Practical Issues in Implementing Fast and Accurate SystemC-Constructed Virtual Platform Simulation

Yu-Fu Yeh [+] [Ψ]
d94943035@ntu.edu.tw

Chung-Yang (Ric) Huang [+]
ric@cc.ee.ntu.edu.tw

[+] *Graduate Institute of Electronics Engineering, National Taiwan University*
[Ψ] *Information and Communications Research Laboratories, Industrial Technology Research Institute*

ABSTRACT

A SystemC-constructed virtual platform simulation usually encounters an issue in trading off simulation efficiency and accuracy. This paper first introduces a thought to both accelerate simulation and offer accurate outcome by integrating our proposed ultra synchronization checking method and a trace-driven simulation technique. However, realizing the thought in SystemC-constructed virtual platform simulation must change the original simulation scheme that the primitive SystemC kernel preforms. To ensure the correctness of simulation outcome without worrying about the change of the different simulation schemes, we then propose QuteVP+, a simulation framework, to achieve the thought for the use of SystemC designs. When a SystemC virtual platform of MPSoC designs on QuteVP+, the experimental results shows that our QuteVP+ speeds up the simulation as high as 121X. Moreover, simulation result is still maintained with cycle accuracy.

Keywords—SystemC; Virtual platform simulation; Transaction-Level Modeling (TLM); Asynchronous discrete event simulation; Trace-driven simulation; Multi-Processor System on Chip (MPSoC).

I. INTRODUCTION

SystemC [1] is a well-known language to conduct virtual platform simulation. To simulate hardware components, a SystemC-constructed virtual prototype creates independent threads to represent different hardware simulation processes (HSPs). Then a primitive SystemC simulator manipulates these HSPs with synchronous discrete event simulation (sync-DES) [2] scheme to mimic concurrent behavior of hardware components. Based on sync-DES, HSPs are synchronized with simulation kernel at every simulation timestamp as shown in Fig. 1(a). This means that a SystemC simulator regularly updates simulation time and state of HSPs; therefore, SystemC-constructed virtual platform simulation can ensure accurate simulation outcome. However, the cost of synchronization (thread context switches between HSPs and a SystemC kernel) is expensive. Moreover, the number of synchronization is significant because the simulator following sync-DES evokes synchronization(s) frequently, especially the simulation with a short timestamp period. Consequently, the significant number of synchronization causes heavy simulation overhead, leadings to a serious degradation in simulation efficiency.

To accelerate virtual platform simulation, previous works (i.e. [3] [4]) suggest asynchronous discrete event simulation (async-DES) scheme [5]. As shown in Fig. 1(b), async-DES allows each HSP to asynchronously (out-of-orderly) advance simulation time. Virtual platform simulation with async-DES can then reduce synchronization. Owing to the benefit of synchronization reduction, async-DES can improve simulation efficiency.

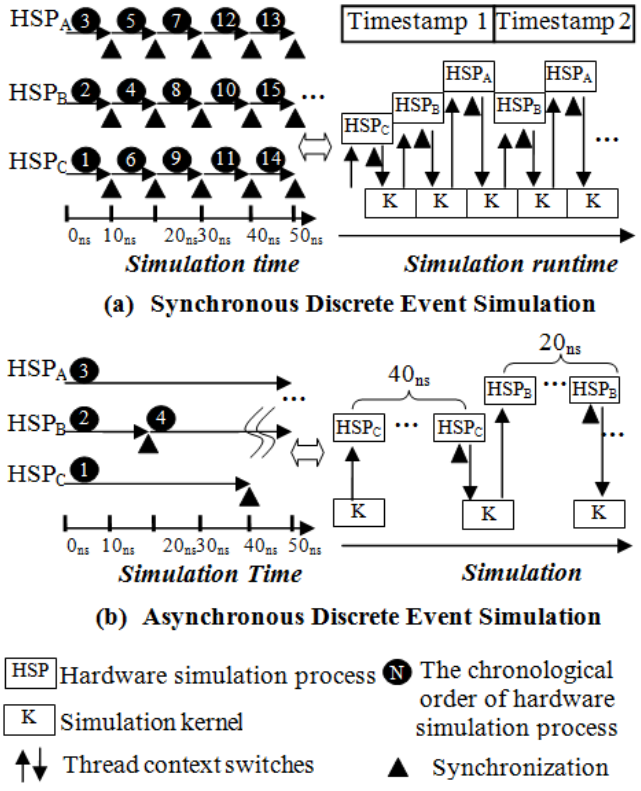

Fig.1 Synchronous discrete event simulation scheme versus asynchronous discrete event simulation scheme

Nevertheless, scheduling HSPs without synchronization ruins functional accuracy and temporal accuracy. This is because lacking necessary synchronization evokes data-dependency violation while HSPs are accessing the same

memory block. (Notably, data-dependency violation leads to incorrect simulation outcome and then ruins functional accuracy.) Additionally, if an HSP individually executes its hardware simulation without synchronizing other HSPs, communication delay among HSPs are ignored. Due to the disregard of communication delay, simulation outcome loses temporal accuracy.

