Hardware Acceleration for A Singularly Valuable Decomposition TYRION

Chae Jubb
ecj2122@columbia.edu
Columbia University

Ruchir Khaitan
rk2660@columbia.edu
Columbia University

Introduction

TYRION is a hardware accelerator designed to compute the Singular Value Decomposition (SVD). SVD is a matrix decomposition that has many uses in machine learning, natural language processing, computer vision and other exciting fields of research. We have implemented the 2-sided Jacobi algorithm to compute the SVD, and our accelerator takes as input a square matrix, and outputs an array containing the singular values, and two matrices containing the left and right eigenvectors. We used SystemC, a system level modeling framework, to implement the algorithm, and tested it on an Cyclone SocKit board.

SVD Overview

The singular value decomposition allows us to decompose a matrix A into:

$$A = U\Sigma V^*$$

where Σ is a diagonal matrix containing the nonnegative singular values sorted in descending order, and U and V_t are sqis a square matrix whose columns are the left and right singular vectors of A. The singular values are the square roots of the eigenvalues of both A^*A and AA^*

This decomposition is valuable because it allows one to create a low-rank approximation for the matrix A using only the first k singular values

$$\tilde{A_k} = U_k \Sigma_k {V_t}^*$$

SVD Algorithm

We use a version of the 2-sided Jacobi algorithm to compute the SVD for square $n \times n$ matrices. This algorithm generates a series of plane rotations, where plane rotations are just a orientation preserving orthogonal transformation that can be implemented with two $n \times n$ Jacobi rotation matrices, and systematically applies them with multiplaction to get rid of the off-diagonal elements of the matrix A.

The Jacobi rotation matrix $J(p, q, \phi)$ contains all '1's on the diagonal, except that it has $\cos\phi$ on j_{pp} and j_{qq}. Furthermore, it contains $\sin\phi$ at j_{pq} and $-\sin\phi$ at j_{qp}.

The algorithm then iteratively applies these matrices to the largest current off-diagonal values, until the remaining matrix is diagonal, and the product of the left rotation matrices is U and of the right is V_t. [1] used the exact same algorithm to implement SVD on FPGAs in 2005.

Applications

Unsurprisingly, there are a wide variety of applications that utilize SVD in order to efficiently process and extract useful information from data, and these applications come from fields as diverse as machine learning, natural language processing, computer vision, seismography, and many others. We chose two, latent semantic analysis and image compression, to try todemonstratewith our implementation, however, only image compression had small enough space requirements to fit on the board.

Latent semantic analysis is a method for classifying documents into topics, and performing queries on sets of documents. Essentially, it represents a corpus of documents as a matrix D where the rows are individual documents, and the columns are words, and each element at indices i, j indicates the number of times word j occurs in column i. Naturally, such a matrix would be very large, however, using the truncated SVD, we now have a much lower rank approximation, and can use cosine similarity to try to cluster documents into topics, and also search queries in the "topic space".

Image compression is an even more natural application for SVD, since the data to be compressed is already a matrix, and significantly, the matrix almost always has few high magnitude signular values. This means that a $n \times n$ bitmap can be stored in $(2n+1)k$ space, if we again compute the SVD, and then use only the K largest values to reconstruct the image. Note that a large portion of the storage saving comes because once we remove the remaining singular values, we can also remove the corresponding rows and columns in matrices U and V_t respectively. In most images, choosing k to be roughly one-fourth of n allows for a very close reconstruction of the image while still compressing by a factor of 2.

Hardware Implementation

We implemented our SVD accelerator using SystemC, and architected it to be modular, and have a simple interface with the main computer. To that end, we also provide a wrapper over the main module to simplify communications with the device driver.

SystemC

SystemC is a system-level modeling language that provides a much higher level of abstraction compared to hardware description languages like VHDL or SystemVerilog that may be applied for similar purposes. Using SystemC enabled us to design the system as a whole, while using more intuitive notions for time that enabled us to thing of the hardware as 'threads' of execution, which is a more intuitive way of thinking about algorithms.

SVD Module

The SVD module presents a pretty simple interface, which is listed below.

```
sc_in<bool> clk; //global clock
sc_in<bool> rst; //global reset

// DMA requests interface from memory to device
sc_out<unsigned> rd_index;
// array index (offset from base address)
sc_out<unsigned> rd_length;
// burst size
sc_out<bool>
rd_request; // transaction request
sc_in<bool>      rd_grant;
// transaction grant

// DMA requests interface from device to memory
sc_out<unsigned> wr_index;
// array index (offset from base address)
sc_out<unsigned> wr_length;
// burst size
sc_out<bool>
wr_request; // transaction request
sc_in<bool>      wr_grant;
// transaction grant

// input data read by load_input
get_initiator<SVD_CELL_TYPE> bufdin;

// output data written by store output
put_initiator<SVD_CELL_TYPE> bufdout;

//configuration signals
sc_in<unsigned> conf_size;
sc_in<bool>     conf_done;
// computation complete
sc_out<bool>    svd_done;
```

This module consists of four SystemC threads, that execute in parallel as much as possible. The reader thread handles all of the signals prefixed by 'rd' and is understandably responsible for requesting input and

accepting it. Similarly, the writer thread handles all the signals prefixed by 'wd' and is responsible for handling write transactions. The data is stored in two 32 bit buffers for input and ouput. The configuration thread is responsible for setting up the hardware on power on and then doing nothing, and the meat of the work is unsurprisingly done by a processing thread, which actually performs the two-sided Jacobi algorithm. The input/output threads communicate via four-way handshakes , an example of which is shown below (this is taken from the reader-thread initiating a read:

```
//request a read
rd_request.write(true);
//wait till writer grants a read
//the do-while ensures
//you wait at least once
do { wait(); }
while(!rd_grant.read());
//request to stop reading
rd_request.write(false);
//wait till the writer confirms
do{ wait(); }
while(rd_grant.read());
```

This pattern is frequently used throughout our device to communicate with drivers and other testbenches.

SVD Wrapper

In order to simplify driver development, we implemented a wrapper over the main SVD module that presents a more streamlined interface, that eventually read from a FIFO, and wrote into a single FIFO. The wrapper presented only four external signals, clock and rest for both input and output, and handled all of the communication with the SVD module. The final interface enabled us to logically input one matrix, and receive 3 as output.

Software Implementation

We did not implement very much functionality in software, since our hardware handled all the work in computing the SVD. Nevertheless, we solved some problems neccessarily in hardware

Conversion to Fixed Point

Since FPGAs don't always support floating point numbers, floating point operations are almost always slower, and we don't actually require floating point precision, we decided to covert our data to fixed point both before sending it to the FPGA, and after receiving it from the device. We chose to represent our data in 64 bit wide vectors, with 40 bits of fractional data. We chose this unorthodox split largely through trial and error to try to find the maximal error bounds neaded to achieve reasonably corect singular values. Note that this means our implementation won't work with values that are greater in magnitude than 2^{23} as the fixed point representation will overflow. We used a naive iterative subtraction algorithm to convert floating point numbers to their appropriate bit vector representations.

Drivers

We wrote two versions of our device drivers. The initial version was a very trivial driver that could only read or write one 32 bit value to the device at a time using the `ioread32()` and `iowrite32()` functions. To avoid the overhead of having so many `ioctl()` calls and presnet a generally more elegant solution we also wrote a memory mapped driver that utilized `copy_from_user()` and `copy_to_user()` and let us read and write entire matrices at a time, which is both more performant and better matches the semantic of our device.

Development Process

We mostly stuck to our original development schedule, and split up responsibilities according to prior background. Mr. Jubb was the lead on the hardware implementation and wrote a majority of the SystemC, while Mr. Khaitan wrote most of the application level code. We also did a ot of the debugging and testing steps together.

Toolchain

Since we chose to use SystemC, we had a very convoluted toolchain to get our code onto the device, and toolchain integration was one of the hardest parts of the project. First, we wrote our SystemC code, and compiled using the Cadence CtoS compiler, which generates Verilog as output. In order to actually synthesize this output, we had to be careful to only use the synthesizable subset of the SystemC framework and unfortunately, this renders a lot of the language unusable. Then, the generated Verilog has to be manually modified sometimes, and after that it was analyzed with Qsys, and then compiled and programmed onto the device with Quartus tools.

Testing

We were able to leverage many of SystemC and Cadence's existing tools to generate very robust tests and simulations to prove the correctness of our implementation without having to run on the device.

CARGO Virtual Linux

CARGO is a new simulator developed at the System Level Design Group at Columbia University that simulates both devices and drivers, each running on their own software thread. This lets developers write prototype device drivers as they are concurrently writing the devices themselves, and allowed us to complete the driver code much earlier than otherwise possible.

Testbench

We wrote a fully randomized testbench also using SystemC that would interface with our device and tested them with Cadence SystemC simulation tools. Our testbench verified the correctness of our implementation by also testing those random inputs against a pure software implementation that was known to be correct. While this simulation was very helpful, and allowed us to be sure that there were no obvious bugs in our implementation, id obviously couldn't catch all hardware specific bugs, nor could it simulate tests with very large matrix inputs, since simulation times were prohibitively slow.

Hardware Functionality Testing

When we finally ran our device on the actual hardware, we encountered many challenges actually communicating through the FIFOs. This meant that we could only establish functionality tests that showed that nonzero inputs were being received by the device, and nonzero outputs were being sent by the device, but we could not verify whether or not those outputs were actually correct at the time of this writng.

Implementation Difficulties

We encountered a number of strange bugs that ended up taking significant time to resolve, and greatly increased the overall difficulty of the implementation.

Porting from C

Porting the original C source code into SystemC capable of being synthesized was difficult as we had to eliminate all dynamic memory allocation, and tweak settings in order to get the algorithm to converge on a set of singular values. This was made more difficult by the fact that the author of the original C source code seemed to view memory as a boundless resource, so much so that we were able to almost halve overall memory consumption from the original. Finally, CtoS has a lot of functionality that allows developers to 'break out' loops and pipeline them, however managing that optimization while still generating correct outputs proved challenging.

Generated RTL

The generated RTL used an active high reset, which lead to many hours of frustration while debugging, as we assumed the existence of an active low reset. Also, the generated Verilog code is so large that standard Quartus runs out of memory while trying to compile it. Fortunately, we learned that it is pretty easy

to use 64-bit Quartus once you set an environment variable.

Bus Woes

The FIFOs used the Avalon MM protocal which had read latency 1, while the device used Avalon streaming protocol, and the SystemC abstraction of flex channels, both of which had read latency 0. Getting the two to communicate correctly proved to be very difficult.

Performance

The algorithm performs best on dense matrices (there are more efficient solutions for sparse matrices) and converges in quadratic time relative to the dimension of the array. Our implementation takes a significant amount of time, averaging 25 seconds on a 50 MHz clock to compute the SVD of a 64×64 matrix, which is inarguably slow. In terms of space, our algorithm uses $O(5n^2)$ space, and uses approximately 40 percent of the block RAM available on the board. This restricts our implementation to arrays no larger than $n = 100$ which renders it unusable for most machine learning or natural language processing applications. Image compression still works as the image can be tiled and compressed in independent small chunks.

Future Optimizations

We would like to actually implement parallel computation of the SVD, as currently we are doing in entirely serially. This would dramatically increase the memory requirements, and so its unclear how feasible this is.

Conclusions

While this implementation of the SVD isn't very practically useful, it was an iteresting exercise and great learning experience in hardware development and general testing and prototyping.

Perhaps the two greatest lessons we learned are that first, combining toolchains is devilishly hard. This one is a particularly devious problem because initially it seems so simple and yet, it was the source of a majority of the bugs encountered while working on this project. The second large lesson is to never trust any piece of software like a simulator that attempts to make claims about how hardware will behave. Overall, we enjoyed the hours and late nights spent on this project, and it was a fitting end to our undergraduate careers.

