HYBRID FUNCTIONAL VERIFICATION OF A USB HOST CONTROLLER

Primož Puhar[1], Andrej Žemva[2]

[1]LEA, d.o.o., Lesce, Slovenia

[2]University of Ljubljana, Faculty of Electrical Engineering, Slovenia

Key words: functional verification, SystemC, TLM, ABV, SCV, simulation, USB

Abstract: With everyday growing demands, complexity of electronic devices has been constantly increasing. Functional verification has become the major bottleneck in the design and verification flow. In order to respond to modern demands, new devices are made of standard pre-verified reusable IP blocks created by using abstract TLM. The paper proposes a three-step design and verification flow based on a reusable test bench. It enables a short design time for a fast-simulating functionally-verified TL model, to be used in early SW development, and a functionally-verified RTL model, ready for HW implementation. The approach is demonstrated on a USB host controller design.

Hibridno funkcionalno preverjanje USB gostitelj krmilnika

Kjučne besede: funkcionalno preverjanje, SystemC, TLM, ABV, SCV, simulacija, USB

Izvleček: Zaradi vsak dan večjih zahtev postajajo elektronske naprave vse bolj kompleksne. Funkcionalno preverjanje je zato postalo najožje grlo v postopku načrtovanja in verifikacije le-teh. Da bi se prilagodili modernim zahtevam, moramo nove naprave sestavljati iz standardnih pred-preverjenih IP blokov, ki smo jih ustvarili s pomočjo abstraktnega TLM. Članek predlaga tristopenjski postopek za načrtovanje in preverjanje, ki temelji na večkratno uporabnem testu. Postopek omogoča načrtovanje hitrega funkcionalno preverjenega TL modela, uporabnega za zgodnji začetek načrtovanja programske opreme. Dodatno omogoča načrtovanje funkcionalno preverjenega RTL modela, pripravljenega za implementacijo. Pristop je prikazan na primeru USB gostitelj krmilnika.

1 Introduction

Though electronic devices, such as smartphones, multimedia players and others, already combine a lot of different functions, the market incessantly demands new functionalities like new audio and video decoders, new accessibility features and support for new interfaces. In future, functionality of any single device shall have to be improved, meaning that its complexity will be drastically increased. A higher level of complexity requires more effort in a device design and verification. Considering also the extreme time-to-market pressures, there is no doubt that new advanced solutions shall have to be provided.

Functions of electronic devices can be assured either by software (SW) or hardware (HW) components or a combination of both. Though there have been new verification techniques developed, simulation is still the most used approach to functional verification. By using slow-simulating Register Transfer Level (RTL) HW models verification times have increased to the level when they now take up to 70% of the device design time and cost /1 – 3/.

In order to respond to the constantly growing new demands, several approaches have been proposed. By using standard, already verified building blocks in new designs, the verification time can be considerably reduced. This applies to both the HW and SW blocks. Another approach to reducing the device design time is the Transaction Level (TL) Modeling (TLM) /1/. It can be used for HW/SW modeling, co-design and co-verification. Compared to the HW model described at the RTL, TLM uses abstract communication with approximate timing thus enabling faster simulation and verification /4/. Another reason to use TLM is availability of the executable models early in the development cycle. They confirm the expected functionality and can be used for design space exploration and early SW design.

Though the TL models can be easily debugged due to their abstractness, the Assertion-Based Verification (ABV) is added to the Simulation-Based Verification (SBV) to allow for faster and easier debugging. The use of assertions helps pinpointing the bug in the design more than SBV itself.

The focus of the proposed Design and Verification Flow (DVF) is on the HW-supported functionality. It captures all the above improvements, i.e. TLM, SBV and ABV for new Intellectual Property (IP) block design.

There have been several design and verification approaches proposed in literature. Vaumorin et al. /5/ favourise the design flow from a specification through SystemC and RTL to the FPGA implementation. Verification takes place at several stages while RTL design described in the HW Description Language (HDL) is verified by using co-simulation with RTL described in SystemC. No assertions are used.

Wei et al. /2/ describe the design and verification scheme for IEEE 802.15.3 MAC and Carbognani et al. /6/ for ARM AMBA. Both groups of authors present their model in TLM

and RTL with support for the reusable test bench; none uses ABV.

Habibi et al. /7/ propose a verification approach with the SystemC model translated to the AsmL language. Only ABV is used. Assertions are described with the PSL language.

Our focus in this paper will be on a hybrid DVF of an IP block. Our approach was verified on a Universal Serial Bus (USB) host controller design constructed according to /11/.

2 Design and verification flow

In this section, we present a three-step DVF (Fig. 2). It starts with a non-formal specification in a written or verbal form of the model behavior and operating conditions. The model behavior specification is used for model description and test-bench construction and the operating condition specification is used for the functional-coverage metric (FCM) definition.

In the first step, the Verification Environment (VE) is constructed enabling efficient Test-Bench (TB) generation to cover the FCM. Therefore, the first FCM is defined according to the model operating conditions. The impact of well-defined operating conditions on the verification quality and time consumed by it is considerable. If some operating conditions are not captured by the specification, the model is not verified for them. If operating conditions are set wider than necessary, VE performs verification for the non-existent conditions and unnecessarily wastes time for that (Fig. 1).

