Modeling of Data Flow Graph for Logic Circuits

Pruthvi Gowda Thorehosur Appajigowda
Electrical and Computer Engineering
University of Arizona
Tucson, AZ
Pruthivi1990@email.arizona.edu

***Abstract*—Scheduling plays an important role in the VLSI design flow because it determines the amount of concurrency of the resulting implementing thus impacts the performance. It also plays an important role in determine the amount of hardware resources required thus impacts the area. This paper talks about designing and generating synthesizable high level state machine description from the Data flow graph in Verilog while providing scheduling alternatives like LIST_L and LIST_R using modeling Paradigm. The user can construct Data flow graph (for ex data flow graph for 6th order FIR filter) using Atoms. Atoms can be logical operators such as adder, subtractor and multipliers etc. Atoms act as graph nodes and the edges can be used to define the relationship between them in the Data flow graph.**

Once the user construct the data flow graph, equivalent high level state machine description which adheres to the chosen scheduling algorithm will be generated as Verilog code. The Verilog code generated can be synthesizable on Xilinx.

***Keywords*—Modeling, Scheduling Algorithm, List_L and List_R, VLSI Design Flow, GME**

I. INTRODUCTION

During High Level Synthesis of the logic circuits, the input specification is generally in some HDL (Hardware Definition Language) like Verilog, VHDL. The HDL Specification can be represented using modelling paradigms Like DFG (Data Flow graph) for the data path intensive operations. Data flow graph is a directed graph with a set of nodes and set of directed edges which is adept for efficient scheduling, allocation and binding procedure.

Verilog is one of the most popular programming languages used for building software models of hardware system. It can be used to specify concurrent and sequential designs at different levels of abstraction; low level structural elements as well as high level behavioral programming.

High-level synthesis is a process of transforming a behavioral description into a structural description comprising data path logic and control logic [1]. First, a behavioral description is converted into a control data flow graph (CDFG). Then operations are scheduled in clock cycles (scheduling), a hardware module is assigned to each operation (module assignment), and registers are assigned to input and output variables (register assignment).

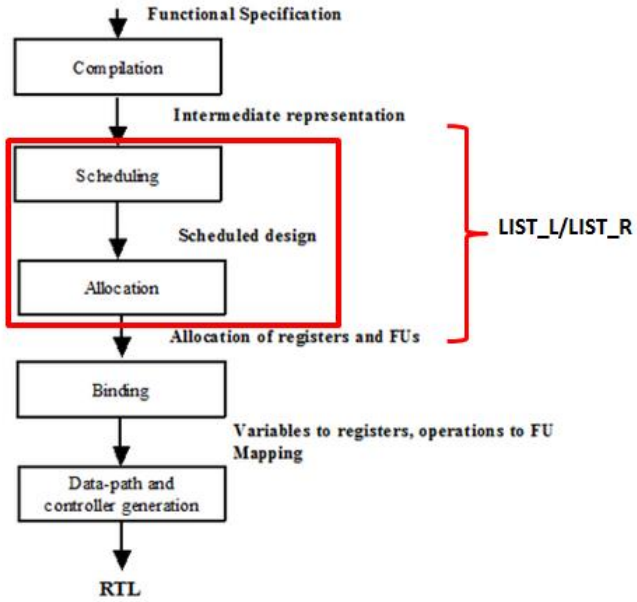

Fig 1: Procesing steps in High Level Synthesis(HLS)

In this paper, a techinque is proposed to Model Data Flow graph to generate Verilog High level state mahine description for the chosen scheduling algorithm (LIST_L/LIST_R). The rest of the paper is structured as follows, Section II gives brief overview about the related work inclined in the area of tools designed for modelling VLSI design flow. Section III gives the overview of Metamodel design used in designing the Interpreter.

Section IV discuss about the algorithm used in designing Interpreter . Section V demonstrates the case study with example. Section VI discuss about the results conclusions, followed by references in Section VII.

