UML based Evaluation of Reconfigurable Shape Adaptive DCT for Embedded Stream Processing

Xianhui HE[1], Yongxin ZHU[1], Zhenxin SUN[2], and Yuzhuo FU[1]

[1] School of Microelectronics
Shanghai Jiao Tong University
{hexianhui, zhuyongxin, fuyuzhuo}@ic.sjtu.edu.cn
[2] School of Computing
National University of Singapore
sunzhenx@comp.nus.edu.sg

Abstract. Multimedia stream standards evolve rapidly as stream applications prosper in embedded systems. A key component of standards, discrete cosine transform is being replaced by SA-DCT, whose complexity results in a large design space. The paper describes a UML 2.0 based design approach to quick evaluation of SA-DCT implementations containing both hardware and software, which are hard to describe and verify in C, Verilog and VHDL. Using the approach, we manage to study the partitioning, reconfigurability as well as performance and hardware cost. The design specifications in UML can be translated into SystemC models consisting of simulators and synthesizable code under proper style constraints. The paper demonstrates the feasibility of quick specifications, verification, evaluation and generation of embedded system designs.

1 Introduction

Since the new millennium, multimedia applications have been gaining their popularity as well as complexity. The complexity is represented by growing computation demands. Due to the busty nature, video stream processing tops the multimedia applications in terms of complexity. A lot of efforts in standards and implementations therefore are dedicated to handling challenges incurred by the complexity.

Video coding standards evolve in the same pace as multimedia applications. To cope with the computation demands of the applications, MPEG-4 was developed and adopted widely after its precedents MPEG1 and MPEG2. One of major features that identifies MPEG-4 from MPEG series standards is the Shape-Adaptive Discrete Cosine Transform (SA-DCT) proposed by Sikora and Makai in [13]. SA-DCT is able to code irregular video regions, compared to traditional block-based DCT which is used in earlier releases of MPEG standards, MPEG-1 and MPEG-2 as well as MPEG's sister standards, H.261 and H.263. The irregular video regions are boundaries between a stationary background and moving forward objects. MPEG-4 relies on SA-DCT to achieve object-based texture encoding, and operations on the objects.

SA-DCT implementations differ significantly due to many concerns, among which adaptability, reconfigurability, functional completeness, and efficiency are considered as major ones. Tseng et al. in [17] consider the SA-DCT implementation [11] proposed by Le et al. is incomplete functionally. It is also unclear whose reconfigurability is better between the design by Chen et al. [6] and the strategy by Gause et al. [7]. As such, it is hard to quickly evaluate various SA-DCT architectures.

An approach to taming the complexity of evaluation is to raise the level of abstraction to that of system-level designs. This approach incurs the needs to specify abstract systems. The Unified Modelling Language (UML) is considered fit to play this role.

Another important issue in specifying SA-DCT architectures is that the specifications must be reconfigured easily. To fit the shape of video object, SA-DCT calculations are reconfigurable and flexible in nature. Due to the arbitrary boundary of the video object, the hardware required by SA-DCT depends on the number of pixels occupied by the object within the 8×8 block involved in SA-DCT calculation.

To study the reconfigurability of SA-DCT architectures in UML, embedded system designers need to describe the architectures clearly with less efforts using UML than text-based specifications. Interestingly, specifications of SA-DCT architectures in UML are intuitive to reconfigure graphically so long as the designers are knowledgable about object-oriented programming, which is the common sense for engineers since 1990s.
Our Scope of Work and Related Work: In the domain of UML for system designs, a large body of work has been reported. Some UML extensions from UML 1.x are proposed in [10]. Two instances of UML extension and translation are [2, 18]. A previous effort [16] focused on hardware generation from UML models without explicit UML specifications of stream processing. Another earlier work [12] mainly studied the model driven architecture approach via SystemC using UML 1.x. A recent work [14] addressed the clocking issues involved in clocked circuit design in UML. A more recent work [19] explored the suitability of UML 2.0 for designs of stream processing, and established a design workflow that takes SystemC as an intermediate language.

