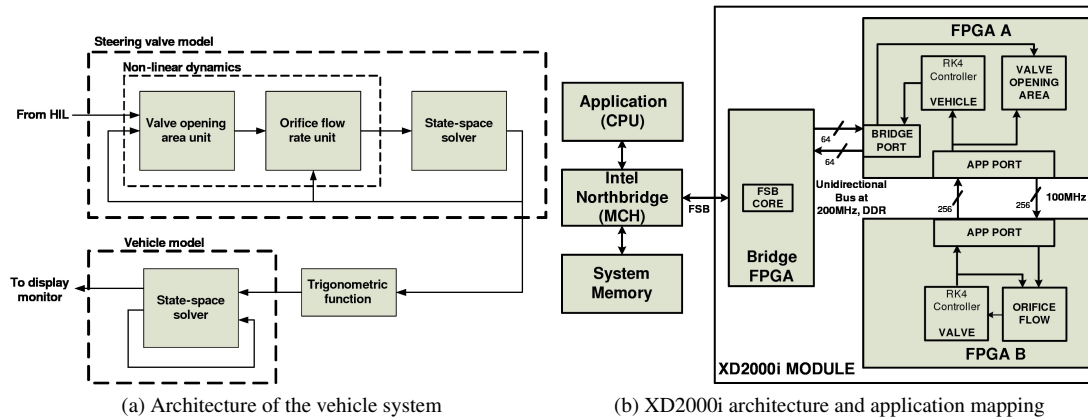


Figure 4: Vehicle system and XD2000i Architecture

Fig. 4b shows the partitioning of the design across the two application FPGAs. The partitioning algorithm is governed by two factors. First, implementation on the XtremeData platform requires that the same application port is used to exchange data between the application FPGA and the software controller. Hence, the valve opening area component, which receives the system input from the software controller, and the state space solver of vehicle model, which generates the simulation output i.e. the new state of the vehicle, are implemented on the same FPGA. Second, the application of the methodology to the vehicle system, the fixed-point representation for which the relative error reaches an error value close to the given range is $I=49$, $F=47$ for the steering valve model and $I=10$, $F=46$ for the vehicle subsystem. For this fixed-point representation the percentage resource utilization for RK4 valve is 38%, RK4 vehicle is 29%, Valve opening areas is 11%, Orifice flow rate is 50% and Trigonometric is 5%. Resource utilization of FPGA A is thus estimated to be $29+11+5=45\%$ and has the capacity to accommodate more components. However, if the orifice flow (which uses the square root core) and state space solver for the steering valve model are implemented on separate FPGAs, there will be a communication delay involved in sending data of eight states after every iteration from one FPGA to the other. To counter this problem and also to efficiently utilize resources of both the FPGAs, we implemented the orifice flow rate and state-space solver on FPGA B with an estimated resource utilization of $38+50=88\%$. Recall that the state of the valve model which is used to compute the system input for the vehicle model represents the piston displacement. To obtain the equivalent angular displacement, an inverse sin function is applied before the input is fed to the vehicle model.

To run the RTS of the vehicle system using the FPGA-based simulator h_{valve} is set to 10^{-6} s and $h_{vehicle}$ is set to 2^{-3} s. The software controller sends the steering wheel angle, A_s to FPGA A every 20ms. This includes the communication delay over FSB to send the system input and receive the simulation output and also the time required to compute new state of the vehicle. After sending the output, the simulator stalls until it receives a new system input from the controller. Once the controller receives the simulation output it stalls until 20ms have completed before it can send a new system input.

4. Simulation and Synthesis Results

The maximum hardware clock frequency supported by the XtremeData XD2000i platform is 100 MHz. However the design was only able to meet the timing constraints at clock frequency of 55 MHz (18.18ns time period). We first compared the time taken by a single iteration on FPGA, using the Modelsim simulator, with the MATLAB-based model running on an Intel Core2 Quad 2.83 GHz processor. The data exchange between the two FPGAs uses FIFOs. The only instance when the data is written by FPGA A to its exit FIFO, is when it has to send 512 bits containing the four valve opening area results to FPGA B. It is important to note here that FPGA A does not write continuously to this FIFO, and by the time the second set of 512 bits is written, the first one has already been read, so we do not have scenario where the FIFO will become full. Similarly, FPGA B writes to its exit FIFO when it has to send 256 bits containing 2 states of the valve model to FPGA A. The inflow to the FIFO is 256 bits per iteration and by the time

the next 256 bits are written, the first packet has already been consumed. We thus assume that the delay associated in sending data across the FPGA is closely approximated in Modelsim simulation to the actual delay on the hardware.