In order to both improve simulation efficiency and ensure simulation accuracy, we introduce a thought which integrates the ultra synchronization checking method (USCM) [4] and trace-driven simulation [6] to conduct fast and accurate virtual platform simulation. However, there are concerns while realizing the thought in SystemC-constructed virtual platform simulation. To overcome the concerns we propose QuteVP+, a simulation work with the related utilities, for achievement of our thought. Finally, the experimental result shows that a Multi-Processor System on Chip (MPSoC) virtual platform simulation with QuteVP+ can achieve simulation acceleration as high as 121X. Moreover, cycle accurate simulation outcome is still be maintained.

The remains of this paper are as follows. Section II introduces a thought for virtual platform simulation with async-DES. Section III introduces how USCM reduces unnecessary synchronization. Then we propose a simulation framework (QuteVP+) in Section IV, as the infrastructure for SystemC-constructed virtual platform. Section V demonstrates the experimental result. Finally, Section VI concludes this paper.

II. PRELIMINARIES

In this section, we first introduce a thought to illustrate how virtual platform simulation schedule hardware simulation processes with async-DES scheme. Then synchronization reduction and time reconstruction techniques are proposed to collocate with async-DES for fast yet accurate virtual platform simulation.

A. Virtual platform simulation with an asynchrnous discrete event simulation (async-DES) flow

In previous works (i.e. [3-4]), virtual platform simulation with async-DES flow contains the kernel phase and hardware simulation phase (Fig.2) to manipulate HSPs as shown in Fig.1 (b). First, simulation flow starts in the kernel phase. After simulation initialization, the simulator triggers an HSP and turns simulation flow from kernel phase to hardware simulation phase. Then the triggered HSP begins to execute its simulation. Fig. 2 shows that the HSP can execute its simulation continuously until the HSP encounters a true synchronization condition. However, once the true synchronization is met, the simulation flow turns back to the kernel phase and synchronization is evoked to halt the executing HSP. Since the execution of each HSP with async-DES executes hardware simulation independently, HSPs are possibly halted at different simulation time. Therefore, communication delay is ignored among HSPs during the continuous hardware simulation. To maintain temporal accuracy, the simulator needs to reconstruct simulation time before triggering the next HSP. Otherwise, accurate outcome cannot be guaranteed. By repeating the introduced simulation flows, async-DES can schedule HSPs to accomplish virtual platform simulation without evoking synchronization at each simulation timestamp.

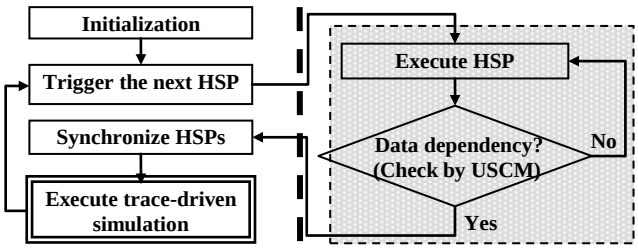

Fig.2 Virtual platform simulation with an async-DES flow

Nevertheless, async-DES needs to collocate with related techniques, which helps determine the necessity of synchronization and reconstruct accurate simulation time. The related techniques are introduced in next subsections.

B. Data-Dependency Checking Method

Virtual platform simulation is generally used for system-level verification and/or design space exploration in early design stage [7]. Research strongly recommends Transaction-Level Modeling (TLM) [8] technique, which can help take trivial signals away and compact complex communication in a transaction, for improvement of simulation efficiency. With the simplification by TLM, the necessity of synchronization can be determined by checking data dependency as in [3-4] [9]. We give an example to explain the principle of data dependency checking below.

Assume that an SoC design contains two hardware modules (an ARM processor and Direct Memory Access Controller (DMAC)). Through memory access analysis, the memory access regions where ARM and DMAC potentially access, can be constructed in a memory access map as Fig. 3 shows. It is obvious that data dependency only occurs while ARM and DMAC access the overlapped memory region (from 0x48000000 to 0x48190000). On the other hand, if intending to access data within the non-overlapped memory regions, an HSP can access data directly without worrying about violating data dependency. Therefore, the simulator can skip synchronization since no data dependency occurs in the non-overlapped memory region.

More details of data-dependency checking are presented in Section III.

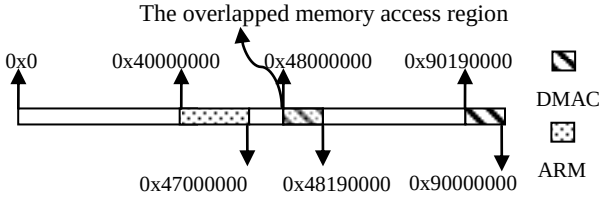

Fig.3 An example of memory access map

C. *Trace-driven simulation*

Because HSPs execute hardware simulation individually when the simulator follows async-DES, communication details among HSPs are ignored. However, disregard of communication details (such as bus contentions), temporal accuracy of simulation outcome is incapable to be guaranteed.