References

[1] W. Ma, M. Kaye, D. Luke, and R. Doraiswami. An fpga-based singular value decomposition processor. In *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, pages 1047–1050, May 2006.

```
sys = {
        devices = {
                --svd0 = {
                --type = "svd",
                --max_size_log = 15,
                --},
                svd0 = {
                        type = "svd-sync",
                        -- synthesizable and thus doesn't need configureable max size
                },
        },
}
// CPSC 445 - Problem Set 1
// Anton Petrov 22-09-11

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "gm_jacobi.h"

void gm_identify (double *matrix, int dimension) {
        int row, col;
        for (row = 0; row < dimension; row++) {
                for (col = 0; col < dimension; col++) {
                        // Elements on diagonal =1
                        if (col == row) {matrix [(row * dimension) + col] = 1.0;}
                        // ... all others 0
                        else {matrix [(row * dimension) + col] = 0.0;}
                }
        }
        return;
}

void gm_copyMatrix (double *a, double *b, int dimension) {
        int row, col;
        for (row = 0; row < dimension; row++) {
                for (col = 0; col < dimension; col++)
                        b [(row * dimension) + col] = a [(row * dimension) + col];
        }
        return;
}

void gm_multiply (double *left, double *right, double *result, int dimension) {
        int leftRow, rightRow; // row numbers of the left and right matrix, respectively
        int leftCol, rightCol; // same as above but for columns
        double tempResult = 0;
```

```
        for (leftRow = 0; leftRow < dimension; leftRow++) {
                for (rightCol = 0; rightCol < dimension; rightCol++) {
                        for (leftCol = rightRow = 0; leftCol < dimension; leftCol++, rig
                                tempResult += left[(leftRow * dimension) + leftCol] * ri
                        result [(leftRow * dimension) + rightCol] = tempResult;
                        tempResult = 0;
                }
        }
        return;
}

void gm_jacobi (double *a, int n, double *s, double *u, double *v) {
        // Arrays that contain the coordinates of the elements of the 2x2
        // sub matrix formed by the largest off-diagonal element. First entry
        // is the row number and second entry is the column number.
        int *a11, *a12, *a21, *a22;
        LargestElement le; // largest element of matrix a
        le.value = le.rowNum = le.colNum = -1;
        int i, j;

        // I could have statically allocated these but I would not have been
        // able to pass them as double pointers: this is a known problem in C
        a11=(int*)malloc(sizeof(int)*2); a12=(int*)malloc(sizeof(int)*2);
        a21=(int*)malloc(sizeof(int)*2); a22=(int*)malloc(sizeof(int)*2);

        gm_identify (u, n);
        gm_identify (v, n);

        if (n == 1) {   // 1x1 matrix is already in SVD
                s[0] = a[0];
                return;
        }

        le = gm_findLargestElement (a, n, &a11, &a12, &a21, &a22);

        int count = 0;
        while (fabs (le.value) > 0.00000000000000000001) {
                count++;
                gm_rotate (a, n, u, v, a11, a12, a21, a22);
                le = gm_findLargestElement (a, n, &a11, &a12, &a21, &a22);
        }

        gm_reorder (a, n, u, v);

        // Copy over the singular values in a to s
```

```
        for (i = 0; i < n; i++) {
                j = i;
                s[i] = a[(i * n) + j];
        }

        free(a11); free(a12); free(a21); free(a22);
        return;
}

void gm_rotate(double *a, int dimension, double *u, double *v, int *x11, int *x12, int
        double a11, a12, a21, a22; // elements of the sub-matrix
        double alpha, beta; // angles used in rotations; alpha is angle of left rotation
        double cosA, sinA, cosB, sinB;
        double *Ui, *Vi; // left and right rotation matrices, respectively
        double *tempResult;
        double X, Y; // temporary values used in calculating angles

        // Assign elements of sub-matrix to the their actual values from a
        a11 = a[x11[0] * dimension + x11[1]]; a12 = a[x12[0] * dimension + x12[1]];
        a21 = a[x21[0] * dimension + x21[1]]; a22 = a[x22[0] * dimension + x22[1]];

        // Calculate angles and sin and cos of those angles using the closed
        // formulas found.
        X = atan((a21 - a12) / (a11 + a22)); Y = atan((a21 + a12) / (a11 - a22));
        alpha = 0.5 * (X + Y);
        beta = 0.5 * (Y - X);
        cosA = cos(alpha); sinA = sin(alpha);
        cosB = cos(beta); sinB = sin(beta);

        // Create left rotation matrix, namely U_i which looks like this
        //
        //              | cosA  sinA |
        //              | -sinA cosA |
        //
        Ui = (double *) malloc(sizeof(double) * (dimension * dimension));
        gm_identify(Ui, dimension);
        Ui[x11[0] * dimension + x11[1]] = Ui[x22[0] * dimension + x22[1]] = cosA;
        Ui[x12[0] * dimension + x12[1]] = sinA;
        Ui[x21[0] * dimension + x21[1]] = (-1.0) * sinA;

        // Create the right rotation matrix, namely V_i which looks like this
        //
        //              | cosB  -sinB |
        //              | sinB   cosB |
        //
        Vi = (double *) malloc(sizeof(double) * (dimension * dimension));
```

```
        gm_identify (Vi, dimension);
        Vi [x11[0] * dimension + x11[1]] = Vi [x22[0] * dimension + x22[1]] = cosB;
        Vi [x12[0] * dimension + x12[1]] = (-1.0) * sinB;
        Vi [x21[0] * dimension + x21[1]] = sinB;

        // Rotate a
        // First on the left with Ui
        tempResult = (double *) malloc (sizeof (double) * (dimension * dimension));
        gm_multiply (Ui, a, tempResult, dimension);
        gm_copyMatrix (tempResult, a, dimension);
        // Then on the right with Vi
        gm_multiply (a, Vi, tempResult, dimension);
        gm_copyMatrix (tempResult, a, dimension);

        // Apply rotation matrix Ui to U
        gm_multiply (Ui, u, tempResult, dimension);
        gm_copyMatrix (tempResult, u, dimension);
        // Apply rotation matrix Vi to V
        gm_multiply (v, Vi, tempResult, dimension);
        gm_copyMatrix (tempResult, v, dimension);

        // Free rotation matrices
        free (Ui); free (Vi); free (tempResult);
        return;
}

LargestElement gm_findLargestElement (double *matrix, int dimension, int **a11, int **a1
        LargestElement *leArray;
        LargestElement leTemp;
        int i, j;

        leTemp.value = 0;
        leTemp.rowNum = leTemp.colNum = -1;

        leArray = (LargestElement *) malloc (sizeof (LargestElement) * dimension);

        // Populate leArray such that the entry at index i contains information
        // about the largest element of row i
        for (i = 0; i < dimension; i++) {
                for (j = 0; j < dimension; j++) {
                        // We are looking for the largest OFF-DIAGONAL element
                        if (j == i)
                                continue;
                        if (fabs (matrix [(i * dimension) + j]) > fabs (leTemp.value)) {
                                leTemp.value = matrix[(i * dimension) + j];
                                leTemp.rowNum = i;
```

```
                                leTemp.colNum = j;
                        }
                }
                leArray[i].value = leTemp.value;
                leArray[i].rowNum = leTemp.rowNum;
                leArray[i].colNum = leTemp.colNum;
                leTemp.value = 0;
        }
        leTemp.value = 0;
        leTemp.rowNum = leTemp.colNum = -1;
        // Iterate over leArray and find the largest element in the matrix as a
        // whole
        for (i = 0; i < dimension; i++) {
                if (fabs (leArray [i].value) > fabs (leTemp.value)) {
                        leTemp.value = leArray [i].value;
                        leTemp.rowNum = leArray [i].rowNum;
                        leTemp.colNum = leArray [i].colNum;
                }
        }

        free (leArray);

        // Determining coordinates of the 2x2 sub matrix formed by this largest
        // element
        if (leTemp.rowNum < leTemp.colNum) { // largest element is above diagonal
                (*a11) [0] = (*a11) [1] = (*a12) [0] = (*a21) [1] = leTemp.rowNum;
                (*a21) [0] = (*a22) [0] = (*a22) [1] = (*a12) [1] = leTemp.colNum;
        }
        else { // below diagonal
                (*a22) [0] = (*a22) [1] = (*a21) [0] = (*a12) [1] = leTemp.rowNum;
                (*a11) [0] = (*a11) [1] = (*a21) [1] = (*a12) [0] = leTemp.colNum;
        }

        return leTemp;
}

void gm_transpose (double *matrix, int dimension) {
        double temp;
        int row, col;
        for (row = 0; row < dimension; row++) {
                for (col = (row + 1); col < dimension; col++) {
                        if (col == row) {continue;} // skip diagonal elements
                        temp = matrix [(row * dimension) + col];
                        matrix [(row * dimension) + col] = matrix [(col * dimension) + r
                        matrix [(col * dimension) + row] = temp;
                }
```

```
        }
        return;
}

void gm_reorder (double *a, int dimension, double *u, double *v) {
        int row, col, x, largestElementRow;
        double temp; // stores the largest singular value
        double *p; // permutation matrix
        double *tempMatrix;

        p = (double *) malloc (sizeof (double) * (dimension * dimension));
        tempMatrix = (double *) malloc (sizeof (double) * (dimension * dimension));

        for (x = 0; x < dimension; x++) {
                temp = 0.0;
                gm_identify (p, dimension);
                for (row = x; row < dimension; row++) {
                        col = row;
                        if (fabs (a [row * dimension + col]) > fabs (temp)) {
                                temp = a [row * dimension + col];
                                largestElementRow = row;
                        }
                }
                gm_swapRows (p, x, largestElementRow, dimension);
                // Reorder a
                gm_multiply (p, a, tempMatrix, dimension); gm_copyMatrix (tempMatrix, a,
                gm_multiply (a, p, tempMatrix, dimension); gm_copyMatrix (tempMatrix, a,
                // Reorder u
                gm_multiply (p, u, tempMatrix, dimension); gm_copyMatrix (tempMatrix, u,
                // Reorder v
                gm_multiply (v, p, tempMatrix, dimension); gm_copyMatrix (tempMatrix, v,
        }
        gm_transpose (u, dimension);
        gm_transpose (v, dimension);
        gm_identify (p, dimension);
        for (row = 0; row < dimension; row++) {
                col = row;
                if (a [row * dimension + col] < 0)
                        p [row * dimension + col] = -1.0;
        }
        gm_multiply (p, a, tempMatrix, dimension);
        gm_copyMatrix (tempMatrix, a, dimension);
        gm_multiply (u, p, tempMatrix, dimension);
        gm_copyMatrix (tempMatrix, u, dimension);
        free (p); free (tempMatrix);
        return;
```

```
}

// Swap row a with row b of matrix m
void gm_swapRows (double *m, int a, int b, int dimension) {
        double temp;
        int col;
        for (col = 0; col < dimension; col++) {
                temp = m [a * dimension + col];
                m [a * dimension + col] = m [b * dimension + col];
                m [b * dimension + col] = temp;
        }
        return;
}
// Anton Petrov
// jacobi.h - contains function declarations

typedef struct {
        double value;
        int rowNum;
        int colNum;
} LargestElement;

// Self-explanatory
void gm_multiply (double *left, double *right, double *result, int dimension);

// Function that performs a number of jacobi rotations on the input matrix a
// and returns the SVD of that matrix in the u, s and v matrices
void gm_jacobi (double *a, int n, double *s, double *u, double *v);

// Find largest off-diagonal element in input matrix. Uses the LargestElement
// struct to store the row, col and value of the element.
LargestElement gm_findLargestElement (double *matrix, int dimension, int **a11, int **a1

// Self-explanatory
void gm_copyMatrix (double *a, double *b, int dimension);

// Loop over matrix and transform it into the identity matrix
void gm_identify (double *matrix, int dimension);

// Perform a single rotation
void gm_rotate (double *a, int dimension, double *u, double *v, int *x11, int *x12, int

// Calculate the transpose of the input matrix in place
void gm_transpose (double *matrix, int dimension);

// Apply a series of permutation matrices to a so that the singular values
```

```
// are ordered in decreasing order. Apply those same permutations in the
// correct order to u and v.
void gm_reorder (double *a, int dimension, double *u, double *v);

// Swap two rows of a matric in place.
void gm_swapRows (double *m, int a, int b, int dimension);
obj-m := svd.o

ccflags-y := -I$(QSIM)/include
CARGO ?= /usr/local/share/cargo
QSIM ?= ${CARGO}/qsim
KSRC ?= $(QSIM)/linux
CFLAGS := -O0 -g3
CFLAGS += -m32 -static
CFLAGS += -I../.. -I../../svd/src -I../
LIBS = -lm -lrt

all: tarball

check:
ifeq ($(KSRC),)
        $(error 'Path to kernel in env variable KSRC not found. Exiting')
endif
.PHONY: check

svd-mmap: svd-mmap.c gm_jacobi.c
        $(CC) $(CFLAGS) -o $@ $^ $(LIBS)

tarball: svd.tar
.PHONY: tarball

svd.tar: svd-mmap svd.ko runme.sh
        tar cf $@ $^

driver: svd.ko

.PHONY: driver

svd.ko: check
        make -C $(KSRC) M=`pwd`

clean help: check
        $(MAKE) -C $(KSRC) M=`pwd` $@
distclean: clean
        $(RM) svd.tar svd-mmap
```

```
.PHONY: all clean help
#!/sbin/ash

/sbin/mark_app

/sbin/insmod svd.ko
/sbin/mdev -s
./svd-mmap &
wait
/sbin/rmmod svd

/sbin/qsim_exit
#include <linux/platform_device.h>
#include <linux/dma-mapping.h>
#include <linux/scatterlist.h>
#include <linux/completion.h>
#include <linux/interrupt.h>
#include <linux/pagemap.h>
#include <linux/device.h>
#include <linux/module.h>
#include <linux/cdev.h>
#include <linux/log2.h>
#include <linux/slab.h>
#include <linux/dad.h>
#include <linux/fs.h>
#include <linux/mm.h>

#include <asm/uaccess.h>

#include "svd.h"

/* device struct */
struct svd_device {
        struct cdev cdev;
        struct dad_device *dad_dev; /* parent device */
        struct device *dev;
        struct module *module;
        struct mutex lock;
        struct completion completion;
        void __iomem *iomem; /* mmapped regs */
        size_t max_size;
        int number;
};

struct svd_file {
        struct svd_device *dev;
```

```
        void *vbuf; /* virtual address of contiguous buffer */
        dma_addr_t dma_handle; /* physical address of that buffer */
        size_t size;
};

static const struct dad_device_id svd_ids[] = {
        { SVD_SYNC_DEV_ID },
        { },
};

static struct class *svd_class;
static dev_t svd_devno;
static dev_t svd_n_devices;

static irqreturn_t svd_irq(int irq, void *dev) {
        struct svd_device *svd = dev;
        u32 cmd_reg;

        cmd_reg = ioread32(svd->iomem + SVD_REG_CMD);
        cmd_reg >>= SVD_CMD_IRQ_SHIFT;
        cmd_reg &= SVD_CMD_IRQ_MASK;

        if (cmd_reg == SVD_CMD_IRQ_DONE) {
                complete_all(&svd->completion);
                iowrite32(0, svd->iomem + SVD_REG_CMD);
                return IRQ_HANDLED;
        }
        return IRQ_NONE;
}

static bool svd_access_ok(struct svd_device *svd,
                const struct svd_access *access) {
        unsigned max_sz = ioread32(svd->iomem + SVD_REG_MAX_SIZE);
        if (access->size > max_sz ||
                        access->size <= 0)
                return false;

        return true;
}

static int svd_transfer(struct svd_device *svd, struct svd_file *file, const
                struct svd_access *access) {
        /* compute the input and output burst */
        int wait;

        unsigned sz = access->size; size_t in_buf_size = SVD_INPUT_SIZE_BYTE(sz);
```

```
        INIT_COMPLETION(svd->completion);

        if (!svd_access_ok(svd, access))
                return -EINVAL;

        iowrite32(file->dma_handle, svd->iomem + SVD_REG_SRC);
        iowrite32(file->dma_handle + in_buf_size, svd->iomem + SVD_REG_DST);
        iowrite32(access->size, svd->iomem + SVD_REG_SIZE);
        iowrite32(0x1, svd->iomem + SVD_REG_CMD);

        wait = wait_for_completion_interruptible(&svd->completion);
        if (wait < 0)
                return -EINTR;
        return 0;
}

static int svd_access_ioctl( struct svd_device *svd, struct svd_file *file,
                void __user *arg) {
        struct svd_access access;

        if (copy_from_user(&access, arg, sizeof(access)))
                return -EFAULT;

        if (!svd_access_ok(svd, &access))
                return -EINVAL;

        if(mutex_lock_interruptible(&svd->lock))
                return -EINTR;

        svd_transfer(svd, file, &access);
        mutex_unlock(&svd->lock);

        return 0;
}

static long svd_do_ioctl(
                struct file *file,
                unsigned int cm,
                void __user *arg) {
        struct svd_file *priv = file->private_data;
        struct svd_device *svd = priv->dev;

        switch (cm) {
                case SVD_IOC_ACCESS:
                        return svd_access_ioctl(svd, priv, arg);
```

```
                default:
                        return -ENOTTY;
        }
}

static long svd_ioctl(
                struct file *file,
                unsigned int cm,
                unsigned long arg) {
        return svd_do_ioctl(file, cm, (void __user *)arg);
}

static int svd_mmap(struct file *file, struct vm_area_struct *vma) {
        struct svd_file *priv = file->private_data;
        struct svd_device *svd = priv->dev;
        unsigned long pfn;
        size_t size;

        size = vma->vm_end - vma->vm_start;
        if (size > priv->size) {
                dev_info(svd->dev, "size: %zu, max size: %zu\n", size, priv->size);
                return -EINVAL;
        }
        pfn = page_to_pfn(virt_to_page(priv->vbuf));
        return remap_pfn_range(vma, vma->vm_start, pfn, size, vma->vm_page_prot);
}

static int svd_open(struct inode *inode, struct file *file) {
        struct svd_file *priv;
        struct svd_device *svd;
        int rc;

        priv = kzalloc(sizeof(*priv), GFP_KERNEL);
        if (priv == NULL)
                return -ENOMEM;

        svd = container_of(inode->i_cdev, struct svd_device, cdev);
        priv->dev = svd;
        priv->size = svd->max_size;

        priv->vbuf = dma_alloc_coherent(NULL, priv->size, &priv->dma_handle, GFP_KERNEL
        if (priv->vbuf == NULL) {
                dev_err(svd->dev, "cannot allocate contiguous DMA buffer of size %zu\n"
                rc = -ENOMEM;
                goto err_dma_alloc;
        }
```

```
        if (!try_module_get(svd->module)) {
                rc = -ENODEV;
                goto err_module_get;
        }

        file->private_data = priv;
        return 0;

err_module_get:
        dma_free_coherent(NULL, priv->size, priv->vbuf, priv->dma_handle);
err_dma_alloc:
        kfree(priv);
        return rc;
}

static int svd_release(struct inode *inode, struct file *file) {
        struct svd_file *priv = file->private_data;
        struct svd_device *svd = priv->dev;

        module_put(svd->module);
        dma_free_coherent(NULL, priv->size, priv->vbuf, priv->dma_handle);
        kfree(priv);
        return 0;
}

/* driver-defined functions for interacting with device */
static const struct file_operations svd_fops = {
        .owner          = THIS_MODULE,
        .open           = svd_open,
        .release        = svd_release,
        .unlocked_ioctl = svd_ioctl,
        .mmap           = svd_mmap,
};

static int svd_create_cdev(struct svd_device *svd, int ndev) {
        dev_t devno = MKDEV(MAJOR(svd_devno), ndev);
        int rc;

        /* char device registration */
        cdev_init(&svd->cdev, &svd_fops);
        svd->cdev.owner = THIS_MODULE;
        rc = cdev_add(&svd->cdev, devno, 1);
        if (rc) {
                dev_err(svd->dev, "Error %d adding cdev %d\n", rc, ndev);
                goto out;
```

```
        }

        svd->dev = device_create(svd_class, svd->dev, devno, NULL, "svd.%i", ndev);
        if (IS_ERR(svd->dev)) {
                rc = PTR_ERR(svd->dev);
                dev_err(svd->dev, "Error %d creating device %d\n", rc, ndev);
                svd->dev = NULL;
                goto device_create_failed;
        }

        dev_set_drvdata(svd->dev, svd);
        return 0;

device_create_failed:
        cdev_del(&svd->cdev);
out:
        return rc;
}

static void svd_destroy_cdev(struct svd_device *svd, int ndev) {
        dev_t devno = MKDEV(MAJOR(svd_devno), ndev);

        device_destroy(svd_class, devno);
        cdev_del(&svd->cdev);
}

static int svd_probe(struct dad_device *dev) {
        struct svd_device *svd;
        int dev_id;
        int rc;
        unsigned max_sz;

        svd = kzalloc(sizeof(*svd), GFP_KERNEL);
        if (svd == NULL)
                return -ENOMEM;
        svd->module = THIS_MODULE;
        svd->dad_dev = dev;
        svd->number = svd_n_devices;
        mutex_init(&svd->lock);
        init_completion(&svd->completion);

        svd->iomem = ioremap(dev->addr, dev->length);
        if (svd->iomem == NULL) {
                rc = -ENOMEM;
                goto err_ioremap;
        }
```

```
        dev_id = ioread32(svd->iomem + SVD_REG_ID);
        if (dev_id != SVD_SYNC_DEV_ID) {
                rc = -ENODEV;
                goto err_reg_read;
        }

        rc = svd_create_cdev(svd, svd->number);
        if (rc)
                goto err_cdev;

        rc = request_irq(dev->irq, svd_irq, IRQF_SHARED, "svd", svd);
        if (rc) {
                dev_info(svd->dev, "cannot request IRQ number %d\n", -EMSGSIZE);
                goto err_irq;
        }

        svd_n_devices++;
        dev_set_drvdata(&dev->device, svd);

        max_sz = ioread32(svd->iomem + SVD_REG_MAX_SIZE);
        svd->max_size = round_up(SVD_BUF_SIZE_BYTE(max_sz), PAGE_SIZE);

        dev_info(svd->dev, "device registered.\n");

        return 0;

err_irq:
        svd_destroy_cdev(svd, svd->number);
err_cdev:
        iounmap(svd->iomem);
err_reg_read:
        iounmap(svd->iomem);
err_ioremap:
        kfree(svd);
        return rc;
}

static void __exit svd_remove(struct dad_device *dev) {
        struct svd_device *svd = dev_get_drvdata(&dev->device);

        /* free_irq(dev->irq, svd->dev); */
        svd_destroy_cdev(svd, svd->number);
        iounmap(svd->iomem);
        kfree(svd);
        dev_info(svd->dev, "device unregistered.\n");
```

```
}

static struct dad_driver svd_driver = {
        .probe      = svd_probe,
        .remove     = svd_remove,
        .name       = DRV_NAME,
        .id_table   = svd_ids,
};

static int __init svd_sysfs_device_create(void) {
        int rc;

        svd_class = class_create(THIS_MODULE, "svd");
        if (IS_ERR(svd_class)) {
                printk(KERN_ERR PFX "Failed to create svd class\n");
