Modeling of a water control system for a tank using SystemC

Matteo Iervasi - VR439212

***Abstract*—This document reports the modeling of a hardware system for controlling the water level of a tank. It has been modeled using a modular approach, focusing on single parts which has been developed and verified separately, then merged and tested as a heterogeneous platform.**

I. Introduction

The system has to be able to stabilize the water level of a tank in a precise range through a controller which can adjust the opening of a valve. The controller reads the current water level from the tank and sends the valve a command among `IDLE`, `OPEN` or `CLOSE` and a threshold, which represents the current maximum aperture of the valve. The controller encrypts the command using the XTEA protocol, so before going to the valve the command is decrypted by a XTEA decipher.

Given that the system is composed by different parts with different technologies, going with SystemC was a good choice, because of its ability to represent a model at different levels of abstraction. Also, the need of having a language able to represent analogue parts brought us using SystemC-AMS, the analogue extension for SystemC.

After separating the various components, each module was developed separately, ensuring it was correct in its behavior. The digital part, composed of the XTEA module, was developed in four variants, namely RTL and the three TLM styles. The analogue part, composed of the valve and the water tank was developed using SystemC-AMS, which provides three different styles for writing analog-mixed signal systems.

I firstly developed the XTEA digital module as a standalone one, then the analogue part, composed of the tank and the valve. After verifying the correct behavior of the two, I created the heterogeneous system.

II. Background

The following technologies were used for the development of the system:

- **SystemC**[1]: Standard C++ library for hardware and software platform modeling at different level of abstraction.
- **SystemC-AMS** [?]: Standard C++ library that extends SystemC digital modeling capabilities with analogue modeling.

The digital components were described using different levels of abstraction:

- **RTL**: the lowest level of description provided by SystemC, which focuses on signals and registers.
- **TLM**: high level of description which focuses on how modules interact each other.

The same for the analogue components:

- **LSF**: stands for *linear signal flow*, describes a continuous non-conservative equation representing the component.
- **TDF**: stands for *timed data flow*, describes a discrete non-conservative equation or function representing the component.

III. Applied methodology

Following the modular approach, the first component to be developed was the digital XTEA module.

A. XTEA TLM

The first step involved the development of a TLM description of the XTEA module. Specifically, three variants were coded:

- TLM UT (untimed model)
- TLM LT (loosely timed model)
- TLM AT4 (approximately timed model)

Although the final module will only have to decrypt commands, the standalone model is a full implementation of XTEA, which uses chunks of 128 bits for the key and 64 bits for the text, so a structure is needed. I also need a flag for setting the mode. In the standalone model of XTEA, I also added a variable for storing the result separately, just for checking the correctness.

The resulting structure was the following:

```
struct iostruct {
  bool mode;
  sc_uint<32> key[4];
  sc_uint<32> text[2];
  sc_uint<32> result[2];
};
```

The module was coded in a single block, using a separate testbench for verifying the correctness, as shown below:

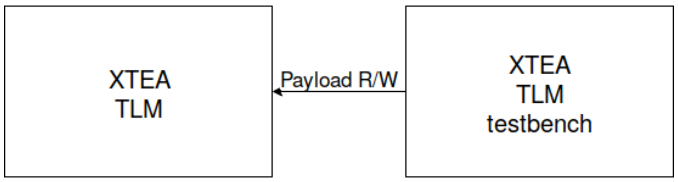


Figure 1. Testbench connection

The testbench acts as an initiator communicating with the target (the XTEA module). It calls target's `b_transport`, sending it data, then waits for the answer. After receiving the answer the initiator can verify the correctness of the data.

Given that I'm using TLM, which focuses on transactions rather than actual implementation, the actual XTEA algorithm is just a C++ function.

Going from untimed TLM representation to a loosely timed one was an easy task. I just added the time notion in the module, simulating the time spent on computations.