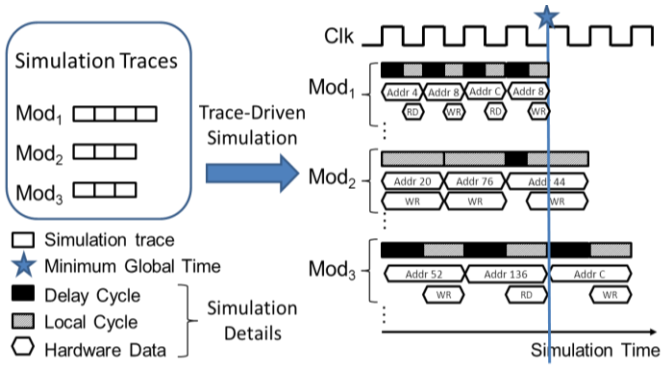

Fig. 4 Time reconstruction with trace-driven simulation

To restore the accurate simulation time [1], timing annotation [10] and trace-driven simulation [6] are two commonly used techniques. Achieving time reconstruction, time annotation technique applies statistic data and inspects immediate state of the simulated HSP to compute communication delay. Basically, applying statistic data and inspecting state of an HSP are not necessary to actually perform interactions among HSPs. Then a time annotation technique can perform time reconstruction efficiently. However, the delay time computation hardly guarantees the exact result if lacking information from the actual interactions. With respect to timing annotation technique, trace-driven simulation records simulation traces and use the traces to re-produce actual interactions for time reconstruction. Thus, trace-driven simulation ensures temporal accuracy.

[1] Simulation time is referred to the logic time of the simulated target on virtual platform; and simulation runtime is referred to the physical time on host machine.

As Fig. 4 shows, the exact timing information (such as local HSP cycle) can be extracted from the recorded simulation traces. Trace-driven simulation can utilize these information and follows communication rules (i.e: bus protocol) to insert delay and then aligns the simulation time for each HSP. After time alignment is done, the HSP with the minimal global time can be figured out. The simulator can schedule the HSP as the next runnable HSP to maintain data dependency.

III. Ultra Synchronization Checking Method

Many data-dependency checking methods, such as [3-4] [9], have been proposed. We suggested adopting Ultra Synchronization Checking Method (USCM) [4] in MPSoC virtual platform simulation. USCM in [4] explains that a hardware module has the authority to access some exclusive memory regions, whereas no other hardware modules can access. This implies a memory exclusive property for data-dependency checking: no data dependency occurs when an HSP accesses data within its exclusive memory regions. With the memory exclusive property, an HSP can just watch its exclusive memory regions to complete data-dependency checking.

For the most precise judgment about data dependency, USCM acquires exclusive memory information from both the hardware and also the program/data storage of the embedded software. Moreover, the memory information is analyzed statically (i.e. at compile time) and dynamically (i.e. during simulation). Hence, a Memory-Exclusivity Table (MET) mechanism is developed to facilitate our analysis of various types of exclusive memory information. Two MET types are presented (i.e. hardware-based and software-based METs), along with a description of how they can be further categorized into hardware static/dynamic METs and software static/dynamic METs, respectively.

A. *Hardware-based Memory-Exclusivity Tables:*

The first exclusive memory information type involves the regulation of memory regions to be read-only or private for HSPs. Such memory information generally exists in the specification of MPSoC. To handle these cases, a format is defined in the hardware specification of the virtual platform, which can be embedded into the header files of the virtual platform implementation. Moreover, the simulator is allowed to parses this information before the simulation starts and stores it into hardware static-MET (HW S.MET).

Although an executing HSP can access its exclusive memory regions without synchronization, some exclusive memory regions dynamically change during simulation. To

avoid incorrect data-dependency checking, exclusive memory information must be dynamically updated. For example, DMAC can exclusively access data while transferring mass data within source memory regions and destination memory regions. Notably, the source and destination memory regions are changeable for different data movements. To handle the dynamic exclusive memory information in simulation, this work offers the ADD_H_D_ExclusiveMem functions for dynamic memory information update (Fig. 5). A designer can then embed these functions into the hardware behavioral function of an HSP as the normal utilization of "pragma". Upon entering a working state, the HSP calls the memory acquiring function to add the exclusive memory regions to hardware dynamic-MET (HW d.MET). When leaving the working state, the HSP calls the memory acquiring function again to remove the memory regions from HW d.MET. Hence, the HSP can utilize the updated exclusive memory information for correct data-dependency checking.

```
void DC::ADD_H_D_ExclusiveMem (unsigned int MemBegin, unsigned int MemEnd) {
  HW_DDDT.push_back(new pair(MemBegin, MemEnd));
}
void DC::DEL_H_D_ExclusvieMem (unsigned int MemBegin, unsigned int MemEnd) {
  itrHWDDDT = HW_DDDT.find(uPair(MemBegin, MemEnd))
  HW_DDDT.erase(itrModDDDT);
}
```

```
void DMA::MassDataMove() {
  // Set the memory regions where only "DMAC" can access
  // Embed function to acquire HW dynamic exclusive memory information
  DDC->ADD_H_D_ExclusiveMem(SourceAddr, SourceAddr+M_Size);
  DDC->ADD_H_D_ExclusiveMem(DistAddr, DistAddr+M_Size);
  cout << "DMAC begins to move mass data .... " << endl;
  for(unsigned int i = 0; i < M_Size; i++){
    ReadMemory(SourceAddr, 4);
    WriteMemory(DistAddr, 4, m_resp_data);
    SourceAddr += 4; DistAddr += 4;
  }
  cout << "DMAC Finishes move data ...." << endl;
  // Embed function to remove HW dynamic exclusive memory information
  pSync->DEL_H_D_ExclusiveMem(SouceAddr-M_Size, SourceAddr);
  pSync->DEL_H_D_ExclusiveMem(DistAddr-M_Size, DistAddr);
}
```

