# Protocol Bus Modeling using inheritance with TLM2.0

H. Posadas, E. Villar
University of Cantabria, GIM, TEISA Dept.
E.T.S.I.I.T. Avda. Los Castros s/n, 39005
Santander, Spain
{posadash, villar}@teisa.unican.es

M. Martínez
DS2 (Design of Systems on Silicon)
Robert Darwin 2, Parque Tecnológico,
Paterna, Spain
{marcos.martinez}@ds2.es

***Abstract*—With the increase of HW/SW system complexity, effective mechanisms for HW/SW co-simulation are required. The system bus is the intermediate element in these communications, so adequate bus modeling is critical. However, this modeling has to be able to drive the system refinement process as easily as possible. Different bus protocols at different description levels have to be integrated and modified in the processor and peripheral models. This work proposes a generic bus model where OO Programming inheritance techniques are used to integrate it into the platform element descriptions. This technique enables the substitution of the different bus models at different abstraction levels in a semi-automatic way.**

## I. INTRODUCTION[*]

The increase of complexity in electronic system design has created new needs in development methodologies and platform definitions. New designs require combining HW and SW capabilities, considering multiprocessor architectures on each chip (MpSoC).

Furthermore, new methodologies have evolved from design flows where HW and SW are developed in separate ways, to new complex flows considering HW and SW refinement together. Functional and non-functional parameters of the whole system have to be analyzed to validate the system [1]. In this context, co-simulation has become one of the most important issues in HW/SW co-design of very complex systems[2]. Co-simulation requires two main elements, a common environment where all HW and SW components can be simulated and HW/SW communication mechanisms.

To create environments where HW and SW can be simulated together, some system-level languages have been proposed. In this work, SystemC[3] will be used as the underlying language. SystemC provides describing elements to model components from system-level to RTL in the HW area.

In this context, the SystemC development group is creating a new standard called TLM2[4], which defines how component communications have to be done at higher levels than RTL. These levels allow compatible and faster communications, and thus they increase the simulation speed of system co-design[5].

However, SW modeling is not directly covered in SystemC. Although some features, such as dynamic threads are provided, it is not enough to model SW components. There are several approaches to integrate SW tasks in the system simulation. One of the most popular solutions is to use a processor model (ISS) to run the binary code of the SW components, including the OS[6][7]. However, this approach is not suitable for all development stages. At higher levels, the execution of an ISS running all the binary code is too slow. Thus, faster solutions are required.

Simulation of SW source code is the solution usually proposed [7][8][9]. To do this, it is necessary to provide an OS model and tools to obtain performance estimations (mainly execution time estimations). Thus, a complete processor model is not required to integrate the SW execution and its effects on the system simulation.

In our approach this second technique has been selected. A library called PERFidiX[9], which is capable of making source-code execution time estimations and annotations, has been used to obtain a timed simulation of the SW code. Thus, HW and SW are co-simulated in a common time axis, making their interactions occur correctly. Processor accesses and interruptions are generated and received at the correct times.

The library also extends SystemC to provide OS features. Thus, it has been used to integrate the refined SW code simulation in the SystemC model. The library can also manage HW interruptions and allows the SW to access the bus peripherals using HW addresses.

However, to model SW/HW accesses in a realistic way, an adequate bus model is required. Directly connecting the SW tasks to the peripherals, several effects such as transfer delays or bus collisions, cannot be modeled. Several generic bus models have been proposed for bus modeling [10-17], covering all TLM abstraction levels. However, these bus models usually present two drawbacks.

First, the bus protocol management has to be included manually in all the peripherals. This means that all communication refinement is manual. Furthermore, bus models do not consider all effects of task executions, such as preemptions during transfers.

Thus, in the present work a new methodology including a generic bus model is proposed to solve these

[*] This work has been supported by the Spanish MICyT under the MEDEA+ 2º 708 LoMoSA project and MEC through the TEC2005-03301 project.

drawbacks. To do this, in section II the communication requirements for HW/SW co-simulation are analyzed. In section III, the proposed bus model is presented. Section IV describes the use of TLM2 in the present work. In section V, the technique for semi-automatically managing the bus protocols in the platform elements is presented. Finally, an application example is presented.

## II. Related work and problem formulation

System co-design requires high description levels. Modeling transfers at signal level is not suitable for managing really complex systems. The resulting simulations can be too slow for the first development stages.

