EECS 507 Project Presentation

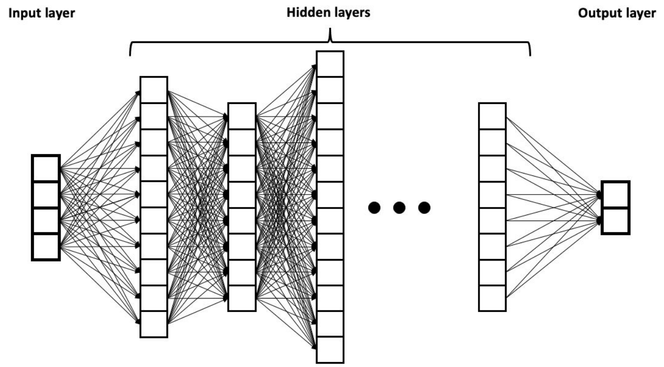
High-Level Synthesis From Tensorflow to FPGA

Group Member: Shenyi Wang

Content

Basic Motivation

Machine Learning Algorithms



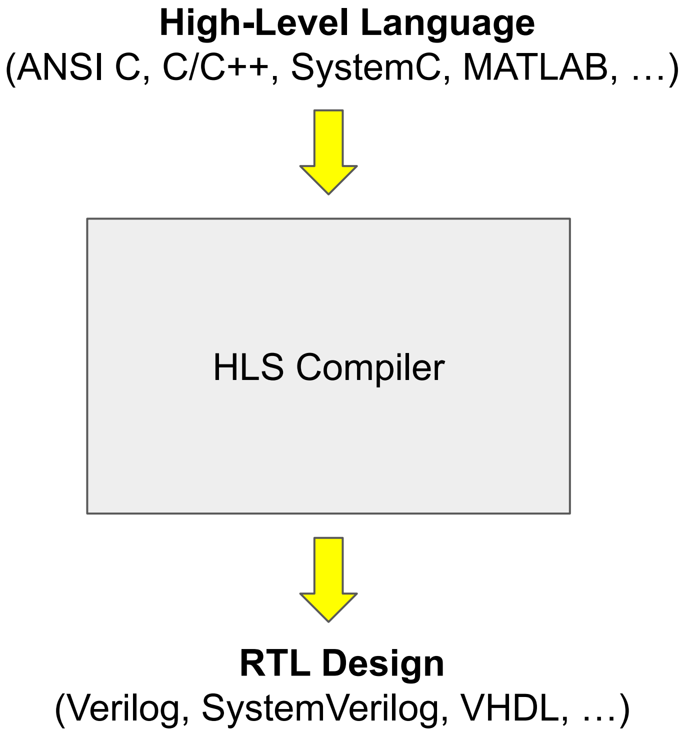
FPGA

HLS
?

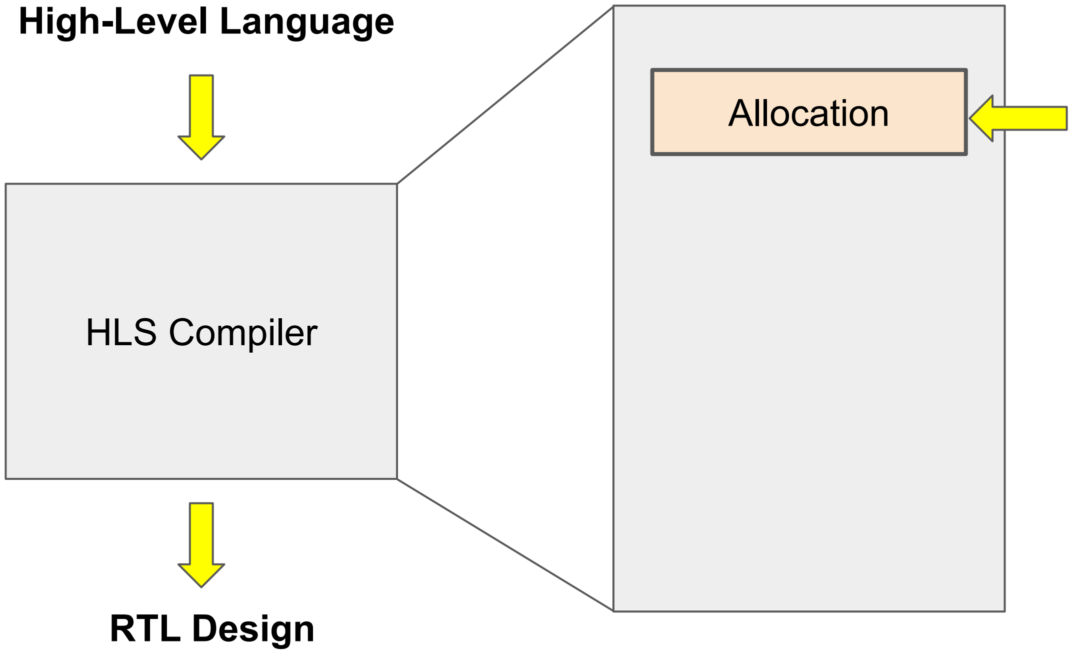
Python
(e.g. Tensorflow)

