Design of Clocked Circuits Using UML

Zhenxin Sun, Weng-Fai Wong, Yongxin Zhu
School of Computing
National University of Singapore
E-mail: {sunzhenx, wongwf, zhuyx}@comp.nus.edu.sg

Santhosh Kumar Pilakkat
Embedded Systems-SDR Lab
Institute for Infocomm Research, Singapore
E-mail: pilakkat@i2r.a-star.edu.sg

Abstract – Clocking is an essential component of any embedded system design. However, traditional design techniques are either short of clocking support or too complex for users. The Unified Modeling Language (UML) has been proposed as design tool in real time system design, but the clocking semantics has not been properly dealt with. In this paper, we will present our experience of using UML to design a clocked system. In particular, UML is used to model the digital down converter, an essential component of software radios. Our tool chain automatically generates the simulation as well as synthesizes the final implementation.

I. INTRODUCTION

With the increasing complexities of embedded systems, designers have been searching for new methodologies that can manage the complexity as well as yielding high productivity.[2] The Unified Modeling Language (UML) is a proven modeling and specification language that has been used widely in development of complex software applications [13].

However, UML lacks natural support for timing semantics. Previous works have tried to use extra notations to specify the clock settings. Most of these notations are not interchangeable and non-reusable.

In this paper, we address this problem by showing how we can use existing UML notations to specify a real time system with clock settings. This design is then automatically translated into detailed implementations that include simulators as well as synthesized hardware. In our framework, UML's class and component diagrams, as well as statecharts are translated into an intermediate form in SystemC. Clock settings are used during the SystemC code generation. At the end of this top-down design flow, synthesizable SystemC models are generated. Therefore, very high level specifications can be lowered to implementations that are very close to hardware [11].

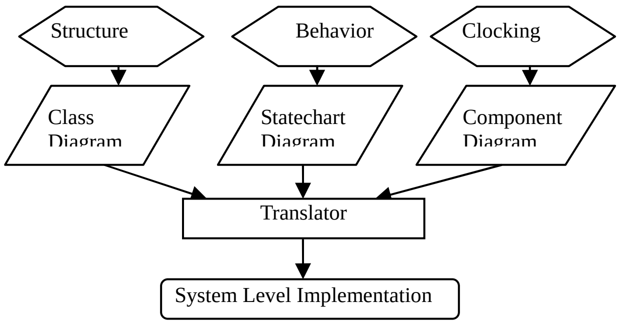


Figure 1 Our Design Flow of the Clocked Chips

Figure 1 shows our design flow. We start` with formal UML models, specifying different aspects and requirements of the system using different UML diagrams. All these diagrams are used as input to the translator. The translator automatically generates the system level SystemC implementations from the model.

II. RELATED WORKS AND SCOPE OF OUR WORK

Lack of support for clocks is a challenge for the several efforts similar to ours that use UML as the design vehicle. In YAML[10], the system structure is modeled using UML notions with extensions. Some results have been reported using both class diagram and statechart to generate SoC designs without clock specifications [3,4,12]. In another approach, extended task graphs were used to capture the system's behavior as well as the clock specification [5].

Our approach differs from others in that rather than formulating another system description language, we used standard UML for the task. In particular, we use I-logix's Rhapsody to build UML models and specify clock setting in the component diagrams. The following features are unique in our approach:

1. To ensure the correctness and reusability, we use the existing UML notations available in Rhapsody 4.2 with customization to build executable UML models.
2. Clock settings of components are specified using component diagram. They are used in the code generation phase.
3. XMI, an interchangeable UML representation, is used as the input of our translator which then generates SystemC code directly.
4. The generated SystemC models can be simulated using the SystemC simulator, and the implementation can then be easily tested and verified.

