International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:3, No:12, 2009

High Level Synthesis of Kahn Process Networks (KPN) for Streaming Applications

Attiya Mahmood, Syed Ali Abbas, Shoab A. Khan
{attiya_t_m, aliabbas_5}@yahoo.com
shoab@ceme.edu.pk
College of E & ME, NUST, Rawalpindi, Pakistan

***Abstract*—**Streaming Applications usually run in parallel or in series that incrementally transform a stream of input data. It poses a design challenge to break such an application into distinguishable blocks and then to map them into independent hardware processing elements. For this, there is required a generic controller that automatically maps such a stream of data into independent processing elements without any dependencies and manual considerations. In this paper, Kahn Process Networks (KPN) for such streaming applications is designed and developed that will be mapped on MPSoC. This is designed in such a way that there is a generic C-based compiler that will take the mapping specifications as an input from the user and then it will automate these design constraints and automatically generate the synthesized RTL optimized code for specified application.

***Keywords*—**KPN, DFG, FPGA

I. Introduction

STREAMING applications are usually represented as a set of simultaneous processes that take a stream of input data and then transforms them into processed output stream of data. Implementing such an application on hardware poses a large challenging modeling problem. For this, KPN is the best ever representation to model such applications on hardware. KPN is a set of independent processes that communicate through point-to-point fashion over unbounded buffers with blocking Read and Non-blocking Write. This provides a very simple mechanism to map an application on hardware or software as KPN. The Reads and Writes also elevate the design from the use of complicated schedules. By this, streaming applications are mapped on independent processes working autonomously after acquiring sufficient data samples on its input buffers. Such data samples are called tokens in KPN's terminology and such an execution is called firing of tokens.

Streaming applications are usually mapped on FPGA and ASIC. The main idea behind this work is to propose a system in which streaming applications will be broken down into set of separate independent processing elements (these processing elements will be set of FPGAs or ASICs), mapped through KPN and inter-process communication between these processing elements will be performed through NOC Switch. This paper demonstrates the mapping constraints on these processing elements and then finally generates a generic customized controller that automatically maps these applications on hardware. By this, the designer can simply add any number of available processors in streaming applications and automatically map different types of streaming applications on hardware without any manual settings and dependencies.

II. Related Work

No C-based compiler has yet been introduced in research that can automatically generate RTL synthesized KPN model based on the specifications of streaming application. But there are so many design issues in KPN implementation. A lot of research has been undergone in KPN buffer sizing, artificial deadlock detection and real time scheduler for KPN.

In 1974, Kahn proposed semantics of simple language for parallel programming. In this work, he proposed a parallel computation model where any application can be modeled into set of concurrent independent processes with unbounded FIFOs (First-In-First-Out) buffers at its inputs and outputs. Theses independent concurrent processing elements can be executed on any parallel processing units without incurring any overhead and dependencies. His main contribution of work is illustrated in [1]. [2] introduced some new features of KPN with regards to its task level parallelism and its deterministic behavior. In real sense, no memory allocation can be unbounded, so a lot of research has been made in optimum buffer sizing for KPN implementation. [3] proposed an automatic buffer sizing for KPN on MpSoC. They proposed an idea of automatic buffer sizing by starting with some fixed sizing and incrementally increase buffer sizing wherever needed. In KPN, FIFO read is blocking and FIFO write is non-blocking based on the assumption of unbounded buffer size. When we start imposing the impact of finite memory sizes then an artificial deadlock issue arises. [4] demonstrated the effectiveness of KPN in media and signal processing applications and presented the method of effective and bounded execution of KPN. [5,6] deals with this artificial deadlock detection when all the processes in the process network are blocked then they claim of finding the effective solution.[7] suggested a new idea of an early detection of artificial deadlocks in the process network of eclipse shape. In recent so many years, KPN has been modified in the set of different DSP designs because it is compositional and it allows parallelism. The output of the KPN is independent of the flow of sequence of execution. [8] presented the idea of designing and analysis of DSP designs using Kahn process networks. [9] proposed the idea of basic transformation of basic DSP designs into Kahn networks, but he did not focus on the task level decomposition of the particular DSP design and also the automatic controller for it. [10] diminished the concept of artificial deadlock in process networks and proposed a design of real time scheduler for process networks on multiprocessor system on chip. Because of KPN's effectiveness, it is consistently used for mapping streaming (Audio or Video) applications on MpSoC. Compaan and Laura

in [11] projected a system design where they take an application written in Matlab and automatically give the transformation which can be mapped on to target platform. YAPI in [12] provided a C++ interface that gives KPN implementation on single processor. [13] offered the idea of KPN exploration on multiprocessor system on chip.