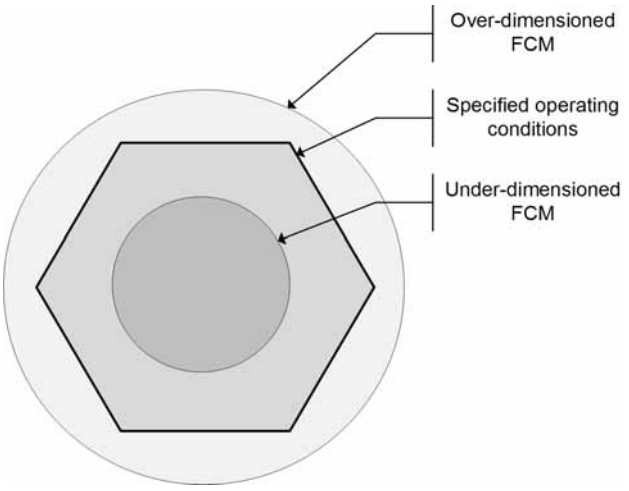

Fig. 1: Model operating conditions space

In the same step, the TB that satisfies the defined FCM is constructed. TB is responsible for generating test vectors that are later applied to the model inputs.

In the following step, the model is translated from the non-formal specification to an executable specification. It is then placed into VE as a Device Under Verification (DUV). DUV and VE are described using TLM which can be employed on many abstraction levels. TLM with an approximate timing (also known as "Programmer's View with Timing" - PVT) was selected for DVF. TLM is therefore approximate-timed and uses abstract transactions for communication. Due to its abstractness, TLM enables a shorter development time and faster simulation.

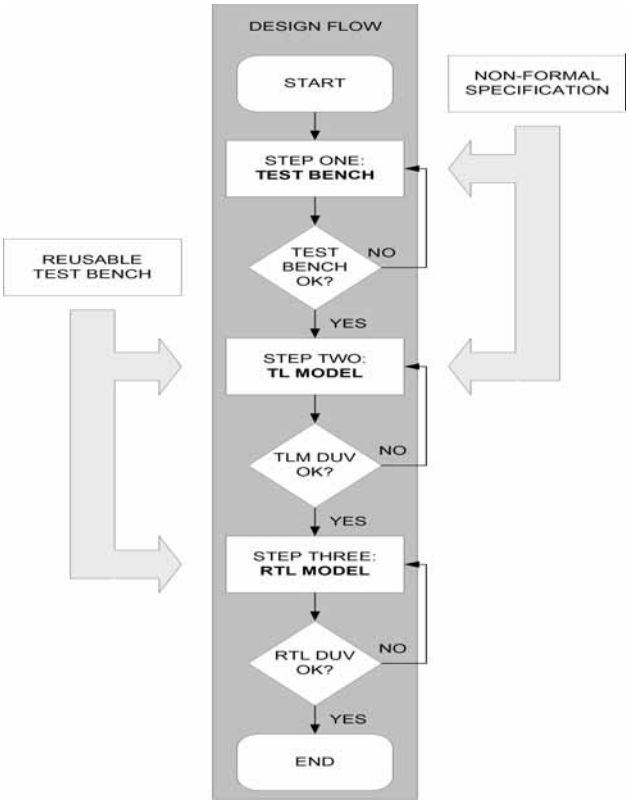

Fig. 2: Design and verification flow

After DUV is placed into VE, the TB composed in the previous step is applied. TL DUV is corrected until it positively passes the test bench.

At the end of the second step, a functionally-verified HW model is available for design-space exploration and early SW development. The TB responses are used as a reference for RTL refinement (golden model).

In order to implement HW with standard tools, the DUV model has to be refined to RTL. This is achieved in the third step. The RTL model is cycle-timed and uses four-state signal logic ('0', 1', Z' or 'X') for communication. In order to reuse TB and the VE on the RTL model, an interface referred to as transactor has to be developed. The transactor adapts the RTL model to the TL environment by translating abstract communication to signal-level communication and provides synchronization. The DUV RTL model is confirmed OK when it positively passes TB.

After using the proposed DVF, the functionally verified RTL model is ready for implementation with standard tools.

2.1 Functional verification

In the next chapters, each step will be explained in detail. The first we shall deal with will be the FCM definition step.

The functional coverage definitions are extracted from the operating conditions specification. FCM states which functions have been exercised. When there are several variables in a function, the problem becomes more complex. The question is whether to test each function with any possible combination of variables or to group the variable values first? An exhaustive verification procedure is much time consuming and therefore expensive. Our solution is to reduce the function coverage space and speed-up the verification process. This allows us to detect the major errors in the design and to reduce the verification time by a few decades, though some of the minor errors might be left undetected. Under the current time-to-market pressures, manufacturers find it hard to perform exhaustive verification (Fig. 3).

