Preprint typeset in JINST style - HYPER VERSION

Architectural modeling of pixel readout chips Velopix and Timepix3

Tuomas Poikela[ab*], Juha Plosila[a], Tomi Westerlund[a], Jan Buytaert[b], Michael Campbell[b], Xavi Llopart[b], Richard Plackett[b], Ken Wyllie[b], Martin van Beuzekom[c], Vladimir Gromov[c], Ruud Kluit[c], Francesco Zappon[c], Vladimir Zivkovic[c], Christoph Brezina[d], Klaus Desch[d], Xiaochao Fang[d] and Andre Kruth[d]

[a] *University of Turku, Department of Information Technology*
FI-20014 Turun yliopisto, Finland
[b] *CERN,*
1211 Geneve, Switzerland
[c] *Nikhef,*
Science Park 105, 1098 XG Amsterdam, the Netherlands
[d] *University of Bonn,*
Nussallee 12, D-53115 Bonn, Germany
E-mail: tuomas.sakari.poikela@cern.ch

ABSTRACT: We examine two digital architectures for front end pixel readout chips, Velopix and Timepix3. These readout chips are developed for tracking detectors in future high energy physics experiments. They must incorporate local intelligence in pixels for time-over-threshold measurement and sparse readout. In addition, Velopix must be immune to single-event upsets in its digital logic. The most important requirements for both chips are pixel size, timing resolution, low power and high-speed sparse readout. We describe the transaction level architectural models of the chips using SystemVerilog. The correctness of the models is ensured using Open Verification Methodology. We will also discuss the advantages gained from transaction level modeling.

KEYWORDS: VLSI Circuits; Digital electronic circuits; Simulation methods and programs.

*Corresponding author.

Contents

1. Introduction

As CMOS scaling continues, more digital electronics are integrated on the pixel chip. This indicates that requirements can also be scaled to provide better performance (accuracy, speed, lower power) and more complex functionality. By adopting transaction level modeling (TLM)[1, 2] at an early stage in the development it is possible to explore and verify different complex architectures in order to meet the specifications. Therefore, we utilised TLM for the development of two different digital readout architectures for pixel readout chips Velopix and Timepix3. In this paper, we introduce the architectures and used modelling principles. We conclude the paper by analysing simulation results from TLM and register-transfer level (RTL) simulations.

Velopix is under development for the LHCb experiment [3], and consists of an array of 256 x 256 pixels (55um x 55um) which detect and tag hits in time with 25ns resolution. Pulse height information is required to improve the tracking resolution and is measured using the time-over-threshold (ToT) technique with a range of 4 bits. To minimize digital logic an array of 4x4 pixels, a super pixel, will share digital resources such as control logic and data storage. To maximise the use of existing on-chip bandwidth, zero suppression and clustering are done within the super pixel, which then transmits data packets of varying length depending on the hit occupancy of the 4x4 pixel area. Resources are also shared within a group of 4 super pixels, with one readout unit per group connected to an 8-bit column bus clocked at 40 MHz. The full chip consists of 64 super columns, each of 16 groups of 4 super pixels. Correspondingly, the periphery of the chip has 64

End-of-Column (EoC) blocks connected to a hierarchy of buses and intermediate FIFOs driving the outputs of the chip.

Timepix3 (see Timepix [4]) has the pixel array and detects and tags hits in time with coarse (25ns) and fine (1.67ns) resolutions, and a ToT range of 10 bits on a pixel level. Each pixel independently detects a hit, but the readout functionality is shared between an area of 2x4 pixels. A data packet consists of data from one pixel only. This shared readout block is connected to a column bus of 160 Mbps. The full chip consists of 128 super columns of 64 super pixels, each containing 8 pixels. The digital periphery of the chip has 128 EoC-blocks each connected to a token arbitrated bus of 1.76 Gbps.