In this paper, our field of interest lies in streaming application mapping. For that, we have proposed an architecture in which we have designed a C-based compiler that can automate the design constraints and automatically generates the synthesized HDL implementation of KPN that is application independent.

III. Design Methodology

As KPN is a network of concurrent processes that communicates through a set of unbounded FIFO (First In First Out) Buffers. As unbounded FIFO size is not realizable in true sense, so confining the size of FIFOs to minimum without affecting the network performance is an interesting research problem and gaining more attention for researchers.

In this research work, The typical KPN structure is implemented on a reconfigurable platform. Fig. 1 demonstrates the basic KPN model in which an application is broken down into set of processing units and their communication is performed through set of FIFOs. Each processing unit is defined with its name and number of cycles or time units, they take in execution. Also, each processing node is connected to set of FIFOs at its input and output. These nodes will not execute as long as it finds desired number of RC (Rate of Consumption) tokens on its input FIFO buffers. The processing units will check on its input FIFOs, when it acquires sufficient tokens, it will start executing. The control is given to this processing unit as long as it executes. After its execution, RP (Rate of Production) number of processed tokens will be written on its output buffers. In this model, Node 'A' is taking the continuous stream of data. When its input FIFO buffer 'F1' will store two tokens, then process A will fire and takes eight cycles for its execution. After completing this processing, it will write two, three and one tokens on FIFOs 'F2', 'F3' and 'F4'. Process 'B' and 'C' will execute when they find two and one tokens on its input FIFO buffers i.e. 'F2' and 'F4'. Process 'D' will not execute until it finds two, three and one tokens on its input buffers i.e. 'F5', 'F3' and 'F6'. Process 'D' is continuously writing data to its output buffer 'F7'. This is how streaming applications are mapped through KPN structure.

Fig. 2 shows the very basic example of JPEG compression. Implementing JPEG is a good example to explain effectiveness of KPN in streaming applications. The raw image taken from the source is saved in FIFO 'F1'. Node '1' performs the RGB to YCbCr conversion and stores the transforms image to FIFO 'F2'. The Node '2' waits to perform the conversion to take place and once FIFO 'F2' acquires this data, it fires and computes DCT and writes the result in FIFO 'F3'.Now Node '3' and '4' sequentially fire and compute Quantization and Entropy coding and write data in FIFO 'F4' and 'F5'. This is how any streaming application can be mapped through KPN structure without incurring any overheads.

