SystemC model for 2 layer LSTM module

Viswanatha Kasyap Pasumarthy (vpasuma@ncsu.edu), Project 2 - ECE 720

Introduction

In this project, I model the behavior of a 2 layer LSTM module in SystemC, using TLM2.0 socket connections. The main aim of this project is to look at the performance benefit of modeling an LSTM module in SystemC versus using a native C program to run the same logic. The LSTM logic computes two layers with 32 inputs in time steps of 16. The project makes use of the spike CPU model designed by Dr. Davis with one TLM initiator socket bound to the address range 0x60000000-0x7FFFFFFF. The project is not designed to maximize the performance of the SystemC model. Rather, it is designed to measure the benefit given by a loosely timed SystemC model over a native C model. This report goes over the design of the modules present in the model, the various delays inserted, and finally looks into the performance metrics mentioned before.

Design Overview

The model consists of four main modules. The spike CPU - provided by Dr. Davis, the LSTM module, the memory module - this models an 8GB DDR4 SDRAM, and the crossbar-style AXI bus. We will now take a deeper look at each of the modules.

Spike CPU

The CPU is a spike chip modeled using the RISC-V RocketTile architecture. There is a TLM initiator socket bound to this CPU. The chip is designed in such a way that any memory transactions within the address range 0x60000000-0x7FFFFFFF are sent to the initiator socket and then on to subsequent components as connected. In our model, we have this initiator socket connected to a TLM simple target socket on the crossbar bus. This is done so that we have control over how the address range is distributed within the other modules connected to the bus.

The CPU is the module which dictates when the LSTM module has to start its computations. It also does error checking once the LSTM module is done with the computation of both layers. The CPU first instructs the LSTM and memory modules to read the ‘input.inc’ file to load the input coefficients into the LSTM module. Next, it instructs both the modules to read the ‘coef1.inc’ file to load the coefficients for computation of layer 1. Once the coefficients are loaded, the LSTM module computes layer 1. Similar steps are followed for computing layer 2, with one exception. Instead of loading input coefficients from ‘input.inc’, we load them from the output h matrix h1 that is computed in layer 1.

LSTM Module

The LSTM module contains two TLM sockets, one initiator, and one simple target socket which work independently. This is so that it can receive transactions from the CPU module while also sending transactions to the memory module. The initiator socket is connected to a simple target socket on the crossbar bus while the simple target socket of the LSTM is connected to an initiator socket on the bus.

This module does the computing for both LSTM layers. First, upon receiving the instruction from the CPU at address 0x7000c100 to load the input coefficients file, the LSTM module sends a transaction to the memory module at address 0x6000b100 to load the 'input.inc' file. There is a delay of 128 cycles inserted here since I have assumed the memory to be able to read at 8 bytes per cycle and there are a total of 512 input coefficients of type 'short', thus leading to a total of 1024 bytes. The input matrix is then assumed to be stored locally.

Once the LSTM module has finished loading the input coefficients, the CPU then instructs the module to load the 'coef1.inc' file to prepare for computation of layer 1. This is done by sending a transaction to address 0x70009000 which is then followed by the LSTM module initiating a transaction to memory at address 0x60008000. There is a delay of 528 cycles inserted here since the coefficient file is of size 4224 bytes. The module then computes the output of layer 1 using the provided lstm_layer function. Since there are a total of 32 timesteps to be processed, there are two delays inserted in this loop. After processing the 16th and the 32nd timestep, the module waits for 8481 cycles per the design specification. Once the computation is complete, the LSTM module writes the output matrices h1 and c1 to memory in address ranges 0x00000000-0x000003fe and 0x00000400-0x000007fe respectively. The h1 matrix is also stored locally for computation of layer 2.

After the CPU gets confirmation that layer 1 has finished computation, it instructs the LSTM module to load the 'coef2.inc' file for computation of layer 2. This transaction is sent to address 0x7000c000 which is then followed by a transaction from the module to memory at address 0x6000b000. As in the case of reading coef1.inc, there is a delay of 528 cycles inserted here since the file is of size 4224 bytes. Once the coefficient file has been processed, the module uses the output matrix h1 of layer 1 as input for layer 2 and starts computing. Similar to layer 1, there are two delays of 8481 cycles each inserted after processing of the 16th and 32nd timestep. As the execution is completed, the module then writes the output matrices h2 and c2 to memory in address ranges 0x00000c00-0x00000ffe and 0x00001000-0x000013fe. The address range for the h1 matrix is obtained from the ECE 564 project specification for Fall 2019 and the others are extrapolated from there. There is a gap of 400 addresses as a buffer between c1 and h1 matrices.