After the LT representation, it was time for the AT4 one. This time things were a bit more complicated. In fact, the AT4 model requires a bidirectional communication between the initiator and the target that involves 4 steps (that's why it's called "AT4"). It was necessary to modify the interface between the initiator and the target so that it was compliant with the style. You can see the AT4 communication in the following picture:

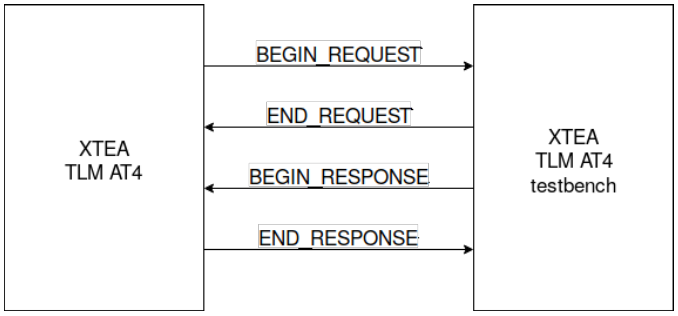
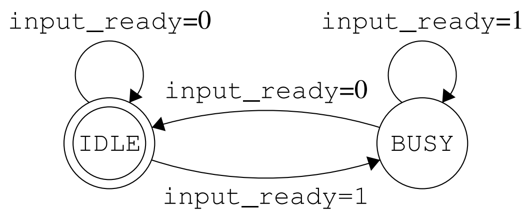


Figure 2. The 4 phases of TLM AT4

B. XTEA RTL

The XTEA module was then modeled in RTL. TLM showed us the needed parts with the structure, which now becomes a series of I/O ports. At this level, I have a clock and a reset that become inputs ports. In theory, in an RTL implementation one should derive from the C++ algorithm a separation between what has to be a finite state machine and the datapath, following some design constraints given from the customer (speed vs space constraint as an example). Here I didn't have any of those, so I decided after taking a look at the XTEA algorithm to just become a datapath. Obviously if I had to synthesize it on real hardware, it would probably be better to change it, because the area of such combinational logic would be immense. The advantage is that the module is very fast and the time to develop it is quite insignificant. The resulting FSM consists of just two states, `IDLE` and `BUSY`.



When the `input_ready` flag is on, it means the module has received the inputs and it can start the computation, so it goes in the `BUSY` state. I decided that as long as the flag is on, the FSM stays on `BUSY` state, so the computation is repeated every time.

The actual RTL implementation was done using an `SC_MODULE` with the following I/O ports:

```
sc_in_clk clk;
sc_in<bool> rst;

sc_in<sc_uint<32>> text_input[2];
sc_in<sc_uint<32>> key_input[4];

sc_out<sc_uint<32>> data_output[2];

sc_in<bool> input_ready;
sc_in<bool> mode;
```

The FSM and the datapath are defined using two separated `SC_METHOD`: the FSM is sensible to the `input_ready` and the `mode` ports, while the datapath is sensible to `clk` and `rst`. The datapath updates the current status and eventually does the computation by calling the `xtea` method. The FSM is only responsible for calculating the next state. The output remains active until `input_ready` is active.

The module was then tested with a testbench.

C. Water tank LSF

After developing the XTEA module, it was time to dive in the AMS part, starting from the water tank, which was suited for LSF style. *Linear Signal Flow* is a continuous time non conservative mixed signal model. The behavior is defined as relations between variables of a set of linear differential algebraic equations. The model is described with a block diagram notation, with signals connecting diagram's blocks. The blocks (or clusters) are defined in the standard.

The differential equation representing the water tank is

$$\dot{x} = 0.6a - 0.03x \tag{1}$$

The implementation was quite straightforward using standard LSF clusters, connected as shown below:

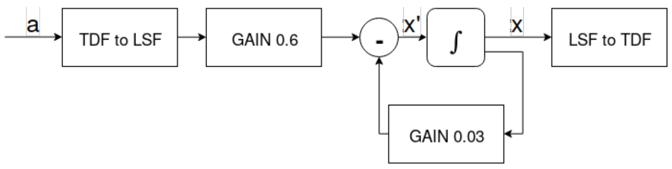


Figure 3. LSF cluster representation

The water tank has also a discrete time input port and an output one, for communicating with the valve and the controller.