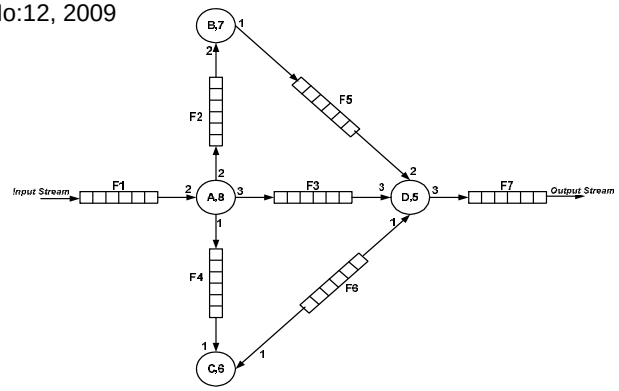

Fig. 1 Basic KPN model

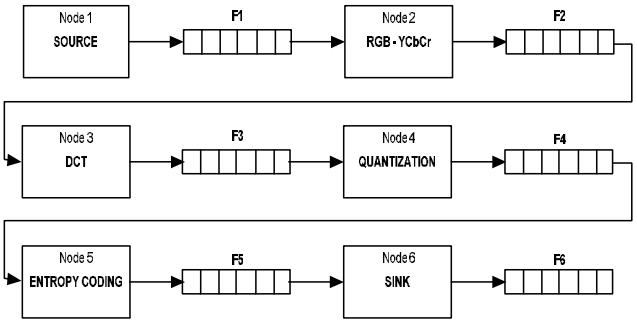

Fig. 2 KPN implementing JPEG compression

The above example shows the effectiveness of KPN in modeling streaming applications on reconfigurable hardware platforms. Thus there is a great need arises to design a controller that automatically maps such diverse applications on reconfigurable platforms like FPGAs or ASICs.

Fig. 3 shows our proposed scheme that is implementing a KPN based mapping of streaming applications on computing platforms. Here, we perform the following steps

- First, we take a configuration file of specific format from the user that lists the requirement specifications to generate automatic KPN i.e. behavior of a node i.e. whether it is combinational, sequential, taking fixed cycles or a dynamic node, how many nodes are present in a network, how many FIFO buffers are required to model this specific application, buffer sizing, rate of consumption and production parameters on each link, number of time units required to perform execution, algorithmic delays, self loops information and input stream of raw data that needs to be processed.
- Then, this configuration file is passed to C-based compiler that will automate this file. This compiler will read the design specifications and automatically generate synthesized RTL Verilog code of KPN controller and its test bench. This

compiler will also automatically generate RTL Verilog codes of all the processing nodes present in this network. "Fig. 3," shows listings of files generated by this compiler. It generates FIFO Verilog File that provides basic FIFO operation that involves simple reads and writes into FIFO. It also gives the information of rate of consumption status of the FIFO. By, this, we can calculate whether RC tokens are accumulated in FIFO buffer or not.

- Next, this compiler generates set of Verilog files depicting all the processing nodes behaviors. N processing nodes files are generated specifying number of data input units, output units and total number of time units by each processing node to perform its successful execution.
- Next, this compiler will generate the main controller file that will pass control signal to each processing module and manage all the coordination and timing constraints among each component. It will continuously view the status of each element and provide the control to each block when is desirable.
- Lastly, this C-based compiler will also generate the Verilog based test bench module that will verify the controller behavior managing all the sub units in the design.
- Thus by this, an automatic mapping of a streaming applications is performed through this C-based compiler. This compiler will allow you to map any type of applications on reconfigurable platforms by specifying different design constraints in configuration file.

A. Algorithm

- **KPN_Controller(); //Main File**

Nodes ← Number of available nodes
Links ← Total number of available links
FIFOs ← Total number of FIFOs required, storing information at links

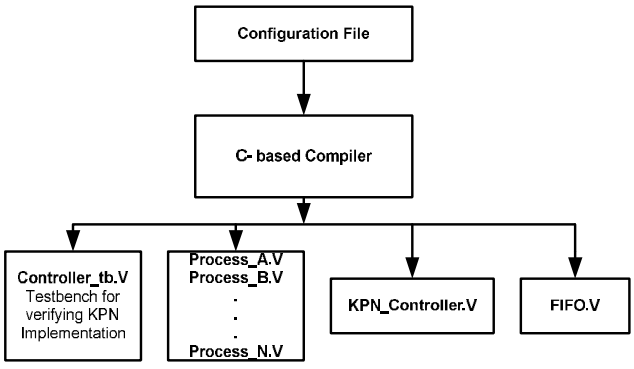

Fig. 3 System Model

```
For i ← 1 to Links
    RC[i] ← Rate of consumption parameter at link i
    RP[i] ← Rate of production parameter at link i
  Delay[i] ← Algorithmic delay at link i
End For

Topology Matrix Generation

For i ← 1 to Nodes
    Throughput[i] ←Execution time for node i
End For

For i ← 1 to Nodes
  If (RC tokens found at its each connected Links)
    While FIFO_Read(); //Read Tokens from FIFO
      If(all input FIFOs are Read)
          Break
      End If
    End While

    Done=Call Process_node(); //Functionality of
                              //node is performed
    If(Done)
      While FIFO_Write(); //Write processed data
                          //tokens
        If(all input FIFOs are Written)
            Break
        End If
      End While
    End If
      End If
End for
```

