SystemC modeling of a water control system

Francesco Alessandro Picchirallo - VR432225

***Abstract*—This document reports the $SystemC$ modeling of a watertank system, controlling and stabilizing the water level. It has been developed with a modular approach, each modules has been developed at different levels of abstraction.**

I. Introduction

The system described in this report has to monitoring and stabilizing the water level between 5 and 8.8 in a tank. The controller receive in input the water level, then send the command, $IDLE$, $OPEN$ or $CLOSE$, and the aperture threshold. The command is encrypted with the $XTEA$ protocol. The decypher decrypt the instruction and send them to the valve, which will execute and send to the water tank the actual valve aperture. The water tank is the dynamic system, whose task is to simulate the water level based on the following function:

$$\dot{x} = 0.6 * a - 0.03 * x$$

where x is the water level and a is the valve aperture.
The entire system is an heterogeneous platform, composed by $SystemC$ and $SystemC\ AMS$ modules, which comunicates through transactors. Each module of the system was developed separately. First the $XTEA$ encrypter-decrypter, written in $SystemC$ at the RTL and TLM abstraction levels, instead the analog part, representing the water tank and the valve, was developed using $SystemC\ AMS$.
The modular approach of the system allows to simulate all the heterogeneous component and meanwhile to design the complete platform, in fact the reuse of component parts is an essential in embedded system design.

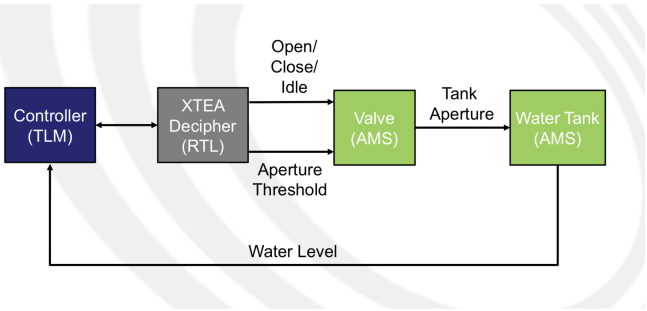

Figure 1. Heterogeneous Platform

II. Background

The system has been developed in C++, in particular using $SystemC$ and $SystemC\ AMS$. $SystemC$[1] is a C++ library for Electronic System Level Design, modeling HW designs at different levels of abstraction which are Register Transfer Level (RTL) and Transaction Level Modeling (TLM). The Register Transfer Level is the lowest level description of $SystemC$, focused on registers and signals, dividing the Finite State Machine (FSM) from the Datapath. Normally used to design modules closed to hardware. The Transaction Level Modeling, instead, describe how the modules communicates each others, providing different styles which are UT (Untimed), LT (Loosely-Timed) and AT (Approximately-Timed). The UT version provide blocking interface and predefined synchronization points, where the simulation time is not tracked. The LT provide the same blocking interface but has two points of synchronization (invocation and return) and introduces the temporal decoupling concept, where individual processes are allowed to run ahead for a certain time slice without advancing general simulation time, with a delayed synchronization with the system. The AT style has a non blocking transport interface, with a timing annotation and multiple phases during the lifetime of a transaction.
$SystemC\ AMS$ is an extension of the $SystemC$ language standard, modeling Analog Mixed-Signal Systems. It defines the following models of computation: Timed Data Flow (TDF), discrete time non-conservative modeling, Linear Signal Flow (LSF), continuous time non-conservative modeling, and Electrical Linear Networks (ELN), continuous conservative modeling.

III. Applied Methodology

The project goal was to develop the system with a modular approach following the abstraction levels of $SystemC$. The characteristics and design choices of each system modules are illustrated below:

A. $Xtea\ RTL$

The eXtended Tiny Encryption Algorithm (Xtea) is a 64-bit block Feistel cipher with a 128-bit key. The single module has been developed from the C++ code, splitting the Datapath from the FSM. This module consists of ten input and three output ports, and include six signal used to connect module ports allowing modules to communicate. The figure 2 below show input and output ports.