Both chips need complex digital architecture requiring, for example, optimal FIFO depths and bus widths. Historically, it has been difficult to simulate such architectures for a full chip with an RTL approach. However, TLM offers this possibility. TLM has many features that allow for efficient modeling of such circuits. The reader is referred to [1, 2], but the main principles that were used in modeling the digital readout architectures of the two chips are described below.

2. Principles of Transaction Level Modeling and Verification

2.1 Initiator and Target

In TLM data is transported via function calls. An initiator is always a component invoking a function, and a target is the component containing the implementation and behaviour of this function. The components are decoupled from each other by using TLM-interfaces. In essence this means that either of the components can be replaced by a new component containing the same type of TLM-port regardless of the actual TLM-function implementation in the component.

2.2 Put-, Get- and Transport-Interfaces

The type of the TLM port indicates the direction of data flow. The initiator always starts the transportation of data as mentioned above, but the data flow can be in either or both directions.

In a Put-interface, an initiator puts (or writes) transactions into the port and a target receives them. In a Get-interface, the data flow is inverted, and is from the target to the initiator. An initiator gets (or reads) transactions from a target. Transport-interface is a combination of both the previously mentioned interfaces. In a single function call an initiator puts data into the port and receives a response transaction once the function returns.

Each of the interfaces can be either blocking or nonblocking. Blocking TLM-implementations are implemented using tasks which can include wait()-statements and timing delays, and nonblocking implementations are functions which return either 0 indicating a failure or 1 indicating a success.

2.3 Open Verification Methodology

Open Verification Methodology (OVM)[5] is a library of verification components. It offers a full set of TLM 1.0 interfaces, a class factory for dynamic selection of instantiated object type, super classes for verification components such as drivers, monitors and scoreboards, and a mechanism to construct complex stimuli for a design-under-test (DUT) using sequencers and layered sequences.

There are also special TLM-interfaces in OVM which were used extensively in this study. These interfaces are intended mainly for testbench purposes and to be used with components found from the OVM library. The library has also its own implementation of a FIFO, namely `tlm_fifo`, which can be connected directly to the TLM-ports. However, it also has a simulation overhead compared to a built-in SystemVerilog (SV) class `mailbox` (see Sec. 3.3).

All components in modeling of both chips were implemented using classes, mainly using the `ovm_component`, from the OVM class library. The testbenches were also developed using the library and verification functions and components. These testbenches were partly reused in RTL verification, which was done using constrained-random verification coupled with functional coverage collection (see [6, 5] for functional coverage). State of the art verification library, a direct derivative of OVM, is called Universal Verification Methodology (UVM) and also supports TLM 2.0 [7] interfaces.

3. Transaction Level Modeling of Pixel Chip

3.1 Transaction and Packet

For architectural modeling, a transaction representing a packet must be defined. Data members of a transaction can be derived from the initial specifications. Additional members can be introduced to facilitate easier debugging and monitoring. For example, additional time stamps were introduced into the transactions to monitor latency between various internal components of the chip. These time stamps are not part of the final implementation of the chip. An example of a packet is shown below:

```
class PixelPacket extends ovm_transaction;
   int tot_value;         // Time-over-threshold.
   int pixel_address;     // Pixel address
   int time_stamp;        // Global time stamp.
   int debug_time_stamp;  // Time stamp for debugging.
endclass
```

For many pixel chip applications, the first three data members shown above are sufficient. Another time stamp, for coarse time for example, can be added easily. By making the transaction as general as possible, without data members needed for particular applications, it was usable for the several different architectures that were studied. This also means that components should not use any additional data in a transaction unless it is needed for the functionality. For example, a high level arbiter that was implemented uses only the address information in the transaction, and the ToT-value and the time stamp are completely transparent to it.

Even if a more detailed transaction is required, all TLM-ports in components should be parametrized with the base transaction class to make transportation of any kind of derived transaction possible. All other transactions should be derived from the base class to utilize the object-oriented (OO) paradigms such as inheritance and polymorphism (see [8, 9] for OO principles).