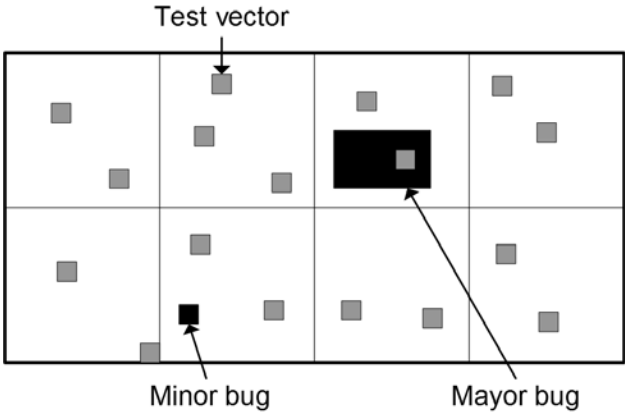

Fig. 3: Verification space

To make it clearer, let's assume the function F(A,B) variables, where A and B can be assigned any value from 0 to 31. The value range is linearly divided into four testing ranges (bags) defined as (0, 7), (8, 15), (16, 23) and (24, 31). Two examples are shown in Fig. 4. If the function parameter is tested within a certain bag, the bag is said to have a hit.

Ranges and numbers of bags of each variable may vary. The number of bags per variable defines the variable resolution. When all the defined bags have at least the predefined number of hits, a full functional coverage is being achieved /9/.

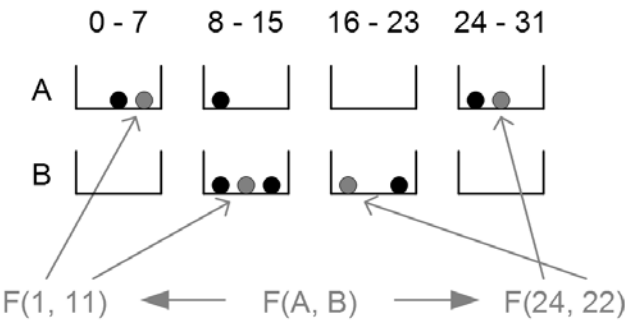

Fig. 4: Function bags

The variable resolutions and the minimum number of hits for a full functional coverage define the verification effectiveness (Fig. 5). When more exhaustive verification is required, higher resolution is selected and more combinations of test vectors are run. On the other hand, when fast and consequently less exhaustive verification is required, lower resolution is selected and fewer combinations of test vectors are run.

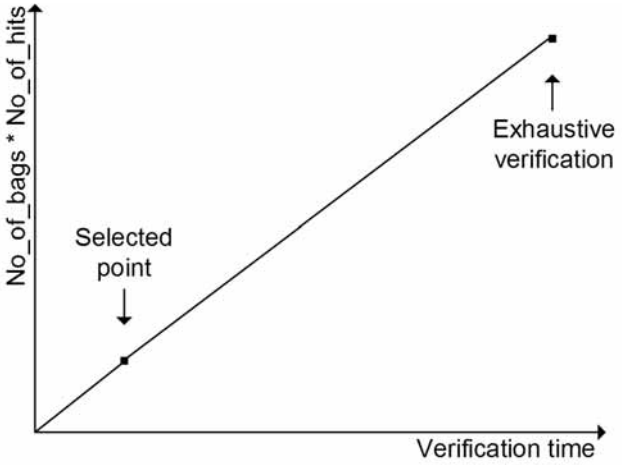

Fig. 5: Parameter selection graph

Since modern devices consist mostly of programmable logic, bug-fixes can be issued in case of missed bugs during first verification.

Verification of DUV runs at TL. The VE in Fig. 6 consists of a test controller, simulator, monitor, evaluator, master (DUV) and multiple slaves when required. Verification runs as follows. The stimulator sends a request to DUV. DUV processes the given request by sending new requests to the slave. After the slave returns response, DUV returns it to the stimulator (Fig. 7). The monitor monitors the stimulator and DUV transactions and forwards them to the evaluator. The evaluator checks correctness of these requests and measures the verification coverage. The test controller controls the environment according to the TB /4/.

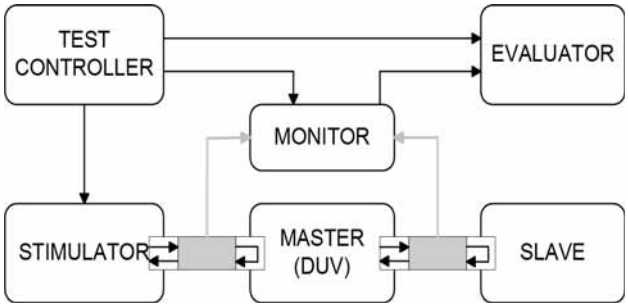

Fig. 6: Verification environment

Test vectors are generated by using the SystemC Verification (SCV) library which enables Constrained Randomization (CR). It shortens the verification time when the pro-