                rc = PTR_ERR(svd_class);
                goto out;
        }

        /* Dynamically allocating dev numbers */
        rc = alloc_chrdev_region(&svd_devno, 0, SVD_MAX_DEVICES, "svd");
        if (rc) {
                printk(KERN_ERR PFX "Failed to allocate chrdev region\n");
                goto alloc_chrdev_region_failed;
        }

        return 0;

alloc_chrdev_region_failed:
        class_destroy(svd_class);
out:
        return rc;
}

static void svd_sysfs_device_remove(void) {
        dev_t devno = MKDEV(MAJOR(svd_devno), 0);

        class_destroy(svd_class);

        /* get rid of dev numbers */
        unregister_chrdev_region(devno, SVD_MAX_DEVICES);
}

static int __init svd_init(void) {
        int rc;
        printk(KERN_INFO "Device-driver initialization\n");
```

```
        rc = svd_sysfs_device_create();
        if (rc)
                return rc;

        rc = dad_register_driver(&svd_driver);
        if (rc)
                goto err;

        return 0;

err:
        svd_sysfs_device_remove();
        return rc;
}


/* shut 'er down */
static void __exit svd_exit(void) {
        printk(KERN_INFO "Device-driver shutdown\n");
        dad_unregister_driver(&svd_driver);
        svd_sysfs_device_remove();
}

module_init(svd_init)
module_exit(svd_exit)

MODULE_AUTHOR("Chae Jubb <ecj2122@columbia.edu>");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("svd driver");

#ifndef _MYDATA_H_
#define _MYDATA_H_

#define SC_INCLUDE_FX
#include <systemc.h>

//#define REAL_FLOAT
//#define SC_FIXED_POINT_FAST
//#define SC_FIXED_POINT
#define CTOS_SC_FIXED_POINT

#ifdef __CTOS__
#define MAX_SIZE    64
#else
#define MAX_SIZE    20
#endif
```

```
#define MAX_ITERATIONS  1000000

#if defined(REAL_FLOAT)
#define SVD_CELL_TYPE double
#elif defined(SC_FIXED_POINT_FAST)
#define SC_FIXED_TYPE sc_dt::sc_fixed_fast
#elif defined(SC_FIXED_POINT)
#define SC_FIXED_TYPE sc_dt::sc_fixed
#elif defined(CTOS_SC_FIXED_POINT)
#include <ctos_fx.h>
#define SC_FIXED_TYPE ctos_sc_dt::sc_fixed
#else
#error "specify fixed or floating point type"
#endif

#ifndef REAL_FLOAT
#define WL  64
#define IWL 24
#define SVD_CELL_TYPE SC_FIXED_TYPE<WL,IWL>
#endif

#define SVD_INPUT_SIZE(__sz) (__sz * __sz)
#define SVD_OUTPUT_SIZE(__sz) (3 * __sz * __sz)
#define SVD_GET_S(__ptr, __sz) (__ptr)
#define SVD_GET_U(__ptr, __sz) (__ptr + __sz * __sz)
#define SVD_GET_V(__ptr, __sz) (__ptr + 2 * __sz * __sz)

#endif
#ifndef _SVD_H_
#define _SVD_H_

#ifdef __KERNEL__
#include <linux/ioctl.h>
#include <linux/types.h>
#else
#include <sys/ioctl.h>
#include <stdint.h>
#ifndef __user
#define __user
#endif
#endif /* __KERNEL__ */

#include "svd_data.h"

#define DRV_NAME  "svd"
```

```
#define PFX     DRV_NAME ": "
#define SVD_MAX_DEVICES   64

#define SVD_REG_CMD         0x00
#define SVD_REG_SRC         0x04
#define SVD_REG_DST         0x08
#define SVD_REG_SIZE        0x0C
#define SVD_REG_MAX_SIZE    0x10
#define SVD_REG_ID          0x14

#define SVD_CMD_IRQ_SHIFT     4
#define SVD_CMD_IRQ_MASK      0x03
#define SVD_CMD_IRQ_DONE      0x02

#define SVD_SYNC_DEV_ID       0x02

/* buffer sizes */
#define SVD_INPUT_SIZE_BYTE(__sz) (sizeof(SVD_CELL_TYPE) * __sz * __sz)
#define SVD_OUTPUT_SIZE_BYTE(__sz) (3 * sizeof(SVD_CELL_TYPE) * __sz * __sz)
#define SVD_BUF_SIZE_BYTE(__sz) \
        (SVD_INPUT_SIZE_BYTE(__sz) + \
         SVD_OUTPUT_SIZE_BYTE(__sz))

struct svd_access {
        unsigned size;
};

#define SVD_IOC_ACCESS  _IOW ('I', 0, struct svd_access)

#endif /* _SVD_H_ */
/*
 * svd-mmap.c
 * Sowftware and hardware-accelerated SVD.
 *
 */
#include <sys/mman.h>
#include <sys/stat.h>
#include <inttypes.h>
#include <stdbool.h>
#include <assert.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
```

```
#include <math.h>

#include "svd.h"
#include "svd_data.h"
#include "gm_jacobi.h"

static const char *devname = "/dev/svd.0";

void compute_golden_model(SVD_CELL_TYPE *golden_input_matrix,
                SVD_CELL_TYPE *golden_matrix, int mat_size) {
        gm_jacobi(golden_input_matrix, mat_size,
                        SVD_GET_S(golden_matrix, mat_size),
                        SVD_GET_U(golden_matrix, mat_size),
                        SVD_GET_V(golden_matrix, mat_size));
}

int main(int argc, char **argv) {
        size_t sz = 4, buf_size;
        char *buf;

        int fd, rc;
        struct svd_access desc;

        SVD_CELL_TYPE *input_matrix, *golden_input_matrix;
        SVD_CELL_TYPE *output_matrix, *golden_matrix;

        unsigned long mismatches = 0;
        size_t i, j;

        srand(time(NULL));

        input_matrix = malloc(sizeof(*input_matrix) * SVD_INPUT_SIZE(sz));
        golden_input_matrix = malloc(sizeof(*golden_input_matrix) * SVD_INPUT_SIZE(sz));
        output_matrix = malloc(sizeof(*output_matrix) * SVD_OUTPUT_SIZE(sz));
        golden_matrix = malloc(sizeof(*golden_matrix) * SVD_OUTPUT_SIZE(sz));

        for (i = 0; i < sz; ++i) {
                for (j = 0; j < sz; ++j) {
                        input_matrix[i * sz + j] = rand();
                }
        }

        memcpy(golden_input_matrix, input_matrix, sizeof(*input_matrix) *
                        SVD_INPUT_SIZE(sz));

        compute_golden_model(golden_input_matrix, golden_matrix, (int) sz);
```

```
fd = open(devname, O_RDWR, 0);
if (fd < 0) {
        perror("open");
        exit(EXIT_FAILURE);
}

buf_size = SVD_BUF_SIZE_BYTE(sz);

buf = mmap(NULL, buf_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (buf == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
}

memmove(buf, input_matrix, SVD_INPUT_SIZE_BYTE(sz));

desc.size = sz;

rc = ioctl(fd, SVD_IOC_ACCESS, &desc);
if (rc < 0) {
        perror("ioctl");
        exit(EXIT_FAILURE);
}

memmove(output_matrix, buf + SVD_INPUT_SIZE_BYTE(sz), SVD_OUTPUT_SIZE_BYTE(sz));

for (i = 0; i < SVD_OUTPUT_SIZE(sz); ++i) {
        SVD_CELL_TYPE diff = output_matrix[i] - golden_matrix[i];
        if (diff < 0) {
                diff *= -1;
        }
        mismatches += !(diff < MAX_ERROR);
}

if (mismatches) {
        printf("Simulation failed with %lu errors\n", mismatches);
} else {
        printf("Simulation success! No errors\n");
}

free(input_matrix);
free(output_matrix);
close(fd);

return 0;
```

```
}
#include <string.h>
#include <errno.h>

#include <ccan/array_size/array_size.h>

#include "device-list.h"
#include "device.h"
#include "exit.h"

extern const struct device_init_operations svd_sync_init_ops;

static const struct device_desc device_list[] = {
        {
                .type           = "svd-sync",
                .description    = "SystemC SVD",
                .ops            = &svd_sync_init_ops,
                .id             = 0x2,
        },
};


const struct device_desc *device_desc_by_type(const char *type)
{
        unsigned int i;

        for (i = 0; i < ARRAY_SIZE(device_list); i++) {
                const struct device_desc *desc = &device_list[i];

                if (!strcmp(type, desc->type))
                        return desc;
        }
        errno = ENODEV;
        return NULL;
}
all: cargo

CARGO ?= /usr/local/share/cargo
QSIM ?= ${CARGO}/qsim

SVD_OBJS = svd-sync.o device-list.o
SVD_CXX_OBJS = svd.opp svd-main.opp svd-wrapper.opp
HDRS = svd.h  svd-wrapper.hpp  svd-sync.h svd_data.h

SVD_DIR := $(dir $(lastword $(MAKEFILE_LIST)))
CARGO_DEV_OBJS += $(addprefix $(SVD_DIR),$(SVD_OBJS))
```

```
CARGO_DEV_CXX_OBJS += $(addprefix $(SVD_DIR),$(SVD_CXX_OBJS))

QSIM ?= qsim
CCAN ?= $(QSIM)/ccan
LUA ?= lua
MCPAT ?= mcpat
SYSTEMC_PATH := $(CARGO)/systemc
SYSTEMC ?= $(SYSTEMC_PATH)/installed

QSIM_LIB := $(QSIM)/lib/libqsim.a
CCAN_LIB := $(CCAN)/libccan.a
LUA_LIB := $(LUA)/lib/liblua.a
MCPAT_LIB := $(MCPAT)/lib/libmcpat.a
SYSTEMC_LIB := $(SYSTEMC)/lib-linux64/libsystemc.a
CSP_LIB := $(CARGO)/csp/lib.a

DEP_LIBS :=

# cargo objs depend on DEP_LIBS, but only the cargo binary depends on ALL_LIBS
ALL_LIBS :=
ALL_LIBS += $(DEP_LIBS)
ALL_LIBS += $(CSP_LIB)

INCLUDE_FLAGS :=
INCLUDE_FLAGS += -I.
INCLUDE_FLAGS += -I$(CARGO)
INCLUDE_FLAGS += -I$(CCAN)
INCLUDE_FLAGS += -I$(QSIM)/include
INCLUDE_FLAGS += -I$(CARGO)/$(LUA)/include
INCLUDE_FLAGS += -I$(CARGO)/$(MCPAT)/include
INCLUDE_FLAGS += -I$(SYSTEMC)/include

CFLAGS = -g3 -ggdb -O0
CFLAGS ?=
include ../svd/sim/Makefile.params
ALL_CFLAGS :=
ALL_CFLAGS += $(CFLAGS)
ALL_CFLAGS += $(INCLUDE_FLAGS)
ALL_CFLAGS += $(LUA_CFLAGS)
ALL_CFLAGS += -fms-extensions
ALL_CFLAGS += -Wall
ALL_CFLAGS += -Wextra
ALL_CFLAGS += -Wmissing-declarations
ALL_CFLAGS += -Wno-missing-field-initializers
ALL_CFLAGS += -Wno-unused-parameter
ALL_CFLAGS += -Wpointer-arith
```

```
ALL_CFLAGS += -Wundef
ALL_CXXFLAGS := $(ALL_CFLAGS)
# C-only flags
ALL_CFLAGS += -Wold-style-definition
ALL_CFLAGS += -Wmissing-prototypes
ALL_CFLAGS += -Wstrict-prototypes

ALL_CXXFLAGS += -Wno-unused-label
ALL_CXXFLAGS += -DSC_INCLUDE_FX

LDFLAGS :=
LDFLAGS += -L$(CCAN)
LDFLAGS += -L$(QSIM)/lib
LDFLAGS += -L$(CARGO)/$(LUA)/lib
LDFLAGS += -L$(CARGO)/$(MCPAT)/lib
LDFLAGS += -L$(SYSTEMC)/lib-linux64

PROFILE_LDFLAGS := -L$(CCAN)

LIBS :=
LIBS += -lccan
LIBS += -lqsim
LIBS += -llua
LIBS += -lmcpat
# libstdc++ needed by mcpat
LIBS += -lstdc++
LIBS += -l:libsystemc.a
LIBS += -lpthread
LIBS += -ldl
LIBS += -lrt
LIBS += -lm
LIBS += $(CARGO)/csp/lib.a

OBJECTS :=
OBJECTS += $(CARGO_OBJS) $(CARGO_DEV_OBJS)
OBJECTS += $(CSP_OBJS)

CXX_OBJECTS :=
CXX_OBJECTS += $(CARGO_DEV_CXX_OBJS)

C_OBJECTS := $(OBJECTS)
ALL_OBJECTS := $(C_OBJECTS) $(CXX_OBJECTS)

CC ?= gcc
CXX := g++
CHECK := sparse
```

```
ifndef V
        QUIET_AR             = @echo '    ' AR $@;
        QUIET_BUILD          = @echo '    ' BUILD $@;
        QUIET_CC             = @echo '    ' CC $@;
        QUIET_CXX            = @echo '    ' CXX $@;
        QUIET_CHECKPATCH     = @echo '    ' CHECKPATCH $(subst .o,.c,$@);
        QUIET_CHECK          = @echo '    ' CHECK $(subst .o,.c,$@);
        QUIET_LINK           = @echo '    ' LINK $@;
endif

# objects depend on libraries we build so that header files can be found
$(C_OBJECTS): %.o: %.c $(missing_dep_dirs) $(DEP_LIBS)
        $(QUIET_CC)$(CC) -g3  -o $*.o -c $(dep_args) $(ALL_CFLAGS) $<

$(CXX_OBJECTS): %.opp: %.cpp $(missing_dep_dirs) $(DEP_LIBS)
        $(QUIET_CXX)$(CXX) -g3 $(ALL_CXXFLAGS) -o $*.opp -c $(dep_args) $<

cargo: $(CARGO_DEV_OBJS) $(CARGO_DEV_CXX_OBJS) $(ALL_LIBS) $(HDRS)
        $(QUIET_LINK)$(CC) -g3 -o$@ $(LDFLAGS) $(CARGO)/libcargo.a $(filter %.opp,$^)
$(filter %.o,$^) $(LIBS)


run: cargo
        @make -C ./test
        @./runcargo svd

.PHONY:

clean:
        $(RM) $(ALL_OBJECTS)
        $(RM) cargo
        $(RM) CARGO-CFLAGS CARGO-LDFLAGS
        $(RM) cargo.power* cargo.data*
        make -C ./test distclean

.PHONY: all clean
#!/bin/bash

export CARGO=/usr/local/share/cargo
export QSIM=$CARGO/qsim

if [ $# -eq 0 ]; then
  echo -e "Usage:\n\t$0 <design_name>"
  exit 1
fi
```

```
DESIGN=$1
CMD="./cargo -b test/$DESIGN.tar \
        -s $CARGO/qsim/tools/qsim-ff/state.1 \
        -c test/config.lua"
echo $CMD && $CMD
#include "svd.h"

#ifdef CTOS_SC_FIXED_POINT
#include "svd_ctos_funcs.h"
#endif

SVD_CELL_TYPE inline fp_abs(SVD_CELL_TYPE in) {
#ifdef CTOS_SC_FIXED_POINT
        return sld::abs(in);
#else
        if (in >= 0)
                return in;
        else
                return -in;
#endif
}

void svd::identify (SVD_CELL_TYPE *matrix, int dimension) {
        int row, col;
IDENTITY_OUTER:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;
IDENTITY_INNER:
                for (col = 0; col < MAX_SIZE; col++) {
                        if (col == dimension) break;
                        // Elements on diagonal =1
                        if (col == row) {matrix [(row * dimension) + col] = 1.0;}
                        // ... all others 0
                        else {matrix [(row * dimension) + col] = 0.0;}
                }
        }
        return;
}

void svd::copyMatrix (SVD_CELL_TYPE *a, SVD_CELL_TYPE *b, int dimension) {
        int row, col;

COPY_MATRIX_OUTER:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;
```

```
COPY_MATRIX_INNER:
                for (col = 0; col < MAX_SIZE; col++) {
                        if (col == dimension) break;
                        b [(row * dimension) + col] = a [(row * dimension) + col];
                }
        }
        return;
}

void svd::multiply (SVD_CELL_TYPE *left, SVD_CELL_TYPE *right, SVD_CELL_TYPE *result, in
        int leftRow, rightRow; // row numbers of the left and right matrix, respectively
        int leftCol, rightCol; // same as above but for columns
        SVD_CELL_TYPE tempResult = 0;

MULTIPLY_MATRIX_OUTER:
        for (leftRow = 0; leftRow < MAX_SIZE; leftRow++) {
                if (leftRow == dimension) break;

MULTIPLY_MATRIX_MID:
                for (rightCol = 0; rightCol < MAX_SIZE; rightCol++) {
                        if (rightCol == dimension) break;

MULTIPLY_MATRIX_INNER:
                        for (leftCol = rightRow = 0; leftCol < MAX_SIZE;
                                        leftCol++, rightRow++) {
                                if (leftCol == dimension) break;
                                /* TODO reevaluate best way to handle this re: waits */
                                SVD_CELL_TYPE tmp, tmp2;
                                tmp = left[(leftRow * dimension) + leftCol];
                                wait();
                                tmp2 = right[(rightRow * dimension) + rightCol];
                                tempResult += tmp * tmp2;
                                wait();
                        }
                        result [(leftRow * dimension) + rightCol] = tempResult;
                        tempResult = 0;
                        wait();
                }
        }
        return;
}

void svd::jacobi (SVD_CELL_TYPE *a, int n, SVD_CELL_TYPE *s, SVD_CELL_TYPE *u, SVD_CELL_
        // Arrays that contain the coordinates of the elements of the 2x2
        // sub matrix formed by the largest off-diagonal element. First entry
```

```
        // is the row number and second entry is the column number.
        int a11[2];
        int a12[2];
        int a21[2];
        int a22[2];
        LargestElement le; // largest element of matrix a
        le.value = le.rowNum = le.colNum = -1;
        int i, j;

        // I could have statically allocated these but I would not have been
        // able to pass them as double pointers: this is a known problem in C

        identify (u, n);
        identify (v, n);

        if (n == 1) {    // 1x1 matrix is already in SVD
                SVD_CELL_TYPE tmp = s[0];
                wait();
                s[0] = tmp;
                return;
        }

        le = findLargestElement (a, n, a11, a12, a21, a22);

        int count = 0;
        SVD_CELL_TYPE old_value = 0;
        /* FIXME This loop may not be synthesizable */

CONVERGENCE_LOOP:
        while (fp_abs (le.value) >
#ifndef REAL_FLOAT
                        SVD_CELL_TYPE(SVD_PRECISION)
#else
                        SVD_PRECISION
#endif
                        && fp_abs(le.value - old_value) > SVD_CELL_TYPE(MIN_MOVEMENT)

            && count < MAX_ITERATIONS
            ) {
                old_value = le.value;
                count++;
                rotate (a, n, u, v, a11, a12, a21, a22);
                le = findLargestElement (a, n, a11, a12, a21, a22);
#ifndef REAL_FLOAT
                if (!(count % 10))
                cerr << "current iteration: " << count << "; " << le.value.to_double() <
```

```
                        old_value.to_double() << endl;
#endif

        }
#ifndef REAL_FLOAT
                cerr << "current iteration: " << count << "; " << le.value.to_double() <
                        old_value.to_double() << endl;
#endif

        cout << "loop count: " << count << endl;

        reorder (a, n, u, v);
        wait();

        // Copy over the singular values in a to s
COPY_SINGULAR_VALUES:
        for (i = 1; i < MAX_SIZE; i++) {
                if (i == n) break;
                SVD_CELL_TYPE tmp;
                tmp = s[(i * n) + i];
                wait();
                s[i] = tmp;
                wait();
                s[(i * n) + i] = 0;
                wait();
        }

        return;
}

void svd::rotate (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *v, i
        SVD_CELL_TYPE a11, a12, a21, a22; // elements of the sub-matrix
        SVD_CELL_TYPE alpha, beta; // angles used in rotations; alpha is angle of left r
        SVD_CELL_TYPE cosA, sinA, cosB, sinB;
        SVD_CELL_TYPE X, Y; // temporary values used in calculating angles

        // Assign elements of sub-matrix to the their actual values from a
        a11 = a[x11[0] * dimension + x11[1]];
        wait();
        a12 = a[x12[0] * dimension + x12[1]];
        wait();
        a21 = a[x21[0] * dimension + x21[1]];
        wait();
        a22 = a[x22[0] * dimension + x22[1]];
        wait();
```

```
        // Calculate angles and sin and cos of those angles using the closed
        // formulas found.
#ifdef CTOS_SC_FIXED_POINT
        X = sld::atan2_cordic_func<CORDIC_ITER, WL, IWL>((SVD_CELL_TYPE) (a11+a22), (SVI
        Y = sld::atan2_cordic_func<CORDIC_ITER, WL, IWL>((SVD_CELL_TYPE) (a11-a22), (SVI
        //cout << "X, Y: " << X << "; " << Y << "; " << (X + Y) << "; " << (Y - X) << en

        alpha = (X + Y) * SVD_CELL_TYPE(0.5);//sld::div_func<WL, IWL, WL, IWL, WL, IWL>
        beta = (Y - X) * SVD_CELL_TYPE(0.5);//sld::div_func<WL, IWL, WL, IWL, WL, IWL>(
        //cout << "a, b: " << alpha << "; " << beta << endl;

        sld::cos_sin_cordic_func<CORDIC_ITER, WL, IWL>(alpha, cosA, sinA);
        //cosA = sld::cos_cordic_func<CORDIC_ITER, WL, IWL>(alpha);
        //sinA = sld::sin_cordic_func<CORDIC_ITER, WL, IWL>(alpha);
        sld::cos_sin_cordic_func<CORDIC_ITER, WL, IWL>(beta, cosB, sinB);
        //cosB = sld::cos_cordic_func<CORDIC_ITER, WL, IWL>(beta);
        //sinB = sld::sin_cordic_func<CORDIC_ITER, WL, IWL>(beta);

        //cout << "rotate stats: " << cosA << "; " << sinA << "; " << cosB << "; " << si
#else
        X = atan((a21 - a12) / (a11 + a22));
        Y = atan((a21 + a12) / (a11 - a22));
        alpha = 0.5 * (X + Y);
        beta = 0.5 * (Y - X);

        cosA = cos (alpha); sinA = sin (alpha);
        cosB = cos (beta); sinB = sin (beta);
#endif

        // Create left rotation matrix, namely U_i which looks like this
        //
        //              | cosA  sinA |
        //              | -sinA cosA |
        //
        identify (Ui, dimension);
        Ui [x11[0] * dimension + x11[1]] = Ui [x22[0] * dimension + x22[1]] = cosA;
        Ui [x12[0] * dimension + x12[1]] = sinA;
        Ui [x21[0] * dimension + x21[1]] = -sinA;
        // Rotate a
    // First on the left with Ui
    multiply (Ui, a, tempMatrix, dimension);
    copyMatrix (tempMatrix, a, dimension);
        // Apply rotation matrix Ui to U
    multiply (Ui, u, tempMatrix, dimension);
    copyMatrix (tempMatrix, u, dimension);
    // Create the right rotation matrix, namely V_i which looks like this
```

```
    // Note that I'm using Ui as Vi
    //          | cosB  -sinB |
    //          | sinB   cosB |
    //
    identify (Ui, dimension);
    Ui [x11[0] * dimension + x11[1]] = Ui [x22[0] * dimension + x22[1]] = cosB;
    Ui [x12[0] * dimension + x12[1]] = -sinB;
    Ui [x21[0] * dimension + x21[1]] = sinB;

    // Then on the right with Vi
    multiply (a, Ui, tempMatrix, dimension);
    copyMatrix (tempMatrix, a, dimension);
    multiply (v, Ui, tempMatrix, dimension);
    copyMatrix (tempMatrix, v, dimension);



/*

    // Create the right rotation matrix, namely V_i which looks like this
        //
        //              | cosB  -sinB |
        //              | sinB   cosB |
        //
        identify (Vi, dimension);
        Vi [x11[0] * dimension + x11[1]] = Vi [x22[0] * dimension + x22[1]] = cosB;
        Vi [x12[0] * dimension + x12[1]] = -sinB;
        Vi [x21[0] * dimension + x21[1]] = sinB;

        // Rotate a
        // First on the left with Ui
        multiply (Ui, a, tempResult, dimension);
        copyMatrix (tempResult, a, dimension);
        // Then on the right with Vi
        multiply (a, Vi, tempResult, dimension);
        copyMatrix (tempResult, a, dimension);

        // Apply rotation matrix Ui to U
        multiply (Ui, u, tempResult, dimension);
        copyMatrix (tempResult, u, dimension);
        // Apply rotation matrix Vi to V
        multiply (v, Vi, tempResult, dimension);
        copyMatrix (tempResult, v, dimension);
*/
        return;
}
```

```
LargestElement svd::findLargestElement (SVD_CELL_TYPE *matrix, int dimension, int *a11,
        LargestElement leTemp;
        int i, j;

        leTemp.value = 0;
        leTemp.rowNum = leTemp.colNum = -1;

        // Populate leArray such that the entry at index i contains information
        // about the largest element of row i
FLE_POPULATE_OUTER:
        for (i = 0; i < MAX_SIZE; i++) {
                if (i == dimension) break;

FLE_POPULATE_INNER:
                for (j = 0; j < MAX_SIZE; j++) {
                        if (j == dimension) break;
                        // We are looking for the largest OFF-DIAGONAL element
                        if (j == i)
                                continue;
                        if (fp_abs (matrix [(i * dimension) + j]) > fp_abs (leTemp.value
                                leTemp.value = matrix[(i * dimension) + j];
                                leTemp.rowNum = i;
                                leTemp.colNum = j;
                        }
                }
                //leArray[i].value = leTemp.value;
                //leArray[i].rowNum = leTemp.rowNum;
                //leArray[i].colNum = leTemp.colNum;
                //leTemp.value = 0;
        }
        //leTemp.value = 0;
        //leTemp.rowNum = leTemp.colNum = -1;
        // Iterate over leArray and find the largest element in the matrix as a
        // whole
/*
FLE_ITERATE_LOOP:
        for (i = 0; i < MAX_SIZE; i++) {
                if (i == dimension) break;
                if (fp_abs (leArray [i].value) > fp_abs (leTemp.value)) {
                        leTemp.value = leArray [i].value;
                        leTemp.rowNum = leArray [i].rowNum;
                        leTemp.colNum = leArray [i].colNum;
                }
        }
*/
```

```
        // Determining coordinates of the 2x2 sub matrix formed by this largest
        // element
        if (leTemp.rowNum < leTemp.colNum) { // largest element is above diagonal
                (a11) [0] = (a11) [1] = (a12) [0] = (a21) [1] = leTemp.rowNum;
                (a21) [0] = (a22) [0] = (a22) [1] = (a12) [1] = leTemp.colNum;
        }
        else { // below diagonal
                a22[0] = leTemp.rowNum;
                a22[1] = leTemp.rowNum;
                a21[0] = leTemp.rowNum;
                a12[1] = leTemp.rowNum;

                a11[0] = leTemp.colNum;
                a11[1] = leTemp.colNum;
                a21[1] = leTemp.colNum;
                a12[0] = leTemp.colNum;
        }

        return leTemp;
}

void svd::transpose (SVD_CELL_TYPE *matrix, int dimension) {
        SVD_CELL_TYPE temp, temp2;
        int row, col;

TRANSPOSE_OUTER:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;

TRANSPOSE_INNER:
                for (col = (row + 1); col < MAX_SIZE; col++) {
                        /* TODO rework indexing */
                        if (col == dimension) break;
                        if (col == row) {continue;} // skip diagonal elements
                        temp = matrix [(row * dimension) + col];
                        wait();
                        temp2 = matrix [(col * dimension) + row];
                        wait();
                        matrix [(row * dimension) + col] = temp2;
                        wait();
                        matrix [(col * dimension) + row] = temp;
                        wait();
                }
        }
        return;
}
```

```
void svd::reorder (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *v)
        int row, col, x, largestElementRow;
        SVD_CELL_TYPE temp; // stores the largest singular value
    //Replace p with Ui because why not?



REORDER_OUTER:
        for (x = 0; x < MAX_SIZE; x++) {
                if (x == dimension) break;
                temp = 0.0;
                identify (Ui, dimension);

REORDER_INNER:
                for (row = x; row < MAX_SIZE; row++) {
                        if (row == dimension) break;
                        col = row;
                        if (fp_abs (a [row * dimension + col]) > fp_abs (temp)) {
                                temp = a [row * dimension + col];
                                largestElementRow = row;
                        }
                }
                swapRows (Ui, x, largestElementRow, dimension);
                // Reorder a
                multiply (Ui, a, tempMatrix, dimension); copyMatrix (tempMatrix, a, dime
                multiply (a, Ui, tempMatrix, dimension); copyMatrix (tempMatrix, a, dime
                // Reorder u
                multiply (Ui, u, tempMatrix, dimension); copyMatrix (tempMatrix, u, dime
                // Reorder v
                multiply (v, Ui, tempMatrix, dimension); copyMatrix (tempMatrix, v, dime
        }

        transpose (u, dimension);
        transpose (v, dimension);
        identify (Ui, dimension);
REORDER_FIND_NEG:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;
                col = row;
                if (a [row * dimension + col] < 0)
                        Ui[row * dimension + col] = -1.0;
        }
        multiply (Ui, a, tempMatrix, dimension);
        copyMatrix (tempMatrix, a, dimension);
        multiply (u, Ui, tempMatrix, dimension);
```

```
        copyMatrix (tempMatrix, u, dimension);
/* Old version
REORDER_OUTER:
for (x = 0; x < MAX_SIZE; x++) {
    if (x == dimension) break;
    temp = 0.0;
    identify (p, dimension);

REORDER_INNER:
    for (row = x; row < MAX_SIZE; row++) {
        if (row == dimension) break;
        col = row;
        if (fp_abs (a [row * dimension + col]) > fp_abs (temp)) {
            temp = a [row * dimension + col];
            largestElementRow = row;
        }
    }
    swapRows (p, x, largestElementRow, dimension);
    // Reorder a
    multiply (p, a, tempMatrix, dimension); copyMatrix (tempMatrix, a, dimension);
    multiply (a, p, tempMatrix, dimension); copyMatrix (tempMatrix, a, dimension);
    // Reorder u
    multiply (p, u, tempMatrix, dimension); copyMatrix (tempMatrix, u, dimension);
    // Reorder v
    multiply (v, p, tempMatrix, dimension); copyMatrix (tempMatrix, v, dimension);
}

transpose (u, dimension);
transpose (v, dimension);
identify (p, dimension);
REORDER_FIND_NEG:
for (row = 0; row < MAX_SIZE; row++) {
    if (row == dimension) break;
    col = row;
    if (a [row * dimension + col] < 0)
        p [row * dimension + col] = -1.0;
}
multiply (p, a, tempMatrix, dimension);
copyMatrix (tempMatrix, a, dimension);
multiply (u, p, tempMatrix, dimension);
copyMatrix (tempMatrix, u, dimension);
*/
    return;
}

// Swap row a with row b of matrix m
```

```
void svd::swapRows (SVD_CELL_TYPE *m, int a, int b, int dimension) {
        SVD_CELL_TYPE temp;

SWAP_ROW:
        for (int col = 0; col < MAX_SIZE; col++) {
                if (col == dimension) break;
                temp = m [a * dimension + col];
                m [a * dimension + col] = m [b * dimension + col];
                m [b * dimension + col] = temp;
        }
        return;
}


void svd::config_svd()
{
        // Initialization
        size.write(0);
        init_done.write(false);

        wait();

        // Read configuration until done
        bool done = false;
CONFIG_REGISTER_WHILE:
        do {
                wait();
                done = conf_done.read();
                int a = conf_size.read();
                size.write(a);
        } while (!done);

        // Let other threads run then do nothing
        init_done.write(true);
        while (true) {
                wait();
        }
}

void svd::load_input()
{
RESET_LOAD:
        bufdin.reset_get();

        rd_index.write(0);
        rd_length.write(0);
```

```
        rd_request.write(false);

        input_done.write(false);

        do {wait();}
        while(!init_done.read());

        const int rows = size.read();
        int index = 0;

LOAD_INPUT_WHILE:
        while(true) {
                if (index == rows * rows)
                        // Input complete; wait for reset
                        do { wait(); } while (true);

                int length = rows * rows;

                rd_index.write(index);
                rd_length.write(length);
                index += length;

                // 4-phase handshake
                rd_request.write(true);
                do { wait(); } while(!rd_grant.read());
                rd_request.write(false);
                do { wait(); } while(rd_grant.read());

LOAD_OUTER:
                for (int i = 0; i < MAX_SIZE; ++i) {
                        if (i == rows) break;
LOAD_INNER:
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                if (j == rows) break;
                                SVD_CELL_TYPE cell = bufdin.get();
                                matrix_in[i * rows + j] = cell;
#ifdef VERBOSE
                                cout << "DUT GET: (index, length, i, j, val)" << index <
                                        << " " << i << " " << j << " " << cell << endl;
#endif
                                wait();
                        }
                }

                // 4-phase handshake
                input_done.write(true);
```

```
                do { wait(); } while (!process_start.read());
                input_done.write(false);
                do { wait(); } while (process_start.read());
        }
}

void svd::store_output()
{
RESET_STORE:
        bufdout.reset_put();

        wr_index.write(0);
        wr_request.write(false);
        wr_length.write(0);

        output_start.write(false);
        svd_done.write(false);

        do {wait();}
        while(!init_done.read());

        const int rows = size.read();
        int length = rows * rows;
        int index = 0;

STORE_OUTPUT_WHILE:
        while(true) {

                if (index == NUM_OUTPUT_MATRIX * rows * rows) {
                        // DEBAYER Done (need a reset)
                        svd_done.write(true);
                        do { wait(); } while(true);
                }

                // 4-phase handshake
                do { wait(); }
                while (!process_done.read());
                output_start.write(true);
                do { wait(); }
                while (process_done.read());
                output_start.write(false);

                /* S matrix */
                // Send DMA request
                wr_index.write(index);
                wr_length.write(length);
```

```
                index += length;

                wr_request.write(true);
                do { wait(); } while(!wr_grant.read());
                wr_request.write(false);
                do { wait(); } while(wr_grant.read());

OUTPUT_S_OUTER:
                for (int i = 0; i < MAX_SIZE; ++i) {
                        if (i == rows) break;
OUTPUT_S_INNER:
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                if (j == rows) break;
                                SVD_CELL_TYPE cell = s[i * size + j];
                                wait();
                                bufdout.put(cell);
#ifdef VERBOSE
                                cout << "DUT PUT: (index, length, i, j, val)" << index <
                                        << " " << i << " " << j << " " << cell << endl;
#endif
                                wait();
                        }
                }

                /* U matrix */
                // Send DMA request
                wr_index.write(index);
                wr_length.write(length);
                index += length;

                wr_request.write(true);
                do { wait(); } while(!wr_grant.read());
                wr_request.write(false);
                do { wait(); } while(wr_grant.read());

OUTPUT_U_OUTER:
                for (int i = 0; i < MAX_SIZE; ++i) {
                        if (i == rows) break;
OUTPUT_U_INNER:
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                if (j == rows) break;
                                SVD_CELL_TYPE cell = u[i * size + j];
                                bufdout.put(cell);
#ifdef VERBOSE
                                cout << "DUT PUT: (index, length, i, j, val)" << index <
                                        << " " << i << " " << j << " " << cell << endl;
```

```
#endif
                                        wait();
                                }
                        }


                        /* V matrix */
                        // Send DMA request
                        wr_index.write(index);
                        wr_length.write(length);
                        index += length;

                        wr_request.write(true);
                        do { wait(); } while(!wr_grant.read());
                        wr_request.write(false);
                        do { wait(); } while(wr_grant.read());

OUTPUT_V_OUTER:
                        for (int i = 0; i < MAX_SIZE; ++i) {
                                if (i == rows) break;
OUTPUT_V_INNER:
                                for (int j = 0; j < MAX_SIZE; ++j) {
                                        if (j == rows) break;
                                        SVD_CELL_TYPE cell = v[i * size + j];
                                        bufdout.put(cell);
#ifdef VERBOSE
                                        cout << "DUT PUT: (index, length, i, j, val)" << index <
                                                << " " << i << " " << j << " " << cell << endl;
#endif
                                        wait();
                                }
                        }
                }
}

void svd::process_svd()
{
        // Reset
        process_start.write(false);
        process_done.write(false);

        // Wait for configuratin
        do { wait(); } while (!init_done.read());

        const int rows = size.read();
        int svd_row = 0;
```

```
DEBAYER_WHILE:
        while (true) {

                if (svd_row == rows)
                        // Wait for reset
                        do { wait(); } while(true);

                // 4-phase handshake
                do { wait(); }
                while (!input_done.read());
                process_start.write(true);
                do { wait(); }
                while (input_done.read());
                process_start.write(false);

                for (int i = 0; i < MAX_SIZE; ++i) {
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                SVD_CELL_TYPE tmp;
                                tmp = matrix_in[i * rows + j];
                                wait();
                                s[i * rows + j] = tmp;
                                wait();
                        }
                }

                jacobi(s, rows, s, u, v);

                // 4-phase handshake
                process_done.write(true);
                do { wait(); }
                while (!output_start.read());
                process_done.write(false);
                do { wait(); }
                while (output_start.read());

                svd_row += rows;
        }
}
#ifdef __CTOS__
SC_MODULE_EXPORT(svd)
#endif
#ifndef _MYDATA_H_
#define _MYDATA_H_

#define SC_INCLUDE_FX
```

```
#include <systemc.h>

//#define REAL_FLOAT
//#define SC_FIXED_POINT_FAST
//#define SC_FIXED_POINT
#define CTOS_SC_FIXED_POINT

#ifdef __CTOS__
#define MAX_SIZE     64
#else
#define MAX_SIZE     20
#endif

#define MAX_ITERATIONS  1000000

#if defined(REAL_FLOAT)
#define SVD_CELL_TYPE double
#elif defined(SC_FIXED_POINT_FAST)
#define SC_FIXED_TYPE sc_dt::sc_fixed_fast
#elif defined(SC_FIXED_POINT)
#define SC_FIXED_TYPE sc_dt::sc_fixed
#elif defined(CTOS_SC_FIXED_POINT)
#include <ctos_fx.h>
#define SC_FIXED_TYPE ctos_sc_dt::sc_fixed
#else
#error "specify fixed or floating point type"
#endif

#ifndef REAL_FLOAT
#define WL  64
#define IWL 24
#define SVD_CELL_TYPE SC_FIXED_TYPE<WL,IWL>
#endif

#define SVD_INPUT_SIZE(__sz) (__sz * __sz)
#define SVD_OUTPUT_SIZE(__sz) (3 * __sz * __sz)
#define SVD_GET_S(__ptr, __sz) (__ptr)
#define SVD_GET_U(__ptr, __sz) (__ptr + __sz * __sz)
#define SVD_GET_V(__ptr, __sz) (__ptr + 2 * __sz * __sz)

#endif
#ifndef _SVD_H_
#define _SVD_H_

#define SC_INCLUDE_FX
#include "systemc.h"
```

```
#include <ctos_flex_channels.h>
//#include <flex_channels.hpp>

#include "svd_data.h"

#define NUM_OUTPUT_MATRIX 3

#define SVD_PRECISION (0.000000000001)
#define MIN_MOVEMENT  (0.000000000001)
#define CORDIC_ITER 80

//#define VERBOSE

typedef struct {
        SVD_CELL_TYPE value;
        char rowNum;
        char colNum;
} LargestElement;

SC_MODULE(svd) {
        sc_in<bool> clk;
        sc_in<bool> rst;

        // DMA requests interface from memory to device
        sc_out<unsigned> rd_index;   // array index (offset from base address)
        sc_out<unsigned> rd_length;  // burst size
        sc_out<bool>     rd_request; // transaction request
        sc_in<bool>      rd_grant;   // transaction grant

        // DMA requests interface from device to memory
        sc_out<unsigned> wr_index;   // array index (offset from base address)
        sc_out<unsigned> wr_length;  // burst size
        sc_out<bool>     wr_request; // transaction request
        sc_in<bool>      wr_grant;   // transaction grant

        // input data read by load_input
        get_initiator<SVD_CELL_TYPE> bufdin;
        // output data written by store output
        put_initiator<SVD_CELL_TYPE> bufdout;

        sc_in<unsigned> conf_size;
        sc_in<bool>     conf_done;

        // computation complete
        sc_out<bool>    svd_done;
```

```
void config_svd(void);
void load_input(void);
void store_output(void);
void process_svd(void);

SC_CTOR(svd) {
        SC_CTHREAD(config_svd, clk.pos());
        reset_signal_is(rst, false);
        SC_CTHREAD(load_input, clk.pos());
        reset_signal_is(rst, false);
        SC_CTHREAD(store_output, clk.pos());
        reset_signal_is(rst, false);
        SC_CTHREAD(process_svd, clk.pos());
        reset_signal_is(rst, false);

        bufdin.clk_rst(clk,rst);
        bufdout.clk_rst(clk,rst);
}

private:
//Written by config_debayer
sc_signal<unsigned> size;
sc_signal<bool>     init_done;

//Written by load_input
sc_signal<bool>     input_done;

//Written by process_debayer
sc_signal<bool>     process_start;
sc_signal<bool>     process_done;

//Written by store_output
sc_signal<bool>     output_start;

/* SVD functions */
void multiply (SVD_CELL_TYPE *left, SVD_CELL_TYPE *right, SVD_CELL_TYPE *result,
void jacobi (SVD_CELL_TYPE *a, int n, SVD_CELL_TYPE *s, SVD_CELL_TYPE *u, SVD_CE
LargestElement findLargestElement (SVD_CELL_TYPE *matrix, int dimension, int *a1
void copyMatrix (SVD_CELL_TYPE *a, SVD_CELL_TYPE *b, int dimension);
void identify (SVD_CELL_TYPE *matrix, int dimension);
void rotate (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *v
void transpose (SVD_CELL_TYPE *matrix, int dimension);
void reorder (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *
void swapRows (SVD_CELL_TYPE *m, int a, int b, int dimension);

/* scratchpad matrices */
```

```
        SVD_CELL_TYPE Ui[MAX_SIZE * MAX_SIZE];
        //SVD_CELL_TYPE Vi[MAX_SIZE * MAX_SIZE];
        //SVD_CELL_TYPE tempResult[MAX_SIZE * MAX_SIZE];
        SVD_CELL_TYPE tempMatrix[MAX_SIZE * MAX_SIZE];
        //SVD_CELL_TYPE p[MAX_SIZE * MAX_SIZE];
        //LargestElement leArray[MAX_SIZE];

        SVD_CELL_TYPE matrix_in[MAX_SIZE * MAX_SIZE];

        SVD_CELL_TYPE s[MAX_SIZE * MAX_SIZE];
        SVD_CELL_TYPE u[MAX_SIZE * MAX_SIZE];
        SVD_CELL_TYPE v[MAX_SIZE * MAX_SIZE];
};

#endif
extern "C" {
#include <sys/syscall.h>
#include <sys/types.h>
#include <csp/csp.h>

#include "svd-sync.h"
#include "sync.h"
}

#include <flex_channels.hpp>
#include "svd-wrapper.hpp"
#include "svd.h"

#include <sc_dma_controller.hpp>

void svd_main(struct device *dev) {
        struct svd_sync *svd_dev = dev_to_svd(dev);
        sc_clock clk("clk", 1, SC_NS);
        sc_signal<bool> rst("rst");
        sc_signal<bool> rst_dut("rst_dut");

        sc_signal<unsigned>  rd_index;
        sc_signal<unsigned>  rd_length;
        sc_signal<bool>      rd_request;
        sc_signal<bool>      rd_grant;

        sc_signal<unsigned>  wr_index;
        sc_signal<unsigned>  wr_length;
        sc_signal<bool>      wr_request;
        sc_signal<bool>      wr_grant;
```

```
/* DMA */
put_get_channel<unsigned long long> dma_phys_addr;
put_get_channel<unsigned long> dma_len;
put_get_channel<bool> dma_write;
put_get_channel<bool> dma_start;

/* Debayer */
put_get_channel<SVD_CELL_TYPE>  bufdin;
put_get_channel<SVD_CELL_TYPE>  bufdout;
sc_signal<unsigned>  conf_size;
sc_signal<bool>      conf_done;

/* computation complete */
sc_signal<bool> svd_done;


/* modules! */
svd_wrapper wrapper("wrapper", svd_dev);
svd dut("dut");
sc_dma_controller<SVD_CELL_TYPE> dma("dma_controller", &dev->dma_cont);
int budget, budget_on_loan = 0;

sc_report_handler::set_actions("/IEEE_Std_1666/deprecated", SC_DO_NOTHING);


/* wire it up */
dut.clk(clk);
dut.rst(rst_dut);
dut.rd_index(rd_index);
dut.rd_length(rd_length);
dut.rd_request(rd_request);
dut.rd_grant(rd_grant);
dut.wr_index(wr_index);
dut.wr_length(wr_length);
dut.wr_request(wr_request);
dut.wr_grant(wr_grant);
dut.bufdout(bufdout);
dut.bufdin(bufdin);
dut.conf_size(conf_size);
dut.conf_done(conf_done);
dut.svd_done(svd_done);

wrapper.clk(clk);
wrapper.rst(rst);
wrapper.rst_dut(rst_dut);
wrapper.rd_index(rd_index);
```

```
        wrapper.rd_length(rd_length);
        wrapper.rd_request(rd_request);
        wrapper.rd_grant(rd_grant);
        wrapper.wr_index(wr_index);
        wrapper.wr_length(wr_length);
        wrapper.wr_request(wr_request);
        wrapper.wr_grant(wr_grant);
        wrapper.conf_size(conf_size);
        wrapper.conf_done(conf_done);
        wrapper.svd_done(svd_done);
        wrapper.out_phys_addr(dma_phys_addr);
        wrapper.out_len(dma_len);
        wrapper.out_write(dma_write);
        wrapper.out_start(dma_start);

        dma.clk(clk);
        dma.rst(rst);
        dma.in_phys_addr(dma_phys_addr);
        dma.in_len(dma_len);
        dma.in_write(dma_write);
        dma.in_start(dma_start);
        dma.in_data(bufdout);
        dma.out_data(bufdin);

        /* simulation time */
        rst.write(false);
        sc_start(10, SC_NS);
        rst.write(true);

        for (;;) {
                if (unlikely(csp_recv_int(dev->sync, &budget)))
                        die_errno("%s: recv, __func__");
                budget = sync_budget_update(budget, &budget_on_loan);
                if (!budget) continue;
                sc_start(budget, SC_NS);
        }
}

#include "device-list.h"
#include "alloc.h"

#include "svd-sync.h"

static void svd_release(struct object *obj) {
        struct svd_sync *svd = obj_to_svd(obj);
```

```
        free(svd);
}

static int svd_create(const struct device_desc *desc, const char *name) {
        struct svd_sync *svd;

        svd = cargo_zalloc(sizeof(*svd));
        if (svd == NULL) {
                return -1;
        }

        svd->dev.obj.release = svd_release;
        svd->dev.obj.name = name;
        svd->dev.id = desc->id;
        svd->dev.length = SVD_NR_REGS * sizeof(u32);

        if (device_sync_register(&svd->dev, svd_main)) {
                free(svd);
                return -1;
        }

        return 0;
}

const struct device_init_operations svd_sync_init_ops = {
        .create = svd_create,
};

#ifndef SVD_SYNC_H
#define SVD_SYNC_H

#include "device.h"

enum svd_regs {
        SVD_REG_CMD,            /* Command and status;
                                0x1 to start computation; 0x0 to reset */
        SVD_REG_SRC,
        SVD_REG_DST,
        SVD_REG_SIZE,           /* Number of elements per sample */
        SVD_REG_MAX_SIZE,       /* READ-ONLY max allowed size */
        SVD_REG_ID,             /* Device ID from OS */
        SVD_NR_REGS,
};

/* status bits in SVD_REG_CMD */
#define STATUS_DONE BIT(5);
```

```
#define STATUS_RUN  BIT(4);

struct svd_sync {
        struct device dev;
};

void svd_main(struct device *dev);

static inline struct svd_sync *dev_to_svd(struct device *dev) {
        return container_of(dev, struct svd_sync, dev);
}

static inline struct svd_sync *obj_to_svd(struct object *obj) {
        struct device *dev = obj_to_device(obj);

        return dev_to_svd(dev);
}

#endif /* SVD_SYNC_H */
extern "C" {
#include <csp/csp.h>

#include "util.h"
#include "io.h"
}

#include "svd-wrapper.hpp"

void svd_wrapper::ioread32(struct io_req *req, struct io_rsp *rsp) {
        int reg = req->local_offset >> 2;

                obj_dbg(&svd->dev.obj, "%s: size %d offset 0x%x\n", __func__, req->size,

        switch (reg) {
                case SVD_REG_CMD:
                        rsp->val = status_reg;
                        break;
                case SVD_REG_SRC:
                        rsp->val = dma_phys_addr_src;
                        break;
                case SVD_REG_DST:
                        rsp->val = dma_phys_addr_dst;
                        break;
                case SVD_REG_SIZE:
                        rsp->val = conf_size.read();
                        break;
```

```
                case SVD_REG_MAX_SIZE:
                        rsp->val = MAX_SIZE;
                        break;
                case SVD_REG_ID:
                        rsp->val = svd->dev.id;
                        break;
                default:
                        BUG();
        }
}

void svd_wrapper::iowrite32(const struct io_req *req, struct io_rsp *rsp)
{
        int reg = req->local_offset >> 2;

                obj_dbg(&svd->dev.obj, "%s: size %d offset 0x%x val 0x%x\n",
                                                __func__, req->size, req->local_offset,

        rsp->val = req->val;

        switch (reg) {
                case SVD_REG_CMD:
        if (req->val == 1) {
        BUG_ON((status_reg != 0));
                                conf_done.write(true);
                                start_fifo.put(true);
                        }
                        else if (req->val != 0) {