3.2 Pixels

A pixel is the basic building block of a pixel chip. To increase the simulation speed, pixels were modeled at the highest level possible without a loss of information regarding the performance of the

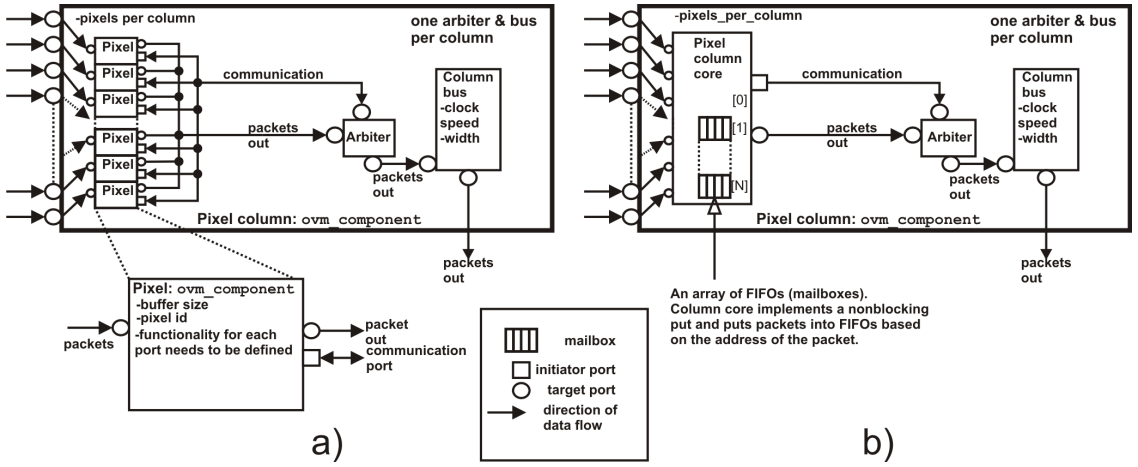

Figure 1. Block diagrams of two different pixel columns: a) Pixels as objects. b) Pixels as mailboxes

Table 1. Build times of different pixel columns.

	128 columns with Pixels (figure1a)	128 columns with Mailbox (figure1b)	128 columns Pixel and 1 `ovm_object`	128 columns Pixel and 2 `ovm_objects`
Runtime	216s	6.6s	544s	616s
Memory	316MB	13MB	332MB	348MB

architecture. For example, modeling of the analog section was reduced to a discriminator output. In an RTL-simulation, the signal is evaluated on every active edge of the clock which causes overhead in a simulation. The overhead was reduced by modeling activity only on rising- and falling edges of the discriminator.

Because a large number of pixels was needed, it was not feasible to implement each pixel as a separate component. For example, implementing a pixel as an `ovm_component` has more overhead than modeling a pixel using a built-in SV-class `mailbox` (see [6]). This distinction is shown in figure 1. It can be seen from the figure that the number of objects (pixels and ports) is greater in a) than in b). Also the number of connections that need to be established is increased. For Timepix3, an SV-class for a full pixel column was created and pixels were implemented using the class `mailbox`. This decreased the duration of build- and connect-phases (see [5] for phase definitions) during the simulation.

The overhead of modeling pixels as an `ovm_component` -class was evaluated using Modelsim 6.5d memory profiler and `time{}` -command. The results are shown in Table 1. Using an `ovm_component` has a significant memory- and runtime overhead when instantiating 65,536 (256 x 256) components. Instantiating one or two objects with each pixel also increases the overall memory footprint and runtime.

Instead of allocating all mailboxes at the beginning of a simulation, a mailbox can also be dynamically allocated when a pixel is hit. This reduces the amount of CPU memory required at the beginning of simulation, but makes the overall memory print dependent on the number of pixels hit. One option is to deallocate the mailbox when a pixel is empty but reallocating a mailbox several

times during the simulation may increase the simulation runtime.