- **FIFO_Read(); //Reading Data from FIFO**

```
If(Read)
    If(FIFO_Empty_Flag)
        Process is Blocked
    Else
        Output ← FIFO(index)
    End If Else
Else
    Do Nothing
End If Else
```

- **FIFO_Write(); //Writing Data to FIFO**

```
If(Write)
    If(FIFO_Full_Flag)
        Process is Blocked
    Else
        FIFO(index) ← Input Data
    End If Else
Else
    Do Nothing
End If Else
```

- **Process(); //Processing Units**

For i ← 1 to throughput
//Processing;
End For
Done =1;
Return (Done);

IV. Implementation Details

Focusing on implementation of our proposed scheme layout, we used some useful tools to put this layout in realization. First, we simulated this design approach in MATLAB version 7.0 for the verification of controller design algorithm. It was very desirable because we need to specify our input variables and all the possible instances of this controller. After this controller design simulation, we then implemented it based on the hardware constraints. For that, it was advantageous for us to first analyze this design on hardware then it would be possible for us to focus on its generic design realization. So, we first designed this controller in Verilog (that is a hardware descriptive language). For that, we used the tool Modelsim version 5.7g. After this hardware design consideration and its successful implementation on hardware, we designed the generic compiler in C-language with the use of Visual C Version 6.0 tool. By this, we designed the C-based compiler that can automatically generate synthesized Verilog code of any streaming application.

For this central controller, we need to first divide the streaming application into set of processing nodes and then classify the nodes as combinational, sequential (taking fixed clock cycles) or dynamic i.e. taking variable number of clock cycles for its execution. A sample clock is the clock that is taking new sample in the logic whereas a circuit clock is usually much faster than a sample clock and executes the logic in each node. A combinational node does not require any control signal except the reset signal that is used to reset any feedback registers in the design. A node requiring pre defined number of clock cycles requires a start signal from the controller whereas the dynamic node requires a start signal from the controller and after its execution it generates the done signal to notify the controller about its completion.

Fig. 4 shows a simple DFG example that has three processing elements/nodes A, B, and C. These processing elements are taking 3, 2 and 1 clock cycles for their execution. Also, these processing nodes will execute only as long as it acquires sufficient number of data tokens on their input buffers. There are four FIFO buffers for temporarily holding data values. Also, nodes are listed with their rate of consumption and production parameters. Rate of consumption is defining the number of data tokens sufficient for node to process and rate of production parameter defines the number of data values that are produced by the respective processing element.