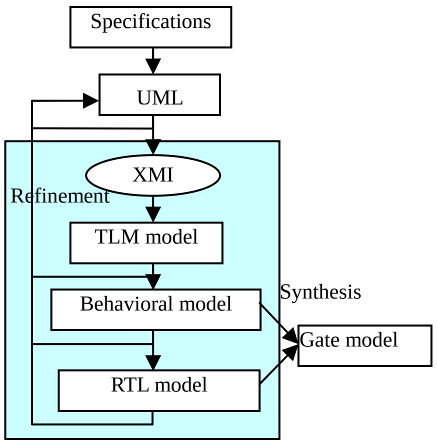


Figure 2 UML to SoC design flow

Figure 2 shows the proposed code generation flow. Starting with the system's specification, UML models are built. Verification and refinement can be done at several levels to test whether the requirements are completely and correctly captured. The system is first translated into transaction level modeling (TLM) level model. This can be done fairly rapidly. When there is sufficient confidence obtained from simulation, the behavioral and RTL level models can be generated. These can be further compiled into gate level models.

In this top-down flow, testing and verification can be done at all the stages. The controlled lowering of the model with extensive testing at every stage is essential for the successful design and deployment of complex embedded solutions.

III: IMPLEMENTATION OF DESIGN FLOW

Our design flow is based on UML notations and SystemC. In the design flow, designers only need to work at the UML level and the rest are automated, thereby boosting productivity. In rest of this section, we will introduce the details of how we capture the system specification and SystemC code generation.

Class Diagram Semantics and Translation: Class diagrams are used to model the structural information of a design.[7] Classes are used to model the system components and communication interfaces. A class has attributes and operations. Each attribute has a type, publicity and static status. Functions have return types, arguments types and names, as well as the publicity and static status.

Each class will be translated into a SystemC sc_module. To model different module elements, we make use of the stereotype of class. There is a mapping from SystemC elements to the UML stereotypes. To model the details of the SystemC design elements, we introduced three extensions using UML stereotypes mechanism. Table 1 shows the mapping between SystemC elements and UML stereotypes.

SystemC elements	UML stereotypes
Modules	Normal class
Interfaces	<interface>
Primitive channel	<pri_channel>
Hierarchical channel	<channel>

Table 1 Mapping from SystemC elements to UML stereotypes

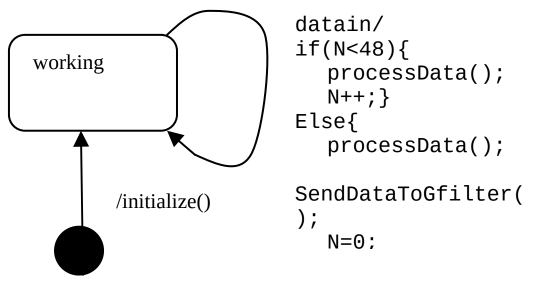
Aggregation is used to model the ‘contains’ information of components. If A has aggregation relation with B, then B will be modeled as component of A.

Associations among the classes are used to model the communication relationships between the components. These relations include association and aggregations. If two components have message or signal exchange, an association will be placed in between. The direction of the association indicates the direction of the communication.

Statechart and Component behaviors: The statechart formalism was introduced by Harel [1]. A statechart design essentially consists of states and transitions like a finite automaton. Statechart diagrams in UML allow for guards on transitions, propagated transitions, actions on transitions, actions on state entry, activities that last as long as a states, and actions on exit. Figure 3 shows an example of a statechart consisting of a simple state and an initial state.

Dynamic behavior of a UML class is expressed in terms of state transition diagrams of simple states which in turn is translated as a process. SystemC provide three different types of processes: sc_thread, sc_cthread, and sc_method, which we use in different levels of abstraction[8]. A local variable called a state is used to hold the current state identifier, and it is assigned to the value of initial state identifier during the initialization stage. Processes keep transiting between states until a final state is reached. When the process enters a new state, it first performs the actions_on_entry. Then the reaction is performed. For most of time, processes stay in one state, waiting for some events. Upon receiving an event, a process will perform the guard action and change the value of state accordingly. When it exits the state, the action_on_exit will be performed.