In this paper, we will discuss two categories of SA-DCT architectures, into which hardware-favored and software-favored approaches fall before we present the specifications in UML 2.0. This discussion is to explain the existence of the tradeoff between the high performance of customized hardware and the flexibility of low cost software.

In the rest of the paper, we will brief the UML based workflow including the translator. Then we will categorize implementations of SA-DCT in terms of software/hardware partitions. That will be followed by details of UML specifications of two instances of representative SA-DCT implementations. According to the UML specifications, we will present the experimental results on performance and hardware costs. Some remarks will be given to conclude the paper.

2 The UML 2.0 based Workflow

The workflow starts with specifications in UML 2.0 using I-Logix's Rhapsody. The specifications are executable within the UML tool at the UML level. Both functional verifications and performance verifications are carried out. As such, the verifications are performed as earlier as possible. Figure 1 shows the steps in the workflow.

After verifying the functionality and performance at UML level, we export the design specifications into intermediate data via the Rhapsody XML Metadata Interchange(XMI) toolkit. These data are parsed by the jdom-based parser [8] to generate

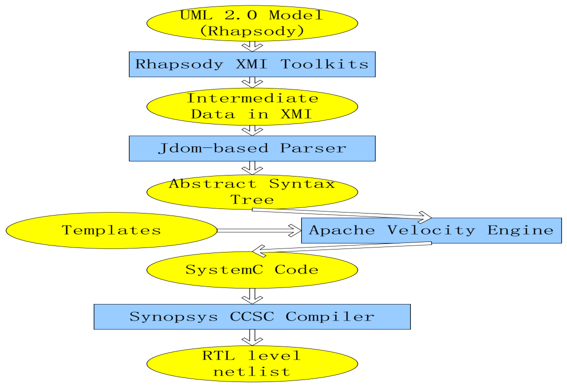

Fig. 1. The UML 2.0 based Workflow

the abstract grammar tree. These information along with a set of translation templates are feeded into Apache Velocity Template Engine [1] to generate the target code, design specifications in SystemC. The SystemC code under proper coding style constraints is acceptable to Synopsys SystemC compiler [15], which translates the SystemC code into synthesizable RTL level netlists in Verilog or VHDL. Additional hardware specification details can be included by the translator on top of the general mapping rules explained above. This action is part of refining process in the design workflow.

3 SA-DCT Classifications

For blocks inside a video object plane (VOP), block-based DCT encoding behaves identically to SA-DCT encoding. SA-DCT saves computation for processing blocks outside the VOP only. The boundary encoding process starts by packing the VOP pixels and aligning them to the upper bound of the 8x8 block. According to [13], given the column length N where $1 \leq N \leq 8$, the u^{th} DCT coefficient $F(u)$ of an 1D N-point DCT for each column is derived as Equ. 1.

$$F(u) = \sqrt{\tfrac{2}{N}} C(u) \sum_{x=0}^{N-1} f(x) \cos[\tfrac{(2x+1)u\pi}{2N}]. \tag{1}$$

$f(x)$ is the data vector, $C(u) = \frac{1}{\sqrt{2}}$ if $u = 0, 1, ..., N-1$, and $C(u) = 1$ otherwise.

To avoid expensive multipliers in terms of power consumption and chip area, many adder-based distributed arithmetic (DA) approaches [4] [3] are proposed as replacements. Given an N-tap inner product with input sequence X_i, output sequence Y_n, and constant coefficient A_i, Equ. 2 expresses the inner production as per [5].

$$Y_n = \sum_{i=0}^{N-1} A_i X_i = \sum_{k=0}^{W_c-1} (\sum_{i=0}^{N-1} A_{i,k} X_i) 2^{-k}. \tag{2}$$

W_c is the word length of A_i and $A_{i,k}$ represents the k^{th} bit of A_i. When $A_{i,k}$ equals 0, computations are saved. The only required operations are bit shifting and addition.

