

An Approach to Reusing Architectural Models

Presenter

Abhilash V Nair

Texas Instruments Inc

Agenda

- ☑ Motivation
- ☑ Description of Architectural model
- ☑ Key Observation from the Architectural model
- ☑ Optimization based on Observation
- ☑ Case study result
- ☑ Conclusion & Future Steps
- ☑ Questions and Discussions

Motivation

- ☑ Reuse of architectural modeling for application development
 - ☒ Early availability of a Cycle Accurate model
- ☑ Improving the performance of the architectural model without disturbing the accuracy of the model
- ☑ Define some standard techniques which can be used for SystemC code optimization

Description of Architectural model

- ✓ The SystemC model represents a Unified Cache model
- ✓ It provides a multi-access cache which is scalable for a number of slaves as well as number of destinations
- ✓ The model is configurable for parameters like
 - ☒ Cache Size
 - ☒ Cache Line Size
 - ☒ Number of Slave ports
 - ☒ Number of Master ports
- ✓ The architecture is 2-staged with tag lookup and memory access
- ✓ It is used as a level1 and level2 cache model and sits very close to the processor
- ✓ The cache model can be used with TI DSP processor or General Purpose Processor like ARM

Key Observation from the Architectural model

- ✓ The model was very low in performance around 22 Kilo Cycles Per Sec (KCPS)
- ✓ Some of the major observations in the model were
 - ⊗ Most of the processes were threads and waiting on a clock, positive or negative edge
 - ⊗ SystemC data types were used extensively
 - ⊗ The slave interface used to connect to the processor was using TLM1.0 FIFO and STL queues

Optimization based on Observation

- ✓ Use SC_METHOD instead of SC_THREAD
 - ✗ [Case 1](#)
 - ✗ Case 2
 - ✗ Case 3
 - ✗ Most of the sc_thread was converted to sc_method using the above method
- ✓ Standard C++ Data types instead of SystemC data types
 - ✗ [Case 4](#)
- ✓ Event based model than time based model. Reduce the use of sc_clock.
 - ✗ [Case 5](#)
 - ✗ The model becomes complex as we move from clock based model to event based model. But this will improve the speed.
 - ✗ Cycle accuracy may get effected if not done properly with good knowledge of the architecture
- ✓ Use a simple FIFO rather than STL or TLM1.0 when the interface needs to be used as a data store unit with no timing details

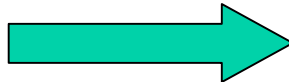


SC_THREAD To SC_METHOD

- ✓ A Thread with n wait statement can be split into n+1 method each waiting on single even
- ✓ Some time it may be required to check if event1 has already occurred before event2 to start the func2

```
sc_thread (my_thread);
```

```
void my_thread{  
wait(event1)  
Statement1;  
wait(event2)  
Statement2  
wait(event3)  
statement3  
}
```



```
sc_method(func1)  
sensitive << event1  
sc_method(func2)  
sensitive << event2  
sc_method(func3)  
sensitive << event3  
void func1(){  
    Statement1;  
}  
void func2(){  
    Statement2;  
}  
void func3(){  
    Statement3;  
}
```

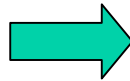
SC_THREAD To SC_METHOD

☑ Avoid wait on clock positive or negative edge

```
sc_thread (my_thread1);  
sc_thread (my_thread2);
```

```
void my_thread1(){  
    while(1){  
        Wait(event1);  
        Wait(clk.posedge);  
        Statement1;  
    }  
}
```

```
void my_thread2(){  
    while(1){  
        Wait(event1);  
        Wait(clk.posedge);  
        Statement2;  
    }  
}
```



```
sc_method (my_method1);  
sc_method (my_method2);  
    sensitivity << my_method_event  
sc_method (my_method_interm);  
    sensitivity << event1;
```

```
void my_method_interm{  
    Static count = 0;  
    if (count == 0){  
        next_trigger(clk.posedge);  
        count = 1;  
    }  
    else {  
        my_method_event.notify();  
        count = 0;  
    }  
}
```

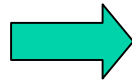
```
void my_method(){  
    Statement1;  
}
```

SC_THREAD To SC_METHOD

☑ Merge processes when connected by events

```
sc_thread (my_thread1);  
sc_thread(my_thread2);
```

```
void my_thread1(){  
    while(1){  
        Wait(event1);  
        Statement1;  
        event2.notify();  
    }  
}  
void my_thread2(){  
    while(1){  
        wait(event2);  
        statement2;  
    }  
}
```



```
sc_method (my_method1);  
    sensitivity << event1  
sc_method (my_method2);  
    sensitivity << event1
```

```
void my_method1{  
    statement1;  
    statement2;  
}  
void my_method(){  
    statement2;  
}
```

☑ The above change can not be done if the notification is after a time unit.