Verilog

Background: What is High-Level Synthesis (HLS)?

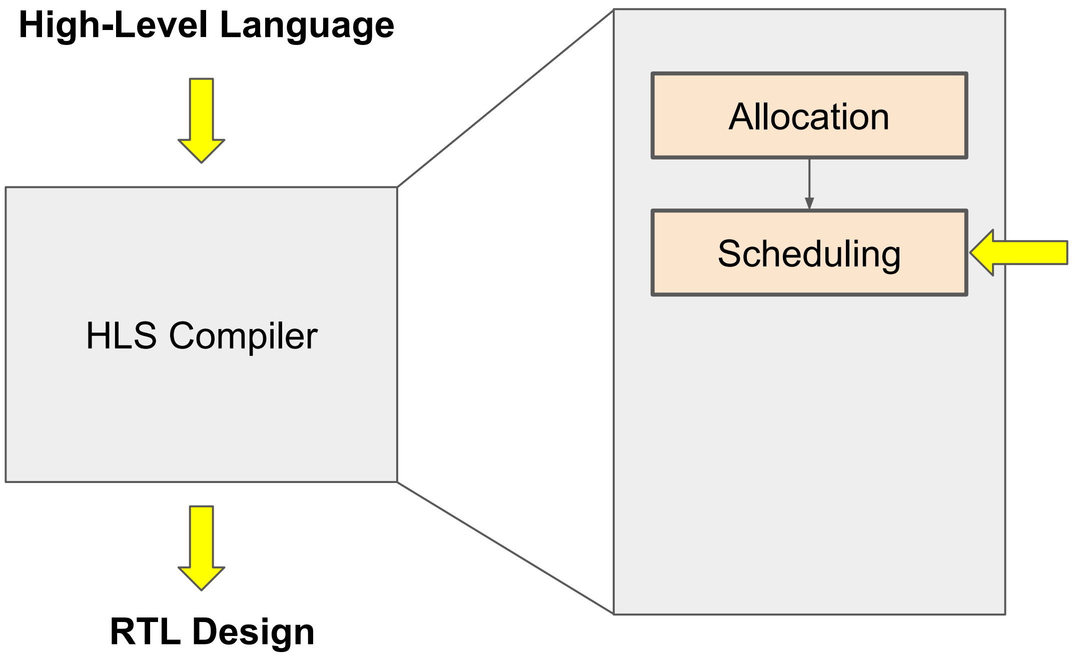

High-Level Language
(ANSI C, C/C++, SystemC, MATLAB, …)
HLS Compiler
RTL Design
(Verilog, SystemVerilog, VHDL, …)

Background: What is High-Level Synthesis (HLS)?



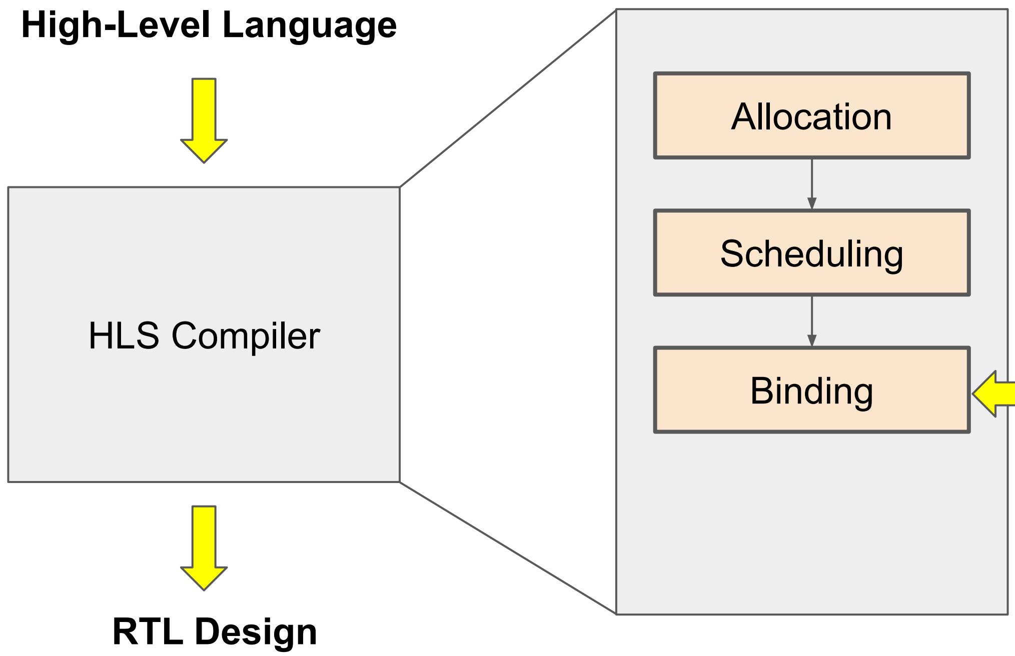
Define the constraints on the generated HW
- Constraints on the generated hardware.
- User-supplied constraints.

