Application Note:

Modeling and Exploring Software Applications using VisualSim

Current generation products contain a number of SoC and configurable Platforms. These products can be easily configured to be used for a wide application space and large profiles of customers. These platforms must be at lowest cost, meet required performance targets and generate minimal power consumption. Information about the platform including functionality and performance must be provided to sales engineers to help in design wins. The competitive marketplace requires companies to differentiate on product innovation through the use of early design exploration tools.

These requirements can be addressed through performance analysis and architecture exploration early in the design cycle and prior to scheduling implementation. VisualSim is the first industry solution to focus on "Idea to Specification" part of the design solution. Mirabilis Design customers have been able to reduce product cost and widen the application of existing products. The average product schedule has been reduced by around 30% resulting in millions of dollars of additional revenue sooner with greater profit margins. The major areas of focus for Platform-based design will be:

1. New Design: The design of the new platform requires the architecture to be explored in terms of lower price, highest performance and minimal power consumption. If early assumptions prove incorrect during implementation, there will be significant delays in project schedule. For example, a dual ARM-9 design is constructed and the bus is too slow or the memory hierarchy needs to be modified, this will cause considerable slowdown in the video delivery. A recent customer design compared a dual ARM-7 vs. to an ARM-8 Cortex processor topology. The initial impression was that the dual ARM-7 would be too slow. At the end of a one week analysis using VisualSim, the customer determined that dual ARM-7 was able to attain the required performance at 1/10 of the power consumed by the ARM-8 architecture. This saved this European company considerable cost, both in die space and cooling/thermal expense.
2. Mapping a platform to an Application: Every customer has a unique set of requirements. The sales engineer must quickly select the right platform based on the requirements. The engineer can do a simple analysis on a spreadsheet or write extensive C code to create the new application. Spreadsheets will be highly inaccurate and C-code will take 3 months for even a simple project. A good example is a VisualSim customer that was using the cell processor platform for a DSP application. The application required movement of large frames of data. The proposed memory interface was too slow and had to be replaced with a high-speed serial interface.
3. Adding new applications to an existing platform: System companies would like to add new applications using software on an existing hardware platform. When a new application is proposed, they will quickly evaluate if the current platform will support the added loading. Once they have decided that the application can be written entirely in software or identified which parts of the application might use a hardware accelerator, they can start the implementation flow. A VisualSim customer developed an entire RADAR tracking system in software. When this software was added to an existing PowerPC-based platform, they found the performance was inadequate. They had to either partition the new application into hardware acceleration or move to a higher performance platform. This delayed the project by 6 months and cost over $6 million in added costs.

Early design exploration for performance, power and functionality provides a validation of a new innovative design concept. This can be used to validate the specification. Also the same model in VisualSim can be used for performance exploration of derivative designs. These VisualSim models can be used to get early design feedback from customers before scheduling implementation. This also becomes the proof of concept for new design wins.

Modeling Overview and Abstraction

We are modeling an automobile system that consists of a single ECU platform with a microprocessor and DSP, and a network connecting all the devices and sensors. On this platform, we will be experimenting with different sets of applications. The purpose of this modeling is to conduct architecture exploration to select the right architecture platform and to optimize the software operation for real-time performance. Figure 1 shows the block diagram of the system with the starting architecture and a proposed distribution of tasks.