II. RELATED WORK

Typical VLSI design flow starts with system specification which is technical representation of design intent. HLS (High Level Synthesis) algorithms are used to convert specification in to RTL (Register Transfer Level) circuits. HLS interprets the design intent written in some high level hardware definition language like System C or Verilog. Xilinx ISE is a software tool used for synthesis and analysis of such HDL (Hardware Description Language). The output of the HLS is RTL circuit. After RTL is verified to be equivalent of the system specification, logic synthesis is performed by CAD tools like Cadence. Cadence is a backend tool used to define the Physical layout, floor planning, and placement and routing.

Scheduling problem can be of four types namely, unconstrained, time constrained, resource constrained and time-resource constrained. There are many algorithms which are proposed to solve the above scheduling algorithms, they are broadly classified into two types as heuristics and exact.

Exact algorithms like Integer Liner Programming for scheduling, provides optimal schedule but consumes high processing time. In practical cases, these exact algorithms for HLS (High level Synthesis) take prohibitive amount of execution time. To cater to the execution time issue, several algorithms based on greedy strategies have been developed that make a series of local decisions, selecting at each point the single "best" operation-control step pairing without backtracking or look-ahead. So they may miss the globally optimal solution, however, they do produce results quickly, and those results are generally be sufficiently close to optimal to be acceptable in practice. Such algorithms are called heuristic algorithms (for HLS). Examples for heuristic algorithms for HLS comprise As Soon As Possible (ASAP), As Late As Possible (ALAP), List Scheduling (LS) and Force Directed Scheduling (FDS).

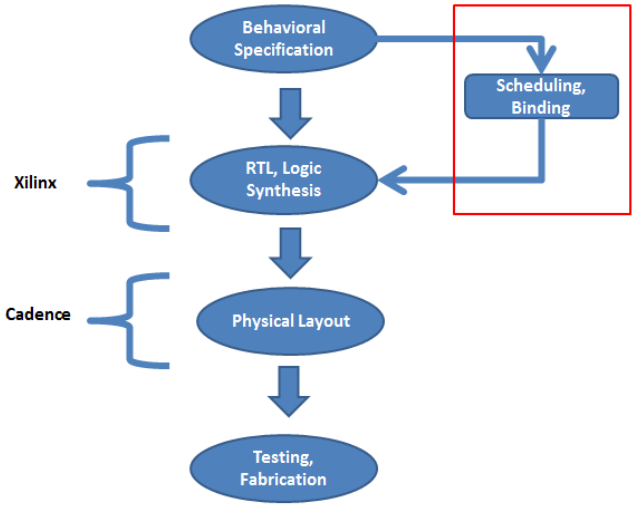

Fig 2: VLSI design flow, highlighting the scope of the tool developed

My proposed work involves modelling the Dataflow graph at scheduling stage so that user can generate HLSM description of the Scheduled Data Flow Graph (LIST_L/LIST_R) in Verilog which can be synthesized in Xilinx and can be used to determine the amount of concurrency that can be achieved and the hardware resource that are required for the operations with the modelled latency and resource constraints.

III. METAMODELING WITH GME

GME is used to model the Data Flow Graph to generate the HLSM (High Level State Machine Description) of the scheduled graph. These models are then automatically translated in to GME configuration information through metamodel interpretation. The metamodeling paradigm is based on unified modeling (UML). The syntactic definitions are modeled using pure UML class diagrams.

Domain Specific models implemented by GME are based on the generic concepts implemented by the GME itself. GME supports various concepts to build complex models. This includes MARS (Model, Atom, Reference, Sets) and explicit constraints

The UML class diagram in the figure 3 shows relationship between the different concepts involved in the design of GME metamodel, this includes hierarchy, multiple inheritances, containment, models, atoms and connection. Project contains set of Folders; Folders are like containers which contain several models. Atom