Background: What is High-Level Synthesis (HLS)?



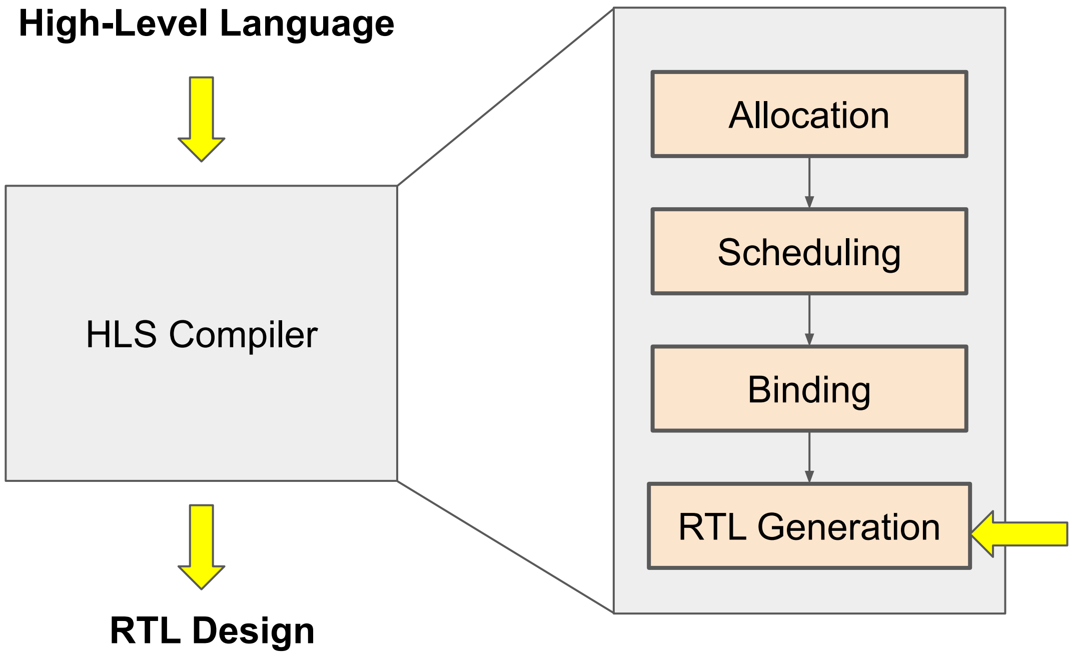
Assign operations into clock cycles such that the operations in each cycle does not exceed the target clock period, in order to meet the user constraint

Background: What is High-Level Synthesis (HLS)?



Associate (bind) each computation in the software with a specific unit in the hardware

Background: What is High-Level Synthesis (HLS)?