To address this, Transfer Level Modeling (TLM) techniques have been proposed [6]. Using these, each transfer only requires a function call, instead of several signal accesses. Thus, simulation speed is improved. Several generic bus models using TLM features have been developed [10-13], as long as specific bus ones, as AMBA [14-16], CAN [16] or STBus [17] models.

Furthermore, TLM does not represent a single abstraction level. Transfers can be applied at different levels of abstraction. Usually three levels are defined [12]. In the first one, bus-cycle accurate (BCA), each clock cycle is modeled independently. Thus, each transfer requires a different function call. This technique is suitable when executing SW code in processor models. Each time the SW wants to send a word through the bus, one transfer is done.

Given that ISSs of specific processors are required, this level is commonly used with specific bus models. In [14] a simple BCA bus for AMBA specification is presented. [16] presents a technique called Result Oriented Modeling(ROM), where internal bus states are omitted and the end result is optimistically predicted. In [15], Cycle Count Accurate at Transaction Boundaries (CCATB) technique replaces the bus cycle accuracy. It tries to increase the abstraction level a bit without losing the cycle accuracy. In [13] a non-specific bus is proposed for early IP integration, connecting 3rd party IPs.

In the other two higher abstraction levels, each data transfer does not need to be considered independently. Several data transfers can be modeled together. Each function call can contain several words (payload) to be transferred in a single operation. The difference between the two levels is mainly that one considers transfer delay times but the other does not. The timed one is called PVT (Programmer View Timed) and the un-timed one is PV.

These transfers are more suitable for SW source-code simulations than for an ISS. While source code transfers are commonly done using buffers, assembler code only makes single-word loads and stores. Thus, it cannot optimize the use of payloads.

In [10] a PVT approach based in Master-Slave libraries where the bus arbiter receives all requests is presented. In [11] an efficient environment is presented, based on Conservative Parallel Discrete Events, where the SystemC simulation clock is ignored in the simulation.

However, none of these methods are oriented to system refinement. All bus models are focused on the improvement of a single design level. In fact, there are few works oriented to models with several levels, where system refinement is possible. In [12] a multiple-level bus model for PV and BCA levels is presented. However, it is only focused on the bus and not on the bus interfaces or the bus protocol managers in the connected elements.

Furthermore, high-level bus models only model payload transfers in burst mode. That is, where several masters access the bus, a transfer cannot start until the previous one ends. However, in real operation, transfers can be interleaved, sharing the bus bandwidth.

Thus, although there have been several works on bus modeling at low levels (including BCA), the use of payloads can be improved. Thus, this work is focused on PV, and especially on PVT. This is the reason why a source-code approach has been selected to model SW instead of an ISS, as explained in section I.

A TLM2 bus model has been developed to address this issue. Although TLM2 is not a standard yet, it has been used for three main reasons. First, it represents an advance from TLM1 as a more complete specification, and more describing elements are provided. Secondly, this work tries to analyze the TLM version proposed, to check its benefits and find its limitations before the final version will be presented. Finally, the current draft can be considered stable, so minimal modifications are expected in the final version. Thus, minimal changes will be required to adapt it to the final TLM2 draft.

Using the TLM2 terminology, for data transfers, masters are the initiators, and slaves are the targets, but for interruptions it is the opposite: slaves are the initiators and masters the targets. Initiators call the transfer function and targets implement that function, so both processor and peripherals must implement some functions to allow communications.

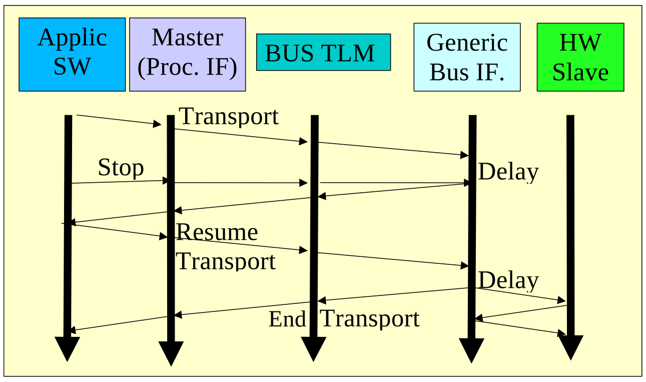

Fig. 1. Transfer modeling

In PVT, delay times have to be considered. Delays in transferring payloads can be caused by three elements: bus propagation delay, transfer time (time of single transfer x size of payload) and peripheral internal delay. Thus, a bus access (Figure 1).