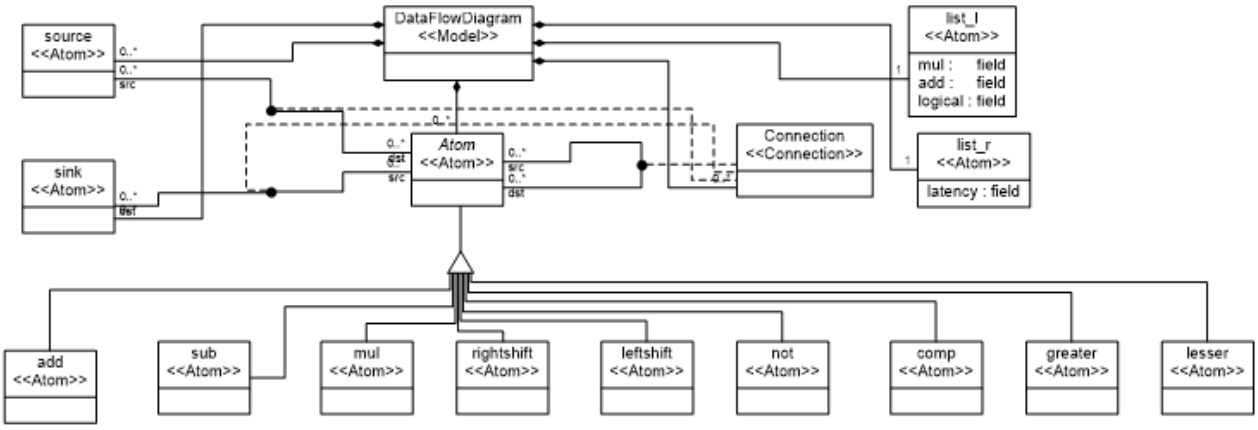

Fig 3: Meta mode design for modeling Data flow graph

References, Sets (MARS) are called as FCO objects. Atoms are elementary objects they cannot contain any items inside it. In the metamodel above the following table defines the atoms contained in the metamodel

Atoms		Semantics
Mul	(*)	Performs product of two inputs
Add	(+)	Performs the sum of two inputs
Sub	(-)	Performs the difference of two inputs
Comp	(== , <, >)	Compare the two inputs for equality, lesser and greater
rightshift	(>>)	Right shift the given input by value specified
leftshift	(<<)	Left shift the given input by the value specified
greater	(>)	Compares the given two input for greater relationship
lesser	(>)	Compares the given two input for lesser relationship
Source		Source of input
Sink		Output register
list_l		Scheduler to opt for LIST_L scheduling
List_r		Scheduler to opt for LIST_ R scheduling

Models are the compound objects; they can contain atoms and inner structure. The part in the model always has a role.

Aspects provide the primary visibility control. Every model has predefined set of Aspects. The simplest way to define the connection between the two objects in the GME is through connection. As shown in fig 4, there is a connection between the Gates which each one of the Gate atoms can act as Source and Destination. These connections are directed, since the Meta model is aimed to design directed Data Flow Graph. The following below constraints are defined for each of the atom which should be strictly obeyed while constructing Data Flow Graph in the Paradigm Sheet of the GME.

Type: Add
Constraints: Two 32 bit inputs [31:0], one 32-bit output [31:0]

Type: Sub
Constraints: Two 32 bit inputs [31:0], one 32-bit output [31:0]

Type: Comp
Constraints: Should have exactly two 32-bit inputs, Output will be one 1-bit output

Type: Ternary Operator
Constraints*:* Should have exactly two 32-bit inputs, and One 1-bit select input, output will be one 32-bit output

Type: Multiplier
Constraints: Should have exactly two 32-bit inputs, Output will be one 64-bit output [63:0]

Type: rightshift
Constraints: Should have one 32-bit inputs and one 1-bit input. Output will be one 32-bit right shifted output by the units specified by 1-bit input

Type: leftshift
Constraints: Should have one 32-bit inputs and one 1-bit input. Output will be one 32-bit left shifted output by the units specified by 1-bit input

Type: list_l
Constraints: should specify number of multiplier, number of adder and number of logical operator before interpreting the model

Type: list_r
Constraints: should specify amount of latency with in which the operation needs to be scheduled before interpreting the model

IV. INTERPRETER DESIGN

Interpreter for this project will be developed using Object Oriented C++ Programming which involves two main functionalities i) Navigating the object network ii) Querying for Individual objects. The object network is traversed, typically the containment relationship, inheritance, connections are traversed to build the dependency equation from the Data Flow Graph. During traversal, objects are queried to get the attributes. Once the dependency equations are built, it is used to generate the HLSM description in Verilog of the chosen scheduling algorithm