Fig.5 An example of exclusive memory information update functions in our DMAC controller model

B. Software-based Memory-Exclusivity Tables:

When the MPSoC virtual platform simulation is performed, in addition to hardware modules, multiple embedded software programs running on multiple processor models must be considered as well. Failing to consider the effects of software programs in data dependency checking leads to a conservative synchronization checking mechanism. For example, most processor models can access all of the shared memory regions. Therefore, when only referring to HW-Based METs, synchronizations are performed in almost all cycles.

As well known in software analysis, only when the software program possesses data-exchanging behavior (e.g., mutex and semaphore) in shared memory should one consider its data-dependency related issues. In other words, if a function of a software program contains only computations within the processor model, the corresponding hardware simulations do not result in data dependencies with other processor models. Therefore, to characterize how software programs impact the synchronization mechanism, the software functions that perform data exchanges with other modules (i.e. the communication functions) should be distinguished from those computation functions. The proposed memory-exclusivity checking mechanism stores the program memory information of the communication functions in the software static-MET. During simulation, if the current program counter address does not match the memory information recorded in the software static-MET, we can infer that the current simulating function is a computation function. Thus no data dependency issue is presented.

During the simulation in which a simulating processor executes communication functions, this work attempts to acquire another type of exclusive memory information to check data- dependency more detail and achieves the better effectiveness of synchronization reduction. Thoroughly analyzing the software program reveals that a simulating processor can exclusively access some dedicated variables such as local variables and constant variables. Our results further indicate that these memory blocks for the dedicated variables do not involve data dependency, even when a simulating processor executes communication functions. This finding suggests that unnecessary synchronization can be further reduced if the above-mentioned exclusive memory information can be obtained.

In our work, local variable are noted by manually inserting "progma" in software program. With compiling commands, a compiler can output the information, such as symbol, text and register tables, to denote local variables. Therefore, addresses of local variables/arguments can be dynamically and exactly captured when a simulating CPU executes the entry of a software function with a disassembling or debugging tool [14]. Notably, the exclusive memory information of the variable is changeable when a simulating processor executes a communication function at different times. Therefore, the simulator must update the exclusive memory information while an HSP of the processor model executes the communication function again. With the mentioned procedures, our work acquires the exclusive memory information as another MET type, called software dynamic-MET (SW d.MET). By checking SW d.MET, synchronization reduction can be more aggressive.

C. Memory Exclusivity Checking Flow:

To utilize METs for memory exclusivity checking, the proposed memory exclusivity checking method looks up four types of METs. The exclusive memory regions of

individual METs are stored in tables with a start and an end memory address. When data dependency is checked, the simulator can search the tables and determine whether the memory address of communication transaction falls within any of the exclusive memory regions. As Fig. 6 shows, a "true" synchronization condition is the one which the details are confirmed by checking all of four METs to determine the memory access with data dependency. On the other hand, a situation in which any one of the data dependency checks determines that the memory address is in exclusive memory regions implies that the transaction of memory access does not evoke data dependency. The simulator can stop checking other METs and skips synchronization. Finally, since synchronization can be reduced without checking all METs, we recommend checking static METs before dynamic METs. This is owing to that checking static METs in log time complexity is more efficient than checking dynamic METs in linear time complexity.

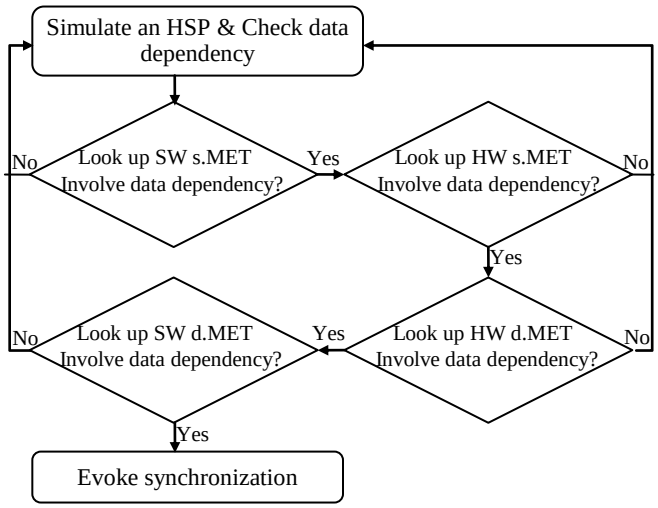

Fig.6 The memory exclusivity checking flow

IV. Our Simulation framework to Conduct Fast and Accurate MPSoC Virtual Platform Simulation

The preceding section presents the thought of async-DES flow with USCM and trace-driven simulation to conduct fast and accurate virtual platform simulation. To realize the thought, compatibility and modification efforts are two critical concerns while modifying the simulation scheme in the primitive SystemC kernel. Then we propose a simulation framework (QuteVP+) and QuteVP+ utilities to overcome the concerns.