The problem is that the system state can change during the transfer, modifying it. If the SW task that is performing the transfer is preempted or killed during the access, it has to be stopped or definitively aborted. To manage these situations, a protocol has to be implemented in masters and slaves. When a stop signal is received the slave has to inform about the amount of information accepted considering the time spent since the request arrived. When the task is re-scheduled, the transfer will be resumed to send the remaining information. When the task is killed, the transfer is aborted and the information is lost.

To implement this in an easy and portable way, the use of some interfaces is proposed. The use of these interfaces will be presented in section V.

Furthermore, to allow correct modeling, the first word of the payload should be received before considering the transference time. Some peripherals operate each word at the moment when it is received. The entire payload does not have to be received before computing.

Furthermore, the bus model has to allow several masters and slaves, as there can be several processors and peripherals in the same bus.

## III. INTERNAL BUS MODELING

The bus model implemented transports data and interruptions among several masters and slaves (Figure 2). To extract the target of each transfer, a memory map is included in the model. For interruptions, the bus deploys the request depending on the interruption number.

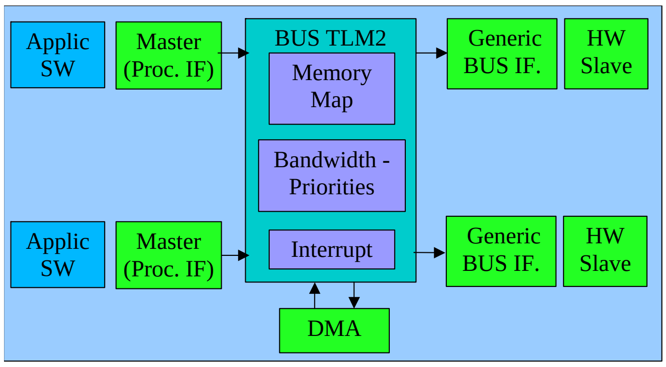

Fig. 2. TLM2 Bus model

For data transfer, each transfer has a memory address associated. The memory map contains the upper and lower bounds of memory addresses for each peripheral and identifies the target peripheral. Furthermore, data transfers consider bandwidths and priorities to model the bus.

As explained above, time modeling is one of the most interesting aspects in bus modeling. Although the propagation and peripheral delays can be estimated statically, the transfer time has to be defined during the simulation. This time depends on the amount of information transferred and the state of the bus. If there are several transfers at the same time, they will share the bus, and so the transfers will be slower.

To model these times, the bus proposed is based on defining bandwidths. A maximum bandwidth value is associated to each bus. Then, during the simulation this bandwidth is shared out among all current transfers. First, it is fairly divided among transfers with maximum priority depending on the bandwidth required by each transfer. Then, the remaining bandwidth is delivered to the remaining transfers depending on their priorities and requirements.

The required bandwidth is obtained considering the amount of information to be transferred and the ideal time of the transfer. For example, in a simple transfer, the processor will make a request to the bus for size “S” and time “0”. That is, the processor wants the transfer to be immediate. However, a common bus does not allow this. Thus, the corresponding peripheral receives a request of size “S” and time “S/B”, where “B” is the available bandwidth of the bus.

To divide the bandwidth correctly, each time a transfer ends or a new one starts, all current transfers with the same or lower priority are stopped. To stop the transfers, the same protocol explained in section II is used. Thus no new functionality is required for internal bus modeling. Then, the bandwidth is shared out again and the active transfers are resumed.

This technique also enables the chaining of buses. The request received by one bus considers the available bandwidth in the previous ones. Furthermore, this technique models the inefficiency provoked when connecting slower buses to faster ones.

## IV. USE OF TLM2.0 INTERFACES

To model both data transfers and interruptions, TLM2.0 interfaces and structs have been used.

### *A. Data transfers*

To model bus transfers one of the standard SystemC TLM2 interfaces, “tlm_annotated_transport_if”, has been used. The “transport (request, response, time)” function is called from the processor interface and served by the peripherals.