Generate a description of the circuit in a hardware description language

High-Level Motivation

High-Level Synthesis emerged as a promising productivity booster

- Companies such as Intel and Xilinx heavily investing in HLS tools (Vivado HLS, Intel HLS Compiler)
- Efforts are being put to enable software designers to program hardware

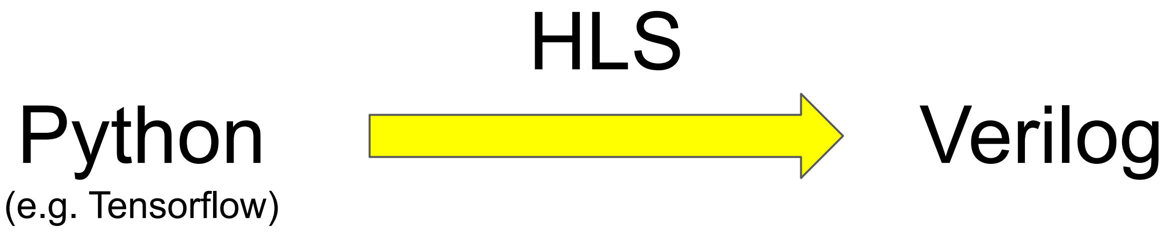
Deep Learning gained increased attention

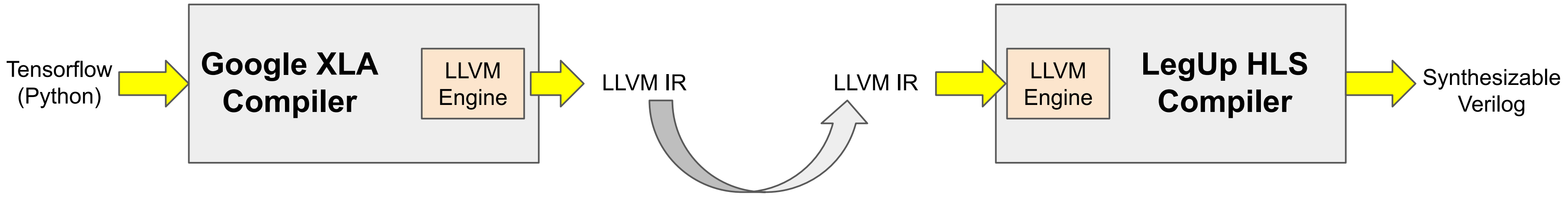
- Modern frameworks allow abstraction of implementation details
- High-level frameworks allow non-experts to experiment with state-of-the-art DNNs

Modern HLS tools are still far from High-Level machine learning frameworks

- Non-experts in machine learning faced with complex implementation details when using HLS tools
- To allow broader FPGA adoption in the ML community a higher level of abstraction is needed

Methodology





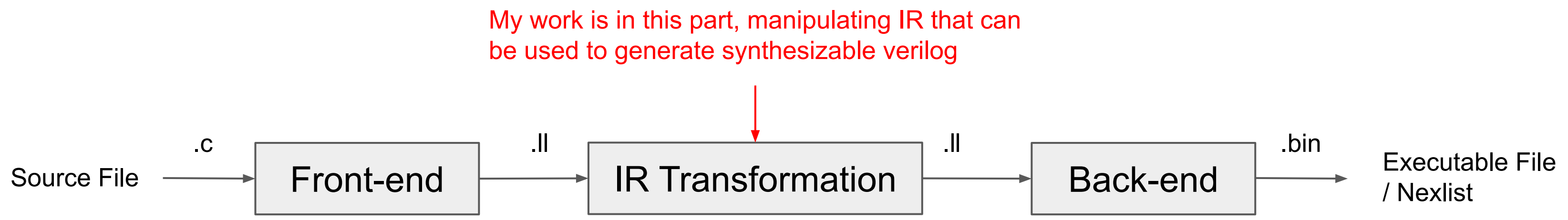
Bridge This!

XLA: a domain-specific compiler for linear algebra that optimizes Tensorflow computations

LegUp: Open source HLS tool being developed at the University of Toronto. It can synthesize most of C language to hardware

Background: LLVM IR and Common Compile Process

- **LLVM** is an open source compilation framework
- **IR** stands for **I**ntermediate **R**epresentation



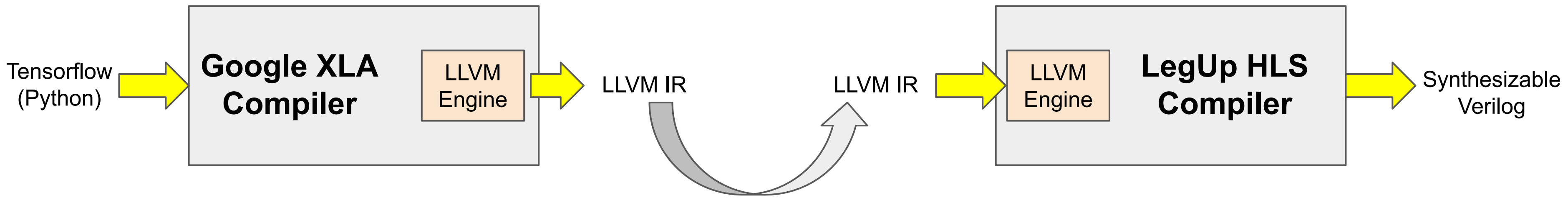
My work is in this part, manipulating IR that can be used to generate synthesizable verilog
Source File
.c
Front-end
.ll
IR Transformation
.ll
Back-end
.bin
Executable File / Nexlist

- Lexical analysis
- Syntax analysis
- Semantic analysis
- ...

