# Project Plan Madhumita

August 23, 2020

## 1 Integrating Melodica as a Coprocessor

### 1.1 Current Design & Way forward

- In the current design of Melodica, the core waits for the reponse after each instruction to dispatch the next one, which adds up to the overall latency of the system.
- Current thought process: All instructions are segregated into two types i.e., Blocking Instructions & Non blocking instructions to facilitate pipelining.
  - **Blocking Instruction** : The core waits for the response before dispatching the next instruction.
    Eg: RD_Q
  - **Non Blocking Instruction** : The core dispatches instructions consecutively (uses the idea of pipelining) and keeps updating the quire.
    Eg: FMA_P , FMS_P , FDA_P , FDS_P
- Subsequent changes are made to the top-level module of Melodica (*PositCore.bsv*)

### 1.2 Pipeline Implementation

- The current model has a 7 cycle latency bubble since the fma/fds modules keep the quire busy for all of it's cycles.
- The fma module is further split up into **multiplier** and **adder** modules to bring down the latency bubble

### 1.3 Instruction Format

- The custom instruction consists of 2 Floating point numbers and an opcode. Currently RST_Q, FMA_P , FMS_P , FDA_P , FDS_P , RD_Q are the available opcodes

## 2 Quire-based Stack Processing

Exploring the advantages of having a quire stack where the intermediate values are stored instead of converting them back to float.
This will add additional sub-opcodes for pop, push and clear
The expectation is that apart from the additional storage form the quire stack, the cost of implementing these operations should be minimal, reusing the quire adder and subtractor already existing.