For the function parameters, the TLM2 structs “tlm_request struct”, and “tlm_response struct” have been used.

- tlm_request parameters:

- Mode: REGULAR, CONTROL and DEBUG
  - Regular mode: Data transfers
  - Control mode: Access to peripheral control and status
- Address (int): Uses the peripheral base address to identify the peripheral. Each bus model contains a memory map that contains the peripheral address and allows the bus to send each request to the specified peripheral. Memory addresses are loaded at the beginning of the simulation. Hot-plugging is also allowed. However, a hot-plug will probably require re-initializing the corresponding OS driver model. Plug and play mechanisms are not completely developed.
- Data (void*): Information transferred
- Priority: Used to assign the bus bandwidth when several transfers are done in parallel. The last bit has been used to indicate if the transfer is normal or in burst mode.
- Block mode: Informs about whether all words of the package will be sent to the same address or the address should be increased on each single word transferred. (Not implemented in the example)
- Others: command type (Read/write), data size, transference id and source id are also used.

- tlm_response parameters:

- Status: ok, error or no_response (peripheral informs that transference size = 0)
- Data, Priority, Size, Transference id, Source id are used as in the tlm_request.

### *B. Interruptions*

To transfer interruptions from the peripherals to the masters, the "tlm_nonblocking_put_if" interface has been used. Specifically the function "nb_put(irq)" is provided by the processor bus interface. The interruption is supposed not to require time to be delivered to the corresponding processor. This interface also contains two more functions. The first one is the "nb_can_put(irq)" that always return true, because it is considered that an IRQ can always be sent. The other one is "ok_to_put(irq)" that returns an event when the interrupt can be sent. However, it is not required as interruptions can always be sent.

This function calls the IRQ handler (creating a new POSIX process), informs the scheduler whether to pre-empt the current task, and asks the annotation engine to stop the time annotation of the pre-empted task. This solution avoids creating a new SC_THREAD to wait for new interruption calls. Instead, the "nb_put" function is called directly by the bus, using an "export" port, and the IRQ manager is executed.

### *C. Non standard TLM2 functions*

Sometimes, it will be necessary to abort or stop a previously decided transfer, because of a change on the system status. Thus, abort and stop functions have been added to the bus interface.

## V. BUS INTERFACES

However, the most tiresome problem when integrating a bus model into a system description is integrating the bus interfaces and protocols into all components during the different steps of the refinement flow. Each time a new bus model is required, all the peripheral connections have to be rewritten.

When analyzing the peripheral interfaces of all components of the same bus, it is usual that they are mostly similar, and only a few of them present some differences. This means that the main part of the protocol manager can be reused among the different modules.

Thus, the first possibility could be to integrate part of the protocol management within the bus model. However, this is unsuitable for two reasons. First, the protocol management will be part of the peripheral when it is implemented, so this functionality should be part of the component, and not of the channel. Secondly, this does not allow ad-hoc modifications for certain peripherals that require a specific protocol implementation.

The proposed methodology is presented in figure 3. First, this methodology creates a new sc_module for each abstraction level ("pvt_prot_manager" or "pv_prot_manager") that integrates the implementation of the TLM interfaces, defining the required ports and protocol functions to connect it to the bus. Furthermore, it provides the designer with communication functions that are independent of the internal bus protocol, to connect the peripheral descriptions and this new module.

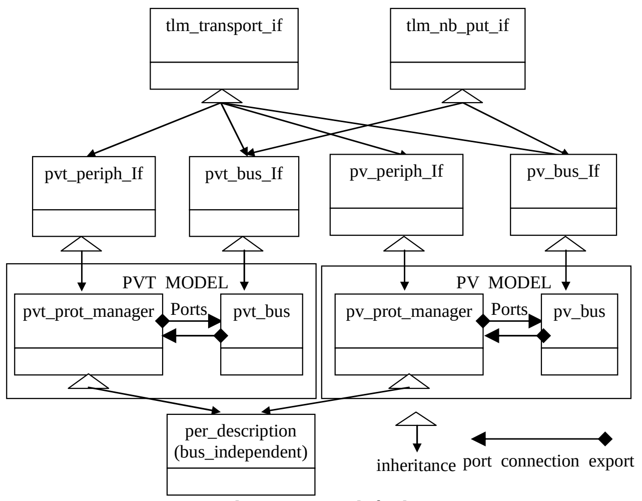

Fig. 3. Inheritance graph for bus connection

The new sc_module has to be inherited in all peripherals that want to access the bus ("per_description"). Thus, neither the ports nor the protocol functions have to be described explicitly in each module. They are inherited.