SystemC To C++ Data Types

- ☑ Use simple C++ data types rather than System C data types when possible

- ☒ The below data type was used to represent the tag memory

```
sc_biguint <MAX_TAG_MEM_WIDTH> tag_mem[MAX_NUM_LINES_PER_WAY];
```



```
unsigned int tag_mem[MAX_NUM_LINES_PER_WAY][MAX_NUM_WAY];  
MAX_TAG_MEM_WIDTH = TAG_WIDTH * MAX_NUM_WAY
```

- ☒ Some mapping between SystemC and C++ data types

- ☒ *sc_uint* <1> => bool
- ☒ *sc_uint* <2> ... *sc_uint* <32> => unsigned int
- ☒ *sc_biguint* <N> => unsigned int [N/32]

- ☒ Range function was used extensively with the SystemC data type. It can be replaced by the simple macro like

```
#define RANGE(x, end_index, start_index) \  
((x & ((1 << (end_index + 1)) - 1)) >> start_index)  
#endif
```



Clock Based Model To Event Based Model

- ✓ Event Based rather than Clocked based model
- ✓ Notify when all the prerequisite for the thread or method to run is completed
- ✓ More prerequisite means more flag to check if all conditions are satisfied

```
sc_thread (my_thread);  
void my_thread(){  
    while(1){  
        wait (clk.posedge)  
        If( my_req == true ){  
            Statement1  
        }  
    }  
}
```



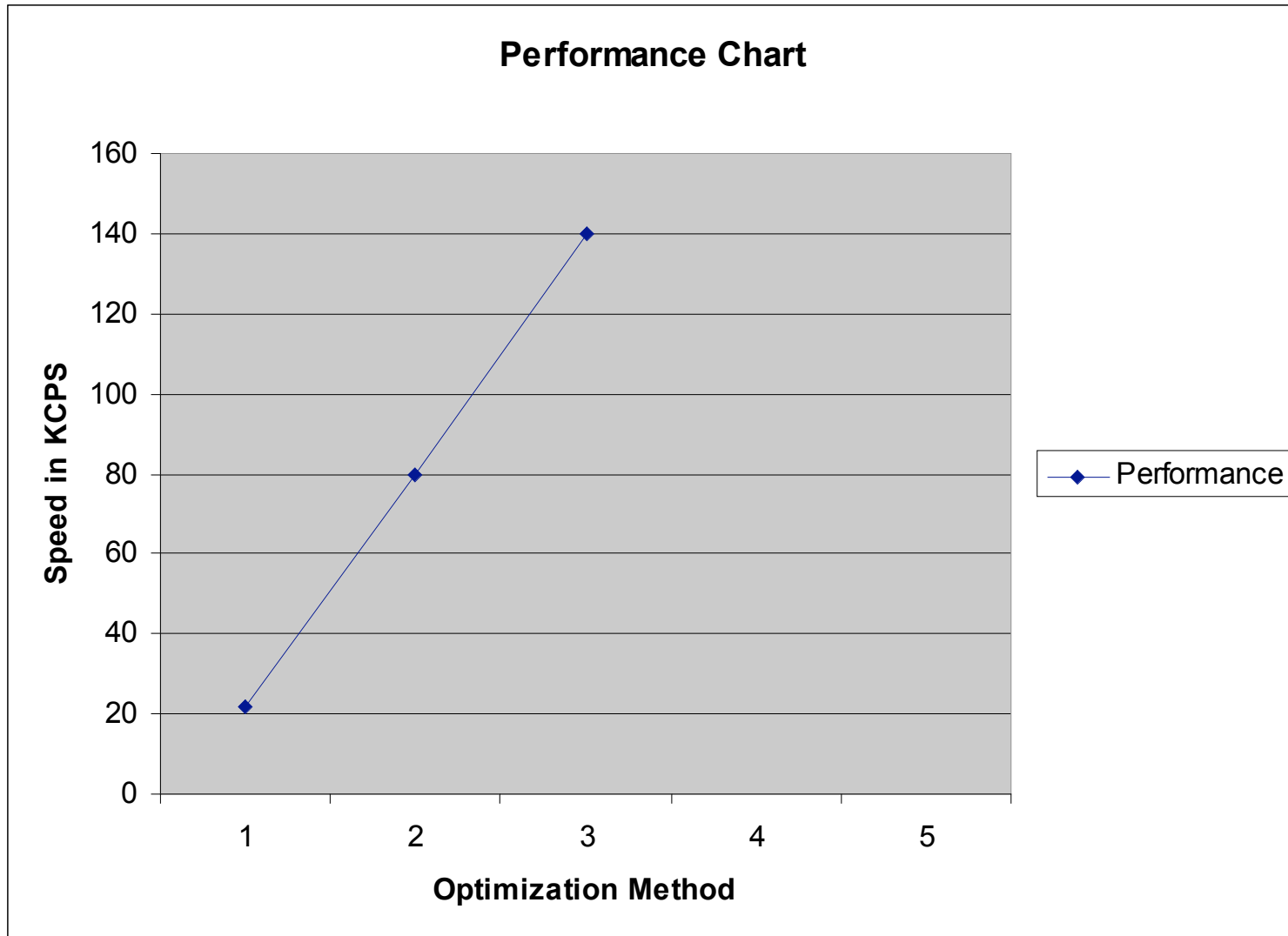
```
sc_method (my_method);  
    sensitivity << my_req
```

```
void my_method {  
    static count = 0;  
    if (count == 0){  
        next_trigger(clk.posedge);  
        count = 1;  
    }  
    else {  
        statement1;  
        count = 0;  
    }  
}
```

```
/* Notify my_req when it is set to 1 */  
my_req.notify();
```



Case Study Result



Conclusion

- ✓ High performance model can be developed using SystemC language
- ✓ Performance can be improved without compromising the accuracy of the model
- ✓ Reuse of an architectural model, previously used for hardware evaluation, for software development
 - ⊗ Reduced the development of a Cycle Accurate model from almost 1 year to 3 months.

Future Steps

- ✓ Use of `sc_time` instead of `sc_clock`
- ✓ Provide Macros to substitute the `SC_THREAD` to optimized `SC_METHOD` without change in user code
 - ⊗ Model can be used for RTL synthesis
 - ⊗ Easy of development

Questions and Discussions