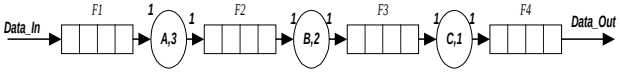

Fig. 4 An example of DFG mapped by KPN

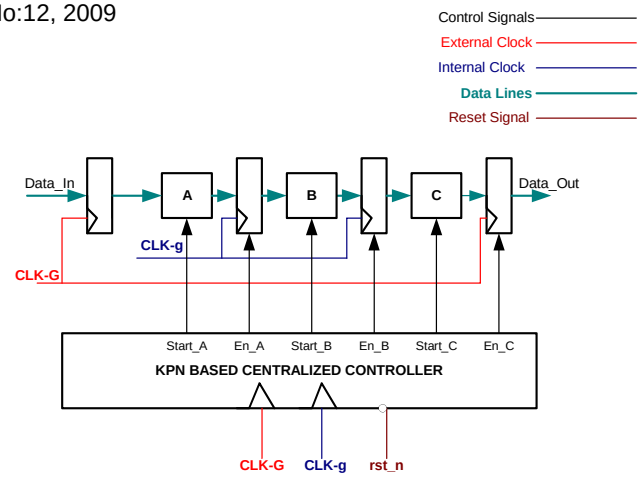

Fig. 5 KPN based centralized controller for above example

Fig. 5 shows the block diagram of the centralized controller that is automatically generated for this application and can be executed and verified by C-based compiler. This controller shows that there are three processing elements that are running on circuit clock that is usually faster than sample clock. Sample clock CLK_G is the clock on which every new sample is acquired for processing while CLK_g is the circuit clock on which every DSP component is mapped. Controller will send start signals to respective nodes when they have acquired sufficient tokens on their input buffers. After this start to execute signals, the process node will take throughput number of clock cycles for their execution. Following these throughput number of clock cycles, the controller automatically enables the respective output buffers for holding the processed output data for temporary storage. This is how any DSP application can be divided in to sub-units. By making its execution independent, now, this DSP design can be mapped on set of processors running concurrently.

Fig. 6 shows the simulation results for this example. C-based compiler is designed in Microsoft Visual C++ version 6.0 professional edition. This diagram shows that our C-based compiler has generated synthesized Verilog code of the above DSP design that is then run in Modelsim version 5.7 (Tool for simulating Verilog based design). This diagram demonstrates that data stream is coming to this application domain. At each positive edge of clock, RC parameters of all nodes are checked. Once RC parameter has attained for particular node then the Process_enable signal is asserted. After this, the controller waits for throughput number of clock cycles for process's execution, and then Process_done signal is asserted. This is how information is processed in our network designed model.

Fig. 7 shows another example that is used here just for demonstration purpose of our proposed work [14]. The DFG (Data Flow Graph) symbolizes best a streaming application. In DFG, an application is represented as directed graph G = <V, E> where V is the Vertex or the set of nodes and E is the directed edge representing interconnections between set of vertices. We are assuming that this is a streaming application whose graph is shown in Fig. 4. By taking this graph, we will