A. The concerns from implementation and utilization persepectives

Section II presents a thought to conduct fast and accurate virtual platform simulation by adopting async-DES with USCM and trace-driven simulation. Nevertheless, to realize the thought on a SystemC-constructed virtual platform simulation, it is necessary to modify SystemC kenrel because a primitive SystemC follows sync-DES. However, the implementation to modify SystemC kernel is not trivial. Here, we discuss two critical concerns as follows.

Firstly, there are function libraries set to connect with SystemC kernel for synchronization (such as wait() and notify() in SystemC library). If the SystemC kernel is modified with async-DES, it is doubtful that SystemC library can perform the same function for the use of the original SystemC designs. Moreover, SystemC kernel is possibly updated in a new version. The modification in the current SystemC kernel may be incompatible to the updated SystemC kernel.

Furthermore, a modification effort is inevitable to the original SystemC designs to adapt an async-DES scheme. However, the modification efforts should be reduced as less as possible because designers are end-users. It is NOT necessary for designers to know details about how the modified SystemC kernel works. Therefore, the approaches for easy-to-modify are required to specify.

Next, we propose QuteVP+ and related utilities to overcome the mentioned concerns.

B. The implementation of QuteVP+

As mentioned, it is hard to ensure the functional consistence in SystemC library after modifying the primitive SystemC kernel. Moreover, the modification in SystemC kernel potentially evokes incompatibility if SystemC is updated with new version. To overcome the concerns, we implement a simulation framework, called QuteVP+. QuteVP+ contains a simulation engine (QuteVP+ engine) to schedule HSPs with async-DES scheme. As Fig. 7 shows, QuteVP+ engine and the SystemC kernel are independent so that the arrangement can both prevent the issues of function inconsistence and avoid incompatibility with new SystemC version.

Each HSP can asynchronously execute it hardware simulation in QuteVP+ because QuteVP+ engine adopts USCM to help determine the necessity of synchronization. For the implementation, we devise QuteVP+ interface so that a SystemC prototype can inherit the interface to communicate with QuteVP engine. Then an HSP can enable QuteVP+ to determine the necessity of synchronization if the HSP request a memory access. With the benefit of data-dependency checking by USCM, HSPs can continue its simulation for many cycles until certain necessary

synchronization condition are met. As a result, a significant synchronization reduction contributes a great promotion in simulation efficiency.

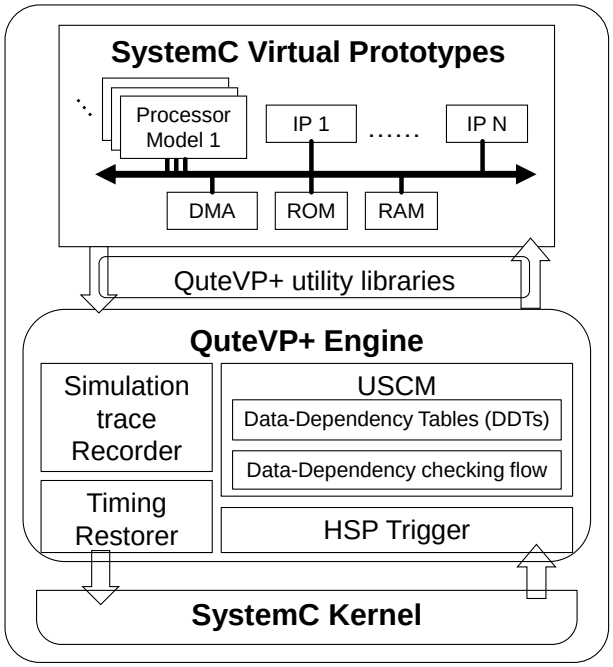

Fig. 7 The block diagrams of QuteVP+

Since our simulation framework replaces the simulation scheme by out-of-order execution approach, each HSP is allowed to advance with different simulation timestamps, instead of referring to global time as the primitive SystemC configures. To maintain temporal accuracy, the simulation engine lets each HSP maintain its local time. Then our simulation engine builds a simulation trace recorder to store acting events based on local time of each HSP. Once the engine needs to maintain temporal accuracy, the timing restorer can utilize the recorded simulation traces to reconstruct accurate simulation time by trace-driven simulation.

In SystemC-constructed virtual platform simulation, enabling synchronization must use the waiting functions. However, some waiting functions with timing configurations involve with the scheduling. Avoiding these waiting functions affecting scheduling, our simulation framework rules that each HSP for synchronization needs use the time-irrelevant wait() function (i.e: wait(event_1)) or wait(SC_ZERO_TIME) function to keep global time always stopping at zero. Therefore, the simulation engine can schedule HSPs with out-of-order execution in "Delta Cycle2".