A hardware-favored implementation of DA is the work of Kinane et al. [9], an architecture consisting completely of hardware is proposed to optimize the usage of adders. The architecture is illustrated in Fig. 2. To make our discussion concise, we refer the

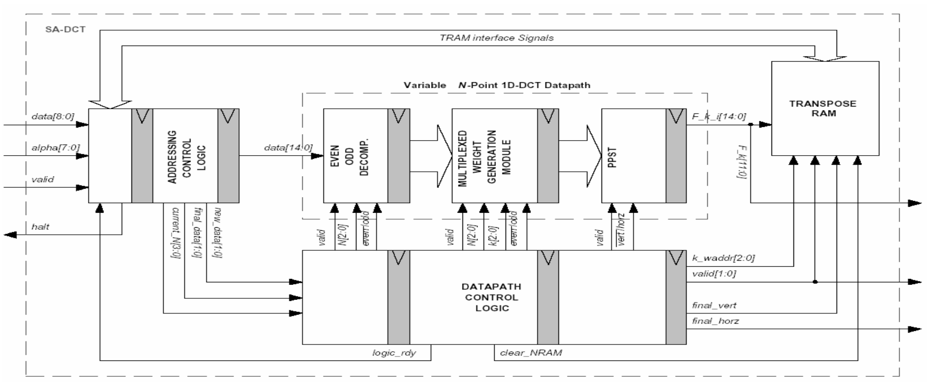

Fig. 2. A SA-DCT architecture containing hardware only, courtesy of Kinane et al. [9]

architecture of Kinane et al. as method one ($m1$) in the paper. In the architecture of $m1$, the multiplexed weight generation module (MWGM) has a reconfigurable adder-based distributed arithmetic structure adjustable to the computation of the distributed weights for N-point DCT efficient k using a 6-bit vector $\{k, N\}$. According to $\{k, N\}$, the multiplexers select the proper values of the weights. The primary adder array consists of 6 two-input adders. The secondary array includes no more than 5 two-input adders. This array combines a subset of the possible primary adder sums with elements of the selected vector.

Another software-favored example is the energy aware IP core design proposed by Chen et al. [5]. The architecture of the design is illustrated in Fig. 3. This architecture is a typical representative of SA-DCT architectures part of whose functions are implemented in software. In this paper, we refer this work as method two ($m2$). The program memory in the architecture of $m2$ identifies SA-DCT solutions containing both software and hardware partitions. The program memory stores the firmware library, which is the collection of assembly instructions to calculate DCT/IDCT in different length.

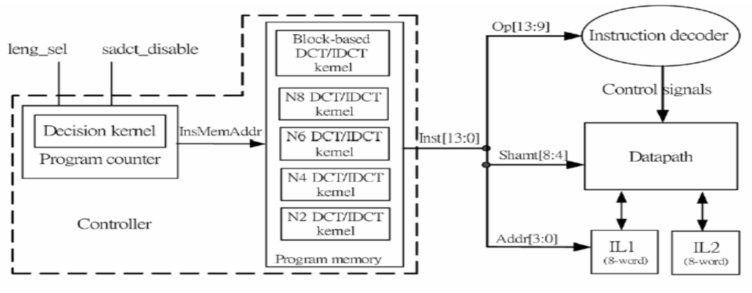

Fig. 3. A SA-DCT architecture with embedded software, courtesy of Chen et al. [6]

4 UML Specifications

Our design specifications of $m1$ and $m2$ architecture are based on UML 2.0 notations and implemented in I-Logix Rhapsdy6.1. We prefer this tool to others since it provides enhanced supports for UML2.0 such as architectural modelling, sub-machines and concurrent state charts, and component based development.