Overall it is assumed that the local memory on the LSTM module is of size 8kB which is more than sufficient to store the input coefficients, the coef.inc file, and the output h matrix per layer. After the layer has been computed and requisite values have been written to memory, these can be overwritten. This approach also means that the module can be scaled to compute as many layers as required

Memory Module

The memory modules models an 8GB DDR4 SDRAM with the following specifications.

Row Address Bits	15
Column Address Bits	10
Bank Address Bits	3
Data Bits	32
Cycle time	1 ns
CL	16
CCD	4
RCD	16
RP	16

The module has one TLM simple target socket that is connected to an initiator socket on the bus. This allows the memory module to receive transactions from both the CPU and the LSTM module.

The memory module reads the input and coefficient files and then passes the data along to the LSTM module. It is assumed that the module can read data at a rate of 8 bytes per cycle. The delays modeled in the memory module are dependent on the specification as mentioned above. There is also an additional one cycle delay added in for the memory module to process the incoming transaction.

Crossbar AXI bus

The crossbar bus is initiated with two simple target sockets and two simple initiator sockets. The connections are listed below

Module port	**Bus port**
CPU.initiator	bus.target[0]
LSTM.initiator	bus.target[1]
memory.target	bus.initiator[0]
LSTM.target	bus.initiator[1]

There is a delay of one cycle inserted into the bus module to model the processing delay for calculation destination port ID. There are no further changes made to the module since we are

only using two modules with target sockets. If the bus needs to be expanded to more modules, then we can make changes to the portID calculation to better distribute the available address range between the ports.

Delays in the model

There are numerous delays inserted at multiple places in the model. Following is a complete list of the delays and the reason for inserting the delay

Location	**Value (in cycles)**	**Justification**
In bus	1	1 cycle delay to account for processing of port ID and address on the bus
In memory	1	Assumed to be processing delay on memory side for any transaction
In memory	(data length/8+data length%8)	Accounted to be the transfer delay for 8-byte transactions over the bus for a write transaction. Since we are assuming a write buffer, we do not need to account for a write delay.
In memory	Varies	Accounted to be the read delay in memory. Varies based on bank and row accesses. Also accounts for the transfer delay for 8-byte transactions over the bus.
In LSTM	8481	Per design specification, each layer waits for this amount of cycles after computing 16 timesteps.
In LSTM	528	Accounts for read time for each coefficients file. Assumes an 8 byte per cycle read speed. Since each coefficient file is 2*((8*16*16)+(4*16)) = 4224 bytes, we have a delay of 4224/8 = 528 cycles
In LSTM	128	Accounts for read time for the input file. Assumes an 8 byte per cycle read speed. As the file is 2*(32*16) = 1024 bytes, we have a delay of 1024/8 = 128 cycles.

Performance comparison

If the LSTM layers are processed on the spike CPU itself, then the total simulation time is found to be 5,273,870ns at a clock frequency of 1GHz, with an execution time of 109.8 seconds. That means the total simulation takes 5,273,870 cycles at 48,029.42 cycles per second to finish execution. If instead, we do the LSTM computing on a separate module, then the simulation takes 137,683ns at a clock frequency of 1GHz with an execution time of 1.238 seconds. This is equal to a total of 137,683 cycles to finish execution at 109,012.67cycles per second.

Both simulations have a total error value of 0. One point of importance is that the makefile has been modified to compile with an optimization level of -O0 instead of -O2 to prevent the compiler from making the program too optimized which could result in certain instructions being skipped.

Conclusion

This project highlights the performance improvement in simulations that can be achieved by using a SystemC model to model the behavior of RTL. This also gives a better understanding of the timing of the system in a manner that is not possible with native C simulations. The way the project is designed, it can be expanded to more than 2 LSTM layers without much effort. It can also be modified to represent various memory architecture delays. Even when the SystemC model is not optimized for performance, we see a drop of 97.38% in simulation cycles along with a drop of 98.87% in execution time. Thus using SystemC not only speeds up the development process but also gives a better understanding of the timing and behavior of speculated RTL design.

Acknowledgments

I would like to sincerely thank Dr. Rhett Davis and Zhiping Wang for their unwavering support and constant guidance throughout the semester and with this project.

References

1. ECE 720 lectures and class notes by Dr. Rhett Davis
2. ECE 564, Fall 2019 project specification file.
3. TLM example on GitHub at engr-ece-720/sc_tlm_tut