High Level Synthesis

A Project Report

PROJECT ABSTRACT

As the name of the project suggests, the project aims at performing high-level synthesis on an input expression written in a high-level format to generate the hardware level code in the VHDL language and some graphical representations for easier viewing. It goes through the processes of Scheduling, Binding and Allocation on a tree produced from the input infix expression while optimizing based on the constraints on the timing and resources to generate an internal representation. This is then worked upon further to generate a synthesizable VHDL component which can be used by the user to perform the operation as desired.

The following report delineates, to a certain extent, the support to be able to understand better the process as followed by the code so that it can be used more efficiently and can be built upon for further development.

Submitted by

Shashank OV
14D070021
IIT Bombay
shashankov@ee.iitb.ac.in

Yogesh Mahajan
14D070022
IIT Bombay
y.mahajan456@gmail.com

Avineil Jain
14D170002
IIT Bombay
avineil96@gmail.com

TABLE OF CONTENTS

PROJECT DESCRIPTION

The Purpose of the Project

Expected Learning

- The various algorithms that are used for the purpose of high-level synthesis
- A basic understanding of the entire flow involving scheduling, binding and allocation
- Good style Python coding sort of :P

Goals of Project

- GUI that takes the input set of expressions, the maximum number of ALU's that can be used
- The main thing: Scheduling Binding and Allocation
- Generate a user-friendly representative image of the scheduled operations
- Generate a synthesizable VHDL code

Testing and Verification

- Manually checked the outputs for all the smaller sub functions of the code, such as binding and scheduling
- Test benches for verifying the synthesizable VHDL code
- The generated VHDL code was synthesized once on Quartus.
- Plotted the expression tree and the generated data path.

Work Partitioning and Distribution

Shashank OV

VHDL Code Generation and VHDL OOP; GUI and Dot File Generation

Yogesh Mahajan

Scheduling; Binding; Allocation

Avineil Jain

VHDL OOP, High-Level Expression Evaluation, Binary Tree Generation

Solution Constraints

Even though our code works, there are some constraints that are to be kept in mind.

- Variable substitution is not allowed, i.e., a sub-expression which is repeated in all the expressions cannot be substituted by some other variable. E.g., if $(a + b)$ occurs a lot in the set of expressions, we can't replace $(a + b)$ by c since then, c would also come in the output and not $(a + b)$
- The output will not always be optimized. E.g., $(a + b + c + \ d)$ uses only one ALU (linear operation), although it could have parallelized this $((a + b) + (c + d))$. However, if the user takes care of the parallelization the input through use of brackets, then it yields a better optimized solution.
- Similar nodes are not always identified while combining trees, E.g., $(a + b) * c$ and $(a * c) + (b * c)$ will not be combined since they will form different sub nodes though mathematically they are identical.