In $m1$, the execution stages are *D*atapath Control stage (DPCtrl), *E*ven Odd Decompose stage(EOD), *P*rimary Adder stage(AdderS1), *S*econdary Adder stage (AdderS2), *W*eight Max Routing stage(WMR), *P*artition Product Summation Tree stage(PPST), and *T*ranspose RAM (TRAM) stage. The details are illustrated in Fig. 4.

In $m2$, there are three major components, i.e. *D*ecision Kernel(DK), *P*rogram Memories(PM)(containing 4 N-DCT software kernels denoted as NxDCT), and *D*ata Path(DP) (containing an adder array and associated control logic). In Fig. 5, DK and DP are specified as individual objects, PM is specified as 4 instances of N-DCT software kernels.

To achieve specification in cycle-level accuracy, a *C*lock Control(clk_ctrl) component is created for both two designs as shown in Fig. 4 and Fig. 5. This clk_ctrl component provides global clock signals to all the rest components in the design.

Besides structure diagrams describing system compositions, we also use parallel statecharts to depict system behaviors. As illustrated in the hierarchical statechart in Fig. 6 shows, the clk_Ctrl object broadcasts new_cycle events to the rest objects. The states 9, 11, 13, 15, 17, and 19 send out the new_cycle events concurrently.

Testing input data are specified in dataPathCtrl class of $m1$, and decisionKernel class of $m2$ respectively. We also specify our NxDCT classes for the simulation of the program memory components of $m2$.

5 Experimental Results

Functional Verifications: The extremely early functionality verification in the design process is executed by checking the logical correctness of events during the execution of the UML specifications. Specifically, we look into the behavior of all the operating components in UML specifications. We pay special attention to all the adders involved in the calculation, especially those might stay idle during execution. More precisely, the primary and secondary adders in $m1$ and the only adder on the datapath of $m2$ are monitored carefully, on the other hand, PPST and TRAM in $m1$ and program memory of $m2$ are of less importance that we do not pay much attention.

Message sequence charts Fig. 7 and Fig. 8 visualize the communications between system components. They are automatically generated by Rhapsody during animated execution. In Fig. 7, new_cycle events, operandReadyforEOD events and addendsReadyforAdder events are examples of trigger messages that clearly demonstrate the system behavior together with their source and destination objects. In Fig. 8, N8sel event is a message example that selects one program memory among four candidates.

According to traces of message passing in Fig. 7 and Fig. 8 and additional debugging information, we verify the UML specifications function correctly as we expected.
Results on Tradeoffs: To make quantitative tradeoffs between performance and hardware cost, we consider two metrics: hardware utilization and execution cycle number.

During the UML execution, we capture information for design evaluation. The numbers of execution cycles are counted and listed in Table 1. The performance of two methods are compared in the line of reconfigurable size of SA-DCT. Though it is not surprising that the hardware-based $m1$ performs significantly better than the hardware-software solution $m2$, we would like to point out that the execution cycles for $m2$ increase more significantly than the counterpart for $m1$. The difference should be attributed to the hardware function invocations initiated by the program memory for $m2$.

no. of points in SA-DCT	1	2	3	4	5	6	7	8
exec. cycles for method1	3	4	5	6	7	8	9	10
exec. cycles for method2	14	14	28	28	66	66	119	119

Table 1. Performance statistics.

no. of points in SA-DCT	1	2	3	4	5	6	7	8
$n_{tot,m1}$	27	54	81	108	135	162	189	216
$n_{tot,m2}$	10	10	20	20	49	49	100	100
$n_{u,m1}$	16	32	48	68	85	117	137	178
$n_{u,m2}$	10	10	20	20	49	49	100	100
utilization $n_{u,m1}/n_{tot,m1}$	59.2%	59.2%	59.2%	63.0%	63.0%	72.2%	72.5%	82.4%
utilization $n_{u,m2}/n_{tot,m2}$	100%	100%	100%	100%	100%	100%	100%	100%

Table 2. Hardware utilization statistics.