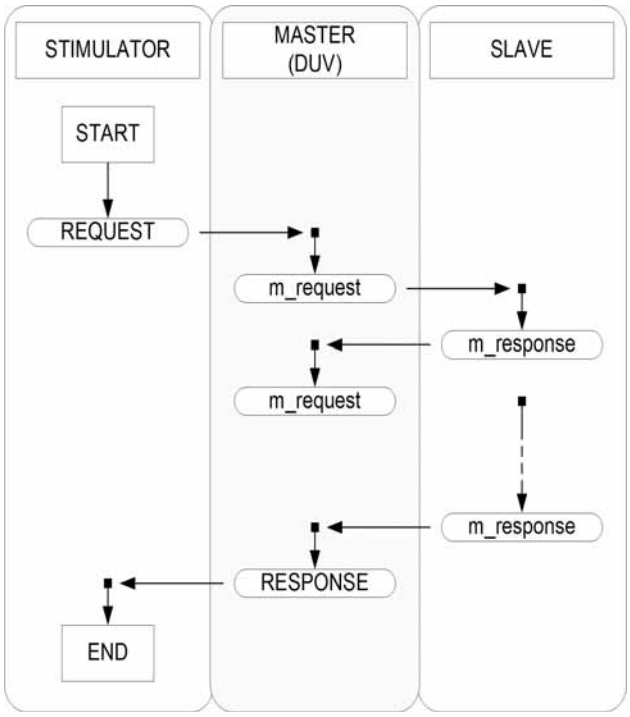

Fig. 7: Transaction-time diagram

posed FCM needs to be used /10/. CR makes it possible to limit test vector generation to a certain range and also to define inter-variable dependencies.

DVF verification consists of SBV and ABV. SBV is responsible for FCM definition, TB construction and storage of DUV requests. It is up to the designer to detect bugs in these requests. ABV is on the other hand responsible for assertion definition which enables automatic bug detection.

Since test vector generation is random, the number of test vectors for full functional coverage (test vector set size) varies. Different test vector sets lead to a full functional coverage. Test vector sets can be combined in the distribution graph.

2.2 TLM & SystemC

SystemC is a C++ library which enables HW modeling. Since it originates from the SW world and supports the HW description, it represents the basis for HW/SW co-design and co-verification.

Similar to other HW developing languages, SystemC also supports architectural design. Modules can be described and connected via input and output ports. The base for the module description is class SC_MODULE. This class consists of several ports, processes and other modules.

SystemC enables separate modeling for computation and communication. Computation is modeled by processes described using either SC_THREAD or SC_METHOD. The computer thread-like processes are described using SC_THREAD. They run all the time and can only be delayed. The RTL-like processes are described using SC_METHOD and are executed only on a trigger event. The trigger events like clock signal are listed in the sensitivity list.

Communication is modeled by channels connected between two ports. The data transmitted over the channels can be very simple like a Boolean value or complex like a second video sample /1, 7/.

Separate modeling of computation and communication is the basis for TLM. The only process running inside the TL model is described using SC_THREAD. From this process, supporting functions are called. The process is used for communication to the VE, while the supporting functions describe functionality and transact with slaves.

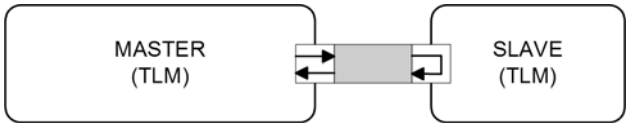

Fig. 8: TLM block scheme

TLM uses request - response transactions for communication between modules. Transactions are abstract data types described by using the custom C++ class. Communication runs as follows. An initiator (master) sends a request to a target and the target (slave) returns the response (Fig. 8).

The process time can be measured in real time (seconds) or in clock cycles. The more the types used in modeling are abstract, the less time is required to simulate the design.

As any other C++ class, the TL model too, is instantiated as an object on the top level and connected to a slave.

2.3 RTL refinement

The RTL model can be described using SystemC or standard HDL (VHDL, Verilog).

The SystemC RTL model is refined from the TL model as shown in Fig. 10. Since the architecture has already been defined at TL, the RTL refinement requires less effort than the RTL design from the start. During refinement, supporting functions are extracted and modeled as SC_METHOD processes. In contrast to the TL model, where transactions are used for communication, the RTL model communicates by 4-level signal buses. When describing the RTL model, each register or the finite-state machine (FSM) has to be described using its own SC_METHOD process.

The RTL model in HDL is identical to the RTL model in SystemC. There are few differences though; when the SystemC model is simulated using any standard compiler, a mixed-language environment is required for HDL model simulation within the SystemC VE (Aldec Riviera, Matlab).

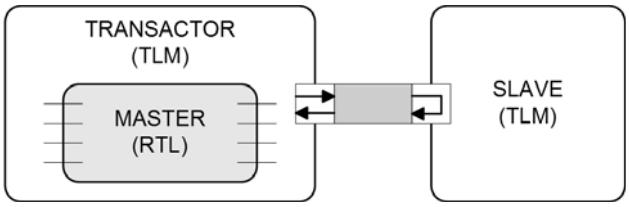

Fig. 9: RTL modeling with transactor

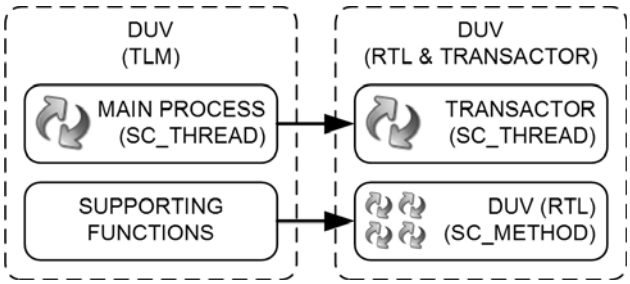

Fig. 10: TL to RTL refinement

In order to connect the less abstract cycle-accurate RTL model to the TL TB, a transactor should be constructed (Fig. 9). The transactor is a process, modeled as an SC_THREAD, originally used as the process in the TL model (Fig. 10). Its duty is to translate requests from the VE into the signal level, signals into requests for a slave, responses from the slave into the signal level, and finally signals into responses for the VE.

