**EE599 Project Phase III:**

**Final Project Report**

**PART A:**

1. **Project Details:**
2. **Project Topic:** Intermittent Computing
3. **Team Members:**

Bilkish Ara Naikodi Phalguni Bhangod Siddharth Gupta

[naikodi@usc.edu](mailto:naikodi@usc.edu) [bhangod@usc.edu](mailto:bhangod@usc.edu) [gupt232@usc.edu](mailto:gupt232@usc.edu)

1. **Problem Description:**

There are many modern applications where there is no continuous supply of power, and

hence stored energy is used to perform computations. Such applications are usually tolerant

towards errors and require fast and low power computations. Intermittent computing is used in

these cases to perform required tasks by taking relevant parts of the data and making

tolerable approximate calculations, in limited time and power constraints.

1. Mathematical Challenges:

The challenges are to determine the advantage in terms of time and power, over

existing Intermittent Computing technology by replacing the standard main memory

with a memristor and exploring new approximation techniques mainly for

multiplication. Multiplication is a basic operation executed maximum number of times in a

piece of code in several applications and is time consuming. Hence, we have

chosen to approximate multiplication operation.

1. Software Challenges:

Implementation of flow of Intermittent Computing architectural model in Python, using

access times/delays calculated and obtained from NVSim and Cacti, for memristor

based memories, and Xilinx ISE design suite for approximate multiplier modelled on FPGA.

1. Algorithmic Challenges:
2. Design of flow of Intermittent computing processor stages including the instruction

fetch, decode, execute, memory and write-back stages and checkpointing in Python.

1. Conversion of high-level source code in Python to Assembly level Language with

replacement of the conventional multiplication operation with operations that

perform approximate multiplication (replacement of MULT operand with SHIFT and

ADD operands).

1. Motivation:

Intermittent Computing applications must be performed with limited energy and time for

execution and are hence more accepting of errors in computations. So, we have chosen to

design an approximate multiplier, which gives an acceptable result, to replace the frequently

occurring multiplication operations, and replace the main memory with a memristor because of

huge access times gap (appx. 100 times faster) in terms of read and write access times.

1. Novelty:

We are replacing the conventional multiplier with an approximate multiplier design, and the

main memory with a memristor based ReRAM. A memristor has been chosen because of

large savings in time required for memory read and write access, as compared to

standard SRAM/DRAM. Because the frequency of multiplication operations is high, we

have chosen to approximate it in a time and power constrained environment.

1. **Project Timeline:**

|  |  |
| --- | --- |
| Stage I | Read the following papers to understand Intermittent Computing and its challenges:   1. Intermittent Computing - Challenges and Opportunities 2. The What’s Next Intermittent Computing Architecture |
| Stage II | We decided to improve upon the Paper 2, so we started reading the following papers on approximate multiplier designs:   1. SiMul: An algorithm Driven approximate multiplier Design for machine learning. 2. RoBA Multiplier: A Rounding-based approximate multiplier for high speed yet energy efficient Digital Signal processing.   We decided to use a memristor in place of main memory to determine its advantages and went through the following paper:   1. A Novel Design for memristor-based logic switch and crossbar circuits. |
| Stage III | 1. Designed a Python model of Memristor cell for understanding. 2. Learned to use Cacti to determine the read/write access times and latencies for cache and RAM. |
| Stage IV | 1. Learned to use NVSim because Cacti does not support non-volatile memory modelling, to calculate the read/write access times and latencies. 2. Simulated basic memristor based RERAM memory model on NVSim. 3. Developed Verilog code and the testbench code for an approximate multiplier in ModelSim and tested it for accuracy. |
| Stage V | Simulated several RERAM models on NVSim to determine the advantage of using a RERAM over the conventional main memory and compared the read/write access times using RERAM v/s conventional SRAM, by modelling it for both Cache and RAM. |
| Stage VI | Simulated the approximate multiplier code on an FPGA model using Xilinx ISE Design suite to obtain the power consumed and area occupied for the multiplier. |
| **Phase III**  Stage VII | To test the ‘shift and add’ approximate multiplier code for accuracy v/s power and latency for different shift value parameters. |
| Stage VIII | Test and understand the checkpointing mechanism implemented in What’s Next Intermittent Computing paper. |
| Stage IX | Implement the Intermittent computing processor stages including the instruction fetch, decode, execute, memory and write-back stages and checkpointing in Python, along with an assembly code to replace the MULT operand with SHIFT and ADD operands, with NVSim access time and latency parameters input in the Python code for RERAM as main memory. |
| Stage X | 1. Check the percentage of improvement for our approximate multiplier design over the What’s Next Intermittent Computing paper design by comparing the multiplication results accuracy. 2. Check the percentage of improvement for the replacement of RERAM over conventional main memory in terms of read/write access times and latencies, by running benchmark programs for each one of them. |