3.3 FIFOs

FIFOs were used in this study as derandomizers and data buffers, and have been used widely elsewhere (see [10, 11]). Due to the specific requirements of FIFOs (asynchronous, multiple clock domains, error correction) in different applications, an RTL-FIFO design can be error-prone. A high-level FIFO can be implemented using `mailbox`-class. This implements blocking- and non-blocking put- and get-functions. No error-prone read- and write-pointer management is required when using `mailbox`. For more complex FIFOs, a specialized class was created which internally used `mailbox` for data management, in addition to user-defined functionality. The `mailbox`-class was also extensively used in the modeling and testbenches of both the chips.

In Velopix, where packets do not have a fixed size, a built-in class `semaphore` was used to model empty bits or slots in a FIFO while an unbounded `mailbox` was used to store the actual transactions. A semaphore is initialized with a certain number of keys, which can be used to represent bits or even complete packets. It has nonblocking functionality for put- and get-functions and also implements a blocking get-task.

3.4 Buses and Arbitration

Buses were used extensively in both architectures and elsewhere in pixel chips (see [12, 10, 13] for example). Sparse readout schemes investigated for Velopix in particular were sensitive to the width and speed of these buses and hence these parameters had to be simulated carefully. Buses were modeled using an unbounded `mailbox`. This functions as a bus with an embedded first-come first-served arbitration. Each component connected to this bus can use the put-interface to put its transaction onto the bus, and the bus takes care of the arbitration using the `mailbox`.

```
class TokenArbiter extends ovm_component;
   ovm_blocking_imp#(PixelPacket, TokenArbiter) get_port;
   ovm_blocking_get#(PixelPacket) data_ports[];
   ...
   task get(ref PixelPacket p);
      while( ! data_found) begin                    // Arbitration algorithm
         data_found = packet_exists_in_pixel(N++);
         #(delay);                                  // A realistic arbitration delay
         if( N > max_num_of_pixels) N = 0;
      end
      data_ports[N - 1].get( p );
   endtask: get
endclass: TokenArbiter
```

Consider the code above describing a token arbiter (see [14, 15] for arbiter schemes). A token arbitration scheme can be implemented at a high-level using one-to-many get-port mapping. A component can request data from the arbiter using a get-port and the arbiter can forward this request to a component holding data using its arbitration algorithm. The algorithm can be replaced by any custom algorithm required for the modeling of a chip. The arbiter does not use any clock but the delay of the arbiter can be realistically modeled using a #-Verilog operator.

Table 2. Simulation times of Velopix TL- and RTL-models.

TLM(full chip)	RTL(1 cols)	RTL(4 cols)	RTL(64 cols)
360s	185s	1251s	140800s[1]

Table 3. Simulation times of Timepix3 TL- and RTL-models.

TLM(full chip)	RTL(2 cols)	RTL(4 cols)	RTL(8 cols)	RTL(128 cols)
1560s	1117s	2883s	7440s	329648s[2]

4. Conclusions and Results

4.1 RTL and TLM Simulation Time

It has been reported that transaction level (TL) models run 10 to 1,000 times faster than RTL-models [2]. Benchmarks in this paper have been been obtained by comparing RTL simulations to TLM simulations. The simulator used was NCSim.

A comparison between Velopix RTL- and TL-simulation times is shown in Table 2, and the comparison of Timepix3 simulation times is shown in Table 3. For RTL-simulations, only modules relevant to the readout architecture were used to reduce the simulation time. For example, configuration logic was modeled at the behavioral level. Simulation times of Timepix3 and Velopix are not comparable because of the differences in hit occupancies and modeling granularity of the pixel functionality.

Velopix has 64 columns (1024 pixels per column) while Timepix3 has 128 columns (512 pixels per column). The durations presented in Table 2 and Table 3 for both the RTL simulations are estimates because no full chip RTL-simulation could be run. The RTL code was implemented using a synthesizable subset of SystemVerilog, and was not optimized for simulation (see [16] for improved simulation efficiency). The performance profiler was also used to ensure that no simulation bottleneck was caused by any of the testbench components. Also no memory benchmark was established for these simulations.