3 Universal Serial Bus

USB is a serial-bus standard to interface devices. Some of its features are: fast data transfer, plug&play capabilities and providing power to low-power devices. It is intended to help retire all legacy varieties of serial and parallel ports. USB connects computer peripherals such as mouse devices, keyboards, PDAs, scanners, digital cameras, printers, personal media players, and flash drives. For many of those devices USB has become the standard connection method. USB was originally designed for personal computers, but it has become commonplace on other devices such as PDAs and video game consoles. In 2004, there were about 1 billion USB devices in the world.

USB specification 1.1 was released in 1996. It includes "low-speed" and "full-speed" data rates. A new specification 2.0 was released in 2000 with some additional features. Among them is the "high-speed" data transfer.

3.1 USB host controller

The USB system has a star topology with a host in the center. The host connects up to 127 functions (devices), with the hub acting as a host for the next level (Fig. 11). Up to five levels are supported. There are several types of devices: hub, human-interface device, mass-storage device, printer, audio, video devices and others. The USB system has a master-slave organization where the master initiates the communication to which the slave can respond.

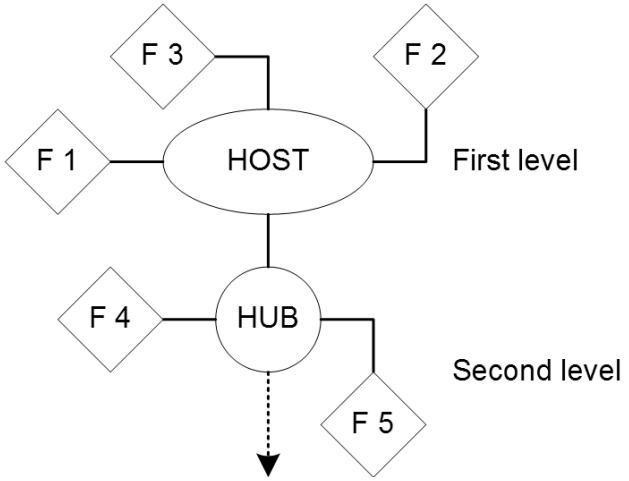

Fig. 11: USB system topography

In the startup operation the host first performs the enumeration of the connected devices and assigns addresses. It then reads the device properties like the device type and number of endpoints. Thereupon, it is ready to transfer the data to or from the devices.

The means to transfer the data are the device endpoints. To enable communication, the host controller creates virtual pipes to these endpoints.

3.2 USB host controller layers

The USB host controller consists of several layers of functionality. They can be implemented either in SW or HW or in a combination of both.

The layers of the USB host controller can be presented in many different ways. We will define layers as shown in Fig. 12. The first layer takes care of the USB electrical part like serialization and new device detection. The second layer composes the packet where the data is added the synchronization field (Sync) and "end of packet" (EOP). The third layer generates the packet data and makes the cyclic redundancy check (CRC) when required. The fourth layer provides different types of transfers. The fifth layer is the functionality layer for connection to the endpoints. The sixth and upper layers are application layers.

When some data from a certain function is needed for an application, a pipe is established. The Pipe layer then calls a specific transfer type, the Packet layer prepares the data required for the operation and adds CRC. The packet is completed with Sync and EOP in the Frame layer. The Base layer serializes the complete data and sends it to the slaves.

3.3 USB host controller model

Our model doesn't include all the features described in the specification, but only certain features from the USB

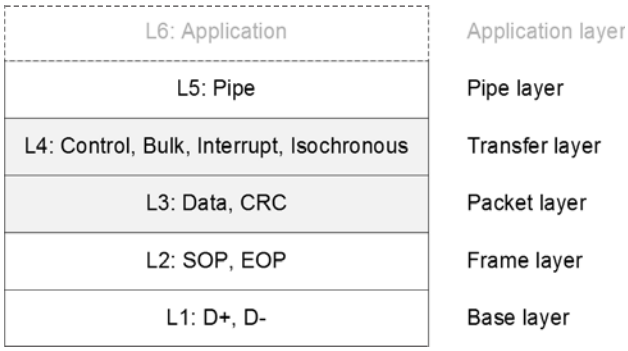

Fig. 12: USB host controller layers

specification 1.1. The model was developed with intention to show how only some selected features of certain layers can be modeled. It allows for an easy upgrading with other features and includes parts of the Packet and Transfer layers (L3 and L4) given in Chapter 8 ("Protocol Layer") of the specification /12/.

First, let's take a look at the common USB packet fields, i.e. the Sync field, Packet ID (PID) field, address (ADDR) field, endpoint (ENDP) field, data (DATA) field, CRC field and EOP field. The Sync field is responsible for synchronization, the PID field identifies the packet, the ADDR field indicates the device the packet is designated for, the ENDP field specifies the device endpoint, the DATA field contains the data, the CRC field detects errors and the EOP field defines the end of the packet.