[2] SystemC defines Delta Cycle, which is typically used for those tasks without the ability to instantaneously change.

In sum, the implemented QuteVP+ engine is independent to the primitive SystemC kernel. Thus, QuteVP+ supports the use for the original SystemC design and avoids the incompatible issue.

C. The utililites of QuteVP+

Basically, functions of an HSP can be categorized to computation and communication functions with transaction-level modeling (TLM). Following the categorization, only the communication function of an HSP influences the behavior of memory access request. This implies that data dependency is only evoked in the communication function. If intending to exploit the ability of QuteVP+ engine for data-dependency checking, an end-user just needs to modify the communication function. Most implementations of the original SystemC virtual platform simulation can be kept and reused.

1. Modify simulation scheme with async-DES in communication function

```
#include <QuteVP_Utility.h>
…. // communication class inheritance
void ARM_ISS::send_request() {
  bool SyncFlag = true;
  SyncFlag = QuteVP_Engine->
DataDependencyChecker (mReq, mResp);
  if (SyncFlag) {
wait(sync_ok_event);
QuteVP_Engine->RequestTransmitter (mReq, mResp);
  }
  if (mReq.get_command()==memory_read)
      M_resp_data = mResp.get_data();
}
```

Fig. 8 An example of sending memory request function with QuteVP+ utility libraries

For the consideration to easily use QuteVP+, we proposed the QuteVP+ utility, reducing the modification effort as less as possible. Here we give an example to show how a processor model applies QuteVP+ utilities to modify an MPSoC design, as shown in Fig.8. Firstly, by including the QuteVP utility library, QuteVP+ utilities can be applied for the target processor model. In this case, the processor model requests memory access through a send_request function call. Then user just needs to modify the send_request function by using the DataDependencyChecker utility. The function of DataDependency Checker utility is to pass the reference of the requesting memory access and check if the memory request involve with data dependency. If the return value of DataDependencyChecker is false, this phenomenon means that the processor model can complete the

transaction of memory request without evoking synchronization. Otherwise, the processor model needs to evoke the untimed synchronization. With the easy modification, the processor model is able to out-of-orderly execute its hardware simulation.

2. Pass Memory Request by Direct Data Access

As is well known for the use of SystemC and OSCI TLM, synchronization will be evoked after a TLM channel requires passing a memory transaction from an initiator port to a target port among HSPs. That is, the OSCI TLM library automatically evokes synchronization to handle a memory request. If a SystemC design adopts the communication approach of TLM in an MPSoC virtual platform simulation, the complex communications often produces a significant number of synchronizations degrading simulation efficiency.

To handle memory transaction more efficiently, we devise a manner, called direct data access to skip unnecessary synchronization. The idea of "direct data access" is to pass a memory request with a function call to directly read/write data from/to the target memory block. Nevertheless, adopting direct data access must overcome a difficulty in letting all HSPs use the same interface to access the data in different hardware models. Then we implement a QuteVP+ interface, which is similar to the interface of transaction level model. Moreover, we implement a function called RequestTransmitter() as Fig.9 shows. With QuteVP+ interface and the RequestTransmitter function, the mentioned difficulty of direct data access can be overcome. In sequel, an example is given to explain how we accomplish the implementation to achieve direct data access in an MPSoC virtual prototype.

Firstly, HSPs in an MPSoC virtual prototype can be categorized as master HSP or slave HSP, with the rule defined in TLM. Our example is that an ARM processor model (as a master HSP) requests a memory access to read data from an RAM model (as a slave HSP). To unify the memory access procedure in the RequestTransmitter function through QuteVP+ engine, our idea is to let the ARM processor model and the RAM model inherit the defined QuteVP+ master interface and QuteVP+ slave interface, which are implemented in QuteVP+ utility libaray. Because all of slave HSPs are ruled to inherit our QuteVP+ interface with an virtual function (get_request()), the ARM processor can access data stored in slave HSP with the unified RequestTransimitter function. Due to the contribution of unification to use the RequestTransimitter function, the modification effort is greatly reduced. Moreover, by inheriting the QuteVP+ interface, QuteVP+ automatically records the ID of master and slave HSP in elaboration phase of the SystemC initialization stage. Then the communication function pointers of HSPs are stored in our defined pointer array. When an HSP requests a memory access through the RequestTranmitter function call, RequestTransmitter function can use the stored function pointer to directly indicate the corresponding get_request() to complete the memory transaction. Due to the bypass, our implemented direct data access mechanism further reduces unnecessary data copy and synchronization; thus, achieving the better simulation efficiency for memory access requests.

```
#include <QuteVP_Utility.h>
// Calculate the request address to seek the target ID
unsigned int pSysc::findIDfromMemMap(unsigned int&
Addr) {
  vector< mMap>::interator itrVec = memMap.begin();
  targetID = 0;
  while (itrVec!=memMap.end()) {
    if (((*itrVec)->first>=Addr) && ((*itrVec)-
>second==Addr)) {
      return targetID;
    }
    ++itrVec; ++targetID;
  }
}
// Using targetID mapping to the corresponding
get_request function
qvp_response
Qutevp_Engine::RequestTransmitter(qvp_request&
mReq) {
  targetAddr = mReq.get_address();
  targetID = findIDfromMemMap(targetAddr);
  return pHSP[targetID].HSPptr()->get_request(mReq);
}
```