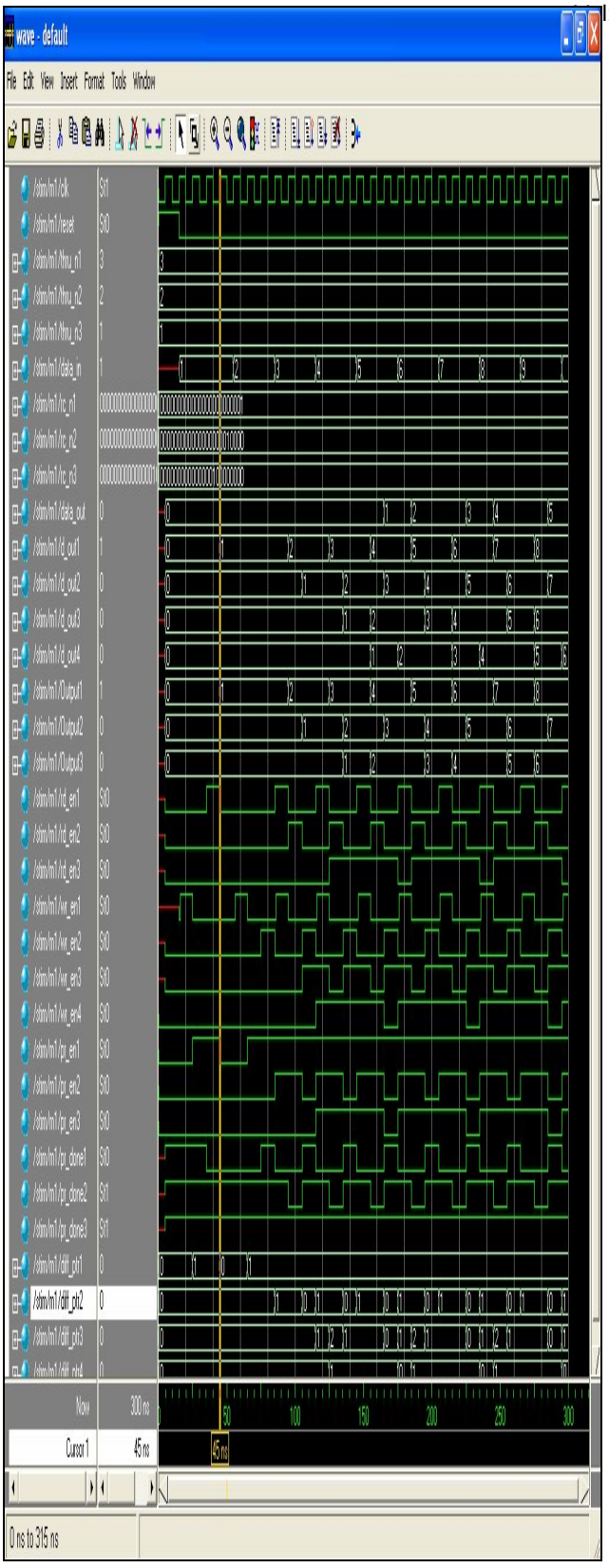

Fig. 6 Timing Diagram of controller generated by C-based compiler

generate an automatic KPN controller for this application that can be mapped on any hardware. In this figure, node 'A' is a combinational node while nodes 'B', 'C', and 'E' take fixed number of seven, eight, and nine predefined number of circuit clock cycles. The node 'D' dynamically executes and takes variable number of circuit clock cycles. Each black dot shows an algorithmic delay where data from previous iteration is used, so in hardware realization, they will run on sample clock. Based on these specifications, a generic controller can easily be designed.

Fig. 8 shows the block diagram of central generic controller that is automatically generated based on this example. Nodes with predefined number of clock cycles like 'B','C', and 'E' require a START signal from the controller and then controller counts for the number of cycles for the node to complete its execution. The controller then asserts the output enable signal to the register (particularly FIFO in this case) at the output of each node. In case of dynamic node 'D', the controller not only notifies the node to start its execution rather the node also after its execution asserts a DONE signal, as for the design in discussion a DONE is asserted and the controller then asserts En_D to latch the output from node 'D' to a register. The input and output to the DFG and the dots on the edges are replaced by the register clocked by sample clock CLK_G whereas the rest of logic in the nodes and the registers at the output of each node is clocked by circuit clock CLK_g. All the feedback registers are reset using a global rst_n signal.