The Packet layer distinguishes between the Token, Data and Handshake packets (Fig. 13). There are some other packet types defined in the USB specification which are beyond the frame of this paper. Each packet type has a different structure. The Token packet consists of the Sync, PID, ADDR, ENDP, CRC and EOP fields, the Data packet of the Sync, PID, DATA, CRC and EOP fields and the Handshake packet of the Sync, PID and EOP fields. Since the Sync and EOP fields occupy every packet, we extracted their assembly from the Packet to a separate Frame layer and will not model them.

Two types of CRC are required for the USB packet. The first is the CRC5 type and is used for CRC calculation of the Token packet. The second one is the standard CRC16 type and is used in CRC calculation of the Data packet. CRC will not be calculated by our model. The abstract model will specify only the CRC type.

As already mentioned above, PID is used to identify the packet. It first identifies whether the packet is of the Token, Data or the Handshake type. It then distinguishes between the SETUP, IN, OUT and the SOF Token packet type, the DATA0, DATA1, DATA2 and the MDATA Data packet type, and the ACK, NAK, STALL and the NYET Handshake packet type. PID occupies eight bits of each packet. PID will be modeled on the abstract level with the PID type only. The reader can find out more on CRC calculation and PID in /11/.

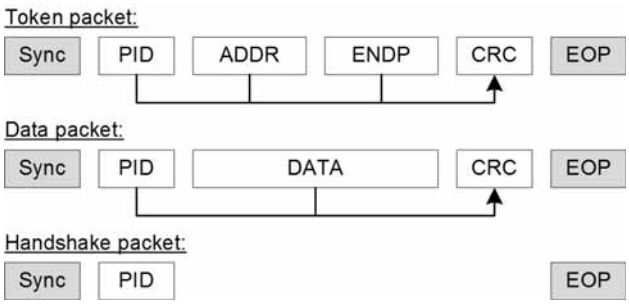

Fig. 13: USB packets and fields

The Transfer layer distinguishes between the four data-flow types: Control, Bulk, Interrupt and Isochronous transfer. The Control transfer is used for properties and status recognition, the Bulk transfer for large data, the Interrupt transfer for interrupts, and the Isochronous transfer for audio and video stream.

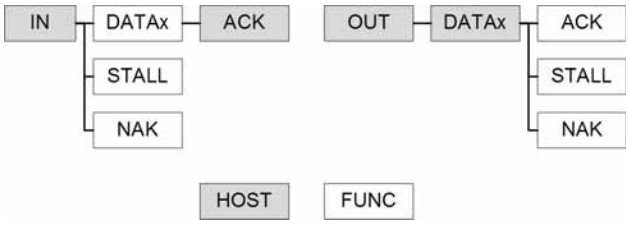

Fig. 14: USB Bulk transfers

Each of the data-flow types has a defined data flow. Since we only modeled the Bulk transfer, we will focus on this data-flow type alone. Bulk transfer can be the IN (read) or OUT (write) transfer. The IN Bulk transfer starts with the host issuing the IN Token packet. The addressed function responds with the Data packet containing the data from the target endpoint. In this case, the host responds with the ACK Handshake packet. In case the addressed function is busy, it responds with the STALL or NAK Handshake packet. The OUT Bulk transfer starts with the host issuing the OUT Token packet and a following Data packet. The function responds with the ACK Handshake packet on a successful reception and with the STALL or NAK one on an unsuccessful reception (Fig. 14). To enable better presentation, our model will only support the DATA0 and the DATA1 Data packet type and the ACK Handshake type. As said above, further functionality can be added later.

3.4 USB function model

The USB function model combines 128 functions with 16 endpoints each. In order to verify the complete specter of functions, the function model responds to all addresses and all endpoints.

3.5 Verification flow

During DVF, the stimulator is used to generate test vectors which model requests from a higher (Pipe) layer. They con-

sist of variables fAddr, fData, fLength, fEndPoint, fDirection and fDataSlot (Fig. 15). The fAddr variable targets the 7-bit ADDR packet field and varies from 0 to 127. The fEndPoint variable targets the 4-bit ENDP packet field and varies from 0 to 15. The fData variable targets the DATA field whose size can at the "full-speed" USB Bulk transfer vary from 0 to 1023 bytes of data. The fData variable is modeled with 1024 bytes of data while the actual size of the transmitted data is defined with fLength. The fDirection and the fDataSlot variables can hold values 0 or 1. For the first variable, value 0 specifies the IN transfer and value 1 the OUT transfer. For the second variable, value 0 specifies the use of the DATA0 transfer and value 1 the use of the DATA1 transfer.

```
class Function {
public:
    unsigned fAddr;
    unsigned char fData[1024];
    unsigned fLength;
    unsigned fEndPoint;
    bool fDirection;
    bool fDataSlot;

    // Constructors
    Function() {}
};
```

Fig. 15: Test vector model

In chapter 3.1, we showed that FCM can have different variable resolutions and different amounts of the required hits per bag for a full functional coverage. We stated that the verification time depends upon the selection of these parameters. Therefore, we ran four different combinations of these two parameters. We selected two different variable resolutions and ran them with a minimum of 100 and 200 hits per bag. The different resolutions are presented in Table 1. The different FCMs are labeled FCM1A (1 for 100 hits, A for resolution A), FCM2A, FCM1B and FCM2B.

Each of the four defined FCMs has many different test vector sets that lead to a full functional coverage. From these test vector sets, the average test vector set size can be calculated. Also, distribution of the test vector set sizes for each FCM can be presented.