In Table 2, we use the utilization rate to evaluate the hardware costs. During execution, we record the number of total running adders(in one N-point DCT) as n_{tot}, and number of useful running adders(in one N-point DCT) as n_u. Since data paths of both designs mainly consist of adders, the hardware utilization rate can be represented by n_u/n_{tot}. We denote n_{tot} for $m1$ as $n_{tot,m1}$, n_{tot} for $m2$ as $n_{tot,m1}$. n_u for $m1$ and $m2$ is denoted as $n_{u,m1}$ and $n_{u,m2}$ respectively. As such, we can tell $m1$ which needs 27 adders takes less execution cycles, while it occupies more hardware staying idle during execution. On the other hand, $m2$ containing only one adder array takes more execution cycles, however it keeps the datapath components fully utilized during execution. The price of the full utilization for $m2$ is the much higher memory cost of $m2$ than $m1$ as $m2$ needs four program memories to store different N-point DCT codes.

As to reconfigurability, both $m1$ and $m2$ behave excellently adapting to the variable number of points N in DCT. There are just enough adders and associated hardware logic for $m1$ to process $1-N$-points DCT. There is no stall in the pipeline processing of the datapath. To adapt to variable number of points N in DCT, $m2$ only needs to change the control signals of the multiplexer.

6 Concluding Remarks

In this paper, we described an approach to specifying embedded stream processing in UML 2.0. Based on the approach, specifications of shape adaptive DCT in both hardware and software partitions are coined. To describe both partitions is hard for C, Verilog or VHDL. It is also feasible to generate SystemC code from specifications in UML

2.0. More importantly, the execution of the specifications enables the early functionality verifications as well as cycle accurate performance evaluation of different architectures. Resource utilization rates are also countable after specifications execution. In our road map, our framework will grow into a complete system specifications and verification solution with more components of embedded multimedia systems.

References

1. Apache_Jakarta_Project_Group. User guide to velocity template engine[online]. In *http://jakarta.apache.org/velocity/docs/user-guide.html*, 2005.
2. F. Bruschi. A systemc based design flow starting from uml models. In *The 6th European SystemC users Group Meeting*, 2002.
3. T. S. Chang, C. S. Kung, and C. Jen. Hardware-efficient dft designs with cyclic convolution and subexpression sharing, Sep. 2000.
4. T. S. Chang, C. S. Kung, and C. Jen. A simple processor core design for dct/idct, 2000.
5. K.-H. Chen, J.-I. Guo, J.-S. Wang, C.-W. Yeh, and J.-W. Chen. An energy-aware ip core design for the variable-length dct/idct targeting at mpeg4 shape-adaptive transforms, 2005.
6. K.-H. Chen, J.-I. Guo, J.-S. Wang, C.-W. Yeh, and T.-F. Chen. A power-aware ip core design for the variable-length dct/idct targeting at mpeg4 shape-adaptive transforms. In *The IEEE ISCAS*, pages 141–144, 2004.
7. J. Gause, P. Cheung, and W. Luk. Reconfigurable computing for shape-adaptive video processing, iee proc.-comput. digit. tech., vol. 151, no. 5,, 2004.
8. JDOM_Project_Group. Jdom and xml parsing[online]. In *http://www.jdom.org/downloads/docs.html*, 2002.
9. A. Kinane, V. Muresana, and N. OConnora. An optimal adder-based hardware architecture for the dct/sa-dct, 2005.
10. L. Lavagno, G. Martin, and B. Selic. *Uml for Real: Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, 2003.
11. T. Le and M. Glesner. Flexible architectures for dct of variable-length targeting shape-adaptive transform, ieee transactions on circuits and systems for video technology, vol.10, no. 8, 2000.
12. K. Nguyen, Z. Sun, P. Thiagarajan, and W. Wong. Model-driven soc design via executable uml to systemc. In *The 25th IEEE Int'l Real-Time Systems Symp.*, pages 459–468, 2004.
13. T. Sikora and B. Makai. Shape-adaptive dct for generic coding of video, ieee transactions on circuits and systems for video technology, vol.5, no. 1, 2002.
14. Z. Sun, W. Wong, Y. Zhu, and S. Pilakka. Design of clocked circuits using uml. In *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific Volume 2*, pages 901–904, 2005.
15. Synopsys_Inc. Cocentric systemc compiler rtl user and modeling guide. In *Synopsys Incorporation*, 2003.
16. W. H. Tan, P. S. Thiagarajan, W. F. Wong, Y. Zhu, and S. K. Pilakkat. Synthesizable systemc code from uml models. In *UML for Soc Design, DAC 2004 Workshop, www.comp.nus.edu.sg/∼zhuyx/usoc04.pdf*, June 2004.
17. P.-C. Tseng, C.-T. Haung, and L.-G. Chen. Reconfigurable discrete cosine transform processor for object-based video signal processing. In *The IEEE ISCAS*, pages 353–356, 2004.
18. Q. Zhu, A. Matsuda, and M. Shoji. An object-oriented design process for system-on-chip using uml. In *The 15th Int'l Symp. on System Synthesis*, pages 249–254, 2002.
19. Y. Zhu, Z. Sun, A. Maxiaguine, and W. Wong. Using uml 2.0 for system level design of real time soc platform for stream processing. In *IEEE 11th Int'l conference on Embedded and Real-Time Computing Systems and Applications*, pages 154–159, 2005.