Dependencies

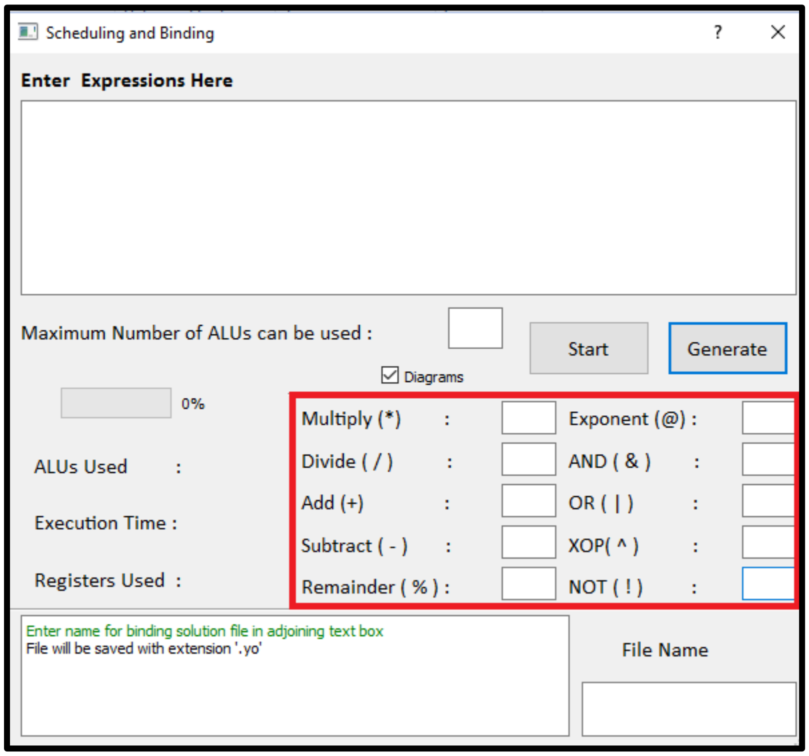
For running the executable: Graphviz

For compiling the entire project: Graphviz; Pydot; PyQt5; Pickle; Python 3.5 (obviously – though note the version)

FUNCTIONAL DESCRIPTION

User Interface

The user interface was built using a standard Python Library, PyQt5. Refer to the following image:



Enter name for binding solution file in adjoining text box
File will be saved with extension '.yo'

User Guide

Here is a short guide on how to use the interface.

- The expressions are to be entered in the big text box on top, each expression on a new line.
- You can also specify the number of maximum ALU's that can be used, if there are resource constraints.
- The file name box will save a local copy of the file that contains the binary output of the binding.
- There are 2 buttons, Start and Generate.
    - Start: Starts the whole process, and generates the output '.yo' file
    - Generate: Generates the synthesizable VHDL code.
- The remaining input boxes are for the time that each operation. Thus, flexibility is provided and different operations may result in different times.

The YO File

This is a python object file created using Pickle. It contains the schedule and bind output and shouldn't be a concern for a general user

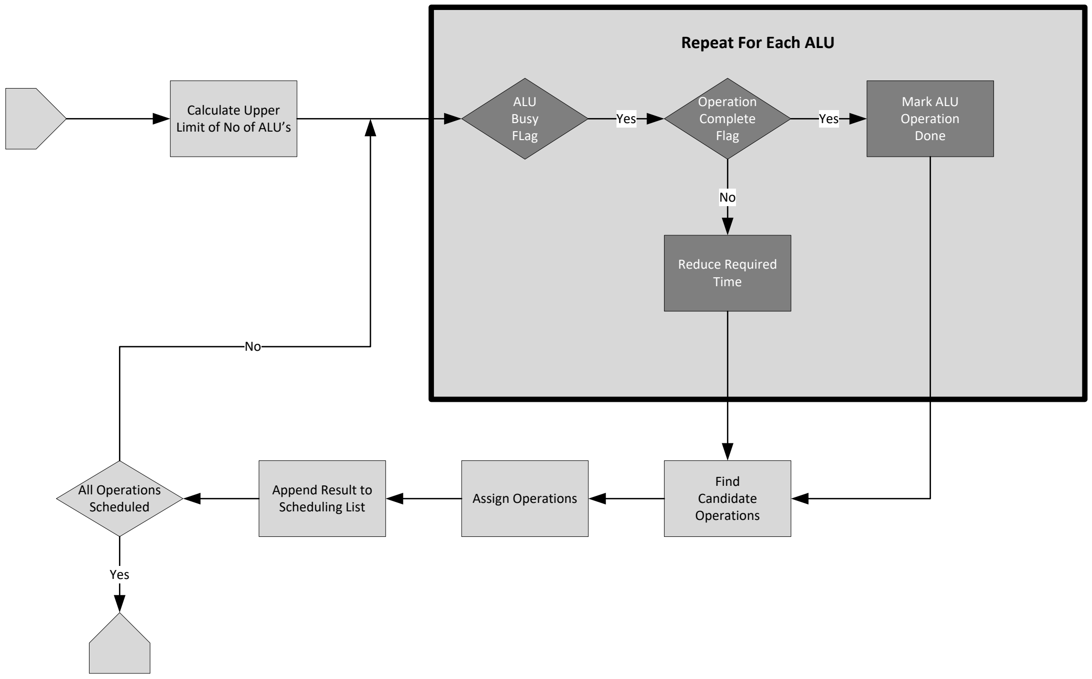
VHDL FILES

data_path.vhd	The data path
control_path.vhd	The control path
"".vhd	The main top level file
"".cmp	The component of top level
""_inst.vhd	Sample Instance
ALU.vhd and REG.vhd	To be filled by the user, more details in the files themselves