- General Optimization
- Transformation
- …

- Hardware Specification Optimization
- Instruction Scheduling
- Code Generation
- …

Challenge



Bridge This!

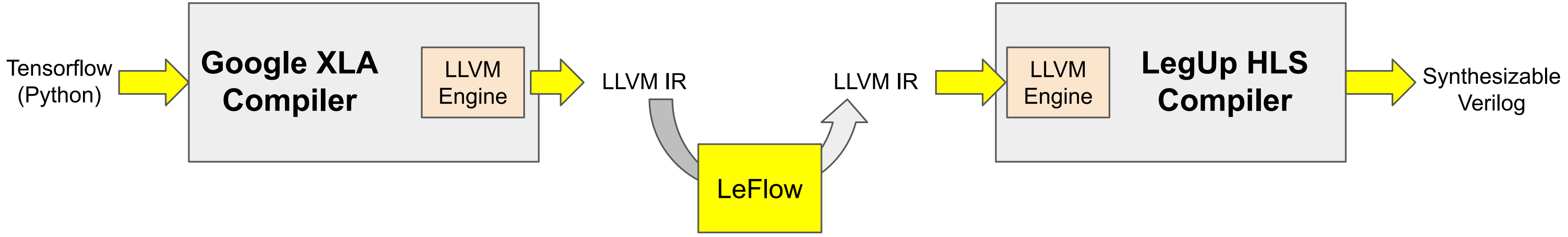
Technical Challenge:

- Transform software-like interface to an interface more suitable for HW
- Handle unsupported kernels
- Partition high dimensionality of inputs/weights
- ...

Engineering Challenge:

- LLVM version mismatch
- Python 2.7 is used in LegUp
- Tensorflow version is a disaster
- ...

Existing Work: LeFlow



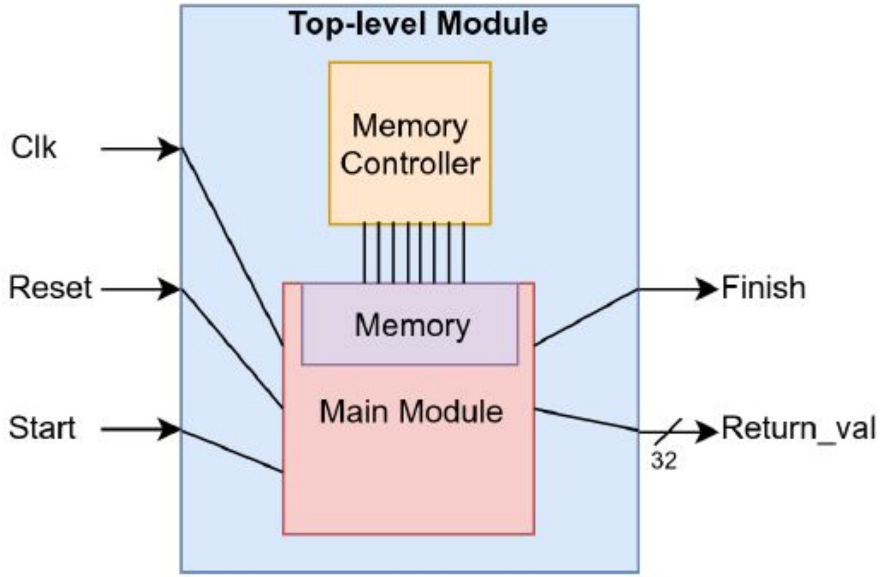
Key Insights in LeFlow:

1. Create stand-alone hardware unit
2. Remove pre-defined kernels that implement a particular Tensorflow operation
3. Memory partitioning to improve performance.