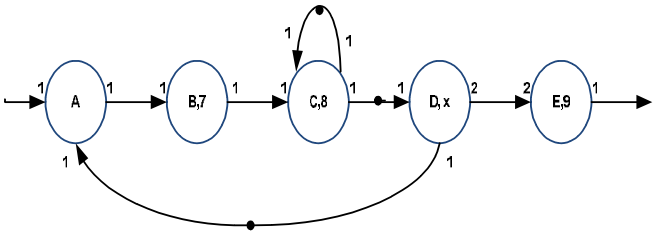

Fig. 7 An example of DFG mapped by KPN

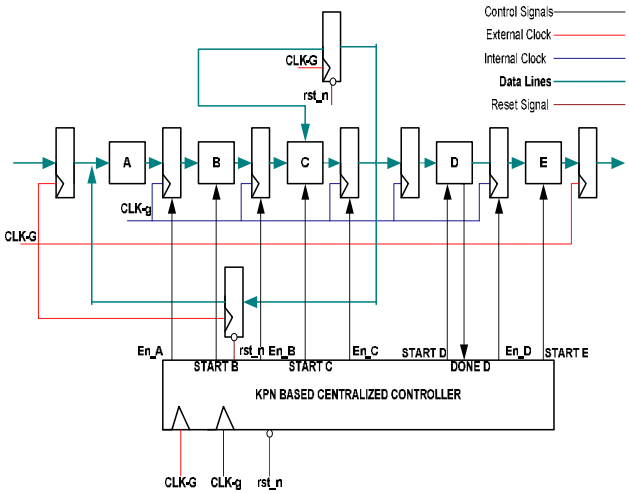

Fig. 8 KPN based centralized controller for above example

Fig. 9 shows the timing diagram of the KPN controller for the above quoted design example generated by the C-based Compiler. This diagram shows that when reset is active high, all the FIFO's and register contents are initialized to zero. There is no valid output at the data-out vector. There is high impedance indication 'Z' at the output shown with red line at the start of data_out signal in timing diagram. When reset signal is reasserted, then at the next positive edge of clock, input stream (data_in) vector starts to be written in internal FIFOs. This diagram lists the throughput clock cycles that are needed for each processing element for its execution. Data_In is the vector array that is carrying the raw information to be processed by this network. When the Rate of consumption parameters of respective process is satisfied then the write signal is asserted and computation units begin taking data from dedicated buffers/FIFOs. These computation units then take throughput number of circuit clock cycles for their execution and final processed data is to be written on output buffers/FIFOs. At that time, there is no valid output because processing is performed in its internal processing units, so the output is held zero. After the execution of internal processing units, the continuous stream of valid output occurs on output (data_out) vector. This is how any DSP application can be scheduled by this C-based compiler. The outcome of this work is that designer can map any set of DSP application on this platform and rescheduled accordingly.

V. Future Work

In this scheme, structure of KPN that can easily be mapped on any reconfigurable platforms have developed. For that, we ask the specification from the user of specific format, and then the compiler reads it and gives an automatic HDL based controller for hardware mapping. Future work will be the automatic generation of this specification file. By simply viewing the streaming application, an automatic design file will be generated that will be then passed to this generalized compiler which is giving the hardware implementation of KPN framework.

VI. Conclusion

In this paper, framework of KPN have proposed and implemented taking the input specifications of streaming applications resulting into automatic synthesized RTL code generation. This is essential because the actual critical streaming application is constituent of thousands or millions of such independent components or processing elements and managing/controlling their processing is a big challenge. Also, Execution time of such application requires more than a week and when a matter of designing a manual controller for such application comes, it becomes a huge overhead. Lastly, if the design goes fail then all your effort will go down. For this consideration, we have come to this point that designers must have application specifications/demands and based on that, an automatic synthesized RTL optimized controller must be generated without any manual considerations and overheads fulfilling their current design demands and if required, then it can easily be upgraded according to restructured design.