                        BUG();
                        }
                        status_reg = req->val;
                        break;
                case SVD_REG_SRC:
                        dma_phys_addr_src = req->val;
                        break;
                case SVD_REG_DST:
                        dma_phys_addr_dst = req->val;
                        break;
                case SVD_REG_SIZE:
                        conf_size.write(req->val);
                        break;
                default:
                        BUG();
        }
}
```

```
void svd_wrapper::copy_from_dram(u64 index, unsigned length) {
        obj_dbg(&svd->dev.obj, "%s\n", __func__);

        /* byte address */
        out_phys_addr.put(dma_phys_addr_src +
                        (index * sizeof(SVD_CELL_TYPE /* DMA token */)));
        out_len.put(length);
        out_write.put(false);
        out_start.put(true);
}

void svd_wrapper::copy_to_dram(u64 index, unsigned length) {
        obj_dbg(&svd->dev.obj, "%s\n", __func__);

        out_phys_addr.put(dma_phys_addr_dst +
                        (index * sizeof(SVD_CELL_TYPE /* DMA token */)));
        out_len.put(length);
        out_write.put(true);
        out_start.put(true);
}

void svd_wrapper::start() {
        rst_dut.write(false);
        wait();
        rst_dut.write(true);

        for (;;) {
                start_fifo.get();
                obj_dbg(&svd->dev.obj, "CTL start\n");
                drive();
                obj_dbg(&svd->dev.obj, "SVD done\n");
        }
}

void svd_wrapper::drive()
{
        for (;;) {
                do {
                        wait();
                } while (!rd_request.read() && !wr_request.read() && !svd_done.read());

                if (svd_done.read()) {
                        rst_dut.write(false);
                        wait();
                        rst_dut.write(true);
                        // Set bits 5:4 to "10" -> accelerator done
```

```
                status_reg &= ~STATUS_RUN;
                        status_reg |= STATUS_DONE;
                        device_sync_irq_raise(&svd->dev);
                        break;
                }
                if (rd_request.read()) {
                        unsigned index = rd_index.read();
                        unsigned length = rd_length.read();

                        rd_tran_cnt++;
                        rd_byte += length * sizeof(SVD_CELL_TYPE);

                        rd_grant.write(true);

                        do { wait(); }
                        while (rd_request.read());
                        rd_grant.write(false);
                        wait();

                        copy_from_dram((u64) index, length);

                } else {
                        // WRITE REQUEST
                        unsigned index = wr_index.read();
                        unsigned length = wr_length.read();
                        wr_tran_cnt++;
                        wr_byte += length * sizeof(SVD_CELL_TYPE);

                        wr_grant.write(true);

                        do { wait(); }
                        while (wr_request.read());
                        wr_grant.write(false);
                        wait();

                        copy_to_dram((u64) index, length);
                }
        }
}

void svd_wrapper::io()
{
        struct io_req req;
        struct io_rsp rsp;

        for (;;) {
```

```
                /*
                 * Most of the time the channel will be empty; we speed things
                 * up by just peeking directly at the queue, avoiding syscalls.
                 * We do this every 10 cycles to simulate faster.
                 */
                wait(10);
                if (csp_channel_is_empty(svd->dev.io_req))
                        continue;

                if (unlikely(io_recv_req(svd->dev.io_req, &req)))
                        die_errno(__func__);

                //BUG_ON(req.size != 4); /* XXX */

                rsp.local_offset = req.local_offset;
                rsp.size = req.size;

                if (req.write){
                        iowrite32(&req, &rsp);
                } else {
                        ioread32(&req, &rsp);
                }
                if (unlikely(io_send_rsp(req.rsp_chan, &rsp)))
                        die_errno(__func__);
        }
}

#ifndef SVD_WRAPPER_HPP
#define SVD_WRAPPER_HPP

extern "C" {
#include "svd-sync.h"
}

#include <flex_channels.hpp>

#include "svd.h"

SC_MODULE(svd_wrapper) {
        sc_in<bool> clk;
        sc_in<bool> rst;
        sc_out<bool> rst_dut;

        /* DMA requests interface */
        sc_in<unsigned>  rd_index;
        sc_in<unsigned>  rd_length;
```

```
        sc_in<bool>       rd_request;
        sc_out<bool>      rd_grant;

        sc_in<unsigned>   wr_index;
        sc_in<unsigned>   wr_length;
        sc_in<bool>       wr_request;
        sc_out<bool>      wr_grant;

        /* DMA */
        put_initiator<unsigned long long> out_phys_addr;
        put_initiator<unsigned long>      out_len;
        put_initiator<bool>               out_write;
        put_initiator<bool>               out_start;

        sc_out<unsigned>  conf_size;
        sc_out<bool>      conf_done;

        /* computation complete */
        sc_in<bool> svd_done;

        void iowrite32(const struct io_req *req, struct io_rsp *rsp);
        void ioread32(struct io_req *req, struct io_rsp *rsp);

        void drive();
        void copy_from_dram(u64 index, unsigned length);
        void copy_to_dram(u64 index, unsigned length);

        void io();
        void start();

        typedef svd_wrapper SC_CURRENT_USER_MODULE;
        svd_wrapper(sc_core::sc_module_name, struct svd_sync *svd_) {
                SC_CTHREAD(io, clk.pos());
                reset_signal_is(rst, false);

                SC_CTHREAD(start, clk.pos());
                reset_signal_is(rst, false);

                svd = svd_;
        }

private:
        struct svd_sync *svd; /* driver interface */

        u32 dma_phys_addr_src;
        u32 dma_phys_addr_dst;
```

```
        size_t dma_size_src;
        size_t dma_size_dst;

        u32 status_reg; /* [0] go command; [4-5] specified in svd-sync.h */

        tlm_fifo<bool> start_fifo; /* handshake btwn driver and device */

        unsigned rd_tran_cnt;
        unsigned long long rd_byte;
        unsigned wr_tran_cnt;
        unsigned long long wr_byte;
};

#endif /* SVD_WRAPPER_HPP */
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "fifo.h"

#define DRIVER_NAME "fifo0"

/*
 * Information about our device
 */
struct vga_led_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        u8 segments[VGA_LED_DIGITS];
} dev;

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_led_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
```

```
{
        int value;

        switch (cmd) {
        case VGA_LED_WRITE_DIGIT:
                if (copy_from_user(&value, (int *) arg,
                                   sizeof(int)))
                        return -EACCES;

                iowrite32(value, dev.virtbase);
                break;

        case VGA_LED_READ_DIGIT:
                value = ioread32(dev.virtbase);
                if (copy_to_user( (int *) arg, &value,
                                  sizeof(int)) )
                        return -EACCES;
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_led_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = vga_led_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_led_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &vga_led_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_led_probe(struct platform_device *pdev)
{
        static unsigned char welcome_message[VGA_LED_DIGITS] = {
```

```
                0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00};
        int i, ret;

        pr_info(DRIVER_NAME ": probe\n");

        /* Register ourselves as a misc device: creates /dev/vga_led */
        ret = misc_register(&vga_led_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        printk("DEVICE RESOURCE START %x END %x \n",dev.res.start,dev.res.end);

        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
                goto out_release_mem_region;
        }

        printk("VIRTUAL ADDRESS OF RESOURCE: %x \n",dev.virtbase);

        return 0;

out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&vga_led_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int vga_led_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
```

```
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&vga_led_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_led_of_match[] = {
        { .compatible = "altr,fifo0" },
        {},
};
MODULE_DEVICE_TABLE(of, vga_led_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_led_driver = {
        .driver = {
                .name   = DRIVER_NAME,
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(vga_led_of_match),
        },
        .remove = __exit_p(vga_led_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_led_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&vga_led_driver, vga_led_probe);
}

/* Called when the module is unloaded: release resources */
static void __exit vga_led_exit(void)
{
        platform_driver_unregister(&vga_led_driver);
        pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Chae Jubb");
MODULE_DESCRIPTION("FIFO0: WRITE END");
#include <linux/module.h>
#include <linux/init.h>
```

```
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "fifo.h"

#define DRIVER_NAME "fifo1"

/*
 * Information about our device
 */
struct vga_led_dev {
        struct resource res; /* Resource: our registers */
        void __iomem *virtbase; /* Where registers can be accessed in memory */
        u8 segments[VGA_LED_DIGITS];
} dev;

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_led_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
        int value;

        switch (cmd) {
        case VGA_LED_WRITE_DIGIT:
                if (copy_from_user(&value, (int *) arg,
                                   sizeof(int)))
                        return -EACCES;

                iowrite32(value, dev.virtbase);
                break;

        case VGA_LED_READ_DIGIT:
                value = ioread32(dev.virtbase);
                if (copy_to_user( (int *) arg, &value,
                                  sizeof(int)) )
```

```
                        return -EACCES;
                break;

        default:
                return -EINVAL;
        }

        return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_led_fops = {
        .owner          = THIS_MODULE,
        .unlocked_ioctl = vga_led_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_led_misc_device = {
        .minor          = MISC_DYNAMIC_MINOR,
        .name           = DRIVER_NAME,
        .fops           = &vga_led_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_led_probe(struct platform_device *pdev)
{
        static unsigned char welcome_message[VGA_LED_DIGITS] = {
                0x3E, 0x7D, 0x77, 0x08, 0x38, 0x79, 0x5E, 0x00};
        int i, ret;

        pr_info(DRIVER_NAME ": probe\n");

        /* Register ourselves as a misc device: creates /dev/vga_led */
        ret = misc_register(&vga_led_misc_device);

        /* Get the address of our registers from the device tree */
        ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
        if (ret) {
                ret = -ENOENT;
                goto out_deregister;
        }

        printk("DEVICE RESOURCE START %x END %x \n",dev.res.start,dev.res.end);
```

```
        /* Make sure we can use these registers */
        if (request_mem_region(dev.res.start, resource_size(&dev.res),
                               DRIVER_NAME) == NULL) {
                ret = -EBUSY;
                goto out_deregister;
        }

        /* Arrange access to our registers */
        dev.virtbase = of_iomap(pdev->dev.of_node, 0);
        if (dev.virtbase == NULL) {
                ret = -ENOMEM;
                goto out_release_mem_region;
        }

        printk("VIRTUAL ADDRESS OF RESOURCE: %x \n",dev.virtbase);

        /* Display a welcome message */
        /* for (i = 0; i < VGA_LED_DIGITS; i++) */
        /*       write_digit(i, welcome_message[i]); */

        return 0;

out_release_mem_region:
        release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
        misc_deregister(&vga_led_misc_device);
        return ret;
}

/* Clean-up code: release resources */
static int vga_led_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&vga_led_misc_device);
        return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_led_of_match[] = {
        { .compatible = "altr,fifo1" },
        {},
};
MODULE_DEVICE_TABLE(of, vga_led_of_match);
```

```
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_led_driver = {
        .driver = {
                .name   = DRIVER_NAME,
                .owner  = THIS_MODULE,
                .of_match_table = of_match_ptr(vga_led_of_match),
        },
        .remove = __exit_p(vga_led_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_led_init(void)
{
        pr_info(DRIVER_NAME ": init\n");
        return platform_driver_probe(&vga_led_driver, vga_led_probe);
}

/* Called when the module is unloaded: release resources */
static void __exit vga_led_exit(void)
{
        platform_driver_unregister(&vga_led_driver);
        pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_led_init);
module_exit(vga_led_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Chae Jubb");
MODULE_DESCRIPTION("FIFO1: READ END");
#ifndef _VGA_LED_H
#define _VGA_LED_H

#include <linux/ioctl.h>

#define VGA_LED_DIGITS 8

typedef struct {
  unsigned char digit;    /* 0, 1, .. , VGA_LED_DIGITS - 1 */
  unsigned char segments; /* LSB is segment a, MSB is decimal point */
} vga_led_arg_t;

#define VGA_LED_MAGIC 'q'
```

```
/* ioctls and their arguments */
#define VGA_LED_WRITE_DIGIT _IOW(VGA_LED_MAGIC, 1, int *)
#define VGA_LED_READ_DIGIT  _IOR(VGA_LED_MAGIC, 2, int *)

#endif
#include <stdio.h>
#include "fifo.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdint.h>

//#define CORRECTNESS_CHECK

int fifo0_fd;
int fifo1_fd;

#define MAX 8388608

void double_to_bv(double d, uint64_t* fixed) {
        double sub = MAX;
        *fixed = 0;
        int i;
        for (i = 63; i >= 0; i--) {
                if (d > sub) {
                        //set the bit and substract the value from sub
                        d -= sub;
                        *fixed |= ( 1UL << i);
                }
                sub /= 2.0;
        }
}

void convert_to_bv(double* fixed) {
        double d = *fixed;
        double_to_bv(d, (uint64_t *) fixed);
}

void bv_to_double(uint64_t fixed, double* d) {
        double sub = MAX;
        *d = 0.0;
        int i;
```

```
        for (i = 63; i >= 0; i--) {
                if ((fixed & (1UL << i))) {
                        *d += sub;
                }
                sub /= 2.0;
        }
}


void convert_to_double(uint64_t* d) {
        uint64_t fixed  = *d;
        bv_to_double(fixed, (double*) d);
}

#define SIZE (2*64*64)
void check_correct() {
        double* test = (double*) malloc(SIZE/2 * sizeof(double));

        uint64_t in, out;
        int input;
        double res;
        int j;
        srand(NULL); //lol time on fpga board
        for (j = 0; j < SIZE/2; j++) {
                test[j] = (double) rand() / 1234543.424;
        }
        for (j = 0; j < SIZE/2; j++) {
                double_to_bv(test[j], &in);

                //should i give the high order bits first?
                input = (int) (in >> 32);
                if (ioctl(fifo0_fd, VGA_LED_WRITE_DIGIT, &input)) {
                        perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
                        return;
                }
                usleep(4000);
                if (j % 64 == 0)
                        printf("%d inserting: %d\n",j,input);

                input = (int) in;
                if (ioctl(fifo0_fd, VGA_LED_WRITE_DIGIT, &input)) {
                        perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
                        return;
                }
                usleep(4000);
                if (j % 64 == 0)
```

```
                        printf("%d inserting: %d\n",j,input);

        }
        printf("finished writing j: %d\n", j);
        sleep(2);
        printf("start reading\n");
        sleep(1);

        int res1, res2;
        for(j = 0; j < 3 * SIZE/2; j++) {
                if (ioctl(fifo1_fd, VGA_LED_READ_DIGIT, &res1)) {
                        perror("ioctl(VGA_LED_READ_DIGIT) failed");
                        return;
                }
                usleep(4000);

                if (ioctl(fifo1_fd, VGA_LED_READ_DIGIT, &res2)) {
                        perror("ioctl(VGA_LED_READ_DIGIT) failed");
                        return;
                }
                usleep(4000);
                out = res1;
                out <<= 32;
                out += res2; //can /should i use an or here

                bv_to_double(out, &res);

                printf("Matrix %d: val %f bv %lu\n", j/(SIZE/2), res, out);
        }
}


int main()
{
        int value_in,value_out;

        static const char filename0[] = "/dev/fifo0";
        static const char filename1[] = "/dev/fifo1";

        printf("FIFO Userspace program started\n");

        if ( (fifo0_fd = open(filename0, O_RDWR)) == -1) {
                fprintf(stderr, "could not open %s\n", filename0);
                return -1;
        }
```

```
        if ( (fifo1_fd = open(filename1, O_RDWR)) == -1) {
                fprintf(stderr, "could not open %s\n", filename1);
                return -1;
        }


        int i, j;
#ifndef CORRECTNESS_CHECK
        for(j = 0; j < SIZE / 2; ++j) {
                value_in = (2 * j) << 12;
                if (ioctl(fifo0_fd, VGA_LED_WRITE_DIGIT, &value_in)) {
                        perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
                        return;
                }
                if (j % 64 == 0)
                        printf("%d inserting: %d\n",j,value_in);

                value_in = (2*j) << 8;
                if (ioctl(fifo0_fd, VGA_LED_WRITE_DIGIT, &value_in)) {
                        perror("ioctl(VGA_LED_WRITE_DIGIT) failed");
                        return;
                }
                if (j % 64 == 0)
                        printf("%d inserting: %d\n",j,value_in);
        }

        printf("finished writing\n");
        sleep(2);
        printf("start reading\n");
        sleep(1);

        for(j = 0; j < 3 * SIZE; ++j) {
                if (ioctl(fifo1_fd, VGA_LED_READ_DIGIT, &value_out)) {
                        perror("ioctl(VGA_LED_READ_DIGIT) failed");
                        return;
                }
                if (j % 64 == 0)
                        printf("j: %d value_out: %d\n",j ,value_out);
        }
#else
        check_correct();
#endif
        printf("FIFO Userspace program terminating\n");
        return 0;
}
ifneq (${KERNELRELEASE},)
```

```
# KERNELRELEASE defined: we are being compiled as part of the Kernel
        obj-m := fifo0.o fifo1.o

else

# We are being compiled as a module: use the Kernel build system

        KERNEL_SOURCE := /usr/src/linux
        PWD := $(shell pwd)

default: module hello

module:
        ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

clean:
        ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
        ${RM} hello
        ${RM} *~

socfpga.dtb : socfpga.dtb
        /usr/src/linux/scripts/dtc/dtc -O dtb -o socfpga.dtb socfpga.dts

.PHONY: run
run:
        insmod fifo0.ko
        insmod fifo1.ko
        ./hello

.PHONY: all
all: module hello run



endif
/*
 * Skeleton device tree; the bare minimum needed to boot; just include and
 * add a compatible value.  The bootloader will typically populate the memory
 * node.
 */

/ {
        #address-cells = <1>;
        #size-cells = <1>;
        chosen { };
```

```
        aliases { };
        memory { device_type = "memory"; reg = <0 0>; };
};
/*
 *  Copyright (C) 2012 Altera Corporation <www.altera.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * dtc -O dtb -o socfpga.dtb socfpga.dts
 */

/dts-v1/;
/include/ "socfpga.dtsi"

/ {
        model = "Altera SOCFPGA Cyclone V";
        compatible = "altr,socfpga-cyclone5", "altr,socfpga";

        chosen {
                bootargs = "console=ttyS0,57600";
        };

        memory {
                name = "memory";
                device_type = "memory";
                reg = <0x0 0x40000000>; /* 1 GB */
        };

        aliases {
                /* this allow the ethaddr uboot environmnet variable contents
                 * to be added to the gmac1 device tree blob.
                 */
                ethernet0 = &gmac1;
        };
```

```
soc {
        clkmgr@ffd04000 {
                clocks {
                        osc1 {
                                clock-frequency = <25000000>;
                        };
                };
        };

        dcan0: d_can@ffc00000 {
                status = "disabled";
        };

        dcan1: d_can@ffc10000 {
                status = "disabled";
        };

        dwmmc0@ff704000 {
                num-slots = <1>;
                supports-highspeed;
                broken-cd;
                altr,dw-mshc-ciu-div = <4>;
                altr,dw-mshc-sdr-timing = <0 3>;

                slot@0 {
                        reg = <0>;
                        bus-width = <4>;
                };
        };

        ethernet@ff700000 {
                status = "disabled";
        };

        ethernet@ff702000 {
                phy-mode = "rgmii";
                phy-addr = <0xffffffff>; /* probe for phy addr */
        };

        i2c1: i2c@ffc05000 {
                status = "disabled";
        };

        i2c2: i2c@ffc06000 {
                status = "disabled";
        };
```

```
                i2c3: i2c@ffc07000 {
                        status = "disabled";
                };

                qspi: spi@ff705000 {
                                compatible = "cadence,qspi";
                                #address-cells = <1>;
                                #size-cells = <0>;
                                reg = <0xff705000 0x1000>,
                                        <0xffa00000 0x1000>;
                                interrupts = <0 151 4>;
                                master-ref-clk = <400000000>;
                                ext-decoder = <0>;  /* external decoder */
                                num-chipselect = <4>;
                                fifo-depth = <128>;
                                bus-num = <2>;

                                flash0: n25q00@0 {
                                        #address-cells = <1>;
                                        #size-cells = <1>;
                                        compatible = "n25q00";
                                        reg = <0>;      /* chip select */
                                        spi-max-frequency = <100000000>;
                                        page-size = <256>;
                                        block-size = <16>; /* 2^16, 64KB */