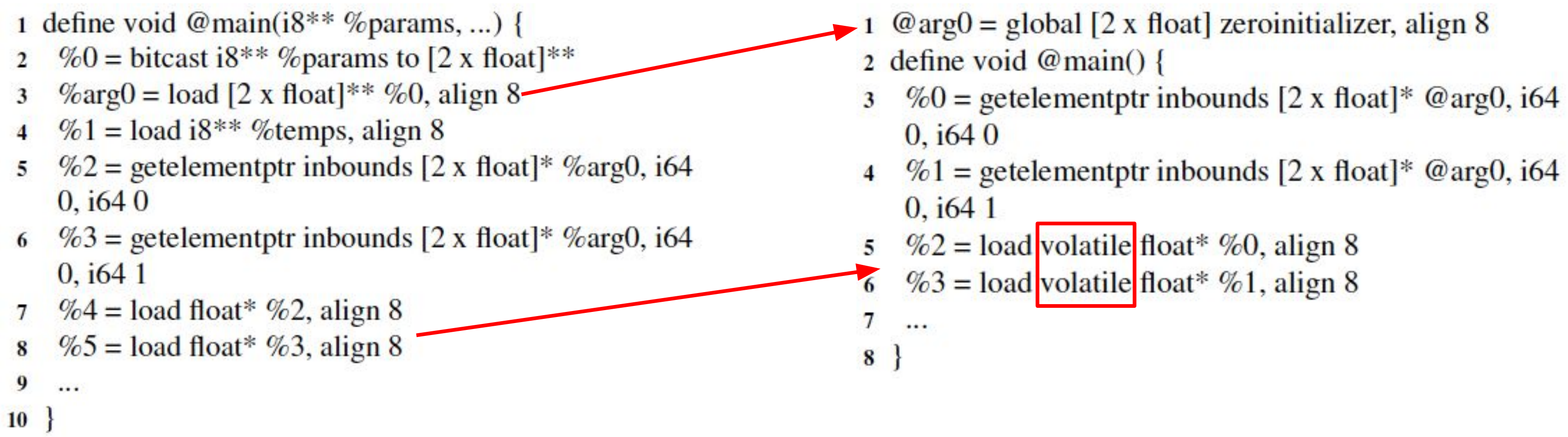
Generate Stand-Alone Hardware Unit

The LLVM IR generated by XLA has a static function's signature (function type) with three main components:

- **Params**: a pointer to an array of addresses containing all input values
- **Temps**: a pointer to an array of addresses for all temporary values (local values)
- **Retval**: a pointer to the temporary variable that is the output of the function.



Transformation Process



Original IR

```
define void @main(i8** %params, ...) {
  %0 = bitcast i8** %params to [2 x float]**
  %arg0 = load [2 x float]** %0, align 8
  %1 = load i8** %temps, align 8
  %2 = getelementptr inbounds [2 x float]* %arg0, i64 0, i64 0
  %3 = getelementptr inbounds [2 x float]* %arg0, i64 0, i64 1
  %4 = load float* %2, align 8
  %5 = load float* %3, align 8
  ...
}
```

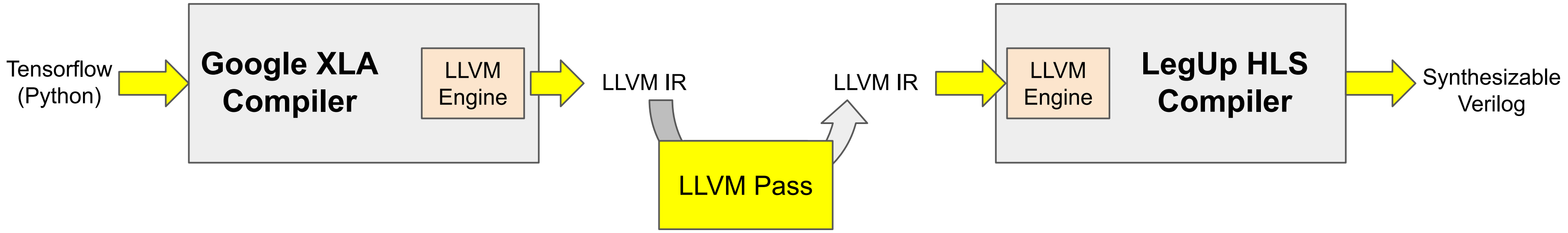
Transformed IR

```
@arg0 = global [2 x float] zeroinitializer, align 8
define void @main() {
  %0 = getelementptr inbounds [2 x float]* @arg0, i64 0, i64 0
  %1 = getelementptr inbounds [2 x float]* @arg0, i64 0, i64 1
  %2 = load volatile float* %0, align 8
  %3 = load volatile float* %1, align 8
  ...
}
```