State transitions are translated into variable assignments to states in main loop. Each transition corresponds to assigning a new state identifier to the state variable. The assignment will be done after the action_on_exit actions are performed.



```
datain/
if(N<48){
   processData();
   N++;}
Else{
   processData();

SendDataToGfilter(
);
   N=0:
```

Figure 3. A Statechart Example

```
if (true)//there is no final state
{
   switch(state)
  {
        case state waitdata:
         wait for event indata
         if (N<48){
            processData();
              N++;
            state=waitdata;
            break;
         }
          else{
              processData();
            N=0;
            State=writeout;
            Break;
        }
         case init:
            if(true) initialization();
        default:
           state=init;
    }
}
```

The above code is the template used to create the state machine based on the statechart in Figure 3. Using this template, the generated code will perform the behaviors defined in the statechart diagram.

Clocking and Component Diagram: Clock setting is an essential component in embedded system design. In the most general case, different hardware components may be clocked differently. Clock rates will affect the overall speed of the final hardware, how the component communicate, cost, power, and other important issues.

Furthermore, in synthesized code, processes are sensitive to clocks. The clock rate will therefore directly determine their behavior.

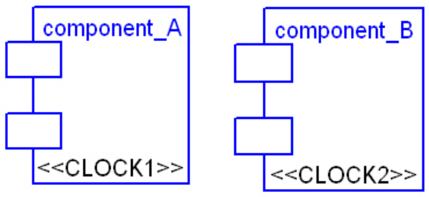


Figure 4 Example of Component Diagram

It is quite natural to describe clock settings as a property of components. Therefore, we chose to use the component diagram to model clock settings. Here again, we make use of stereotypes. In the Figure 4, components are the module instances and the stereotypes of the components are set as CLOCK*x*, where *x* is the period of the clock. In Figure 4, we have the clock settings of two components. There are one instance of Component_A and one instance of Component_B. The clock period of Component_A is set to be 1ns, while Component_B is 2ns. For components without explicit clock setting, a default clock with the period of 1ns is used. The following is the SystemC code for creating clocks with clock period equals to 1ns, 2ns, and 10ns.

```
  sc_clock base_CLK; //default clock
  sc_clock CLOCK2("CLOCk2", 2, 0.5, 0, false);
  sc_clock CLOCK10("CLOCk10", 10, 0.5, 0, false);
```

The clock will be connected to the component when they are defined in the driver class. So far, the clock type is not included here, however, in similar way as the clock period, the processor should be able to process the clock type and translate accordingly.

IV: CASE STUDY

Software Radio: A software radio is a radio whose channel modulation waveforms are defined in software [9]. Software radios employ a combination of techniques that include multi-band antennas and RF conversion; wideband ADC and digital to analog conversion (DAC); and the implementation of IF, baseband and bitstream processing functions in general purpose programmable processors. The resulting software-defined radio in part extends the evolution of programmable hardware, increasing flexibility via increased programmability. It also represents an ideal that may never be fully implemented but that nevertheless simplifies and illuminates tradeoffs in radio architectures that seek to balance standards compatibility, technology insertion and the compelling economics of today's highly competitive marketplace.

Digital Down Converter: We implemented a digital down converter (DDC) for the global system for mobile communications (GSM) - a wireless communication protocol. Digital radio receivers often have fast analog to digital converters delivering vast amounts of data. However, in many cases, the signal of interest represents a small proportion of that bandwidth. A DDC is a filter that extracts the signal of interest from the incoming data stream. Our implementation closely follows the MATLAB example in Xilinx's system generator (see Figure 5).

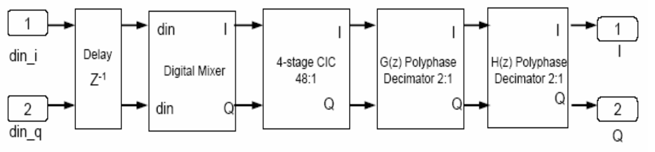


Figure 5 Block Diagram of Digital Down Converter

DDC Structural model and translation: From Figure 5, we can see that the Digital Down Converter consists of a mixer, a cascade integrator-comb and two decimators. The desired channel is translated to baseband using the digital mixer comprised of multipliers and a direct digital synthesizer (DDS). The sample rate of the signal is then adjusted by a multi-stage, multi-rate filter consisting of a cascade integrator-comb (CIC) filter and two polyphase finite impulse response (FIR) filters with a decimation factor of 2. The functions performed in the system are complex multiplication, and multi-rate filtering. The overall down sampling rate of the converter is 192:1.