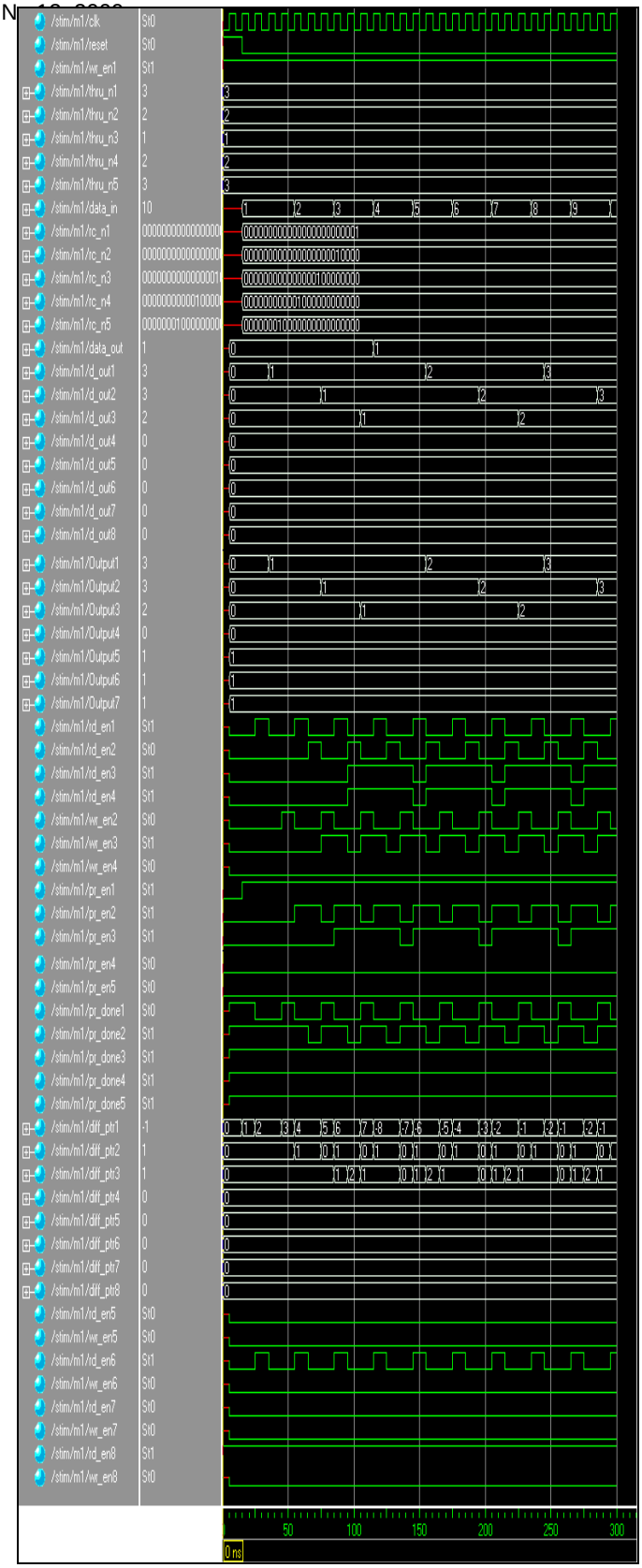

Fig. 9 Timing Diagram of controller generated by C-based compiler

REFERENCES

[1] Gilles Kahn, "The semantics of a simple language for parallel programming". In Jack L. Rosenfeld, editor, *Information Processing 74: Proceedings of the IFIP Congress 74*, pages 471-475. IFIP, North-Holland, August 1974.

[2] Edward A. Lee and Thomas M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–799, May 1995.

[3] Eric Cheung, Harry Hsieh, and Feris Baralin, "Automatic Buffer Sizing for Rate-Constrained KPNApplications on Multiprocessor System-on-Chip," Proceedings of IEEE, pages 37-44, 2007.

[4] Marc Geilen and Twan Basten, "Requirements on the execution of kahn process networks," In *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003*, pages 319-334, Warsaw, Poland, April 2003. Lecture Notes in Computer Science vol. 2618.

[5] Twan Basten and Jan Hoogerbrugge, "Efficient execution of process networks". In A. Chalmers, M. Mirmehdi, and H. Muller, editors, *Proc. Communicating Process Architectures*, pages 1-14, Bristol, UK, September 2001. IOS Press

[6] Thomas M. Parks, "*Bounded Scheduling of Process Networks*," PhD Thesis, EECS Department, University of California, Berkeley, CA, December 1995.

[7] Bharath N., S.K. Nandy, and Nagaraju Bussa, "Artificial Deadlock Detection in Process Networks for Eclipse", Proceedings of 16th International Conference on Application-Specific Systems, Architectures and Processors, IEEE Computer Society, 1063-6862/05, 2005

[8] Cepons J., Kazanavicius E., Mikuckas A., "Design and Analysis of DSP systems using Kahn process Networks," DSP Lab, Kaunas University of technology, ISSN 1392-2114Ultragarsas, Nr .4(45), 2002.

[9] Zvironas A., Kazanavicius E. Partitioning of DSP tasks to Kahn network. KTU. Kaunas. Ultragarsas. ISSN1392-2114, 2002. Nr. 2(43).

[10] Javed DULLOO, Philippe MARQUET, "Design of a Real-Time Scheduler for Kahn Process Networks on Multiprocessor systems," Rapport LIFL # 2003-06, september 2003.

[11] Todor Stefanov, Claudiu Zissulescu, Alexandru Turjan, Bart Kienhuis, and Ed Deprettere, "System Design using Kahn Process Netwroks: The Compaan/Laura Approach," Presented at DATE'04, Paris 16-20 Feb 2004.

[12] E. A. de Kock, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P. Lieverse, K. A. Vissers, and G. Essink, "Yapi: application modeling for signal processing systems," in *DAC '00: Proceedings of the 37th conference on Design automation*. New York, NY, USA: ACM Press, 2000, pp. 402–405.

[13] P. Lieverse, T. Stefanov, P. van der Wolf, and E. Deprettere, "System level design with spade: an m-jpeg case study," in *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 31–38.

[14] Dr. Shoab A. Khan, Book: "Digital Design for Signal Processing Systems" to be published.