This technique presents one advantage with respect to using intermediate modules as transactors. Those peripherals that require modifying some of the standard protocol implementation can redefine the specific part and reuse the rest by overloading the corresponding function. It is not required to create a different transactor type for each especial bus communication.

The inherited module also provides the designer with an interface that is independent of the bus modeling level. Thus, a PV, PVT or BCA bus model can be used in the system simulation just replacing the inheritance.

To maintain the TLM2 standard interfacing capabilities, the connections between the protocol managers and the bus models ("pvt_bus" or "pv_bus") require an interface form the TLM2 standard. Thus, the protocol manager inherits the standard "tlm_transport_if" interface.

Apart from that, the port binding technique is improved. First, only some connections between the bus and the peripherals ports have to be done manually if the change of abstraction level requires modifying the number of bus ports. However, even this requirement can be automatically solved. To avoid this last manual modification, another specific function has been included in the interface module. This function receives a pointer to the bus as a parameter, and internally makes the port binding. Each interface module inherited at each modeling level has its own binding function, so the specific ports for this level are automatically bound.

To allow connection between the HW description and the bus interface, a new set of functions is defined in the pv/pvt_periph_if . This functions connects the peripheral description ("per_description") and the bus protocol manager inherited ("pvt_per_manager" or "pv_per_manager"). This set is used internally to the HW component. The use of this new set of functions, allows changing the inheritance to modify the communication level of abstraction without modifying the HW description. The set provides the functions presented in table 1.

The “read” and “write” functions return and put the data transferred by the bus at the indicated address. Functions “wait_read” and “wait_write” block the task until a read or write access is performed by the bus. An argument indicates if the function has to be unblocked at the beginning or at the end of the transference. Finally, “send_interrupt” can be used to send interruptions to the processors.

TABLE I
BUS INTERFACE SC_MODULE

```
class tlm_bus_module: public sc_module{
    // Ports and Ctor
   void bind_bus(bus){...}
    ...
    //Bus Protocol management
    void   transport (...){...}
       ...
   // HW interface
   int read (addr,data,size){...}
   int write (addr,data,size){...}

   int wait_read (bool){...}
   int wait_write (bool){...}

  int send_interrupt (int irq){...}
  ...
}
```

TABLE II
NEW MACROS TO INTRODUCE THE BUS INTERFACE

| | |
|---|---|
| *#define BUS_MODULE(name)* | *struct name: tlm_bus_module* |
| *#define BUS_CTOR(name)* | *… name(…): tlm_bus_module(...)* |

To automatically introduce these changes new macros following and extending the SystemC philosophy have been created. Thus, replacing the original SystemC macros by these new ones, the bus interface is automatically integrated into the module. These macros can be shown in table 2.

Another interesting point is the implementation of the functions to manage the bus protocol, especially the operation at aborts and stops. These functions are used to allow modeling task preemptions during payload transfers. PVT transfers implies considering transfer time specially with large payloads. The time required is annotated using a wait statement with the expected time. However, during this time, an unexpected event can make the payload transfer to be stopped or aborted. For example, it the SW task which is making the transfer is preempted or cancelled, the transaction, and thus, the wait statement, have to be cancelled.

At aborts, the transfer finishes, and the peripheral status is reset to accept new transfers. The peripheral answer is not important. Thus, the abort execution is done in zero time.

Stops are more complex. First, the peripheral has to inform about the amount of information it has accepted before the stop event. The peripheral answer can require some time, so the stop is not immediate. Furthermore, to allow continuing the transfer, the information received has to be stored in the peripheral. Peripheral state is changed to "in transfer" to avoid other incoming messages being considered as the continuation of the stopped communication, producing an incorrect operation.

The interface for bus masters is similar to the one presented above for peripherals. Instead of implementing the “transport” function, it includes the functions for interrupt management. Furthermore, it is connected to the OS model, so the user does not access it directly.

Finally, modules that are both masters and slaves at the same time, such as DMAs, can include both interfaces to allow sending and receiving requests.