The MATLAB-based model takes 13 μs to complete one iteration of the steering valve and vehicle model. The Modelsim simulation shows that one iteration of the steering valve model takes 4.1 μs in hardware, which includes a delay of 0.27 μs (15 cycles) required to send 512 bits of data from FPGA A to FPGA B. A further delay of 0.234 μs (13 cycles) is observed, while sending the 256 bit output of the valve model to FPGA B. The vehicle model generates an output in 2.214 μs (123 cycles). Apart from the RK4 component for the vehicle model which takes 113 cycles, 7 cycles are consumed by the trigonometric function, 1 cycle is used to obtain the input for the trigonometric function by division of linear piston displacement with the length of the arm, and 1 cycle is consumed to complement the output of the trigonometric function. Thus, the total time estimated from Modelsim to compute a single iteration of the steering valve and each iteration of the vehicle model is $4.1+0.27+0.234+2.214=6.818 \mu\text{s}$. This comparison does show a speedup of $2\times$ over the MATLAB implementation for a single iteration. However, the actual simulation allows the steering valve model to run for 20ms, followed by vehicle model simulation for 20ms. In Modelsim, the time required to generate the final output of the vehicle system simulation for 20ms is computed as 9.21ms which shows a speedup of $2\times$ over the required time of 20ms. The time to send the data over the FSB is negligible as compared to the computation time for this model.

5. Conclusions

In this paper, we introduced a method to improve the simulation time of vehicle systems, to meet the real-time constraints using hardware based implementation of the mathematical models. We presented the methodology adopted to implement these models, for different sized models and estimated the resource usage of the hardware design beforehand to make intelligent decisions about the implementation strategy. We applied our methodology to an 8th order steering valve and vehicle model. The system was successfully implemented on a high-performance reconfigurable computing platform with a speedup of $2\times$ for the overall simulation process. During the process, we designed hardware components that can be further used for implementation of other models. This work forms the basis for the next step of research in this direction which will focus on developing partitioning algorithms to provide different architectures for implementing the models across multiple hardware platforms along with the software integration. While in this paper we considered the implementation across two FPGAs, the work can be formalized to consider any number of hardware platforms in order to investigate more efficient hardware and hardware/software partitions.

References

- [1] S. Zheng, S. Zheng, J. He, J. Han, An optimized distributed real-time simulation framework for high fidelity flight simulator research, in: Proceedings of International Conference on Information and Automation (ICIA), 2009, pp. 1597–1601. doi:10.1109/ICINFA.2009.5205172.
- [2] P. Le-Huy, S. Guerette, L. Dessaint, H. Le-Huy, Dual-step real-time simulation of power electronic converters using an FPGA, in: Proceedings of IEEE International Symposium on Industrial Electronics, 2006, pp. 1548–1553.
- [3] X. Xiaobo, Z. Kangfeng, Y. Yixian, X. Guoai, A model for real-time simulation of large-scale networks based on network processor, in: Proceedings of the Broadband Network Multimedia Technology (IC-BNMT), 2009, pp. 237–241. doi:10.1109/ICBNMT.2009.5348486.
- [4] M. Tavernini, B. Niemoeller, P. Krein, Real-time low-level simulation of hybrid vehicle systems for hardware-in-the-loop applications, in: Proceedings of Vehicle Power and Propulsion Conference (VPPC), 2009, pp. 890–895. doi:10.1109/VPPC.2009.5289753.
- [5] J. Maroto, E. Delso, J. Felez, J. Cabanellas, Real-time traffic simulation with a microscopic model, IEEE Transactions on Intelligent Transportation Systems 7 (4) (2006) 513–527. doi:10.1109/TITS.2006.883937.
- [6] M. Lerotic, S.-L. Lee, J. Keegan, G.-Z. Yang, Image constrained finite element modelling for real-time surgical simulation and guidance, in: Proceedings of Biomedical Imaging: From Nano to Macro, 2009, pp. 1063–1066. doi:10.1109/ISBI.2009.5193239.
- [7] M. Karkee, B. L. Steward, Open and closed loop system characteristics of a tractor and an implement dynamic model, in: American Society of Agricultural and Biological Engineers (ASABE), 2008.
- [8] R. Yates, Fixed-point arithmetic: An introduction, <http://www.digitalsignallabs.com/fp.pdf> (2007).
- [9] A. Percy, Advantages of the Virtex-5 FPGA 6-Input LUT Architecture, http://www.xilinx.com/support/documentation/white_papers/wp284.pdf (December 2007).
- [10] M. Karkee, M. Monga, B. Steward, J. Zambreno, A. Kelkar, Real-time simulation and visualization architecture with field programmable gate array (FPGA) simulator, in: Proceedings of the ASME 2010 World Conference on Innovative Virtual Reality, 2010, pp. 8–13.
- [11] XtremeData, Xd2000iTM development system, <http://www.xtremedata.com/products/accelerators/in-socket-accelerator/xd2000i>.