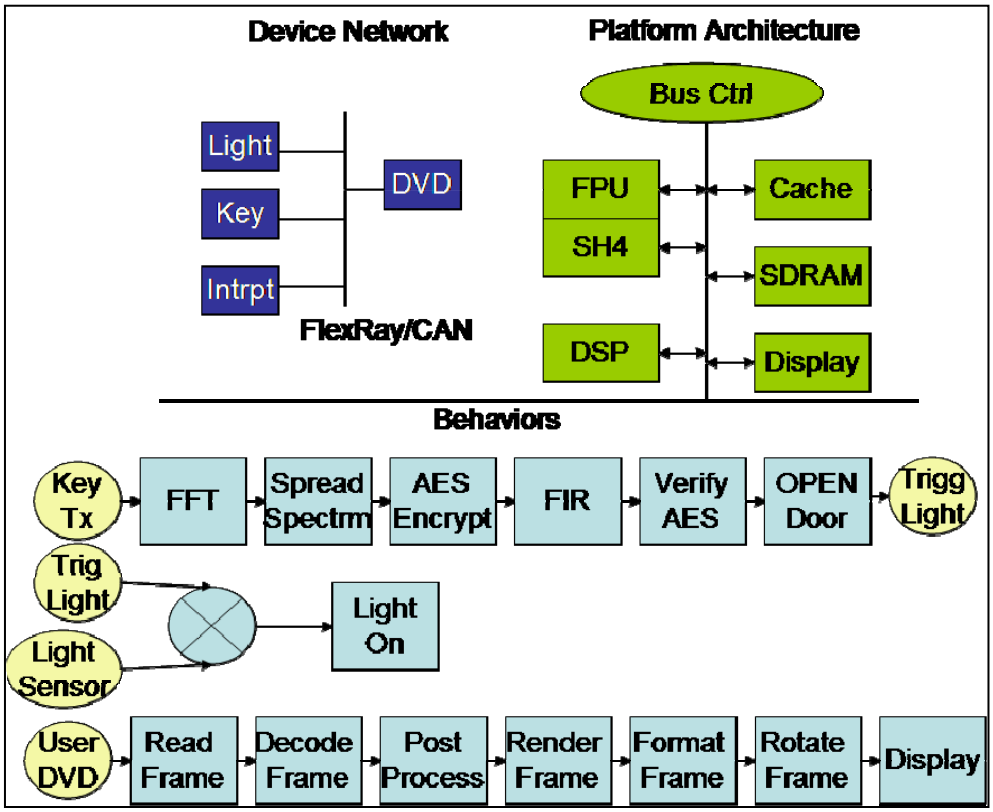

Figure 1 Block Diagram of the Application, Platform and Device Network

Figure 2 shows a model of an automotive application modeled in VisualSim. This model shows a detailed behavior and user activity.

The model is built in stages. First the software behavior, user activity and Statistics are built up similar to a UML diagram procedure. The details of the behavior were initially described in relative events. The behaviors are triggered by a User Activity diagram that is separate from the behavior flows. The model has multiple software behaviors. This is simulated as a pure functional execution to make sure that all the connections and the flow are correct. Also, a check is done for the right outputs and the correct branching.

A middle layer could be used to describe the communication between the software modules. This could be a CORBA, RMI, Datagram or a direct connection. For the sake of simplicity, this has not been included in the model. There are blocks in VisualSim that support Serial I/O, Datagram and CORBA.

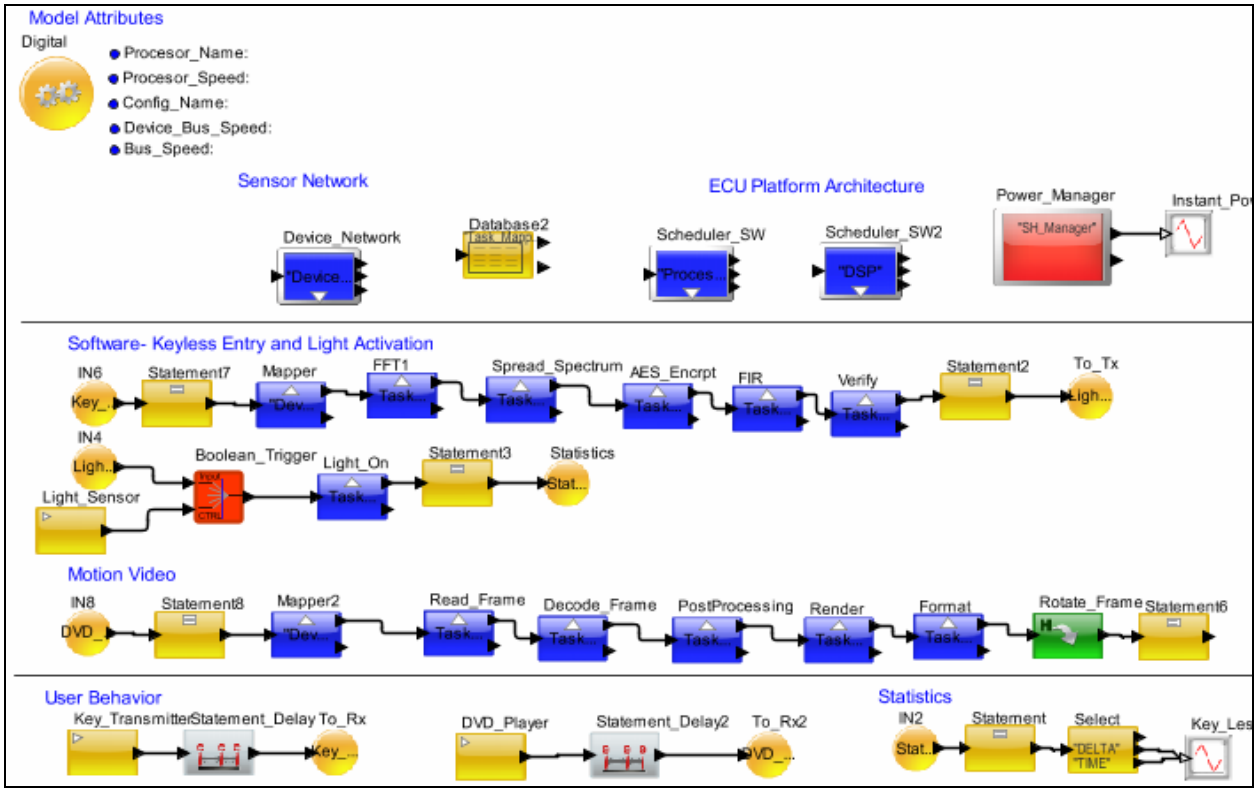

Figure 2 Abstract or Transaction-Level Model of the Automotive Application

The Transaction-Level (TLM) definition of the architecture is done using schedulers. A Scheduler is used to represent the processing resources at this level of abstraction. The ECU platform architecture consists of two devices- one scheduler to represent the Processor and another for the DSP. The entire device network is abstracted using a single Scheduler. There are two power states for each scheduler - Active and Standby. The power is analyzed for this model in parallel with the performance and functionality.