Each of the components is modeled as a class, and they communicate through event sending (see Figure 8). The model has been translated into both TLM, behavioral and RTL levels. We could not find the source code for a similar DDC in UML or SystemC for comparison. Hence we have compared just the FIR module of our design with an FIR example provided by Synopsys. The only modification we did to the Synopsys code was to ensure that the coefficients and the bit-widths of the ports are the same as those of our FIR model. The codes were compiled into gate-level net-list using Synopsys tc6a_cbacore library, which targets cell-based array architectures [11]. The same timing constraints were used on the synthesis runs of both. By comparisons of the final synthesized hardware, we found that our generated code uses about 33.25% more resources than the hand-coded version. We believe that this is an acceptable overhead given the fact we input the model using the Rhapsody tool with UML notations.

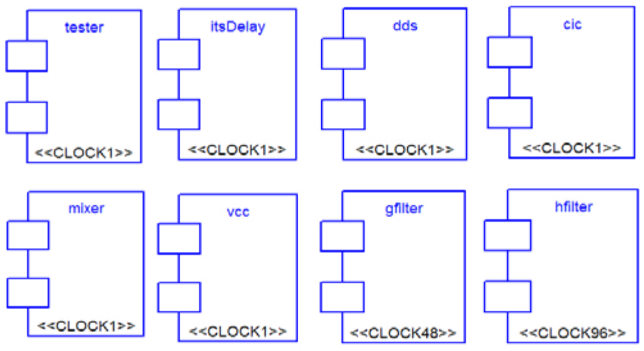


Figure 6 Component Diagram of Digital Down Converter

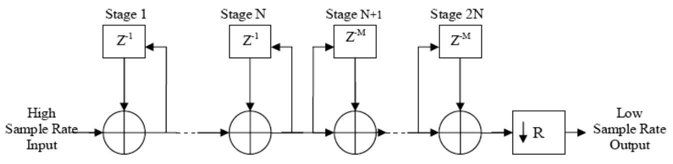


Figure 7 N-stage CIC Structure for Analysis Purposes [9]

DDC component model and clock settings: We assign different clock rate of different DDC components. Figure 6 shows the clocking setting of the DDC components. The first several components form a stream pipeline. Every cycle, each of them will process one data and send a data to its following components. The components will have rate decimation by factor of 48 after CIC. Therefore, every 48 cycles, CIC only gives 1 output and gfilter has the clock period of 48ns. The two decimators are running on low clock rate. Each of them decimates the sample rate by half. They help to further reduce the sampling rate. The Digital Down Converter can bring the incoming signal rate down from 1MHz to 5.21KHz.

Cascade Integrator-comb filter (CIC) can be used to reduce the sample rate by a large factor. In our example, a 4-stages CIC with 48:1 rate changing is modeled. **Error! Reference source not found.** shows a general structure of CIC. In our example, R=48, N=4 and M=1. We can see that there is a rate conversion module in the CIC component. The rate change is modeled as clock rate change. The sampling rate will be decrease to 1/48, and the component following the CIC is deployed on clock with 48 slower clock rate. Figure 7 shows a general analytical structure of CIC.