1. Extracts the input and output registers and declares them as global variables

2. Reads from the global variables will be marked as *volatile*

My Work



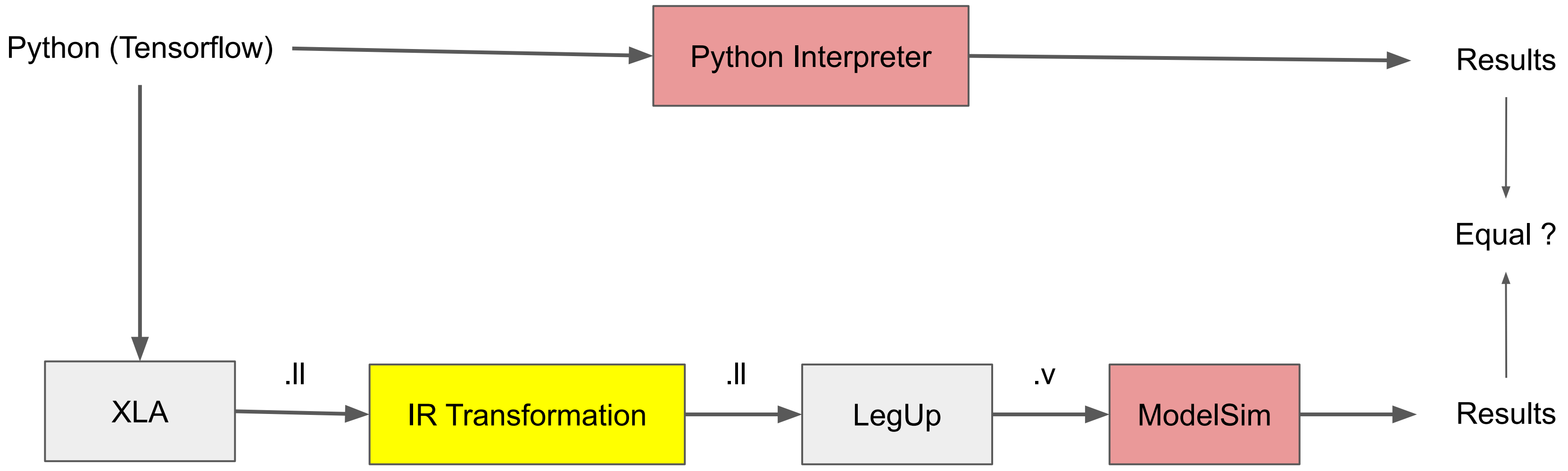
Some Limitations of LeFlow:

1. They use Python scripts to deal with LLVM IR in form of string editing. This is not scalable to large file and structure.
2. They don't implement vectorize operation transformation
3. They use Python 2.7 which is not officially supported anymore.

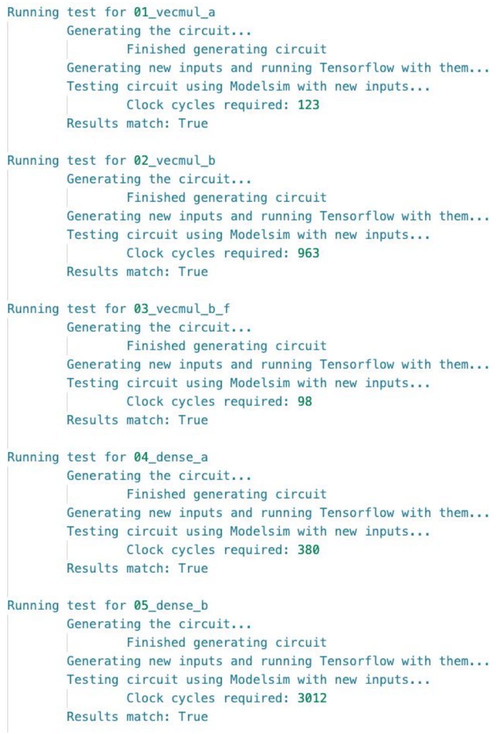
My Work:

1. Reimplement LeFlow in C++ with new LLVM framework and incorporate the entire process into one LLVM Pass.
2. Add fixed-point bit width support by using profiling