Internal signals are described as:

- $counter$: 6 bit signal, used for the 32 iterations necessary for the both mode, cipher and decipher.
- $delta$: 32 bit signal, used to memorize the delta value.
- $v0$: 32 bit signal, used to cipher and decipher the word0
- $v1$: 32 bit signal, used to cipher and decipher the word1
- sum: 64 bit signal, used to compute at each iteration which key will be used.
- $temp$: 32 bit signal, used to temporarily store the key which will be used.

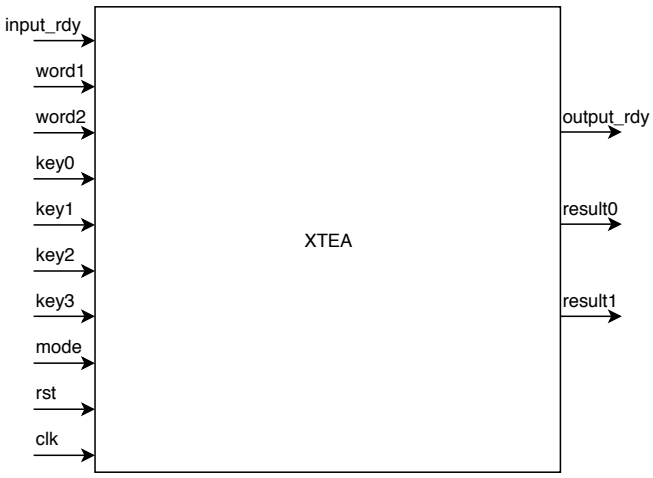

Figure 2. Xtea Module

This module is described as an 13 states EFSM, where the datapath is represented from the automa states (nodes), the transitions and conditions on them represents the control logic part. The FSM and datapath are defined using two different SC_METHOD: the datapath is sensible to clk and rst signals, instead the fsm is sensible to STATUS signals and input_rdy port. While the input_rdy is 0 the FSM stay in ST_0, when it changes to 1 the FSM will pass to ST_1 where the words to be encrypted/decrypted will be saved in v0 and v1 signals. Here the FSM is divided into two parts, one for encrypt and one for decrypt. The FSMD is described in Figure 3 below.

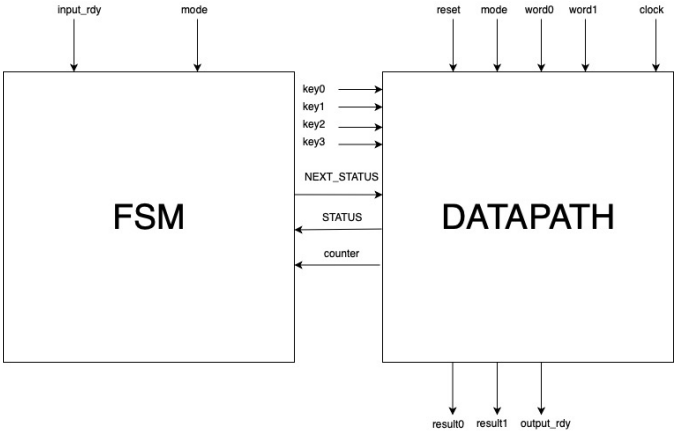

Figure 3. Xtea FSMD

B. $Xtea\ TLM$

This Xtea version has been developed using $SystemC\ TLM$ coding styles, UT, LT and $AT4$. The general idea is to let the testbench (initiator) communicate with the $xtea$ module (target), through the $b\_transport$ function. In this version a supported structure, called $iostruct$, was created:

```
struct iostruct {
  sc_uint<1> mode;
  sc_uint<32> key0;
  sc_uint<32> key1;
  sc_uint<32> key2;
  sc_uint<32> key3;
  sc_uint<32> word0;
  sc_uint<32> word1;
  sc_uint<32> result0;
  sc_uint<32> result1;
};
```

In each transaction a payload with the structure information is sent.

The first TLM style developed was the Untimed version. As already mentioned, the Untimed version has a blocking interface and predefined synchronization point. The initiator calls the $b\_transport$ function implemented in the target, adding to the transaction, the payload. It should be noted that in this version, the notion of time is not necessary.