Table 1: Resolutions of variables

Variable	Range	Resolution A	Resolution B
fAddr	0 – 127	8	16
fLength	0 – 1023	8	16
fEndPoint	0 – 15	4	8
fDirection	0, 1	2	2
fDataSlot	0, 1	2	2

```
class Packet {
public:
    enum PIDType {
        USB_NULL, USB_OUT, USB_IN,
        USB_SOF, USB_SETUP, USB_DATA0,
        USB_DATA1, USB_DATA2, USB_MDATA,
        USB_ACK, USB_NAK, USB_STALL,
        USB_NYET};
    enum CRCType {CRC5, CRC16};

    unsigned pMask [5];
    PIDType pPID;
    unsigned pAddr;
    unsigned char pData[1024];
    unsigned pEndP;
    CRCType pCRC;
    unsigned pLength;

    // Constructors
    Packet() : pPID(USB_NULL) {}
};
```

Fig. 16: Abstract packet model

The USB host and USB function communicate via the abstract packet. The packet fields are modeled using the following variables: pMask, pPID, pAddr, pData, pEndP and pCRC and pLength (Fig. 16). The pMask variable masks the presence of the major five variables and the pLength variable defines the actual size of the pData variable. The abstract packets are stored to a file and represent a golden reference of SBV. Due to the large amount of the overall data, the data is modeled using only its length (pLength). The complete data (pData) is verified using ABV.

For ABV, five assertions were implemented in the VE. The assertion monitor checks for correct packet contents, sequence of packets, Address, Data, Slot & Length and Direction & Destination. Assertions are checked on the packet sent from the host and are logged for further analysis. Assertions that have not been met report an error.

3.6 Results

Using the proposed methodology, the design of the USB host controller models was efficient. We spent 16 hours for the VE description, 30 hours for the TL model description and 12 hours for the RTL model refinement.

The minimum amount of test vectors for full functional coverage (minimum test vector set size) can be calculated by multiplication of the required number of hits per bag and the maximum number of bags per variable of a defined FCM. The numbers are 800 for FCM1A, 1600 for FCM1B and FCM2A and 3200 for FCM2B. After exercising 1000 random test vector sets, we calculated the average test vector set sizes. The results are presented in Table 2. We

also compared deviations of the average test vector set sizes from the minimum test vector set sizes in percents.

Table 2: Average test vector set sizes

FCM	Average size	Deviation
FCM1A	941	17.6%
FCM2A	1796	12.2%
FCM1B	1941	21.3%
FCM2A	3684	15.1%

Distribution for FCM1A is shown in Fig. 17. The values were grouped for better presentation. The step for the x axis is 2% of the minimum test vector set size; for FCM1A the number is 16. The y axis shows the number of test vectors within the group. As seen from this particular case, there were 129 out of 1000 test vector set sizes between 896 and 911 (labeled "904").

To have results from different FCMs better analyzed, we normalized the results to the minimum test vector set size. The normalized results are shown in Fig. 18. From these results and those from Table 2, FCMs with resolution A converge faster than FCMs with resolution B. Also, FCMs with the required 200 hits per bag converge faster than FCMs with the required 100 hits per bag. We can conclude that FCM2A finishes faster than FCM1B, although it might miss more bugs than the latter. For better interpretation of these results, a more in-depth study would be needed since they depend upon SCV CR.

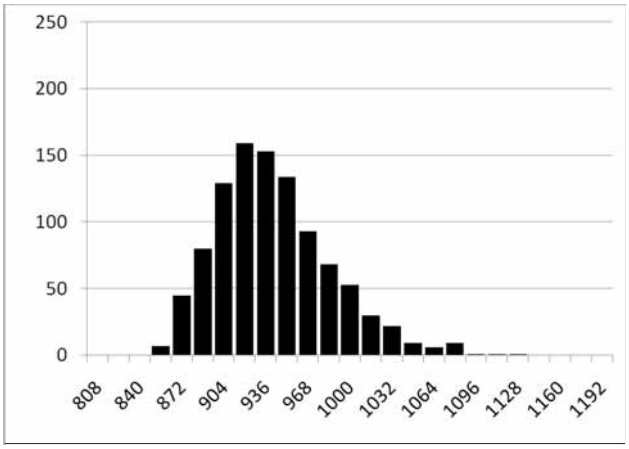

Fig. 17: FCM1A distribution chart

By using DVF, we created fast-simulating TL and RTL models of the USB host controller with the described features. The RTL models are described by using SystemC and VHDL. Table 3 shows that the TL model in average simulates six times faster than the RTL models. Since the design used in our verification was simple, the absolute time difference is small. If we added all the features from the USB specification 2.0, the verification times would be much longer and the difference would be more obvious.