                                        quad = <1>;
/* 1-support quad */
                                        tshsl-ns = <200>;
                                        tsd2d-ns = <255>;
                                        tchsh-ns = <20>;
                                        tslch-ns = <20>;

                                        partition@0 {
                                                /* 8MB for raw data. */
                                                label = "Flash 0 Raw Data";
                                                reg = <0x0 0x800000>;
                                        };
                                        partition@800000 {
                                                /* 8MB for jffs2 data. */
                                                label = "Flash 0 jffs2 Filesystem";
                                                reg = <0x800000 0x800000>;
                                        };
                                };

                        };
```

```
sysmgr@ffd08000 {
        cpu1-start-addr = <0xffd080c4>;
};

timer0@ffc08000 {
        clock-frequency = <100000000>;
};

timer1@ffc09000 {
        clock-frequency = <100000000>;
};

timer2@ffd00000 {
        clock-frequency = <25000000>;
};

timer3@ffd01000 {
        clock-frequency = <25000000>;
};

serial0@ffc02000 {
        clock-frequency = <100000000>;
};

serial1@ffc03000 {
        clock-frequency = <100000000>;
};

usb0: usb@ffb00000 {
        status = "disabled";
};

usb1: usb@ffb40000 {
        ulpi-ddr = <0>;
};

i2c0: i2c@ffc04000 {
        speed-mode = <0>;
};

leds {
        compatible = "gpio-leds";
        hps0 {
                label = "hps_led0";
                gpios = <&gpio1 15 1>;
```

```
                        };

                        hps1 {
                                label = "hps_led1";
                                gpios = <&gpio1 14 1>;
                        };

                        hps2 {
                                label = "hps_led2";
                                gpios = <&gpio1 13 1>;
                        };

                        hps3 {
                                label = "hps_led3";
                                gpios = <&gpio1 12 1>;
                        };
                };

                lightweight_bridge: bridge@0xff200000 {
                        #address-cells = <1>;
                        #size-cells = <1>;
                        ranges = < 0x0 0xff200000 0x200000 >;

                        compatible = "simple-bus";

                        fifo_0: fifo0@0x100000008 {
                                compatible = "ALTR,fifo0";
                                reg = < 0x00000008 0x00000008 >;
                        }; //end fifo@0x100000008 (fifo_0)

                        fifo_1: fifo1@0x100000000 {
                                compatible = "ALTR,fifo1";
                                reg = < 0x00000000 0x00000008 >;
                        }; //end fifo@0x100000000 (fifo_1)

                };
        };
};

&i2c0 {
        lcd: lcd@28 {
                compatible = "newhaven,nhd-0216k3z-nsw-bbw";
                reg = <0x28>;
                height = <2>;
                width = <16>;
                brightness = <8>;
```

```
        };

        eeprom@51 {
                compatible = "atmel,24c32";
                reg = <0x51>;
                pagesize = <32>;
        };

        rtc@68 {
                compatible = "dallas,ds1339";
                reg = <0x68>;
        };
};/*
 *  Copyright (C) 2012 Altera <www.altera.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

/include/ "skeleton.dtsi"

/ {
        #address-cells = <1>;
        #size-cells = <1>;

        aliases {
                ethernet0 = &gmac0;
                ethernet1 = &gmac1;
                serial0 = &uart0;
                serial1 = &uart1;
                timer0 = &timer0;
                timer1 = &timer1;
                timer2 = &timer2;
                timer3 = &timer3;
        };
```

```
cpus {
        #address-cells = <1>;
        #size-cells = <0>;

        cpu@0 {
                compatible = "arm,cortex-a9";
                device_type = "cpu";
                reg = <0>;
                next-level-cache = <&L2>;
        };
        cpu@1 {
                compatible = "arm,cortex-a9";
                device_type = "cpu";
                reg = <1>;
                next-level-cache = <&L2>;
        };
};

intc: intc@fffed000 {
        compatible = "arm,cortex-a9-gic";
        #interrupt-cells = <3>;
        interrupt-controller;
        reg = <0xfffed000 0x1000>,
              <0xfffec100 0x100>;
};

soc {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "simple-bus";
        device_type = "soc";
        interrupt-parent = <&intc>;
        ranges;

        amba {
                compatible = "arm,amba-bus";
                #address-cells = <1>;
                #size-cells = <1>;
                ranges;

                pdma: pdma@ffe01000 {
                        compatible = "arm,pl330", "arm,primecell";
                        reg = <0xffe01000 0x1000>;
                        interrupts = <0 180 4>;
                };
        };
```

```
clkmgr@ffd04000 {
                compatible = "altr,clk-mgr";
                reg = <0xffd04000 0x1000>;

                clocks {
                        #address-cells = <1>;
                        #size-cells = <0>;

                        osc: osc1 {
                                #clock-cells = <0>;
                                compatible = "fixed-clock";
                        };

                        f2s_periph_ref_clk: f2s_periph_ref_clk {
                                #clock-cells = <0>;
                                compatible = "fixed-clock";
                                clock-frequency = <10000000>;
                        };

                        main_pll: main_pll {
                                #address-cells = <1>;
                                #size-cells = <0>;
                                #clock-cells = <0>;
                                compatible = "altr,socfpga-pll-clock";
                                clocks = <&osc>;
                                reg = <0x40>;

                                mpuclk: mpuclk {
                                        #clock-cells = <0>;
                                        compatible = "altr,socfpga-perip
                                        clocks = <&main_pll>;
                                        fixed-divider = <2>;
                                        reg = <0x48>;
                                };

                                mainclk: mainclk {
                                        #clock-cells = <0>;
                                        compatible = "altr,socfpga-perip
                                        clocks = <&main_pll>;
                                        fixed-divider = <4>;
                                        reg = <0x4C>;
                                };

                                dbg_base_clk: dbg_base_clk {
                                        #clock-cells = <0>;
```

```
                compatible = "altr,socfpga-perip
                clocks = <&main_pll>;
                fixed-divider = <4>;
                reg = <0x50>;
        };

        main_qspi_clk: main_qspi_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&main_pll>;
                reg = <0x54>;
        };

        main_nand_sdmmc_clk: main_nand_sdmmc_clk
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&main_pll>;
                reg = <0x58>;
        };

        cfg_s2f_usr0_clk: cfg_s2f_usr0_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&main_pll>;
                reg = <0x5C>;
        };
};

periph_pll: periph_pll {
        #address-cells = <1>;
        #size-cells = <0>;
        #clock-cells = <0>;
        compatible = "altr,socfpga-pll-clock";
        clocks = <&osc>;
        reg = <0x80>;

        emac0_clk: emac0_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&periph_pll>;
                reg = <0x88>;
        };

        emac1_clk: emac1_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
```

```
                clocks = <&periph_pll>;
                reg = <0x8C>;
        };

        per_qspi_clk: per_qsi_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&periph_pll>;
                reg = <0x90>;
        };

        per_nand_mmc_clk: per_nand_mmc_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&periph_pll>;
                reg = <0x94>;
        };

        per_base_clk: per_base_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&periph_pll>;
                reg = <0x98>;
        };

        s2f_usr1_clk: s2f_usr1_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&periph_pll>;
                reg = <0x9C>;
        };
};

sdram_pll: sdram_pll {
        #address-cells = <1>;
        #size-cells = <0>;
        #clock-cells = <0>;
        compatible = "altr,socfpga-pll-clock";
        clocks = <&osc>;
        reg = <0xC0>;

        ddr_dqs_clk: ddr_dqs_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-perip
                clocks = <&sdram_pll>;
                reg = <0xC8>;
```

```
                };

                ddr_2x_dqs_clk: ddr_2x_dqs_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip
                        clocks = <&sdram_pll>;
                        reg = <0xCC>;
                };

                ddr_dq_clk: ddr_dq_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip
                        clocks = <&sdram_pll>;
                        reg = <0xD0>;
                };

                s2f_usr2_clk: s2f_usr2_clk {
                        #clock-cells = <0>;
                        compatible = "altr,socfpga-perip
                        clocks = <&sdram_pll>;
                        reg = <0xD4>;
                };
        };

mpu_periph_clk: mpu_periph_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&mpuclk>;
        fixed-divider = <4>;
        };

mpu_l2_ram_clk: mpu_l2_ram_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&mpuclk>;
        fixed-divider = <2>;
        };

l4_main_clk: l4_main_clk {
        #clock-cells = <0>;
        compatible = "altr,socfpga-gate-clk";
        clocks = <&mainclk>;
        clk-gate = <0x60 0>;
        };

l3_main_clk: l3_main_clk {
```

```
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>;
                };

        l3_mp_clk: l3_mp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>;
                div-reg = <0x64 0 2>;
                clk-gate = <0x60 1>;
                };

        l3_sp_clk: l3_sp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>;
                div-reg = <0x64 2 2>;
        };

        l4_mp_clk: l4_mp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>, <&per_base_clk>;
                div-reg = <0x64 4 3>;
                clk-gate = <0x60 2>;
                };

        l4_sp_clk: l4_sp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&mainclk>, <&per_base_clk>;
                div-reg = <0x64 7 3>;
                clk-gate = <0x60 3>;
                };

        dbg_at_clk: dbg_at_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                div-reg = <0x68 0 2>;
                clk-gate = <0x60 4>;
                };

        dbg_clk: dbg_clk {
                #clock-cells = <0>;
```

```
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                div-reg = <0x68 2 2>;
                clk-gate = <0x60 5>;
                };

        dbg_trace_clk: dbg_trace_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                div-reg = <0x6C 0 3>;
                clk-gate = <0x60 6>;
                };

        dbg_timer_clk: dbg_timer_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&dbg_base_clk>;
                clk-gate = <0x60 7>;
                };

        cfg_clk: cfg_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&cfg_s2f_usr0_clk>;
                clk-gate = <0x60 8>;
                };

        s2f_user0_clk: s2f_user0_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&cfg_s2f_usr0_clk>;
                clk-gate = <0x60 9>;
                };

        emac_0_clk: emac_0_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&emac0_clk>;
                clk-gate = <0xa0 0>;
                };

        emac_1_clk: emac_1_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&emac1_clk>;
```

```
                clk-gate = <0xa0 1>;
                };

        usb_mp_clk: usb_mp_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&per_base_clk>;
                clk-gate = <0xa0 2>;
                div-reg = <0xa4 0 3>;
                };

        spi_m_clk: spi_m_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&per_base_clk>;
                clk-gate = <0xa0 3>;
                div-reg = <0xa4 3 3>;
                };

        can0_clk: can0_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&per_base_clk>;
                clk-gate = <0xa0 4>;
                div-reg = <0xa4 6 3>;
                };

        can1_clk: can1_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&per_base_clk>;
                clk-gate = <0xa0 5>;
                div-reg = <0xa4 9 3>;
                };

        gpio_db_clk: gpio_db_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&per_base_clk>;
                clk-gate = <0xa0 6>;
                div-reg = <0xa8 0 24>;
                };

        s2f_user1_clk: s2f_user1_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
```

```
                clocks = <&s2f_usr1_clk>;
                clk-gate = <0xa0 7>;
                };

        sdmmc_clk: sdmmc_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&f2s_periph_ref_clk>, <&main_nand_sdmm
                clk-gate = <0xa0 8>;
                };

        nand_x_clk: nand_x_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&f2s_periph_ref_clk>, <&main_nand_sdmm
                clk-gate = <0xa0 9>;
                };

        nand_clk: nand_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&f2s_periph_ref_clk>, <&main_nand_sdmm
                clk-gate = <0xa0 10>;
                fixed-divider = <4>;
                };

        qspi_clk: qspi_clk {
                #clock-cells = <0>;
                compatible = "altr,socfpga-gate-clk";
                clocks = <&f2s_periph_ref_clk>, <&main_qspi_clk>
                clk-gate = <0xa0 11>;
                };
        };
};

dcan0: d_can@ffc00000 {
        compatible = "bosch,d_can";
        reg = <0xffc00000 0x1000>;
        interrupts = <0 131 4>;
};

dcan1: d_can@ffc10000 {
        compatible = "bosch,d_can";
        reg = <0xffc10000 0x1000>;
        interrupts = <0 135 4>;
};
```

```
gmac0: ethernet@ff700000 {
        compatible = "altr,socfpga-stmmac", "snps,dwmac-3.70a", "snps,dw
        reg = <0xff700000 0x2000>;
        interrupts = <0 115 4>;
        interrupt-names = "macirq";
        mac-address = [00 00 00 00 00 00];/* Filled in by U-Boot */
        clocks = <&emac0_clk>;
        clock-names = "stmmaceth";
};

gmac1: ethernet@ff702000 {
        compatible = "altr,socfpga-stmmac", "snps,dwmac-3.70a", "snps,dw
        reg = <0xff702000 0x2000>;
        interrupts = <0 120 4>;
        interrupt-names = "macirq";
        mac-address = [00 00 00 00 00 00];/* Filled in by U-Boot */
        clocks = <&emac1_clk>;
        clock-names = "stmmaceth";
};

gpio0: gpio@ff708000 {
        compatible = "snps,dw-gpio";
        reg = <0xff708000 0x1000>;
        interrupts = <0 164 4>;
        width = <29>;
        virtual_irq_start = <257>;
        interrupt-controller;
        #interrupt-cells = <2>;
        gpio-controller;
        #gpio-cells = <2>;
        clocks = <&per_base_clk>;
};

gpio1: gpio@ff709000 {
        compatible = "snps,dw-gpio";
        reg = <0xff709000 0x1000>;
        interrupts = <0 165 4>;
        width = <29>;
        virtual_irq_start = <286>;
        interrupt-controller;
        #interrupt-cells = <2>;
        gpio-controller;
        #gpio-cells = <2>;
        clocks = <&per_base_clk>;
};
```

```
gpio2: gpio@ff70a000 {
        compatible = "snps,dw-gpio";
        reg = <0xff70a000 0x1000>;
        interrupts = <0 166 4>;
        width = <27>;
        virtual_irq_start = <315>;
        interrupt-controller;
        #interrupt-cells = <2>;
        gpio-controller;
        #gpio-cells = <2>;
        clocks = <&per_base_clk>;
};

hps_0_fpgamgr: fpgamgr@0xff706000 {
        compatible = "altr,fpga-mgr-1.0", "altr,fpga-mgr";
        transport = "mmio";
        reg = <0xFF706000 0x1000
                          0xFFB90000 0x1000>;
        interrupts = <0 175 4>;
};

i2c0: i2c@ffc04000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
        reg = <0xffc04000 0x1000>;
        interrupts = <0 158 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

i2c1: i2c@ffc05000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
        reg = <0xffc05000 0x1000>;
        interrupts = <0 159 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

i2c2: i2c@ffc06000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
```

```
        reg = <0xffc06000 0x1000>;
        interrupts = <0 160 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

i2c3: i2c@ffc07000 {
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "snps,designware-i2c";
        reg = <0xffc07000 0x1000>;
        interrupts = <0 161 4>;
        emptyfifo_hold_master = <1>;
        clocks = <&per_base_clk>;
};

L2: l2-cache@fffef000 {
        compatible = "arm,pl310-cache";
        reg = <0xfffef000 0x1000>;
        interrupts = <0 38 0x04>;
        cache-unified;
        cache-level = <2>;
        arm,tag-latency = <1 1 1>;
        arm,data-latency = <2 1 1>;
};

mmc: dwmmc0@ff704000 {
        compatible = "altr,socfpga-dw-mshc";
        reg = <0xff704000 0x1000>;
        interrupts = <0 139 4>;
        fifo-depth = <0x400>;
        #address-cells = <1>;
        #size-cells = <0>;
        clocks = <&l4_mp_clk>, <&sdmmc_clk>;
        clock-names = "biu", "ciu";
};

nand: nand@ff900000 {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "denali,denali-nand-dt";
        reg = <0xff900000 0x100000>, <0xffb80000 0x10000>;
        reg-names = "nand_data", "denali_reg";
        interrupts = <0 144 4>;
        dma-mask = <0xffffffff>;
        clocks = <&nand_clk>;
```

```
        };

pmu {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "arm,cortex-a9-pmu";
        interrupts = <0 176 4>, <0 177 4>;
        ranges;

        cti0: cti0@ff118000 {
                compatible = "arm,coresight-cti";
                reg = <0xff118000 0x100>;
        };

        cti1: cti1@ff119000 {
                compatible = "arm,coresight-cti";
                reg = <0xff119000 0x100>;
        };
};

rstmgr@ffd05000 {
        compatible = "altr,rst-mgr";
        reg = <0xffd05000 0x1000>;
};

spi0: spi@fff00000 {
        compatible = "snps,dw-spi-mmio";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0xfff00000 0x1000>;
        interrupts = <0 154 4>;
        num-chipselect = <4>;
        bus-num = <0>;
        tx-dma-channel = <&pdma 16>;
        rx-dma-channel = <&pdma 17>;
        clocks = <&per_base_clk>;

        spidev@0 {
                compatible = "spidev";
                reg = <0>;      /* chip select */
                spi-max-frequency = <100000000>;
                enable-dma = <1>;
        };
};

spi1: spi@fff01000 {
```

```
        compatible = "snps,dw-spi-mmio";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0xfff01000 0x1000>;
        interrupts = <0 156 4>;
        num-chipselect = <4>;
        bus-num = <1>;
        tx-dma-channel = <&pdma 20>;
        rx-dma-channel = <&pdma 21>;
        clocks = <&per_base_clk>;

        spidev@0 {
                compatible = "spidev";
                reg = <0>;
                spi-max-frequency = <100000000>;
                enable-dma = <1>;
        };
};

sysmgr@ffd08000 {
        compatible = "altr,sys-mgr";
        reg = <0xffd08000 0x4000>;
};

/* Local timer */
timer@fffec600 {
        compatible = "arm,cortex-a9-twd-timer";
        reg = <0xfffec600 0x100>;
        interrupts = <1 13 0xf04>;
};

timer0: timer0@ffc08000 {
        compatible = "snps,dw-apb-timer-sp";
        interrupts = <0 167 4>;
        reg = <0xffc08000 0x1000>;
};

timer1: timer1@ffc09000 {
        compatible = "snps,dw-apb-timer-sp";
        interrupts = <0 168 4>;
        reg = <0xffc09000 0x1000>;
};

timer2: timer2@ffd00000 {
        compatible = "snps,dw-apb-timer-osc";
        interrupts = <0 169 4>;
```

```
        reg = <0xffd00000 0x1000>;
};

timer3: timer3@ffd01000 {
        compatible = "snps,dw-apb-timer-osc";
        interrupts = <0 170 4>;
        reg = <0xffd01000 0x1000>;
};

uart0: serial0@ffc02000 {
        compatible = "snps,dw-apb-uart";
        reg = <0xffc02000 0x1000>;
        interrupts = <0 162 4>;
        reg-shift = <2>;
        reg-io-width = <4>;
};

uart1: serial1@ffc03000 {
        compatible = "snps,dw-apb-uart";
        reg = <0xffc03000 0x1000>;
        interrupts = <0 163 4>;
        reg-shift = <2>;
        reg-io-width = <4>;
};

usb0: usb@ffb00000 {
        compatible = "snps,dwc-otg";
        reg = <0xffb00000 0xffff>;
        interrupts = <0 125 4>;
        dma-mask = <0xffffffff>;
        host-rx-fifo-size = <512>;
        dev-rx-fifo-size = <512>;
        dev-nperio-tx-fifo-size = <4096>;
        dev-perio-tx-fifo-size = <512 512 512 512 512 512
                512 512 512 512 512 512 512 512 512>;
        dev-tx-fifo-size = <512 512 512 512 512 512
                512 512 512 512 512 512 512 512 512>;
};

usb1: usb@ffb40000 {
        compatible = "snps,dwc-otg";
        reg = <0xffb40000 0xffff>;
        interrupts = <0 128 4>;
        dma-mask = <0xffffffff>;
        host-rx-fifo-size = <512>;
        dev-rx-fifo-size = <512>;
```

```
                        dev-nperio-tx-fifo-size = <4096>;
                        dev-perio-tx-fifo-size = <512 512 512 512 512 512
                                512 512 512 512 512 512 512 512 512>;
                        dev-tx-fifo-size = <512 512 512 512 512 512
                                512 512 512 512 512 512 512 512 512>;
                };
        };
};
# A Tcl script for the Qsys system console

# Start Qsys, open your soc_system.qsys file, run File->System Console,
# then execute this script by selecting it with Ctrl-E

# The System Console is described in Chapter 10 of Volume III of
# the Quartus II Handbook

# Alternately,
# system-console --project_dir=. --script=syscon-test.tcl
#
# system-console --project_dir=. -cli
#   and then "source syscon-test.tcl"

# Base addresses of the peripherals: take from Qsys
set fifo0 0x8
set fifo1 0x0

puts "Started system-console-test-script"

# Using the JTAG chain, check the clock and reset"

set j [lindex [get_service_paths jtag_debug] 0]
open_service jtag_debug $j
puts "Opened jtag_debug"

puts "Checking the JTAG chain loopback: [jtag_debug_loop $j {1 2 3 4 5 6}]"
jtag_debug_reset_system $j

puts -nonewline "Sampling the clock: "
foreach i {1 1 1 1 1 1 1 1 1 1 1 1} {
    puts -nonewline [jtag_debug_sample_clock $j]
}
puts ""

puts "Checking reset state: [jtag_debug_sample_reset $j]"

close_service jtag_debug $j
```