Fig. 9 The implementation for direct data access

3. Maintain temporal accuracy by modifying the defined TimeAlign function of QuteVP+

To maintain temporal accuracy, QuteVP+ engine automatically records simulation traces in back-end for time reconstruction. Furthermore, TimeAlign() function in QuteVP+ utility library is a virtual function, in which simulation traces are then available for implementation of trace-driven simulation. Because of the variant in different architecture, users can modify the TimeAlign() function with corresponding communication rule to calculate delay cycle. Time reconstruction for different SystemC design can be achieved.

```
#include <QuteVP_Utility.h>
unsigned int QuteVPplus_engine::TimeAlign () {
  while () {
  // Pop out the simulation traces of each HSP
    ….
  // Compare the memory access request time
    ….
  // Use round-robin protocol to choose the serviced HSP
      ……
  // Insert delay cycle
    ……
  // Check if minimal global time is found
    If (time reconstruction is done==true) break;
  } // time reconstruction ends
  return HSP_ID; // HSP_ID is determined by the result
of minimal global time computation
}
```

Fig. 10 An example of maintaining temporal accuracy in QuteVP+ TimeAlign()

Fig. 10 is an example to show how our test case modifies TimeAlign() to complete trace-driven simulation for an MPSoC with a "round-robin" bus architecture. Firstly, the memory access request time is extracted by the popped-out simulation traces of each HSP. The following process compares the request time of HSPs to examine if a bus contention occurs. Notably, the test case modifies TimeAlign() function by implementing a bus arbitration protocol to determine the HSP which can be serviced. Therefore, communication delay can be exactly inserted for those HSPs with bus contention. By repeating the above steps, TimeAlign() completes time reconstruction. Finally, TimeAlign() returns a HSP_ID to indicate the next runnable HSP.

V. Experimental Results

To validate the robustness of our purposed async-DES scheme, called USCM on QuteVP+, we conduct experiments to compare the simulation efficiency with the sync-DES scheme in the primitive SystemC kernel, called Clock-Step Simulation Method (CSSM). We compare the synchronization counts and the simulation speed while applying different simulation methods in an MPSoC virtual platform simulation. Moreover, we further compare the simulation efficiency improved by USCM.

The experimental settings are in followings. First, we build a virtual MPSoC prototype by SystemC v2.2 and TLM v1.0, and modify public programs, JPEG Encode [10] and Sparse Matrix Multiplication [11]), as the parallel programs for the target MPSoC. We then modify the target MPSoC on QuteVP+, so that the simulation scheme becomes replaceable while keeping the original functionalities of hardware modules intact. Furthermore, the target MPSoC contains two memory systems commonly utilized in MPSoC: the distributed shared memory system and the uniform shared memory system. Then we can examine various types of synchronization conditions in the target MPSoC. Finally, we model ARM v5Te processor as Instruction Set Simulator and other hardware peripherals in the target MPSoC with cycle accuracy. Such arrangements ensure the virtual platform simulation in high simulation accuracy.

These experiments were conducted on a Linux workstation with Intel Xeon 2.2 GHz and 16 GB RAM.

A. The improvement of simulation speed

Table 1 demonstrates the experimental results on the comparison of synchronization count (Sync-Count). We compare CSSM and USCM on two parallel software programs (JPEG encoding and Sparse Matrix Multiplication) with numbers of simulation processor models (CPUs) from 1 to 32. The number of the total simulated instructions and the simulation cycle (i.e. the time unit in the target MPSoC) are presented in the 2, 3, 6 and 7 columns. Please note that we implement USCM with cycle accuracy. Therefore, their instruction counts and target time are the same as CCSM's.

Columns 4, 5, 8, and 9 are the counts of synchronization (Sync-Count) for CSSM and USCM, respectively. We can see that the number of synchronizations increases with the increasing numbers of CPUs, no matter which simulation

Table 1 The counts of synchronizations versus different numbers of processors for SMM and JPEG-Enc simulations

#CPU	SMM				JPEG-Encode			
	#Inst	Simulation Cycles (10ns)	CSSM Sync-Count	USCM Sync-Count	#Inst	Simulation Cycles (10ns)	CSSM Sync-Count	USCM Sync-Count
1	169,314,561	254,261,472	722,859,678	28,423,967	512,513,774	771,969,678	2,212,558,094	87,409,800
2	121,085,896	156,417,989	460,436,669	12,508,290	386,970,594	507,338,577	1,520,454,360	45,790,479
4	107,747,016	124,834,536	365,547,740	7,564,476	314,622,268	376,936,991	1,093,803,850	21,489,726
8	105,501,921	114,852,199	335,037,092	3,891,842	287,006,102	327,838,743	926,966,612	10,945,309
16	112,941,976	119,699,619	349,944,254	2,075,848	272,113,534	308,490,886	860,594,492	5,436,347
32	135,121,633	145,001,298	419,101,187	1,220,659	264,576,313	295,742,881	831,027,316	2,918,037

scheme performs. However, the Sync-Count in our USCM algorithm is usually 1 to 2 orders less than those of CSSM. This indicates that our framework can greatly reduce the numbers of synchronizations.