4.2 Conclusions and Discussion

It has been shown that TLM can reduce the simulation time for an architectural optimization of a pixel chip. The experience in these projects has shown TL-models to be faster to develop and debug than corresponding RTL-models. It can also be concluded that TLM at the early stage of the projects has facilitated testbench development before RTL design has started. It can also be said from the experience that debugging TL-components is an order of magnitude faster than debugging RTL-modules. Due to high-level synchronization mechanisms such as events, FIFOs and semaphores, the behaviour of TL-components is easier to estimate than the behaviour of RTL-modules based on custom protocols and synchronization.

The biggest disadvantage in using TLM has been the lack of connection between physical implementation and TL-models. Because of this the physical implementation had to be kept in

[1] An estimate from a simulation of 1 and 4 columns.

[2] An estimate from a simulation of 2, 4 and 8 columns.

mind while designing the TL-model of the chip architecture. Even so, the RTL models still had to be written from scratch after the architecture was determined using TLM. The need for a very low-level optimization in a pixel chip, especially in the pixel area, makes a pixel chip unsuitable for automatic refinement from TL-model into RTL-model. However, the same disadvantage exists for non-synthesizable bus functional models written without TLM-interfaces.

Even with this drawback, using TLM for architectural simulations is a viable option, especially if several different architectures need to be developed and simulated. While RTL-models can be scaled down by reducing the number of columns for example, a full chip architectural TL-model offers more accurate estimates in the case of highly non-uniform hit occupancies. The impact of these distributions to data transport in pixel columns and periphery blocks can be modeled without losing cycle accuracy while keeping the simulation performance higher than in RTL-simulations. The work in this paper was carried out using SystemVerilog hardware description language.

References

[1] L. Cai and D. Gajski. Transaction Level Modeling: An Overview. *First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 1, 2003.

[2] F. Ghenassia et al. *Transaction level modeling with SystemC*. Springer, Dordrecht, 2005.

[3] The LHCb Collaboration. Letter of intent for the LHCb upgrade, 2011. CERN-LHCC-2011-001.

[4] X. Llopart et. al. Timepix a 65k programmable pixel readout chip for arrival time, energy and/or photon counting measurements. *NIM A*, 581:485–494, 2007.

[5] Mentor Graphics. Open Verification Methodology (OVM) user's guide. Verification Academy.

[6] SystemVerilog Language Reference Manual. SystemVerilog homepage.

[7] The Open SystemC Initiative. SystemC webpage.

[8] Iain D. Craig. *Object-Oriented Programming Languages: Interpretation*. Springer, 2007.

[9] Avinash C. Kak. *Programming with Objects: A Comparative Presentation of Object-Oriented Programming with C++ and Java*. John Wiley and Sons, 2003.

[10] D. Arutinov et al. Digital Architecture and Interface of the New ATLAS Pixel Front-End IC for Upgraded LHC Luminosity. *IEEE Transactions on Nuclear Science*, 56(2):388–393, April 2009.

[11] G. Dellacasa et al. A silicon pixel readout ASIC with 100 ps time resolution for the NA62 experiment. *J. Inst.*, 6, January 2011. C01087.

[12] Ch. Hu-Guo et al. CMOS pixel sensor development: a fast read-out architecture with integrated zero suppression. *J. Inst.*, 4(4):1–10, April 2009. P04012.

[13] M. Gélin et al. Intermediate Digital Monolithic Pixel Sensor for the EUDET High Resolution Beam Telescope. *IEEE Transactions on Nuclear Science*, 56(3):1677–1684, June 2009.

[14] H. Kariniemi and J. Nurmi. Arbitration and routing schemes for on-chip packet networks. In *Interconnect-Centric Design for Advanced SOC and NOC*, pages 253–282. Kluwer Academic Publishers, 2004.

[15] H. Foster and A. Krolnik. *Creating Assertion-Based IP*. Springer, New York, 2008.

[16] C. Cummings. Verilog coding styles for improved simulation efficiency. *International Cadence User Group Conference*, October 1997.