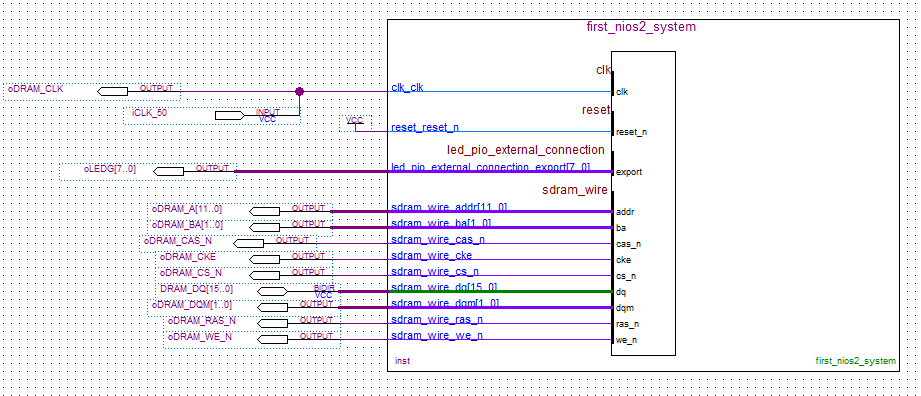
**Part I**

**Setting Up NIOS II**

Setting up the NIOS II system, was a relatively trivial task –following the tutorial correctly was all that was required.

We first performed this on the DE2-70 board, and then later on the DE0 board. As we were one of the first groups to do this – we found there was an issue with the compiling of Quartus projects on the lab computers – they have to be compiled on the H:/ drive rather than the mapped \\ic.ac.uk\homes\ drive. Once this issue was overcome we proceeded on to the next problem.



The system was then tested using count binary and the hello\_world example files set out in the NIOS II Eclipse software.

**Determinant Square Matrix in Software**

The determinant of a 3x3 Matrix performed in software was done on the NIOS II.

float determinant**(**float **\***matrix**,** int dimension**){**

int i**,** j**,** p**;**

float a**,** result**;**

float **\***m**;**

// Let us copy the matrix first

m **=** **(**float **\*)** malloc**(** **sizeof(**float**)\***dimension**\***dimension **);**

memcpy**(**m**,** matrix**,** **sizeof(**float**)\***dimension**\***dimension **);**

// First step: perform LU Decomposition using Doolittle's Method

// This algorithm will return, in the same matrix, a lower unit triangular matrix

// (i.e. diagonals one)

// and an upper trangular matrix

// https://vismor.com/documents/network\_analysis/matrix\_algorithms/S4.SS2.php

**for** **(**i **=** 0**;** i **<** dimension**;** i**++){**

**for** **(**j **=** 0**;** j **<** i**;** j**++){**

a **=** getAt**(**m**,** i**,** j**,** dimension**);**

**for** **(**p **=** 0**;** p **<** j**;** p**++){**

a **-=** getAt**(**m**,** i**,** p**,** dimension**)** **\*** getAt**(**m**,** p**,** j**,** dimension**);**

**}**

putAt**(**m**,** i**,** j**,** dimension**,** a**/**getAt**(**m**,** j**,** j**,** dimension**));**

**}**

**for** **(**j **=** i**;** j **<** dimension**;** j**++){**

a **=** getAt**(**m**,** i**,** j**,** dimension**);**

**for** **(**p **=** 0**;** p **<** i**;** p**++){**

a **-=** getAt**(**m**,** i**,** p**,** dimension**)** **\*** getAt**(**m**,** p**,** j**,** dimension**);**

**}**

putAt**(**m**,** i**,** j**,** dimension**,** a**);**

**}**

**}**

// Second step is to find the determinant.