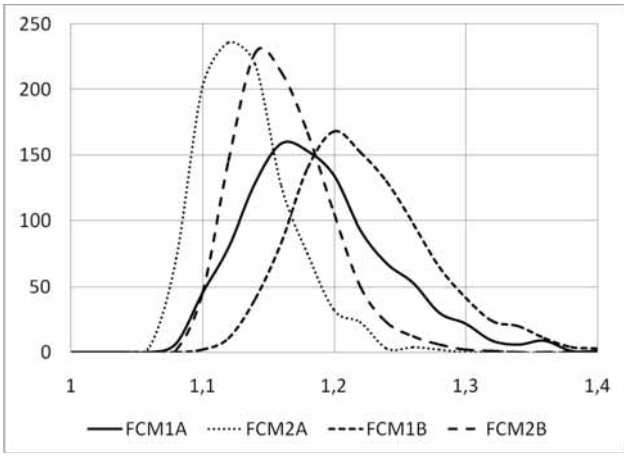

Fig. 18: Normalized distribution chart

Table 3: Average verification times of FCM1A

TLM	RTL	
	SystemC	VHDL
1,2s	6,7s	7,8s

Compared to /13/, the USB host controller shows several differences. Its verification converges much faster than that of /13/. The reason for it can be found in the inter-independent variables of the USB host controller, while those of /13/ are inter-dependent. This is a great challenge for SCV CR. Another difference is in the scope of the design verified by using SBV compared to the share verified by using ABV. The USB host controller design is a good example showing how both SBV and ABV can be efficiently used for verification, whereas using ABV for verification of /13/ is needless.

For simulation purposes, we used a computer with the Intel Pentium M 740 processor running at 1730 MHz with 1 GB of memory. The wall time was used for time measurement.

4 Conclusion and future work

In this paper we propose a hybrid three-step design and verification flow. We prove correctness of our approach on a simplified USB host controller. Further functionality can be added and verified at any time. It can be realized by either SW or HW or a combination of both. For instance, the Transfer layer was implemented in SW, and the Packet layer in HW. For realization in SW, the RTL refinement is not required.

One of the reasons to adopt the proposed approach is to shorten the design time for the verified TL model, which in our case study was 30 hours. Another 16 hours were required for the VE and another 12 hours for the RTL refinement.

We showed the free choice of FCM at the expense of possible missed bugs. Also, ABV was very helpful at pinpointing the bugs by means of which the design took us less time.

The use of the proposed DVF requires no special tools. TL and SystemC RTL models description and simulation are made only by a free C++ compiler. Since no SystemC implementation tool is currently available, the VHDL RTL model in combination with mixed-language simulators is also required.

In our case study we show that the TL model simulates six times faster than the RTL model. The simulation speed becomes more relevant at more complex models that take more time to simulate.

Future work will focus on improving the proposed coverage metric and convergence towards full coverage. The USB host controller model will be added further functionality in order to explore the described FCMs with more complex models. Other nonlinear FCMs will be studied so as to optimize the verification.

5 References

/1/ S. Swan, "SystemC Transaction Level Models and RTL Verification", 43rd ACM/IEEE Design Automation Conference, pp. 90-92, 2006.

/2/ Y. Wei, H. Guanghui, X. Ningyi, Z. Zucheng, "SystemC Transaction Level Modeling and Verification of IEEE 802.15.3 MAC", International Conference on Communications, Circuits and Systems Proceedings, pp. 2554-2558, 2006.

/3/ S. Tasiran, K. Keutzer, "Coverage metrics for functional validation of hardware designs", IEEE Design & Test of Computers, vol. 18, no. 4, pp. 36-45, 2001.

/4/ F. Ghenassia, "Transaction Level Modeling with SystemC", Springer, 2005.

/5/ E. Vaumorin, T. Romanteau, "From Behavioral to RTL Design Flow in SystemC", electronic file available at https://www.systemc.org/, 2004.

/6/ F. Carbognani, C. K. Lennard, C. N. Ip, et al, "Qualifying precision of abstract SystemC models using the SystemC Verification Standard", Design, Automation and Test in Europe Conference and Exhibition, pp. 88-94, 2003.

/7/ A. Habibi, S. Tahar, "Design and Verification of SystemC Transaction-Level Models", Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 14, no. 1, pp. 57 - 68, 2005.

/8/ T. Grotker, S. Liao, G. Martin, S. Swan, "System Design with SystemC", Kluwer Academic Publishers, 2002.

/9/ R. Siegmund, U. Hensel, A. Herrholz, et al, "A Functional Coverage Prototype for SystemC-based Verification of Chipset Designs", 9th European SystemC Users Group Meeting, electronic file available at http://www-ti.informatik.uni-tuebingen.de/~systemc/, 2004.

/10/ Members of the SystemC Verification Working Group, "SystemC Verification Standard Specification", electronic file available at https://www.systemc.org/, 2003.

/11/ USB Specification 2.0, electronic file available at http://www.usb.org/developers/ docs/, 2000.

/12/ C. Peacock, "USB in a Nutshell", electronic file available at http://www.beyondlogic.org/ usbnutshell/usb-in-a-nutshell.pdf, 2002.

/13/ P. Puhar, A. Žemva, "Simulation-based functional verification of a video processing IP block", 43th MIDEM Conference Proceedings, pp. 171-176, 2007.

Primož Puhar, B.Sc.
LEA d.o.o.
Finžgarjeva 1A, 4248 Lesce
primoz.puhar@lea.si

Prof. Dr. Andrej Žemva,
University of Ljubljana
Faculty of Electrical Engineering,
Tržaška 25, 1000 Ljubljana, Slovenia

Prispelo (Arrived): 04.02.08 *Sprejeto (Accepted):* 28.5.08