Evaluation: Correctness Checking



Results



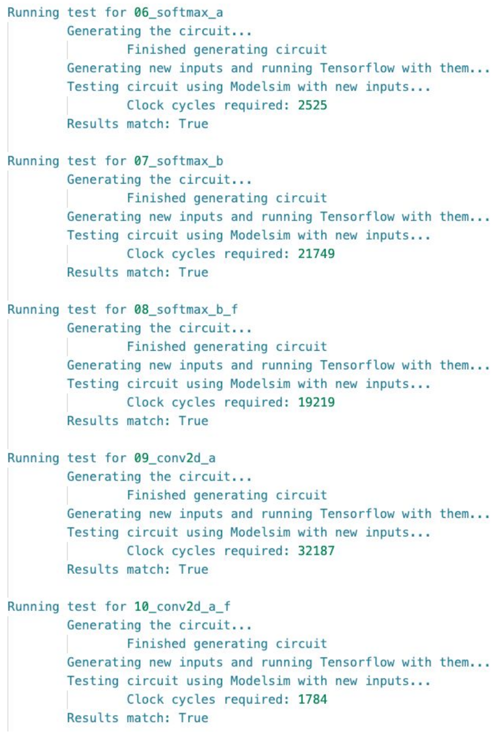
```
Running test for 01_vecmul_a
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 123
        Results match: True

Running test for 02_vecmul_b
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 963
        Results match: True

Running test for 03_vecmul_b_f
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 98
        Results match: True

Running test for 04_dense_a
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 380
        Results match: True

Running test for 05_dense_b
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 3012
        Results match: True
```



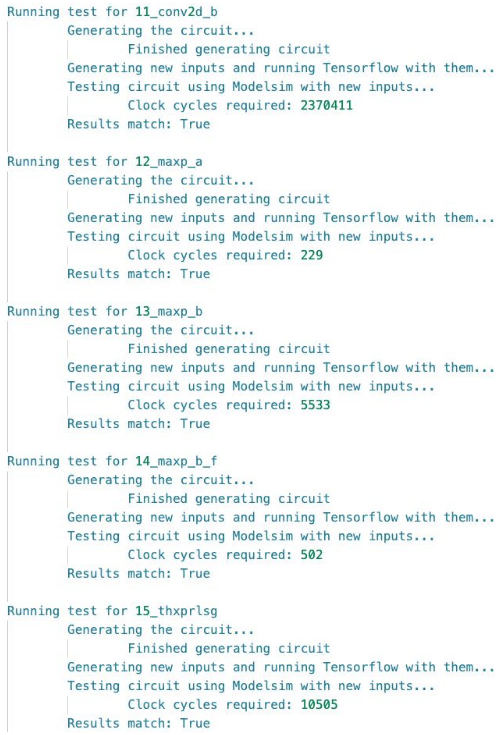
```
Running test for 06_softmax_a
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 2525
        Results match: True

Running test for 07_softmax_b
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 21749
        Results match: True

Running test for 08_softmax_b_f
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 19219
        Results match: True

Running test for 09_conv2d_a
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 32187
        Results match: True

Running test for 10_conv2d_a_f
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 1784
        Results match: True
```



```
Running test for 11_conv2d_b
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 2370411
        Results match: True

Running test for 12_maxp_a
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 229
        Results match: True

Running test for 13_maxp_b
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 5533
        Results match: True

Running test for 14_maxp_b_f
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 502
        Results match: True

Running test for 15_thxprlsg
        Generating the circuit...
                Finished generating circuit
        Generating new inputs and running Tensorflow with them...
        Testing circuit using Modelsim with new inputs...
                Clock cycles required: 10505
        Results match: True
```

Test was done in 1738.61 seconds

Future Work

- The kernel functions used in the XLA compiler is target to CPU and GPU and many optimization are designed for these architectures. It would be beneficial to design some kernel functions targeting FPGA.
- The memory partitioning algorithm currently used is straightforward. A machine learning specific automatic memory partitioning algorithm is needed.
- Backend of FPGA synthesis using LLVM need to support more operation.

Problem I met

- The latest released version of LegUp is 4.0 . It’s too old and with some dependence bugs in it. Fix them needs a lot of efforts.
- Both LegUp and LeFlow use Python 2.7 which is out of support officially. Hard to find available source to download dependencies.
    - Python 3.x is not valid after I put many efforts.
- LLVM version used in LegUp is too old. Find a matching version of XLA compiler is difficult.
- Tensorflow developing so fast so their source codes structure is very messy.

Reference

1. D. H. Noronha, B. Salehpour and S. J. E. Wilton, "LeFlow: Enabling Flexible FPGA High-Level Synthesis of Tensorflow Deep Neural Networks," FSP Workshop 2018; Fifth International Workshop on FPGAs for Software Programmers, 2018, pp. 1-8.
2. Canis, Andrew, et al. "LegUp: high-level synthesis for FPGA-based processor/accelerator systems." *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. 2011.

Q&A