The last element to be considered is how time delays are annotated in the transfer modeling. To maintain generality, the annotation has to be done when the values are received. Peripherals can decide to operate when each value is received or only when the entire payload has been accepted. That is, when the petition starts or after the wait time.

This is easy to model in write accesses, because the peripheral receives the request with the information, waits the corresponding time and returns an acknowledgement. However, in read accesses it is more complicated. The master makes the request and the slave sends the information. Thus, if the time wait is implemented in the slave, the master only receives the information at the end of the transfer, so it cannot operate during the transfer with the data received. If the wait statement is placed in the master, neither the slave nor the bus has information about the status of the current transfer.

To address this issue, read transfers have been implemented in a two-step sequence. First, the read is performed in zero time, and then a new special request is made. The first one performs the transfer, and sends the information to the master. The second request has no functional effect; it is used only to make the time annotation. The bus interfaces automatically manages these two-step transfers, so they are hidden to the user.

## VI. Example

To verify the technique proposed, an example has been implemented. This example models a Vocoder GSM[18]. The Vocoder is divided into two parts, one modeling the coder and the other one including the decoder.

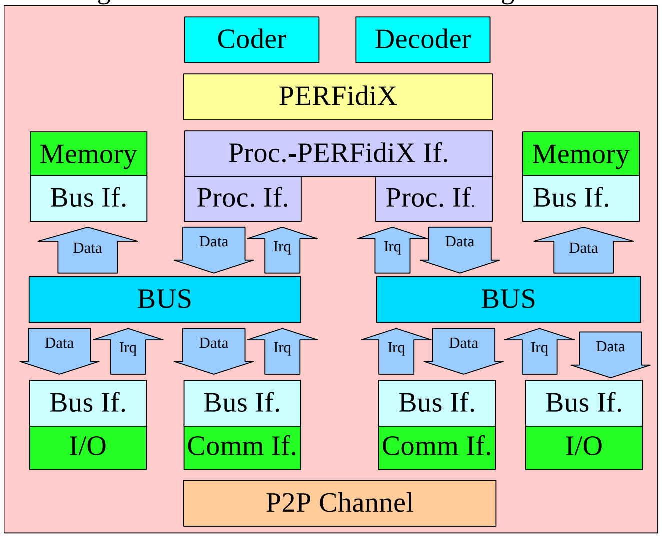

Fig. 4. GSM Vocoder model architecture

Each part has a bus with several elements connected (figure 4). The bus master is a processor. This processor executes the SW code that models the coding or decoding operations. To run the SW code, the tool PERFidiX has been used. This tool provides an OS model and obtains a timed simulation from the un-timed SW code.

The SW code is stored in a memory connected to the same bus. Although the source-code simulation does not require a memory model to run, these bus accesses have been modeled. The effect of these accesses can be really important, especially if there are several masters in the same bus, because collisions can reduce the execution speed. To model them, the number of load and store operations is dynamically estimated at the same time the code is executed based on statistical information.

The information received by the coder is generated in another peripheral that models the system input interface (I/O). The SW calls this peripheral when a new value must be coded. In the decoder case, the output is sent to another peripheral. This peripheral models the system output. It receives the decoder values as they are generated, and verifies if they are correct, comparing them with the original sequence provided to the coder.

Finally, a communication channel connects the coder and decoder. The selected channel is a NoC. To do that, a point-to-point channel has been integrated in the system. This channel is connected to a pair of communication peripherals, one placed on each bus.

Summarizing, there are two nodes with one bus, one processor and three peripherals, connected using the inherited interfaces. The operation is the following: The coder reads the input values from the I/O peripheral, encodes them and sends the codified values to the point-to-point channel. The decoder receives the values de-coded and transfers the results to the output peripheral. Once the decoder receives each coded frame, it sends and acknowledgement to the channel. Communication peripherals inform the software that new information is received using interruptions. PV and PVT bus models have been used. The substitution does not require modifications in the example code, only substituting the bus and the module inheritance in the macros in table II.