The TLM Loosely Time version is similar to the Untimed version except that, in addition to payload, time-related information is added, simulating the time spent on computations. This is necessary because the LT version introduce the temporal decoupling concept, to allow processes to continue their execution without increasing the simulation time.

The last version, Approximately Timed at 4-phases, is the non blocked version of the TLM coding styles. Transactions are broken down into 4 phases, the testbench send data to the $xtea$ module, then wait the ack ($BEGIN\_REQ$, END_REQ). The $xtea$ receive transaction and process the data received, then send the ack to the testbench ($BEGIN\_RESP$, END_RESP). Figure 4 represents the AT4 communication.

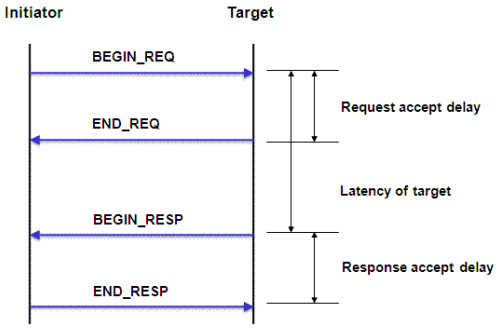

Figure 4. The 4 phases of TLM AT4

C. $Water\ Tank\ System$

Water Tank System is a water monitoring and control system, developed using $SystemC\ AMS$. The system consists of a tank, a valve and a controller. Figure 4 show the diagram representing the AMS system.

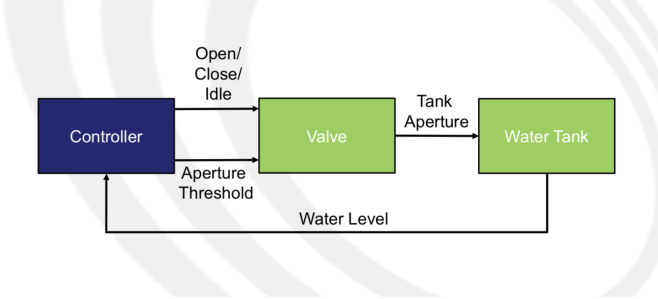

Figure 5. Water Tank system

The $controller$, described as TDF non conservative for a discrete time modeling, receive in input the water level, supplied from the water tank, and has to verify that is within 5 and 8.8. If it is in that range, then send to the valve an $IDLE$ command, if is greater than 8.8, then modify every 5 seconds the valve aperture threshold (t) as follow: :

$$t = t * 0.7$$

and send a $CLOSE$ command. Finally if the water level is lower than 5, modify every 5 seconds the valve aperture threshold as follow:

$$t = t * 1.1$$

and send an $OPEN$ command. The $valve$ module is also described as TDF. The main focus of this module is to implement the commands received in input from the controller, then modify the valve aperture (a) as follow:

- command $IDLE$: a = 0
- command $OPEN$: a = 0.25
- command $CLOSE$: a = -0.25

The $valve$ will send in input to the $water\ tank$ the actual valve aperture, and finally this module will simulate the water level inside itself. The $water\ tank$ module is described as a Linear Signal Flow (LSF) non conservative at continuous time, simulating the water level using the function described in introduction, whose representation in $SystemC\ AMS$ is made using some LSF modules as follow:

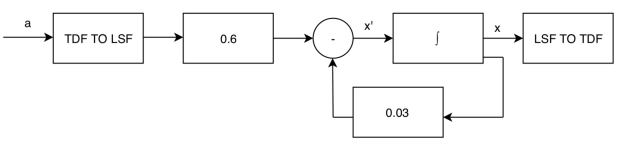

Figure 6. LSF representation

As can be seen from the Figure 5, a TDF to LSF and a LSF to TDF converter has been used to communicate with the other modules, described in discrete time.
The main focus of this AMS system is to stabilized the water level. As can be seen in Figure 7, the water level has stabilized around 210 seconds, with a value of 7.733.