Algorithm:

- Get the name of the FCO model
- Function will have **void return type with the "model name"** as the name of the function
- Create a structure of type "struct gate node {…}; "which will have the data member to hold the input connection of each of the atoms and their names.
- Create an vector array of type <struct gate_node> to push all the struct's of each atom
- Function `void Component::find_num_atoms(std::string atom_name, Model &model)` will be used to get the input connection of each of the atoms present in the paradigm sheet and to store the attribute to "struct" and "push" in to the vector
- Create dependency matrix for each of the atoms excluding source, Once all the atom attributes such as "input connection, atom_name" are all stored in the structs.
- If LIST_R is selected for scheduling, call List_R_Schedule() function to generate corresponding HLSM Verilog code
- If LIST_L is selected for scheduling, call List_L_Schedule() function generate corresponding HLSM Verilog code.

Flow Chart:

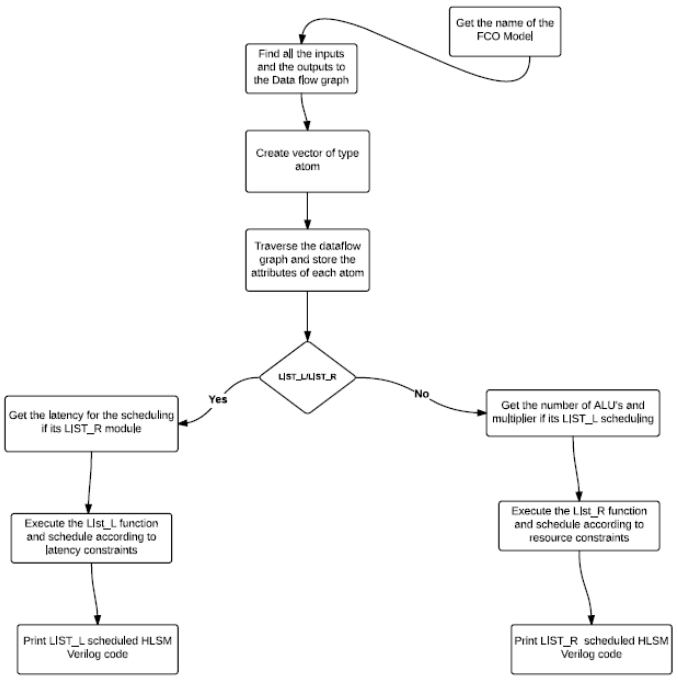

Fig 4: Flow chart for Interpreter Design

V. CASE STUDY WITH EXAMPLE

Framework for modeling Data flow graph for generating HLSM in Verilog for the chosen scheduling algorithm (LIST_L/LIST_R) is built around GME based domain specific modelling environment. A BONComponent interpreter is designed and developed in object oriented C++ language to interpret the models and to generate the Verilog code.