B. The improvement of simulation speed

Table 2 shows the simulation runtime and speed-up ratio (with the following formula) among CSSM and USCM in the columns 2, 3, 4, and 5, 6, 7, for SMM and JPEG-Encode simulations, respectively.

Table 2. The comparisons of speed-up ratio for SMM and JPEG-Enc simulations

# CPU	SMM			JPEG-Enc		
	CSSM	USCM	Speed-up Ratio	CSSM	USCM	Speed-up Ratio
1	14,917	123	121.0	37,282	385	96.8
2	7,420	108	68.6	22,196	313	70.9
4	4,972	107	46.3	15,195	287	53.0
8	3,255	113	28.8	9,660	275	35.1
16	4,641	137	33.9	11,691	302	38.7
32	6,421	198	32.6	11,740	354	33.2

It is clear that USCM greatly outperforms the CSSM. This is due to the effects in the synchronization reductions. The results show that USCM can simulate a single-processor SoC design with about 121X and 96X speed-up for SMM and JPEG Encode. For the 8~32 processor MPSoC virtual platform simulation, the improvements in simulation speed by USCM can be as high as average 30X. This means that USCM robustly offers the benefit of the simulation speed-up for the SystemC-constructed MPSoC designs.

VI. CONCLUSIONS

In this paper, a simulation framework, QuteVP+, is proposed to overcome a trade-off issue of simulation efficiency and accuracy, for SystemC-based virtual platform simulations. The contributions of QuteVP+ are the integrations of USCM and trace-driven simulation techniques to both accelerate simulation and offer accurate outcome. Moreover, the utility libraries of QuteVP+ are proposed to minimize the efforts in modifying a primitive SystemC-based virtual platform with our proposed QuteVP+. The experimental result demonstrates that QuteVP+ improves simulation speed-up as high as 121X for a SystemC-based virtual platform simulation of MPSoC designs. Moreover, QuteVP+ guarantees simulation results with cycle accuracy.

As mentioned, QuteVP+ needs to recognize more types of dependency conditions for better applicability. Moreover, we plan to propose a parallel scheme on QuteVP+, achieving the further improvement of simulation efficiency.

ACKNOWLEDGMENT

This work is supported by Information and Communications Research Laboratories (ICL), Industrial Technology Research Institute (ITRI), Taiwan, under Grant 101-EC-17-A-05-01-1111.

REFERENCES

[1] <http://www.systemc.org/downloads/standards/>

[2] Banks. J, J. S. Carson II, and B.L. Nelson, "Discrete Event System Simulation, 2nd Ed., Prentice-Hall, Englewood Cliffs, 1996.

[3] K.H Lin, S.J. Cai and Ric. Huang, "Speeding Up SoC Virtual Platform Simulation by Data-dependency Aware Virtual Synchronization", in Proc. IEEE Asia and South Pacific Design Automation Conference, Jan. 2010, pp. 143-148.

[4] Y.-F. Yeh, C.-Y (Ric) Huang, C.-A. Wu and H.-C. Lin, "Speeding up MPSoC Virtual Platform Simulation by Ultra Synchronization Checking Method", in Proc. DATE, Mar. 2011, pp. 353-358.

[5] S. Ghosh, E. Debenedictis, "An Asynchronous Distributed Discrete Event Simulation Algorithm for Cyclic Circuits using a Data-Flow", isn Proc. IEEE System, Man, and Cybermetics, 1991, pp. 265-268.

[6] Y. Yi, "Fast and Accurate Cosimulation of MPSoC Using Trace-Driven Virtual Synchronization", in Proc. IEEE Transaction on Computer-Aided Design of Integrated Circuit and Systems, Dec. 2007, pp. 2186-2200.

[7] D. Densmore, R. Passerone, and A. Sangiovanni-Vincentelli, "A platform-based taxonomy for ESL design", IEEE Design Test, 2006, vol. 23, no. 5, pp. 359-374.

[8] Cai et al, "Transaction Level Modeling: An Overview", in Proc. 1st IEEE Intl. Conf. on Hardware/Software Co-design & System Synthesis, Oct. 2003, pp. 19-24.

[9] M.H Wu, "An effective synchronization approach for fast and accurate multi-core instruction-set simulation", in Proc. 5th EmSoft, Nov. 2009, pp. 197-204.

[10] Y. Hwang, S. Abdi, and D. Gajski, "Cycle-approximate re-targetable performance estimation at the transaction level," in Proc. IEEE Design, Automation, and Test in Europe Conference (DATE), Mar. 2008, pp. 3-8.

[11] Lossless JPEG Encode: <http://www.ijg.org/>, 1998.

[12] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris. Understanding the performance of sparse matrix-vector multiplication. In PDP '08: Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and Network-based Processing, 2008.

[13] "ARM Developer Suite Developer Guide", http://infocenter.arm.com/help/topic/com.arm.doc.dui0056d/DUI0056.pdf, 2001.