Figure 7. WaterWave AMS

D. $Heterogeneous\ Platform$

Heterogeneous Platform is the final designed system, which includes RTL, TLM and AMS modules. This is made possible by the transactors, which are translator in communication between modules implemented at different levels of abstraction and allows mixed modeling and testbench reuse. The main idea is to reuse the RTL implementation and the TLM testbench of the $xtea$, the AMS water tank and valve, connect RTL and TLM part with a basic transactor and connect the TLM and AMS with a TLM/AMS transactor. The TLM testbench will be the controller of this system, sending cipher data to the RTL $xtea$, that decipher the data and will send the results to the valve.
The basic transactor has the same number of output ports as the RTL input ports, because the output of the TLM will be the input of the RTL. The values are written when the $b\_transport()$ function is called. The conversion of RTL values to AMS and viceversa are made with $sca\_tdf::sca\_de$ ports, in the interfaces file, such as $iface\_valve$ and $iface\_watertank$. Figure 7 shows the complete $Heterogeneous\ Platform$ schema, instead Figure 9 shows the water level stabilization, the aperture threshold and the valve aperture. Note that the aperture threshold had to be encrypted, but it was chosen not to encrypt it and pass it directly to the $iface\_valve$.

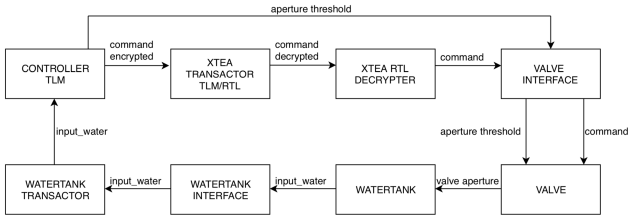

Figure 8. Heterogeneous Platform Schema

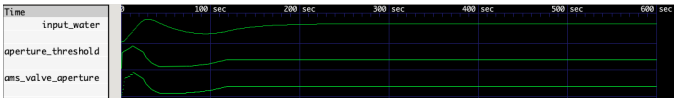

Figure 9. Heterogeneous Platform WaterWave

IV. Results

As already mentioned, the system was designed with a modular approach, in this way the modules were tested first individually, with some different testbench, and then within the heterogeneous platform.
The difficulty of the $Heterogeneous\ Platform$ consists to adapting the modules, developed individually, to communicate each other, therefore they must be re-adapted to send the data

correctly. Considering that the modules are at different levels of abstraction, the synchronization complexity is greater, therefore the simulation time will be slower than those of individual modules, because it's need synchronize TLM modules, that are less accurate but its simulation time is faster, with the RTL module, that is more accurate but slower than TLM. As can be seen from the Figure 10, the RTL module employ 2630 ns to encrypt and decrypt.
Figure 11 show the EFSM of representing the $xtea$ RTL module.

V. Conclusion

Component reuse is an essential to reduce the time required to design an embedded system and hit the Time to Market, $SystemC$ is useful to this. The possibility of developing modules at different level of abstraction and test them separately, the possibility to model both analog components at continuous and discrete time, and digital components to simulate the entire system is a significant benefit for the development of an heterogeneous system.