// Because the lower triangle is a unit triangular matrix

// the determinant is simply a product of all the upper triangle diagonal

// which in this case is exactly the diagonal of m

result **=** 1**;**

**for** **(**i **=** 0**;** i **<** dimension**;** i**++)**

result **\*=** getAt**(**m**,** i**,** i**,** dimension**);**

free**(**m**);**

**return** result**;**

**}**

// Based on i and j, and a float pointer, get the value at row i column j

float getAt**(**float **\***m**,** int i**,** int j**,** int dimension**){**

**return** **\*(**m **+** i**\***dimension **+** j**);**

**}**

// Based on i and j, and a float pointer, put the value at row i column j

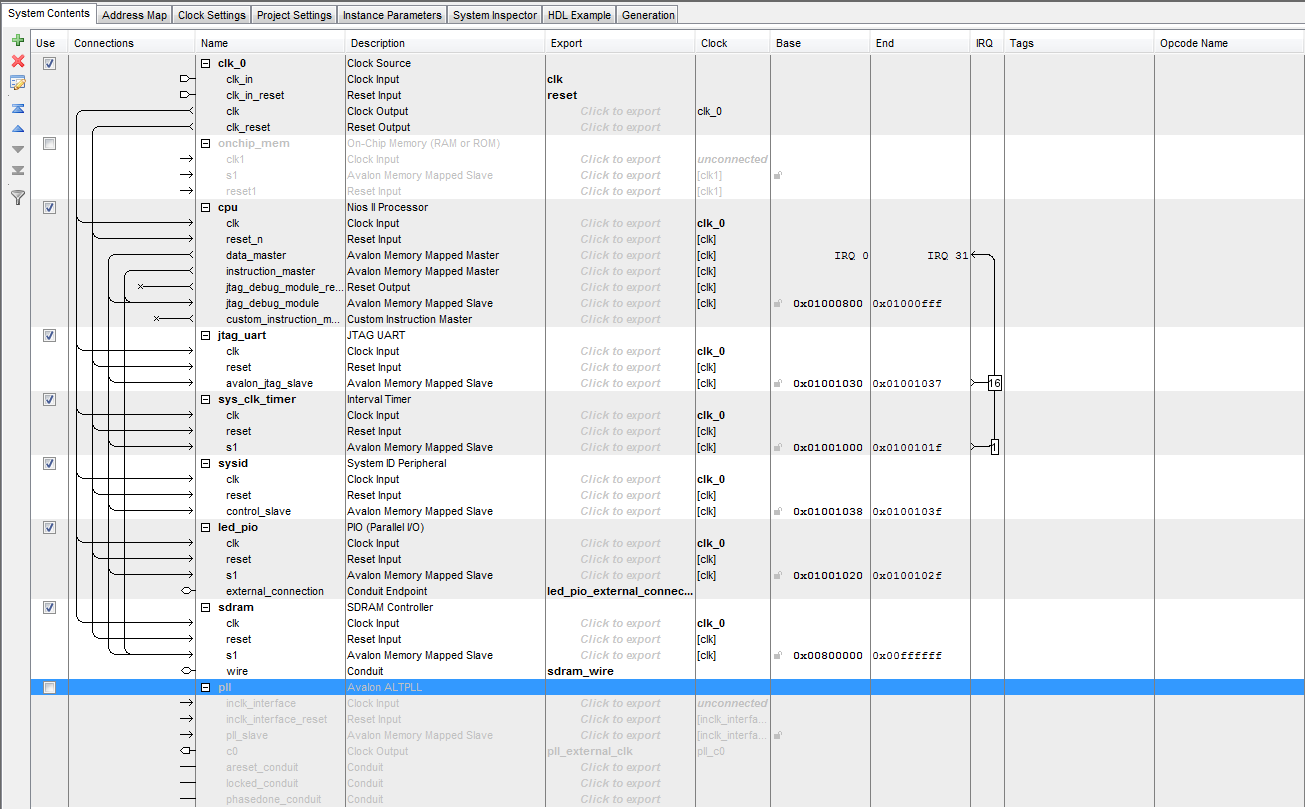
void putAt**(**float **\***m**,** int i**,** int j**,** int dimension**,** float value**){**

**\*(**m **+** i**\***dimension **+** j**)** **=** value**;**

**}**

Calculation of the 3X3 matrix took - 0.009 for 10 iterations.