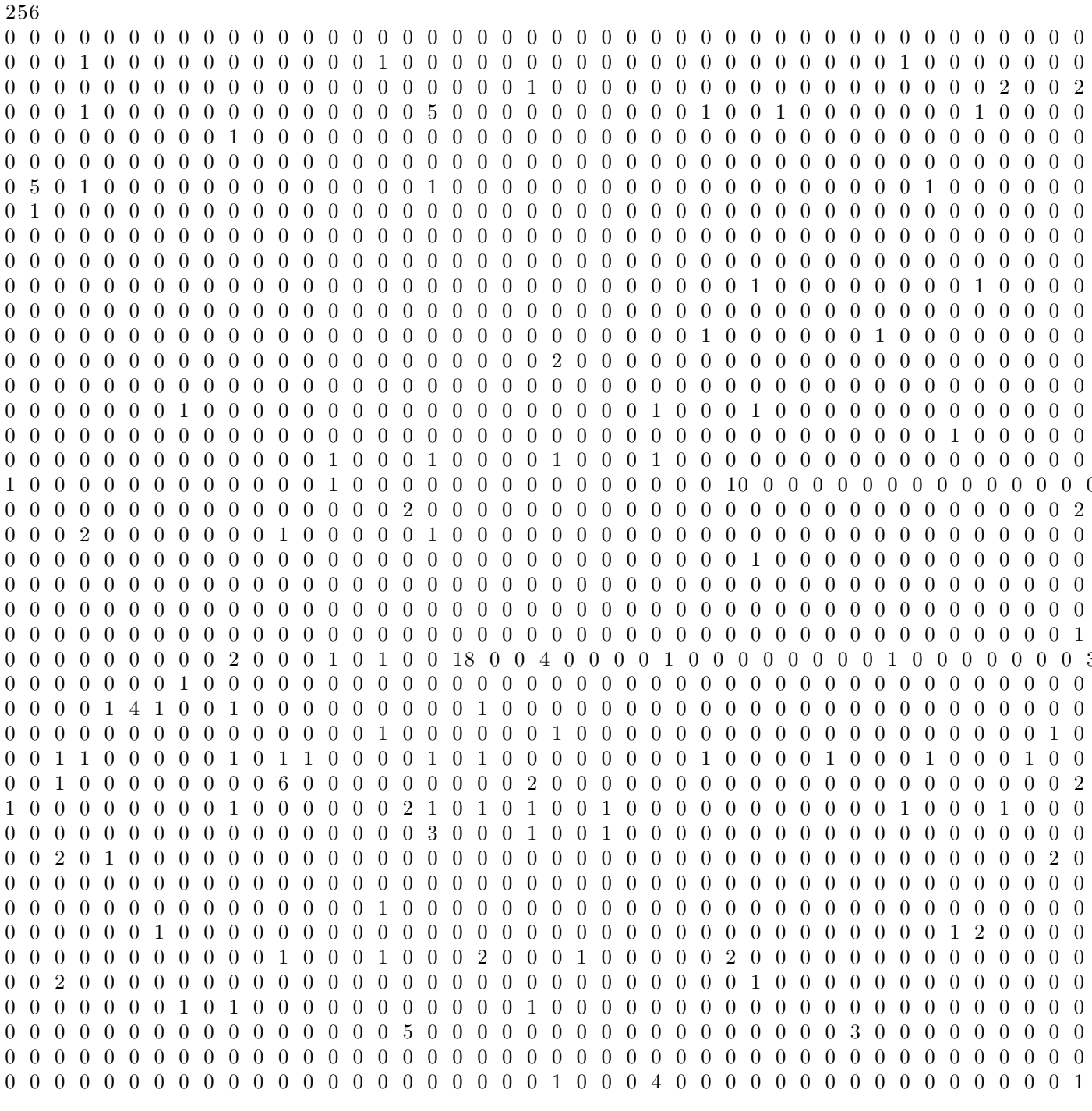
```
puts "Closed jtag_debug"

# Perform bus reads and writes

set m [lindex [get_service_paths master] 0]
open_service master $m
puts "Opened master $m"

# Write a test pattern to the input fifo
foreach {v} {0 1 2 55} {
    master_write_32 $m $fifo0 $v
        puts "wrote $v @ $fifo0"
        after 400
        set ret [master_read_32 $m $fifo1 1]
        puts "I got $ret"
}

close_service master $m
puts "Closed master"
```

```
from PIL import Image
from scipy import misc

from scipy import linalg, dot

def transform(input_matrix, dim):
    print len(input_matrix)
    u, sigma, vt = linalg.svd(input_matrix)
    for index in xrange(dim, len(input_matrix)):
        sigma[index] = 0.0
    return dot(dot(u, linalg.diagsvd(sigma, len(input_matrix), len(vt))), vt)



lena = misc.lena()
U, s, Vh = linalg.svd(lena)
compressed = transform(lena, 64)
import os
import sys

from collections import defaultdict
```