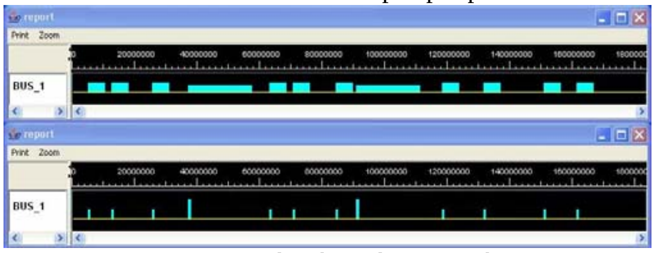

Fig. 5 PVT and PV bus utilization graphs

Once the simulation has finished the bus reports the graph with the bus occupation, as can be seen in figure 5. There, all transfers done through the bus can be shown, including its duration and bandwidth used.

## VI. Conclusions

This paper proposes a technique that allows design refinement at different TLM levels. It considers both the bus model and the peripheral bus protocol managers. This allows easy and partially automatic refinement of HW component bus interfaces. Furthermore, the performance of the SW tasks can be easily obtained with different requirements of accuracy/speed, by automatically modifying the bus abstraction level.

The bus extends the common functionality of PV/PVT bus models allowing interleaved payload transfers, considering bus bandwidths and allowing dynamically stopping and aborting transfers. This increases the accuracy of the model, especially when SW models consider preemption and interrupts during payload transfers.

To allow semiautomatic refinement in the bus connections, inheritance techniques have proven to be effective to replace the required bus protocol managers at the bus peripherals in an easy and fast way. Using a simple interface within the peripheral, common to all TLM description levels, the functionality required to perform requests and responses through the bus is automatically provided.

## VI. References

[1] A. Sangiovanni-Vincentelli and G. Martin: Platform-based design and software design methodology for embedded systems, IEEE Design and Test of Computers. Nov.-Dec., 2001
[2] A.A. Jerraya, S. Yoo, D. Verkest and N. When: “Embedded Software for SoC”, Springer, 2003
[3] IEEE 1666 standard, available at http://www.systemc.org
[4] OSCI Draft for SystemC TLM2, available at http://www.systemc.org
[5] F. Ghenassia (Ed.): “Transaction-Level Modeling with SystemC”, Springer, 2005.
[6] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi and M. Ponzino: “SystemC cosimulation and emulation of multiprocessor SoC design”, IEEE Computer, April, 2003.
[7] Y. Yi, D. Kim and S. Ha: “Fast and time-accurate cosimulation with OS scheduler modeling”, Design Automation of Embedded Systems, N.8, Springer, 2003
[8]S. Yoo, G. Nicolescu, LG. Gauthier and A.A. Jerraya: “Automatic generation of fast timed simulation models for operating systems in SoC design”, Proceedings of the Design, Automation and Test Conference, IEEE, 2002.
[9]H. Posadas, J. Ádamez, P. Sánchez, E. Villar and F. Blasco: “POSIX modeling in SystemC”, Proceedings of the Asian, South-Pacific Design, Automation Conference, IEEE, 2006.
[10] J. Lee & S. Park, “Orthogonalized communication architecture for MP-SoC with global bus”, System-on-Chip for Real-time Applications, 2005
[11] E. Viaud, F. Pecheux & A. Greiner: “An Efficient TLM/T Modeling and Simulation Environment Based on Conservative Parallel Discrete Event Principles”, DATE 2006
[12] W. Klingauf, R. Gunzel, O. Bringmann, P. Parfuntseu & M. Burton, “GreenBus - a generic interconnect fabric for transaction level modelling”, DAC 2006
[13] A. Hoffmann, R. Langridge, D. Machin, “SoC integration of programmable cores”, Int. Symp. on System-on-Chip, 2003
[14] M. Caldari, M. Conti, M. Coppola, S. Curaba, L. Pieralisi, C.Turchetti : “Transaction-level models for AMBA bus architecture using SystemC 2.0”, DATE, 2003.
[15] S. Pasricha, N. Dutt, M. Ben-Romdhane, “Fast exploration of bus-based on-chip communication architectures”, CODES + ISSS 2004.
[16] G. Schirner, R. Domer: “Result Oriented Modeling, a Novel Technique for Fast and Accurate TLM”, IEEE Transactions on Computer-Aided Design of Integrated Circuits, Volume PP, Issue 99, 2007
[17] I. Moussa, T. Grellier, G. Nguyen, “Exploring SW performance using SoC transaction-level modeling”, DATE.2003
[18] GSM Specification: EN 301.245, ETSI, December, 1997.