References

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

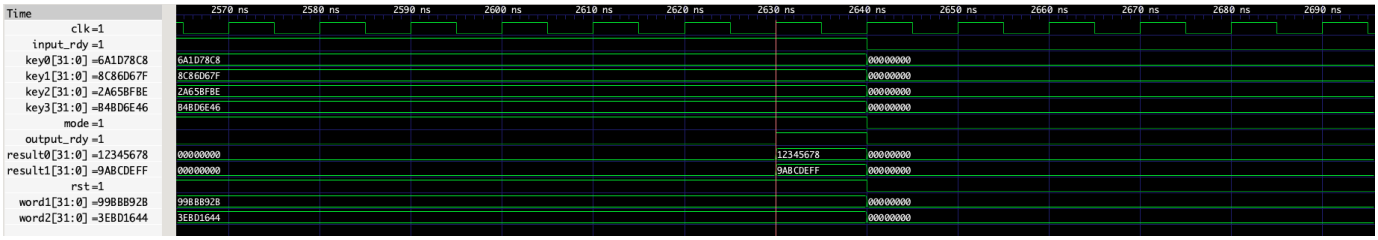

Figure 10. RTL times

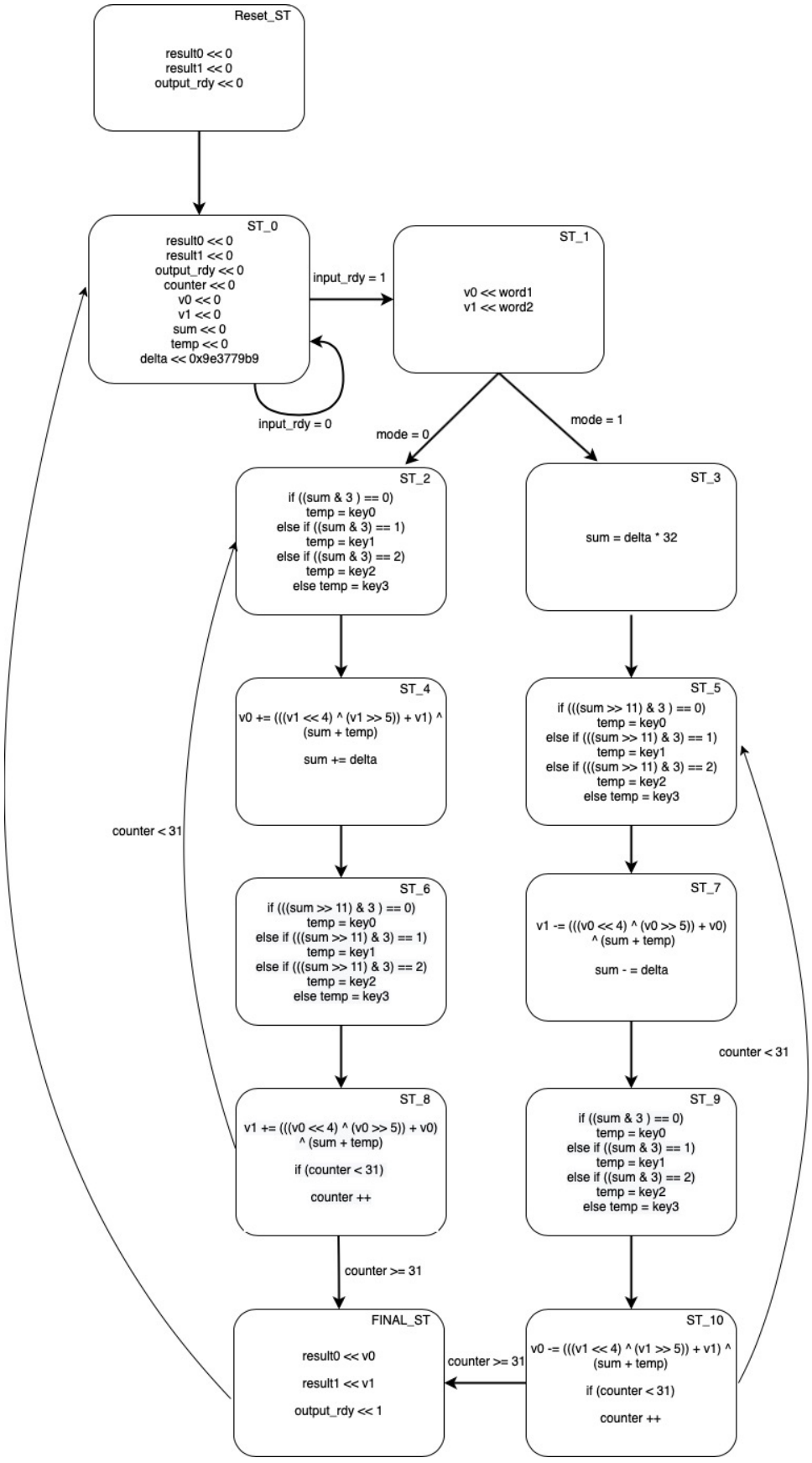

Figure 11. EFSM XTEA RTL module