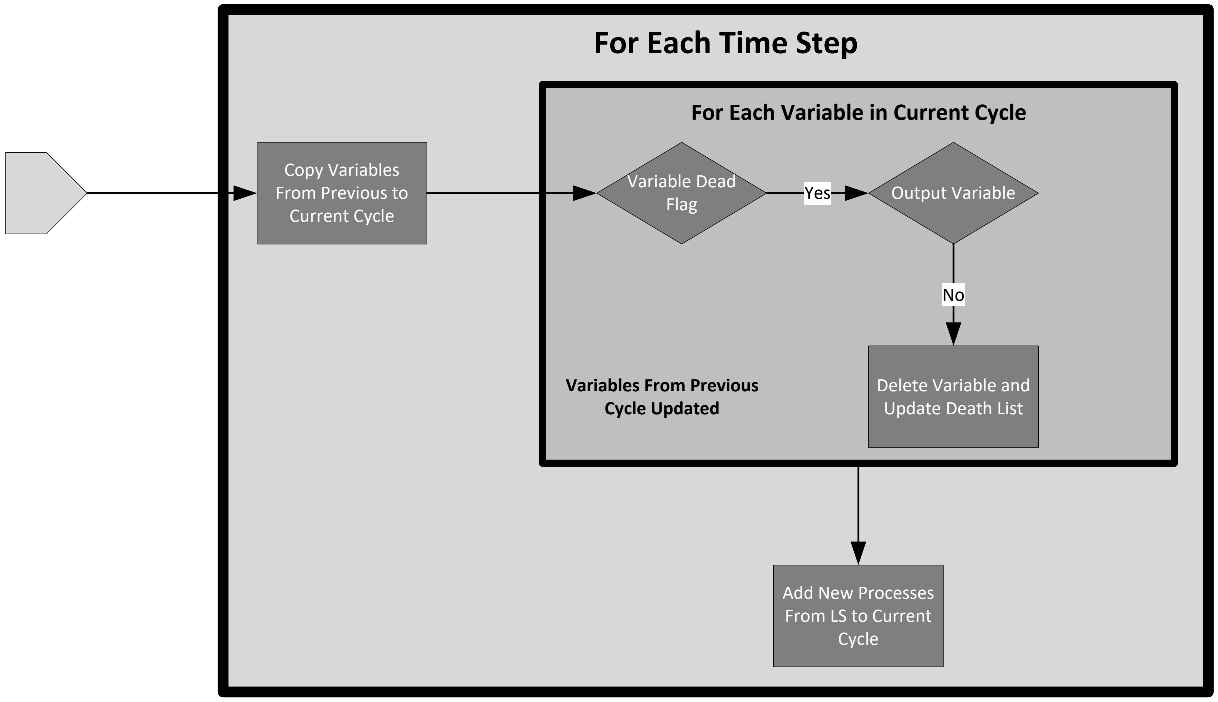
Scheduling and Binding

The code contains a pretty good explanation of what the code is about. However for better understanding, we have also made some flowcharts, which highlight the major aspect of how the code works.

First up is the Scheduling Algorithm. Here is the flowchart:

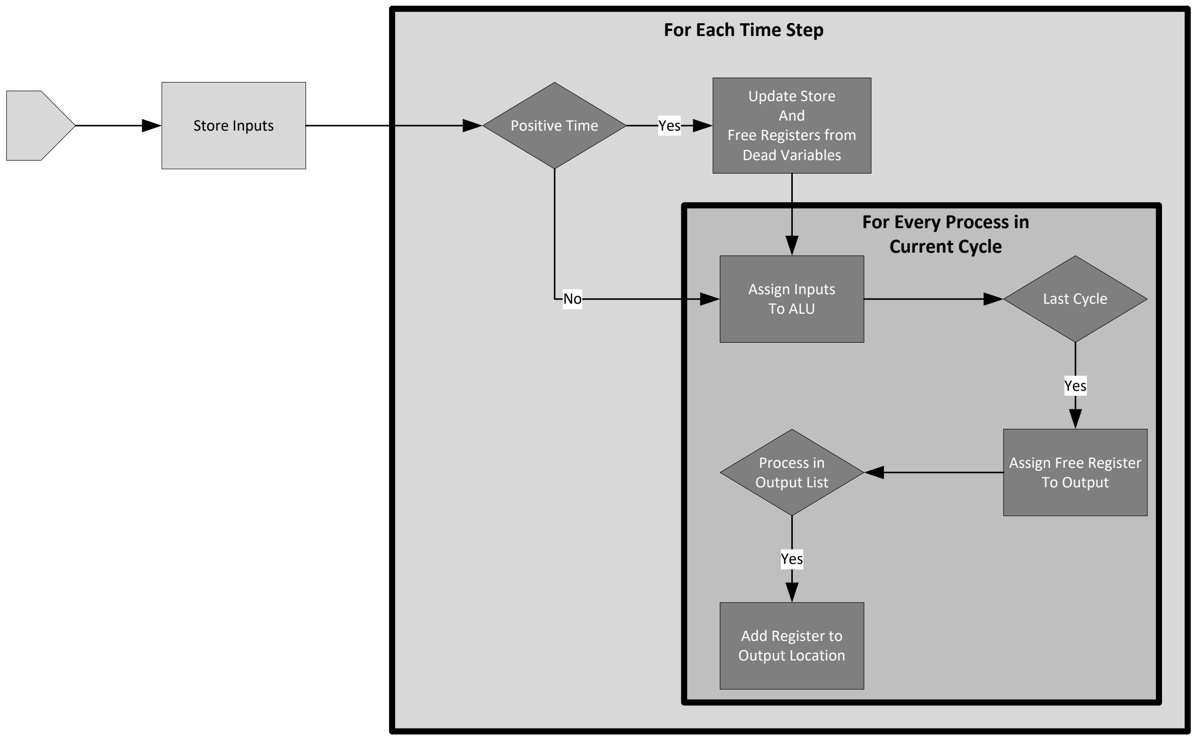


Next is calculation of the lifetime of the variables, which is important since each variable will not be used in all the cycles, so one can dynamically update the variables in each cycle.



For Each Time Step
For Each Variable in Current Cycle
Copy Variables From Previous to Current Cycle
Variable Dead Flag
Yes
Output Variable
No
Delete Variable and Update Death List
Variables From Previous Cycle Updated
Add New Processes From LS to Current Cycle

Next is the binding algorithm, which assigns the operations the required resources which in this case are ALU's


For Each Time Step
Store Inputs
Positive Time
Yes
Update Store And Free Registers from Dead Variables
For Every Process in Current Cycle
No
Assign Inputs To ALU
Last Cycle
Yes
Assign Free Register To Output
Process in Output List
Yes
Add Register to Output Location

VHDL Script Generation

The codes are self-explanatory :P

FUTURE WORK

While this project itself was sufficient for a basic consideration of High-Level Synthesis, there are certain improvements that can be made to make this even better.

- Improving identical node detection while merging detection (maybe some heuristics can be used)
- Improving parallelism (While still trying to use the limited amount of resources required)
- Generated images have placements determined by Graphviz, so sometimes node placements are not optimal.
- All the computations are assumed to be done by ALU, but it is possible that there may be discrete computation elements, such as separate adders and separate subtractors.

REFERENCES

[1] The Dot Language Introduction	http://www.graphviz.org/doc/info/lang.html
	https://en.wikipedia.org/wiki/DOT_%28graph_description_language%29
[2] Attributes of the DOT Objects	http://www.graphviz.org/doc/info/attrs.html
[3] Shapes in DOT Language	http://www.graphviz.org/doc/info/shapes.html
[4] Graphviz and PyDot	https://pythonhaven.wordpress.com/2009/12/09/generating_graphs_with_pydot/
	http://stackoverflow.com/questions/5316206/converting-dot-to-png-in-python
[5] Python 3.5 Documentation	https://docs.python.org/3/tutorial/
[6] PyQT reference	http://pyqt.sourceforge.net/Docs/PyQt5/
[7] Learning Python Reference	http://interactivepython.org/courselib/static/thinkcspy/index.html