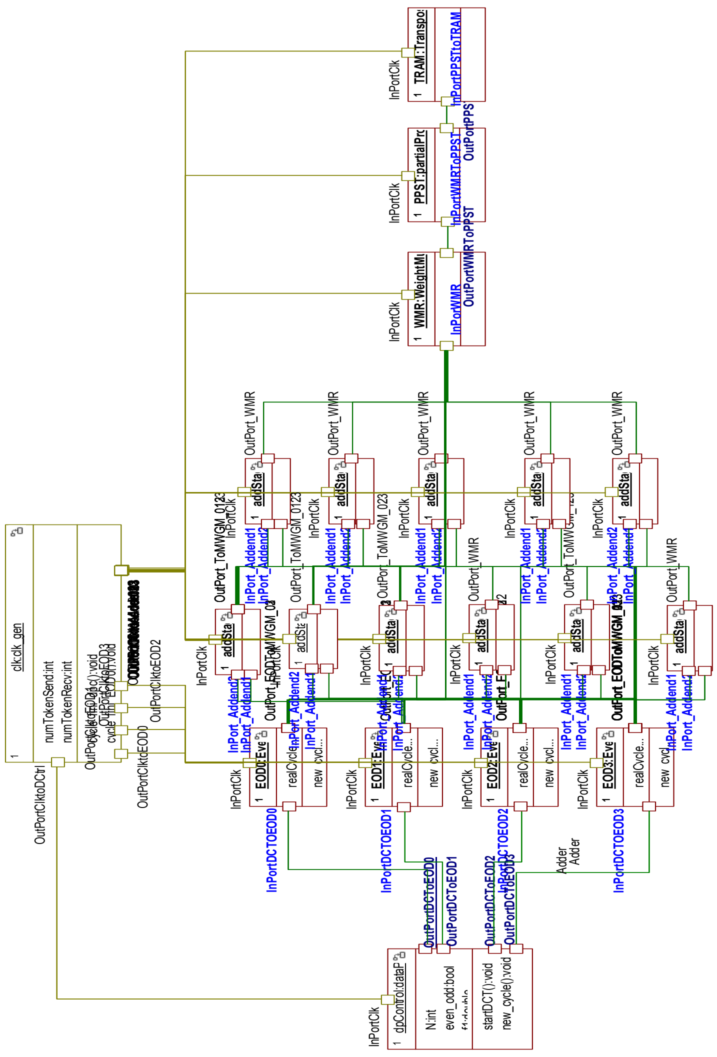

Fig. 4. The object model of the method by Kinane et al. [9]