The cycle-accurate and instruction model is shown in figure 3. This model was constructed by modifying the TLM or abstract model. There are two items that are different with the Transaction-Level model- the mapping table content and the architecture setup.

Based on the sizing information provided by the TLM model, the processor and the peripheral devices are selected. The ECU is now assembled with the refined description of the architecture. The architecture in the transaction level modeled is now refined to cycle-accurate/instruction-accurate. This uses the detailed SH4 processor model that was generated using the Processor Generator Toolkit.

The entire model took 2 days to construct.

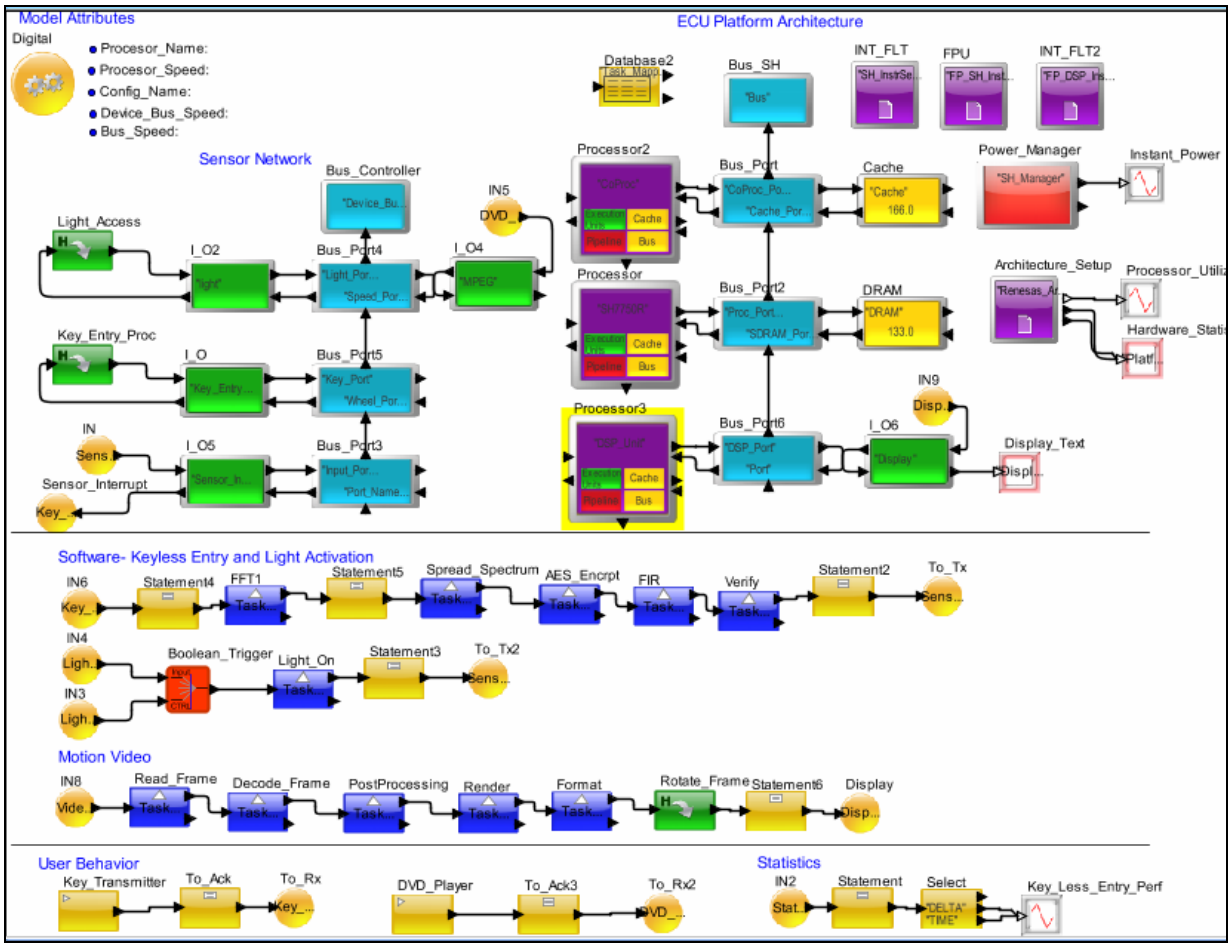

Figure 3 Automotive Model using SH4 processor Family

Application

Automotive: DVD Player, Keyless Entry and Interior Lighting

Hardware

SH7750R (SH7750 Series Hardware Manual and Renesas Web Site – Power diagram)
Generic DSP, Shared CPU Bus, External Cache, External DRAM
Sensors (Key, Light, DVD), Device Network
Interrupt Handler