**SDRAM**

****

3x3 with SDRAM 10 iterations = 0.016s

**Embedded Multipliers**

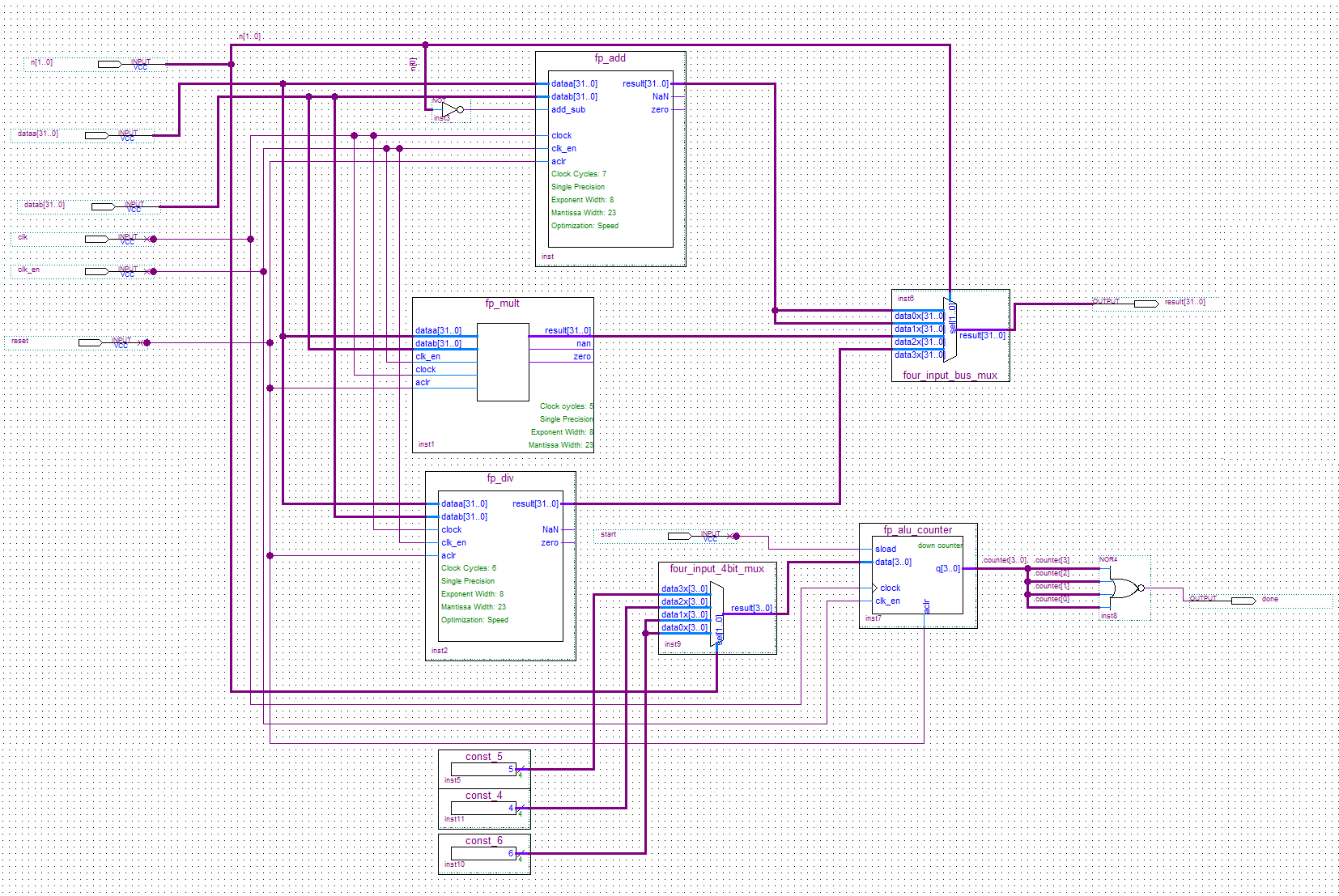
Times and Sizes

**LUT – based Multipliers**

Times and sizes –

Performance loss percentages – make some kind of graph here.

**Floating Point Hardware**

****