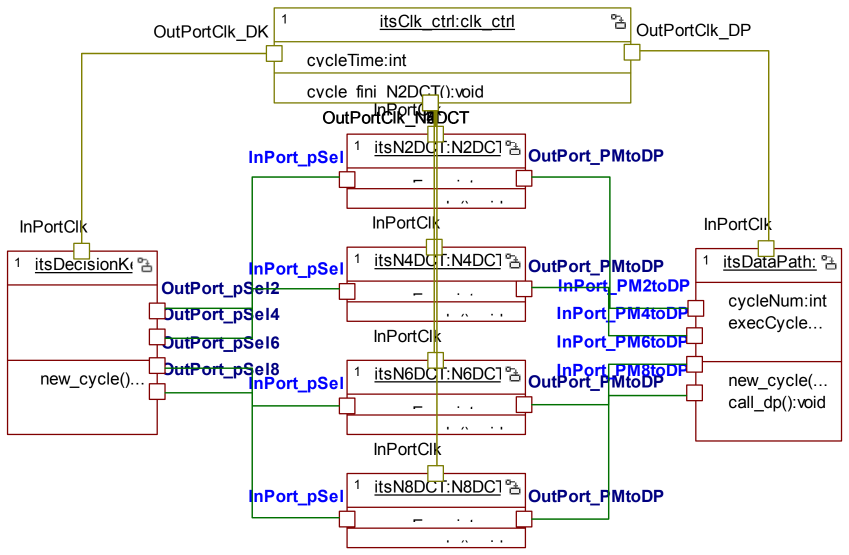

Fig. 5. The object model of the method by Chen et al. [5]

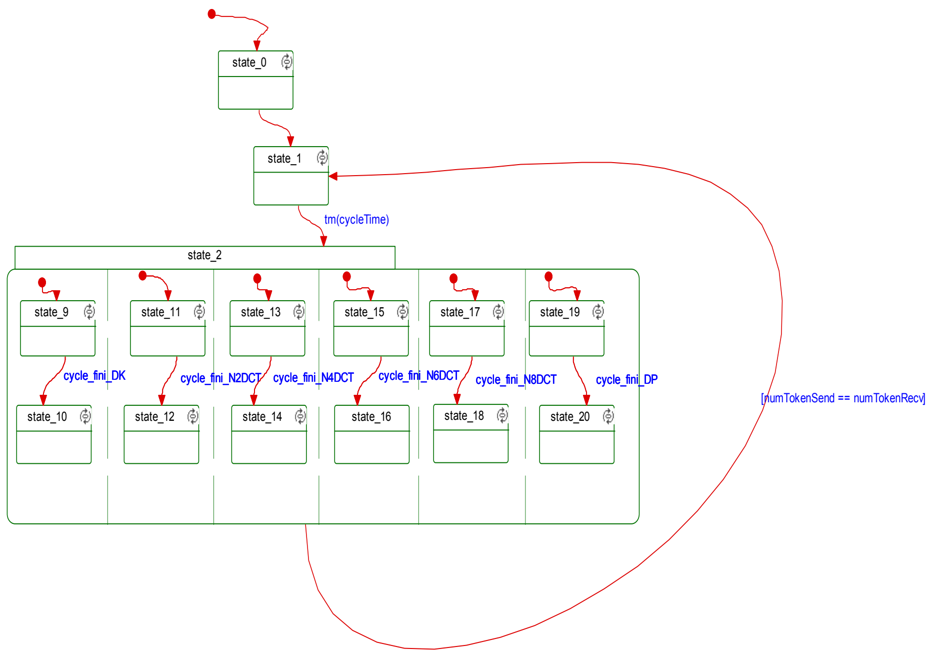

Fig. 6. A hierarchical statechart of the Clk_Ctrl class in the design by Chen et al. [5]