Software

Motion Video task order: Read->Decode->Post Process->Render->Format->Rotate->Display
Keyless Entry task order: FFT->Spread Spectrum-> AES Encrypt-> FIR->Verify
Light task order: (Triggered by Keyless Entry and the Light Sensor) Activate

User Action

Activate Key Transmitter (Random events)
Switch DVD Player on (Single Event)

Statistics

Latency (From Key Transmitter to Door OPEN and Light turned on)
Summary: Device Bus, CPU Bus, SH7750R, Cache, Memory, DSP
Power: SH7750R only

Explorations

The VisualSim model will be used to determine the optimal architecture for the current set of applications and user behavior. A number of characteristics of the model have been parameterized- Processor and bus Speed, Cache size and bus width. This allows for easy exploration. The following model attributes can be modified to perform exploration:

1. Architecture connectivity.
    a. Place the cache on a separate bus connected by a bridge.
    b. Move the DSP to a separate bus with its own dedicated cache.
    c. Add a DMA for doing cache access as opposed to a direct memory access.
    d. Add another ECU. This is done by Copy-Paste the existing ECU and modifying the name. This would replicate the entire processing architecture.
    e. Extend the TLM model by adding a cache definition
2. Device network Architecture
    a. Replace the shared linear bus with a FlexRay network
    b. Combine a CAN and a FlexRay with a gateway between them
3. Software Exploration
    a. Transforming a mathematical algorithm into a set of function or task for concurrent operation, increased parallelism or pipelining
    b. Optimizing loops based on processor stalls (for data retrieval) and the time of execution for each task or loop
    c. Determine the statistics for execution time- minimum, maximum and standard deviation
    d. Determine the impact of function and data dependency on execution time
4. Hardware-Software Partitioning
    a. Modifying the mapping table for different processors, instruction sequence or processing time; and running simulation can provide valuable data on parallelism, concurrency, effect of priority, preemption at the RTOS and the efficiency of different processors.
    b. Move some of the Motion Video tasks from the DSP to the SH4 processor.
    c. Vary the rate of user operations for the key transmitter.
    d. Make the Motion Video to loop through more multiple frames. This is done by simply changing the Transaction_Source for the DVD user behavior from Single_Event to Uniform_Distribution.
    e. Perform the load balance across multiple schedulers in the TLM model
    f. Load balance all the tasks between two ECUs. To load balance simply use the function in the Expression Language called “getNextResource”. This determines the processor with the least task activity and maps the task to this processor.
5. Component selection
    a. The SH4 can either have a single internal data cache or a data cache + scratchpad RAM. Add another cache to the SH4 processor and the FPU blocks. Make the Cache_Miss for the D-Cache to point to this RAM. The I_Cache Miss will point to the external Cache.
    b. Replace SH4 with ARM9 to compare performance difference between the two architectures. This requires a change in the Mapping table, change the names of the instructions in the execution sequence and a new instruction set.
    c. The abstract TLM model can be extended to include the effect of cache access. This is done by connecting a Mapper block to the output of the Scheduler and referencing another Scheduler to represent the memory.

Model Details:

For a detailed discussion on creating an architecture model in VisualSim, review the Mirabilis Design Application Note: *Designing Processor Platform Using the Architecture Modeling Toolkit*. This document will focus on the modeling of a select components, software and user behavior. For more details on each generator or the available statistics, refer to the *Architecture Toolkit* documentation.

Processor:

The Processor Generator block of VisualSim was used to describe the generic DSP and the SH7750R processor. The Floating Point Unit (FPU) in the SH4 was made a separate Processor block that was connected to the Pipeline of the SH4 processor block. The processor was defined using the following information:

1. List of Execution Units by processor

Processor

```
SH4_List  IU  BPU VPU    ; /* All EU listed. Branch and Load-Store Combined */
IU   INT_1                    ; /* Integer Unit */
BPU  INT_2                  ; /* Branch and Load-Store Unit */
VPU  FP_1                   ; /* Floating-Point Unit; Used for routing only */
```

Floating-Point Unit

```
VPU  FP_1             ; /* Single EU.  Same as above */
```

DSP

```
DSP  FP_1             ; /* One EU in the DSP. Assumes sequential execution */
```

2. Instruction set separated by processor and by Execution Unit

Floating Point and DSP alone shown here. Notice the range of cycles for some tasks. This information is used when modeling the number of cycles consumed.