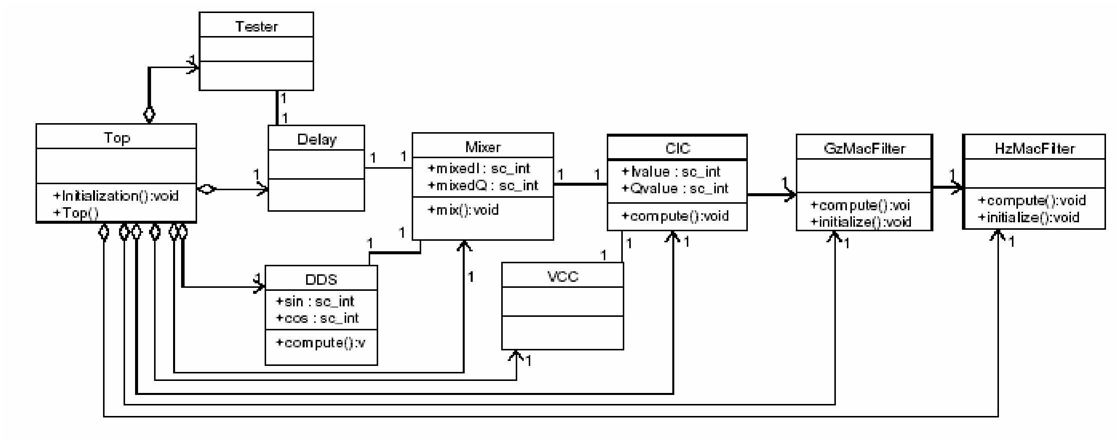


Figure 8 Class Diagram of Digital Down Converter

V: CONCLUSION

In this paper, we outlined a design flow to develop clocked hardware circuits using UML-notations. We use Class, statechart and Component diagrams to model system specifications. Our experience with the extensive case study of the design of digital down converter show that this approach works well in practice. As future work, we would like to explore further how our approach can be tied in with real-time software so as to have an integrated hardware-software specification and modeling tool chain.

REFERENCES

[1] J.R. Beauvais, T. Gautier, P. Le Guernic, R. Houdebine, E. Rutten. “A translation of Statecharts into Signal”. Proceedings of the International Conference on Application of Concurrency to System Design (CSD'98), IEEE Publ., pages 52-62, Aizu-Wakamatsu, Japan, March 1998

[2] P Boulet, J.L. Dekeyser, Cedric Dumoulin, Philippe Marquet. “MDA for SoC Embedded Systems Design, Intensive Signal Processing Experiment”. SIVOES-MDA workshop at UML 2003, 20-24 October 2003 San Francisco

[3] S. Bourduas, F. Khendek and Daniel Vincent. “From MSC and UML to SDL”. COMPSAC. P.153-158 2002

[4] P. N. Green and M. D. Edwards. “The Modelling of Embedded Systems Using HASoC”. in Proceedings of DATE 2002 Conference, Paris, France, March 2002.

[5] S. Klaus, S. A. Huss and T Trautmann, "Automatic Generation of Scheduled SystemC Models of Embedded Systems From Extended Task Graphs", Proc. Int. Forum on Design Languages, Marseille , France, September 2002

[6] G. Martin, L. Lavagno, Jean Louis-Guerin. “Embedded UML: a merger of real-time UML and co-design”, Proceedings of the ninth international symposium on Hardware/software codesign, p.23-28, April 2001

[7] OMG workgroup, UML homepage. http://www.uml.org

[8] The Open SystemC Initiative (OSCI). SystemC homepage. http://www.systemc.org/

[9] J. H. Reed. “Software Radio, a modern approach to radio engineering”. Prentice Hall PTR. 2002

[10] V. Sinha, F. Doucet, C. Siska, R. K. Gupta, S. Liao, A. Ghosh. “YAML: A Tool for Hardware Design Visualization and Capture”. In Proc. International Symposium on System Synthesis, 2000

[11] Synophsis company. “CoCentric SystemC™ Compiler RTL User and Modeling Guide”, Synophsis Inc. 2003

[12] W.H. Tan, P.S. Thiagarajan, W.F. Wong, Y. Zhu and S.K. Pilakkat “Synthesizable SystemC Code from UML Models”. National University of Singapore, School of Computing. 2004

[13] Q. Zhu, A. Matsuda, S. Kuwamura, T. Nakata, and M. Shoji. “An Object-Oriented Design Process for System-on-Chip using UML”. In Proc. of the 15th International Symposium on System Synthesis (ISSS 2002), Kyoto, Japan. p.249-254, 2002