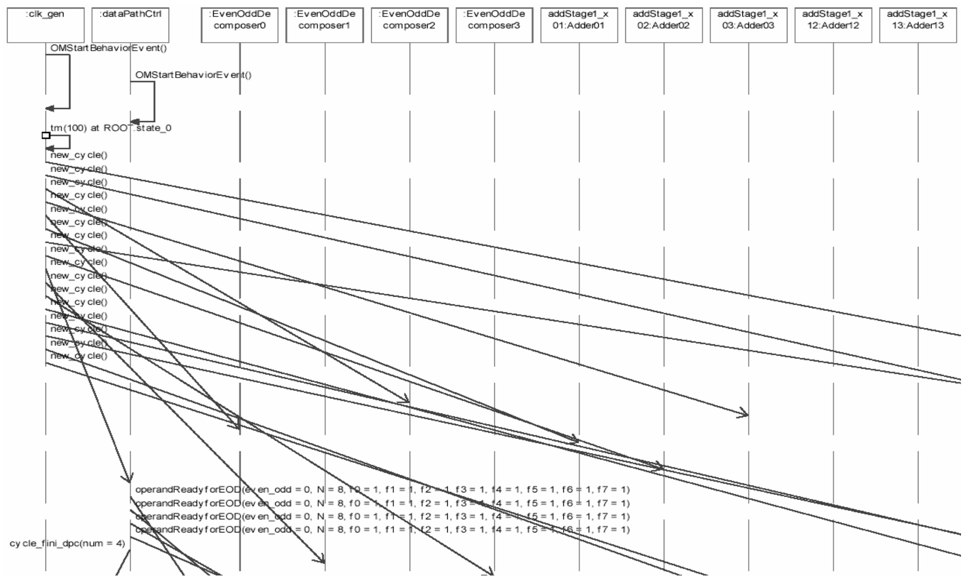

Fig. 7. Animated sequence diagram of the design by Kinane [9] after zooming

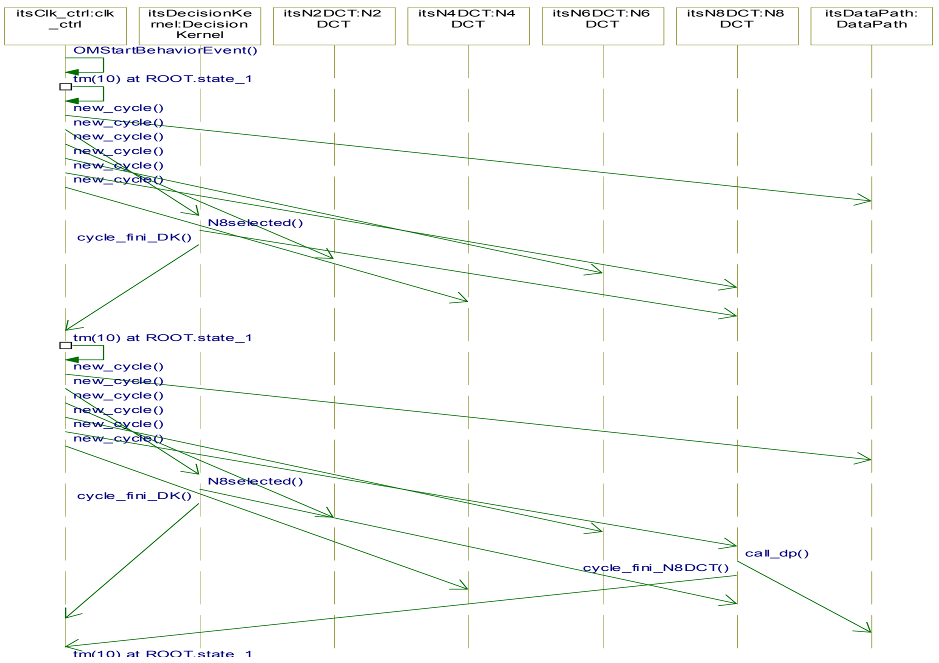

Fig. 8. An animation sequence diagram of the design by Chen et al. [5]