SH4 FP_1

```
begin FP_1              ;  /* Group */
  FADD 3 4          ;  /* Range indicates multiple cycles */
  FCMP 2 4          ;
  FDIV 12 13        ;
  FLOAT 3  4        ;
  FMAC 3  4         ;
  FMUL 3  4         ;
  FSQRT 11 12       ;
  FSUB 3 4          ;
  FTRC 3 4          ;
  DFADD 7 9         ;
  DFCMP 3 5         ;
  DFCNVDS 4 5       ;
  DFCNVSD 3 5       ;
  DFDIV 24 26       ;
  DFLOAT 3 5        ;
  DFMUL 7 9         ;
  DFSQRT 23 25      ;
  DFSUB 7 9         ;
  DFTRC 4 5         ;
  FTRV 7            ;
  GFMOV 1 2         ;
  GFIPR 4 5         ;
  GFRCHG 1 4        ;
  GFTRV 5 8         ;
```

```
        end   FP_1              ;
DSP FP_1
        begin FP_1              ;  /* Group */
          FFT    2000     ;
          DECODE 1500                    ;
          ROTATE 600            ;
          RENDER 250             ;
          PP     425                ;
          FORMAT 225              ;
        end   FP_1              ;
```

3. 4-Stage Pipeline for the SH4 with a call to another pipeline for the FPU

```
/* First row contains Column Names.            */
Stage_Name      Execution_Location      Action  Condition ;
1_PREFETCH      I_1                     instr   none      ;
1_PREFETCH      D_1                     read    none      ;
2_DECODE        I_1                     wait    none      ;
2_REG_READ      D_1                     wait    none      ;
3_EXECUTE       SH4_List                exec    none      ;
3_EXECUTE       CoProc                  task    VPU       ; /* Call to External FPU */
4_MA            SH4_List                wait    none      ;
4_MA            CoProc                  wait    none      ;
5_WB            D_1                     write   none      ;
```

4. FPU pipeline

```
/* First row contains Column Names.            */
Stage_Name   Execution_Location   Action   Condition ;
1_EXECUTE       VPU               exec     none      ;  /* From SH4 */
2_MA            VPU               wait     none      ;  /* To SH4 */
```

5. Separate DSP modeled with its own pipeline, resources and instruction set. The DSP was modeled as a generic component that requires sizing.

```
/* First row contains Column Names.            */
Stage_Name      Execution_Location      Action  Condition ;
1_PREFETCH      D_1                     read    none      ;
2_REG_READ      D_1                     wait    none      ;
3_EXECUTE       DSP                     exec    none      ;
4_MA            DSP                     wait    none      ;
```

6. Resources and Speed definition

SH4:

```
    /* First row contains Column Names.            */
    Parameter_Name              Parameter_Value    ;
    Processor_Instruction_Set:      SH_InstrSet       ;
    Number_of_Registers:          16              ;
    Processor_Speed_Mhz:           240.0             ;
    Context_Switch_Cycles:         100             ;  /* This is a hard to find value and was
                                                         calculated using the Cache response
                                                         value */
    Instruction_Queue_Length:       6              ;
    Number_of_Pipeline_Stages:      5                ;
    Number_of_INT_Execution_Units:  1                  ;
    Number_of_FP_Execution_Units:   0                  ;
```

DSP

```
/* First row contains Column Names.                */
Parameter_Name                  Parameter_Value    ;
Processor_Instruction_Set:      FP_DSP_InstrSet    ;
Number_of_Registers:            2                  ;
Processor_Speed_Mhz:            200.0              ;
Context_Switch_Cycles:          100                ;
Instruction_Queue_Length:       3                  ;
Number_of_Pipeline_Stages:      4                  ;
Number_of_INT_Execution_Units:  0                  ;
Number_of_FP_Execution_Units:   1                  ;
```

7. Cache Definition in the processor

SH4

```
Number_of_Cache_Execution_Units: 2              ;   /* Separate cache for Instr and Data */
I_1:  {Cache_Speed_Mhz=240.0, Size_KBytes=8.0, Words_per_Cache_Line=16,
Cache_Miss_Name=Cache}
D_1: {Cache_Speed_Mhz=240.0, Size_KBytes=16.0, Words_per_Cache_Line=8,
Cache_Miss_Name=Cache}
```

DSP

```
Number_of_Cache_Execution_Units: 1      /* 1 internal cache */          ;
D_1:          {Cache_Speed_Mhz=200.0, Size_KBytes=16.0, Words_per_Cache_Line=32,
Cache_Miss_Name=DRAM} /* Next level of memory is SDRAM and not external Cache */
```

Architecture

The architecture was split into two parts- the Electronic Platform Architecture and the Device Network. The device network was abstracted to a simpler shared bus network. The devices are connected to it. The external remote transmitters trigger the devices, which causes transfers on the Bus. Each sensor arbitrates on the Bus for bandwidth. The outputs from all the Sensors are sent to the Sensor Interrupt port (Bottom-Left) on the Bus. This is a hardware trigger that causes a software behavior to fire. This triggers an input Virtual Flow in the Behavior section. This is identical to a hardware interrupt causing a sequence of instructions to execute. In a real system this bus network would have been a more complex structure such as a CAN or a FlexRay network. There are standard VisualSim library elements to define these networks. The Platform Architecture consists of a shared bus with a SH4, DSP, Cache, DRAM and a Display I/O connected to it. The Cache and DRAM are defined by modifying the parameters of the generator in the VisualSim library.