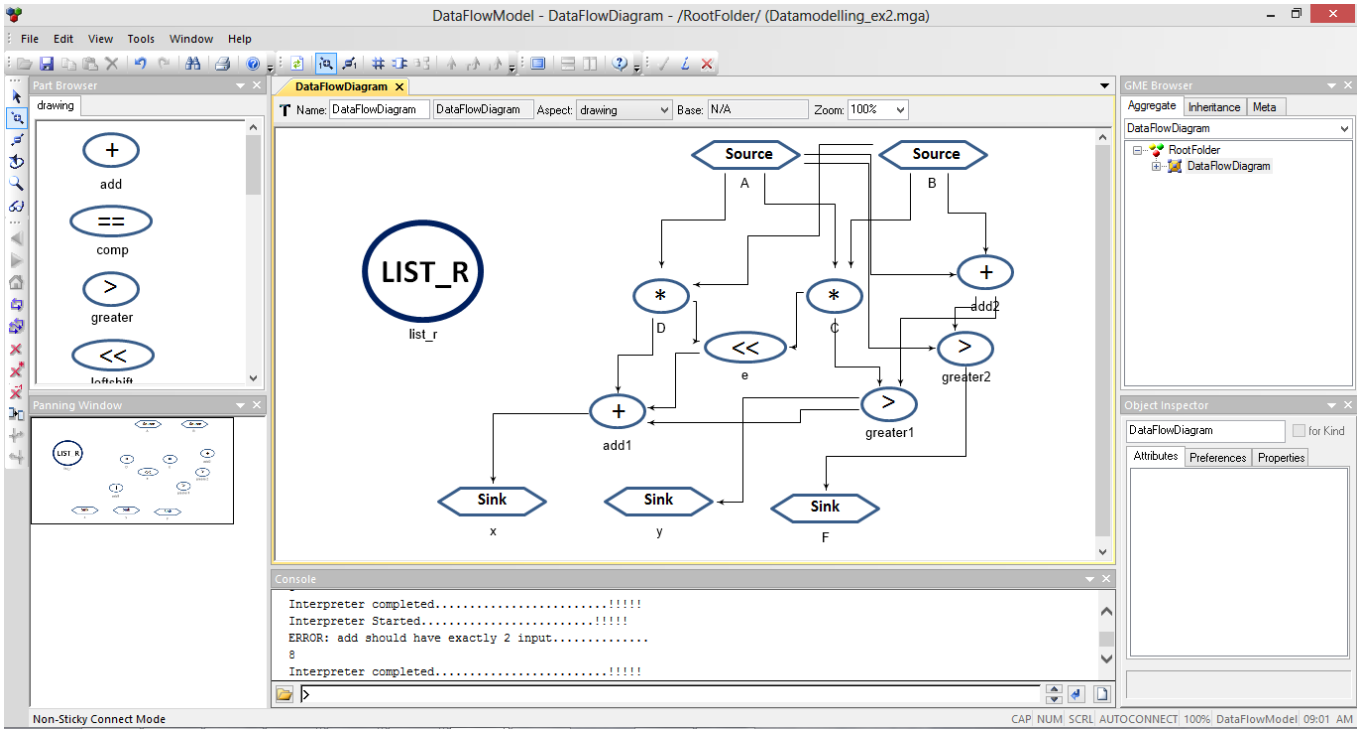

Fig 5: Example of Dataflow model to generate HLSM Verilog code with List_R Scheduling

The modeling paradigm allows the specification of operators that needed to be scheduled for the computation

Figure 5 shows the snapshot of the GME Data flow model used in designing the Dataflow graph to generate HLSM Verilog code. The resource models are hierarchical block diagrams that specify the operator which needed to schedule. Users will have the choice among LIST_L/LIST_R to define the scheduling by providing the latency and resource constraints.

The BON2Component interpreter will then interpret the Dataflow diagram by traversing the object network and querying each of the objects to build the data dependency equations internally.

Once the dependency equations are built by the interpreter internally, it will then schedule these operations based up on the latency constraint specified for the LIST_R scheduling or schedules based on the resource constraint (number of adders, sub and logical operators) for LIST_L scheduling.

Based on the results, User can validate the Data flow graph design against the resource and the latency constraints provided during the Data flow graph modeling.

The generated HLSM Verilog code can be synthesized in Xilinx to do the timing analysis for the given set of operations in the Data flow graph.

If the Dataflow graph is not valid, like if the operations modeled cannot be scheduled with the specified latency for LIST_R scheduling, user will be notified with suitable constraint violation message “ERROR: Operations cannot be scheduled with the specified latency…..” on the GME console.

Overall, it’s a new approach which demonstrates the proof of concept to apply the heuristic List scheduling algorithms for (ALU and logical) operations prior to RTL synthesis via modeling technique. Since the interpreter will take care of algorithm implementation housed in GME tool, Domain experts can take up tool to model the Dataflow graph to generate HLSM Verilog code to draw the conclusion on scheduling without the nuances of the algorithm implementation

HLSM Project Status (12/10/2014 - 11:58:55)			
Project File:	DataFlowGraph.xise	**Parser Errors:**	No Errors
Module Name:	HLSM	**Implementation State:**	Synthesized
Target Device:	xc7a100t-3csg324	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	67 Warnings (0 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	200	126800	0%
Number of Slice LUTs	213	63400	0%
Number of fully used LUT-FF pairs	70	343	20%
Number of bonded IOBs	228	210	108%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48E1s	6	240	2%