D. Valve TDF

After modeling the water tank, I went to the valve. This time, LSF was not very suited, instead I chose to use *Timed Data Flow*. TDF is a discrete time non conservative mixed signal modeling technique. The process is similar to the RTL one: we define the inputs and the outputs inside a specific `SCA_TDF_MODULE` plus an internal variable for storing the current valve's aperture.

The module has two specific methods: `set_attributes()` and `processing()`. With `set_attributes()` I can initialize timesteps and the eventual delays, while `processing()` contains the method executed at every timestep, so it does the actual work.

Valve's behavior can be expressed as following:

- On `IDLE`:
  Do nothing
- On `OPEN`:
  $a = a + 0.25t$ where a is the aperture and t the timestep
- On `CLOSE`:
  $a = a - 0.25t$ where a is the aperture and t the timestep

E. Controller TDF

After developing the water tank and the valve, I needed to test the correct behavior. Though the final project requires a TLM controller, I developed a TDF controller for the standalone AMS project. The development process was very similar to the one of the valve: the controller periodically reads the water level from the tank, and send a command to the valve, together with the valve maximum aperture threshold.

F. Heterogeneous platform

After ensuring everything was behaving correctly, it was time for making the modules work together. The first problem was that for letting the components communicate each other, I needed transactors. They are sort of "translators" that enables us to make different modules communicate (like a digital RTL module with an AMS one). Using transactors comes with a drawback: it makes the simulation slower, that's why I had to decrease the controller speed from 5 sec to 2.

Needed transactor were:

- **TLM-RTL**:
  SystemC TLM module with the same number output ports as the RTL module's input ports. When the `b_transport()` function is called values are written.
- **RTL-TLM**:
  Module with a looping `SC_THREAD`. The TLM controller of the heterogeneous platform is modeled this way.
- **RTL-AMS/AMS-RTL**:
  An AMS module that uses `sca tdf::sca de` ports for converting RTL values to AMS and vice-versa.

Picture 4 shows how the heterogeneous platform is built.

After coding the transactors, it was just a matter of testing and parameter tuning. One notable thing is the valve threshold which, being a real value, was passed directly to the valve instead of being crypted with XTEA, because it would have been difficult otherwise.

IV. Results

For testing the correctness of the system, every module was tested both as a standalone one and as a part of the heterogeneous platform. The testbench tried different inputs and checks for the results, measuring times of simulations. The first tested module was XTEA. The TLM levels were pretty fast and very similar to each other. RTL was slower, but not that much given the fact I implemented it as pure combinational logic. It can encrypt and decrypt in 2ns, as shown in figure 5.

After XTEA I tested the AMS module, which required more time on testing and tuning. Figure 6 shows the correct curve of the water level stabilization.

The heterogeneous platform is shown with two different controller timing in figure 8 and 7.

V. Conclusion

Recycling models and using different level of abstraction can be useful for reducing system modeling and verification, and SystemC makes this straightforward. Also the possibility to test analogue parts before actually synthesizing the digital ones is very useful for having at least a working prototype of the digital synthesizable part.

References

[1] Accellera Systems Initiative *et al.*, "Systemc," *Online, December*, 2013.