Routing

Each architecture element could have multiple connections to the Bus and to communicate with network resources. A Routing Table entry is provided to define the specific path from each source to destination. This table needs to be updated only for those communications that will occur. All other can be ignored.

```
/* First row contains Column Names.                */
Source_Node     Destination_Node     Hop              Source_Port ;
CoProc          Cache                CoProc_Port      bus_out    ;
CoProc          DRAM                 CoProc_Port      bus_out    ;
SH7750R         Cache                Proc_Port        bus_out    ;
SH7750R         DRAM                 Proc_Port        bus_out    ;
DSP_Unit        DRAM                 DSP_Port         bus_out    ;
```

Cache	CoProc	Cache_Port	output ;
Cache	SH7750R	Cache_Port	output ;
DRAM	CoProc	SDRAM_Port	output ;
DRAM	SH7750R	SDRAM_Port	output ;
DRAM	DSP_Unit	SDRAM_Port	output ;
Cache	DRAM	Cache_Port	output ;
DRAM	Cache	SDRAM_Port	output ;
light	Sensor_Input	Light_Port	to_bus ;
Key_Entry	Sensor_Input	Key_Port	to_bus ;
Sensor_Input	light	Input_Port	to_bus ;
Sensor_Input	Key_Entry	Input_Port	to_bus ;

Application Behavior

The behavior can consist of existing or new applications. The blocks in the Behavior flow represent a single task in the application. For existing application, the source code could be compiled and used as an instruction sequence. In the case of a new application, a pseudo code can be constructed and used as an instruction sequence. In both these cases, the list of sequences will be executed on the processor to get the exact performance.

For new application with limited information, VisualSim provides a unique methodology to model the tasks for performance analysis. First the user breaks the application down into a sequence of tasks. Each task in this case is associated with number of time units or cycle count. The flow diagram will contain dependencies on a prior task, data availability or triggers (interrupts). The flows define the tasks that are executed in parallel i.e., execute in synchronous time.

The above dependency graph and parallel execution can also be modeling to optimize existing software functions or pseudo code. The instruction sequence generated form the software execution is broken into sequence of tasks. Once the dependency and flows are charted on the Block Diagram Editor, the user can edit the diagram to move tasks to a more parallel flow. The mapping can also be varied by performing concurrent scheduling and retargeting at a different platform.

Each task in the flow is described using the Software Mapper block. This block gets the timing, priority, target processor and instruction sequence information from a Mapper Database. The database references a text file that contains the information. Refer to the section on mapping to understand the template for this file.

In the model in Figure 3, there are three applications- Motion Video, Keyless entry and Light. The Motion Video and Keyless entry are triggered by a user operation. The Light is activated by the door opening (end of the keyless entry flow) and the light sensor data being available. These two triggers are the dependencies in the Light flow and triggers the Light function to execute. Each blue block in the flow for the Behavior represents one task. The DVD or Motion Video flow is independent of the Keyless entry and light. This uses custom DSP functions and most of the tasks are executed on the DSP.

The mathematics associated with each tasks can be described using the SmartMachine scripting language, C/C++ code or using the Expression blocks. The Statement blocks in the behavior flows are the mathematics for the tasks.

For new algorithms or applications, a standard technique is used in VisualSim to determine the activity time or clock cycles. This is the most important of the software exploration process. The initial estimate will be based on the number of instructions and data size required to address the task algorithm and a band range around it. For example, the # of loops will be a function of the

data size and the number of frames to be processed. So, if the estimate is 4 cycles per loop, then the range can be 2-8 cycles. We can further refine this by stating that 50% of the time it is in this range and 25% of a tail on each side of this range contains one additional cycle. For every data input, the loop is computed dynamically and the cycle count is provided to the Scheduler to processor.