Fig 6: Hardware Utilization report from the generated HLSM Verilog code

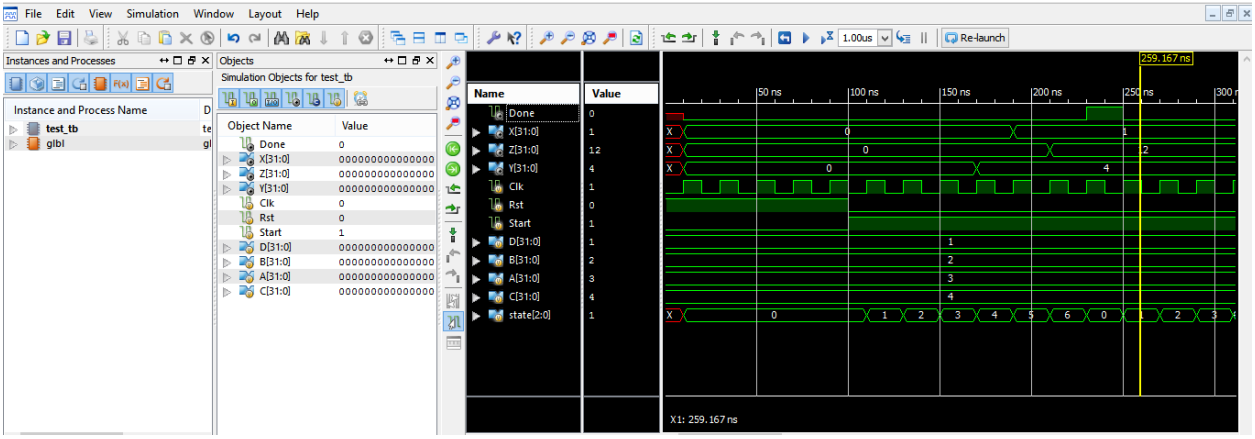

Fig 7: Timing analysis through xilinx from the genrated HLSM Verliog code

VI. RESULTS AND CONCLUSION

A unique approach to model the data flow graph with GME modeling environment has been proposed and developed. It gives the user to test the series of computation which can be scheduled by specifying latency and resource constraints, without the onerous task of implementing the algorithms.

With this modeling approach, user can able to determine the amount of concurrency that can be achieved through timing analysis from the generated HLSM Verilog code as shown in figure 7. Along with this, user will have opportunity to determine resource utilization since the HLSM code generated is synthesizable in Xilinx; the utilization report obtained is as shown in figure 6.

Overall it's a unique approach which can be used to determine the amount of concurrency of the resulting implementing thus impacts the performance. It also can be used to determine the amount of hardware resources required thus impacts the area. So these results in optimal design of circuits by determining amount of concurrent operations and hardware utilization for the operations scheduled without the cost of implementing List heuristic algorithms for the domain expert.

VII. REFERENCES

[1] Zaki, Mohamed, and Sofiène Tahar. "Syntax code analysis and generation for Verilog." Candian Conference on Electrical and Computer Engineering (CCECE 2003). 2003.

[2] Ledeczi, Akos, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. "The generic modeling environment." In Workshop on Intelligent Signal Processing, Budapest, Hungary, vol. 17. 2001.

[3] J. Sztipanovits, G. Karsai: Model-Integrated Computing, IEEE Computer, pp. 110-112, April, 1997.

[4] GME 2000 Users Manual, Vanderbilt University, 2000., available from http://www.isis.vanderbilt.edu/publications. Asp

[5] Y.-W. Hsieh and S. P. Levitan, Control/data-flow analysis for vhdl semantic extraction, in Proc. Of The 4th Asia-Pacific Conference on HardwarevDescription Languages, pp. 68--75, August 1997.

[6] R. K. Brayton et al. VIS: A system for verification and synthesis. In Computer Aided Verification, LNCS 1102, Springer Verlag, 1996, pp.428-432.