Appendix

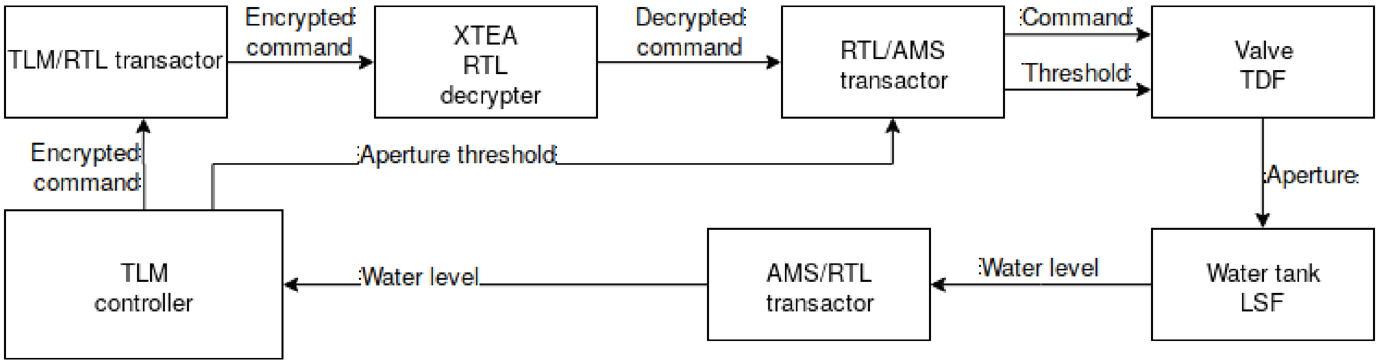


Figure 4. Heterogeneous platform

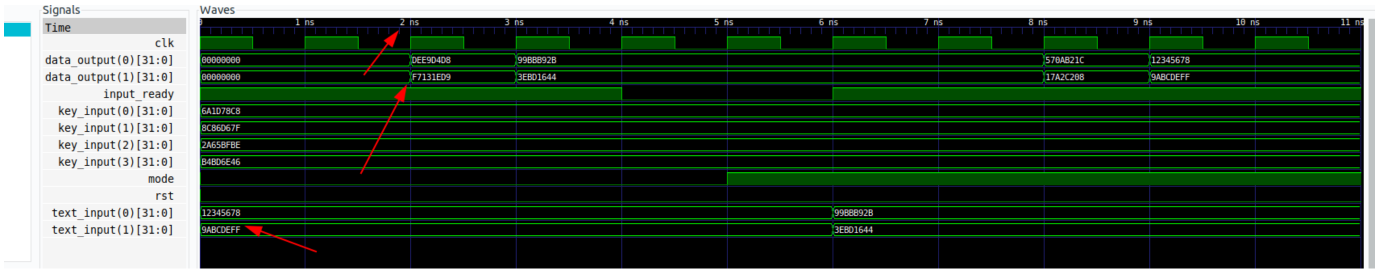


Figure 5. RTL times

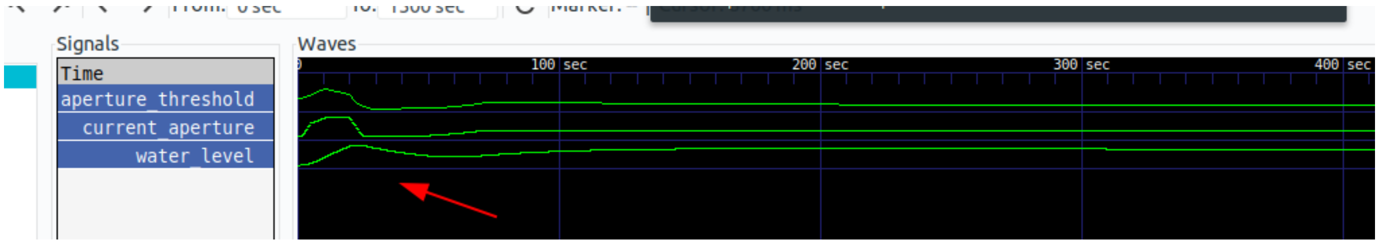


Figure 6. AMS water level stabilization

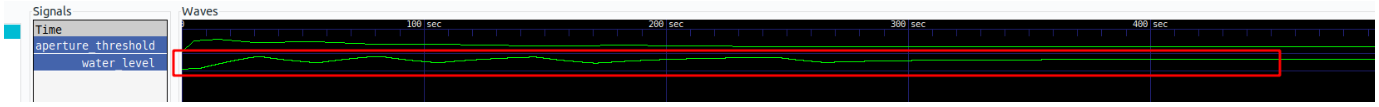


Figure 7. Water level stabilization

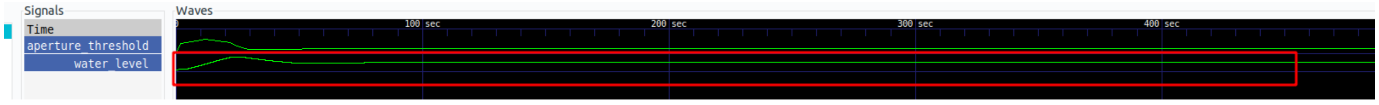


Figure 8. Water level stabilization