Mapping

A mapping table is provided to the model using the Database block. The mapping table along with the associated list of instructions is show.

```
Task_Name   Task_ID    Destination          Instruction ;
SST          2          Renesas_Arch.SH7750R   {"ADD"}    ;
AES          3          Renesas_Arch.SH7750R   {"MOV","XOR","MOV","SHLR","SHLR","XOR"} ;
Verify       4          Renesas_Arch.SH7750R   {"FADD"}   ;
FFT          1          Renesas_Arch.DSP_Unit  {"FFT"}    ;
Light        5          Renesas_Arch.SH7750R   {"ADD,MOV,SHLL"}   ;
Read         6          Renesas_Arch.SH7750R   {"MOV"}    ;
Decode       7          Renesas_Arch.DSP_Unit  {"DECODE"}    ;
PortProc     8          Renesas_Arch.DSP_Unit  {"POSTPROC"}    ;
Render       9          Renesas_Arch.DSP_Unit  {"RENDER"}    ;
Format       10         Renesas_Arch.DSP_Unit  {"FORMAT"}    ;
Rotate       11         Renesas_Arch.DSP_Unit  {"ROTATE"}    ;
FIR          12         Renesas_Arch.SH7750R   {"ADD","SUB","BF","ADD","SUB","DIV","*BF","MUL"}
```

The mapping is scheduled using the destination in the Column 3. The execution destination can be made dynamic for optimal load balancing by using the RegEx function “getNextResource”. The parameters for this function are the list of possible targets. The table has been currently setup to target the SH4 and the DSP processors. Add another SH4 or ARM9 to the architecture and then update this table for the mapping. The instruction in the sequence would have a different name from the ones for the SH4 when targeting to the ARM9. This is easily created by performing a “Find and Replace” operation.

User Trigger or Traffic

This is used to trigger the behavior flows. This describes the user actions such as click on the key transmitter or inserting a DVD into the player and click on “GO”. This is constructed using the Transaction Generators in VisualSim.

Power Exploration

The Power Manager is the Red block in the model. The Power information was captured for only the SH4 processor. The rest of the devices in the architecture were omitted for the purpose of this power study. The other architecture components can be added to the list by entering the power levels for each component in a separate row. The power information for the SH7750R is described as follows in the block:

```
Architecture_Block              Standby  Active  Wait   Idle      Cycles    ;
Renesas_Arch_SH4-7750R          15.0     195.0   39.0   30.0       1        ; /* mWatts */
Power_Manager_Speed: 240 MHz
```

Analysis and Statistics

In this model, we have generated the summary for the Architecture components such as the Bus, SH4, DSP, Cache and memory. This was done by connecting a Text Viewer to the output of the

Architecture Setup block. The latency was calculated for the Keyless action only. This was taken as the time from the user action to the OPEN command. The Statistics at the bottom right of the model was triggered using Virtual Connection from the input to the Light action. The Light action starts immediately after the Keyless entry tasks were completed. The last statistics generated is the Instantaneous power for the SH4 and the energy discharge rate. This was done by connecting XY Viewers to the output of the Power Manager block.

A few of the statistics are displayed below:

Figure 4 Timing Diagram of the SH4 Processor Resources

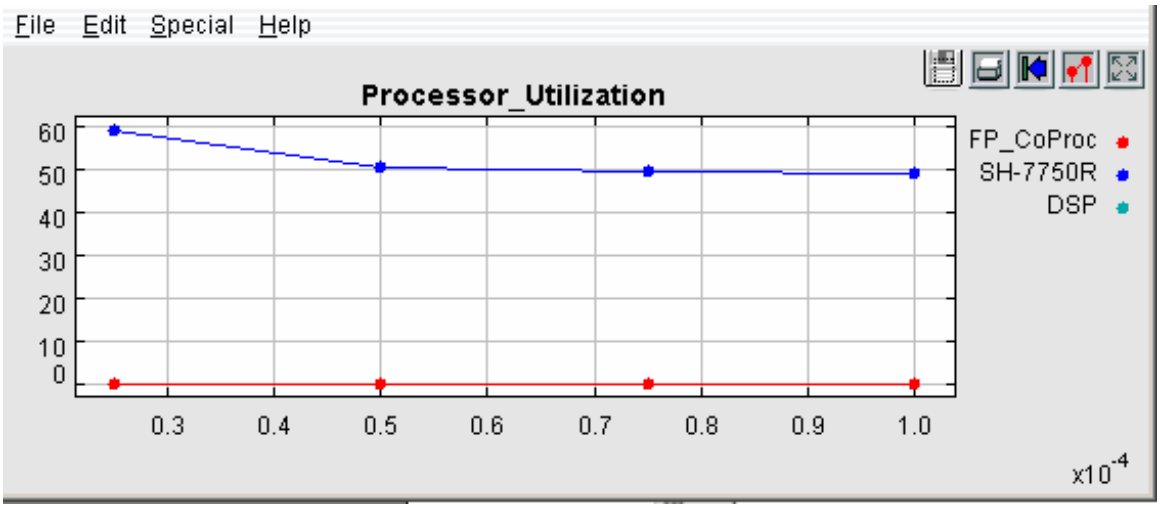

Figure 5 SH4 and Co-Processor utilization

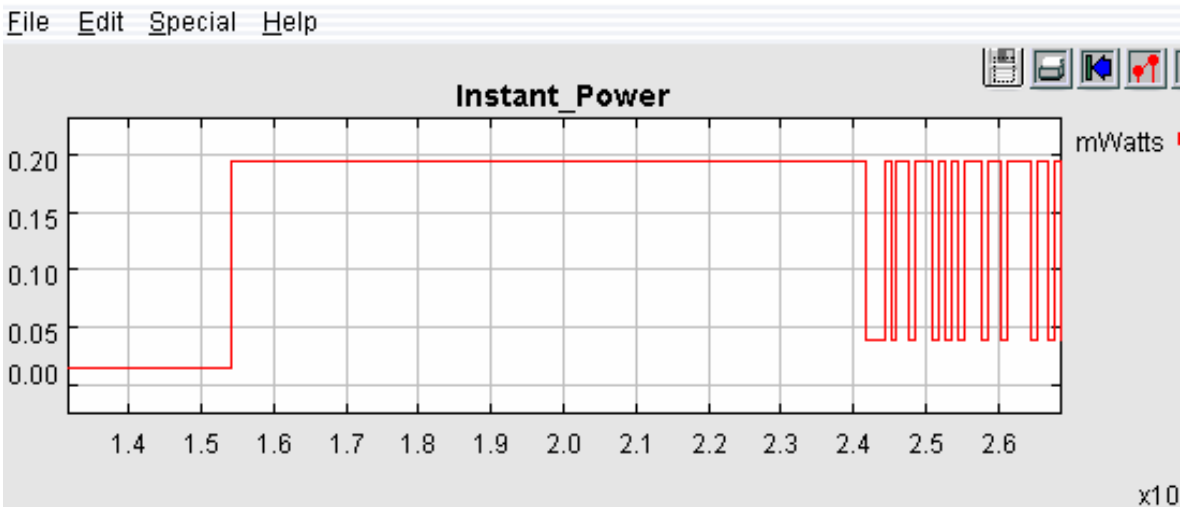

Figure 6 System-Level Instantaneous Power Consumption

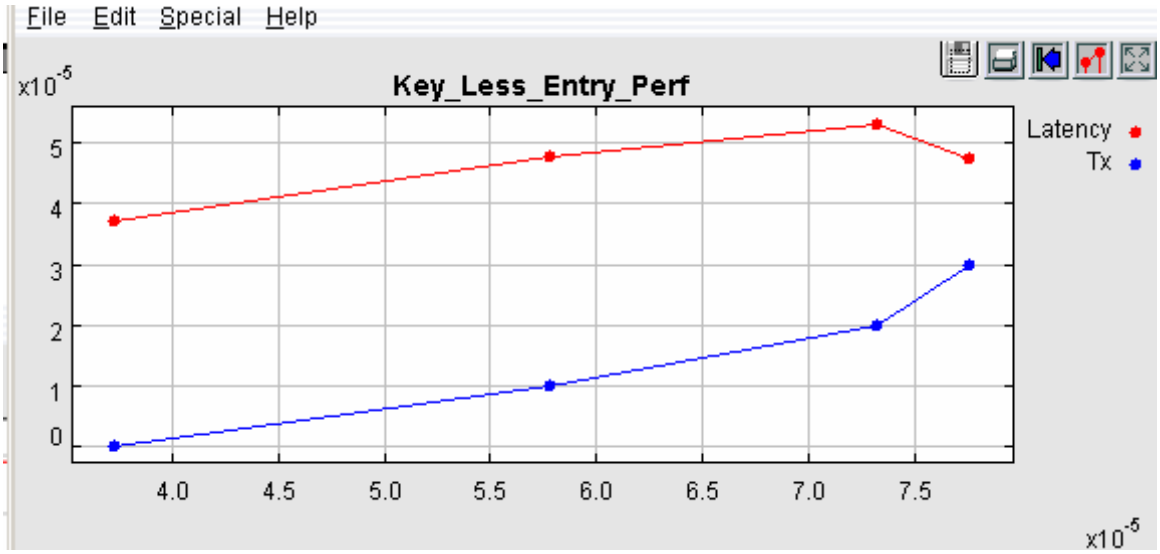

Figure 7 Response for from Transmitter to Door Opening

Benefits

The advantages of using VisualSim to address the above issues and also doing it early in the design cycle is-

1. Customer can minimize the number of platforms being maintained. They can try out different combinations of use-cases with each platform during development. Hence they can characterize the platforms by applications with associated performance metrics.
2. Comprehensive testing without having to build each application. There are a number of applications, use cases and operating conditions that each platform will be used in. It is impossible to create software for each application, develop a prototype for each case, create an operating environment for each condition and test all possible conditions. The risk of lower quality or not operating with the stated performance will be a huge impact on the business, revenue and profits.
3. Demonstrate the product functionality and performance to customers without building prototypes. Each customer will require a different platform variation and different

combination of applications to run on their product. The platform can be used to quickly create a combination of a platform and a set of application models. The simulation can demonstrate the functionality and performance. All of this can be accomplished in a few hours and without requiring that the software be available.