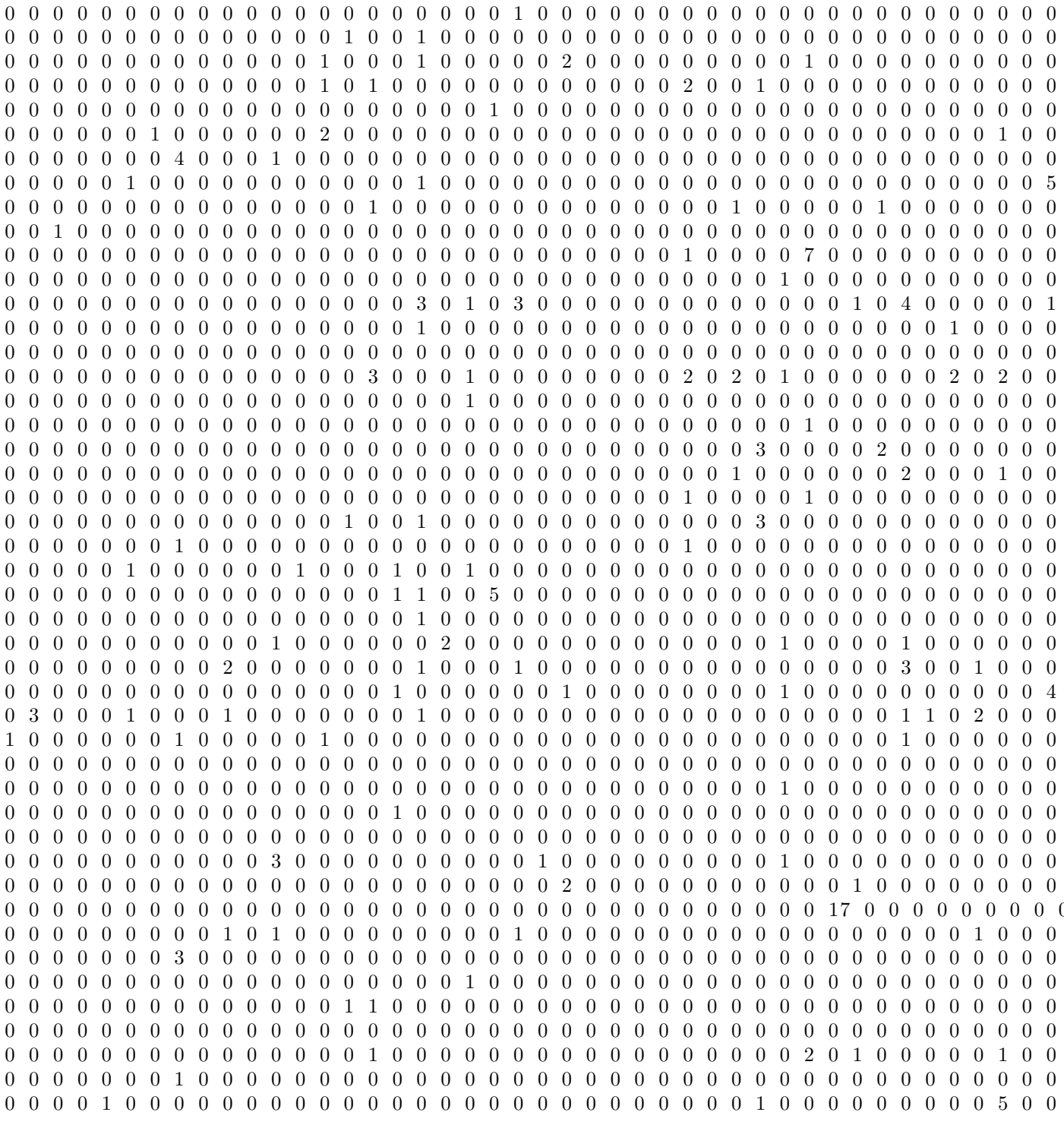
```
def _clean(string):
    """ remove any nasty grammar tokens from string """

    grammar_tokens = [".", ",", "<", ">", "?", "!", ":", ";", "\"", "(", ")", "{", "}",

    for g in grammar_tokens:
        string = string.replace(g, "")

    string = string.replace("\s+"," ")
    string = string.lower()
    return string

def resolve_simple_plurals(wset):
    rlist = []
    for w in wset:
        if w.endswith("s"):
            if w[:len(w) - 1] in wset:
                continue
        else:
            rlist.append(w)
    return set(rlist)

with open(sys.argv[1], "r") as f:
    content = f.read();
    lines = content.split("<DOC>")
    lines = [line.lstrip() for line in lines]
    docs = []
    vocab = defaultdict(int)

    for line in lines:
        doc_lines = line.split("\n")
        doc_lines = [l.lstrip() for l in doc_lines if not l.lstrip().startswith("<")]
        doc_text = " ".join(doc_lines)
        docs.append(doc_text)

        doc_text = _clean(doc_text)

        doc_words = doc_text.split(" ")

        for w in doc_words:
            vocab[w] += 1



    sorted_vocab = sorted(vocab.iteritems(), key = lambda x: x[1], reverse=True)
    sorted_vocab = [x for x in sorted_vocab if x[0].isalpha()]
```

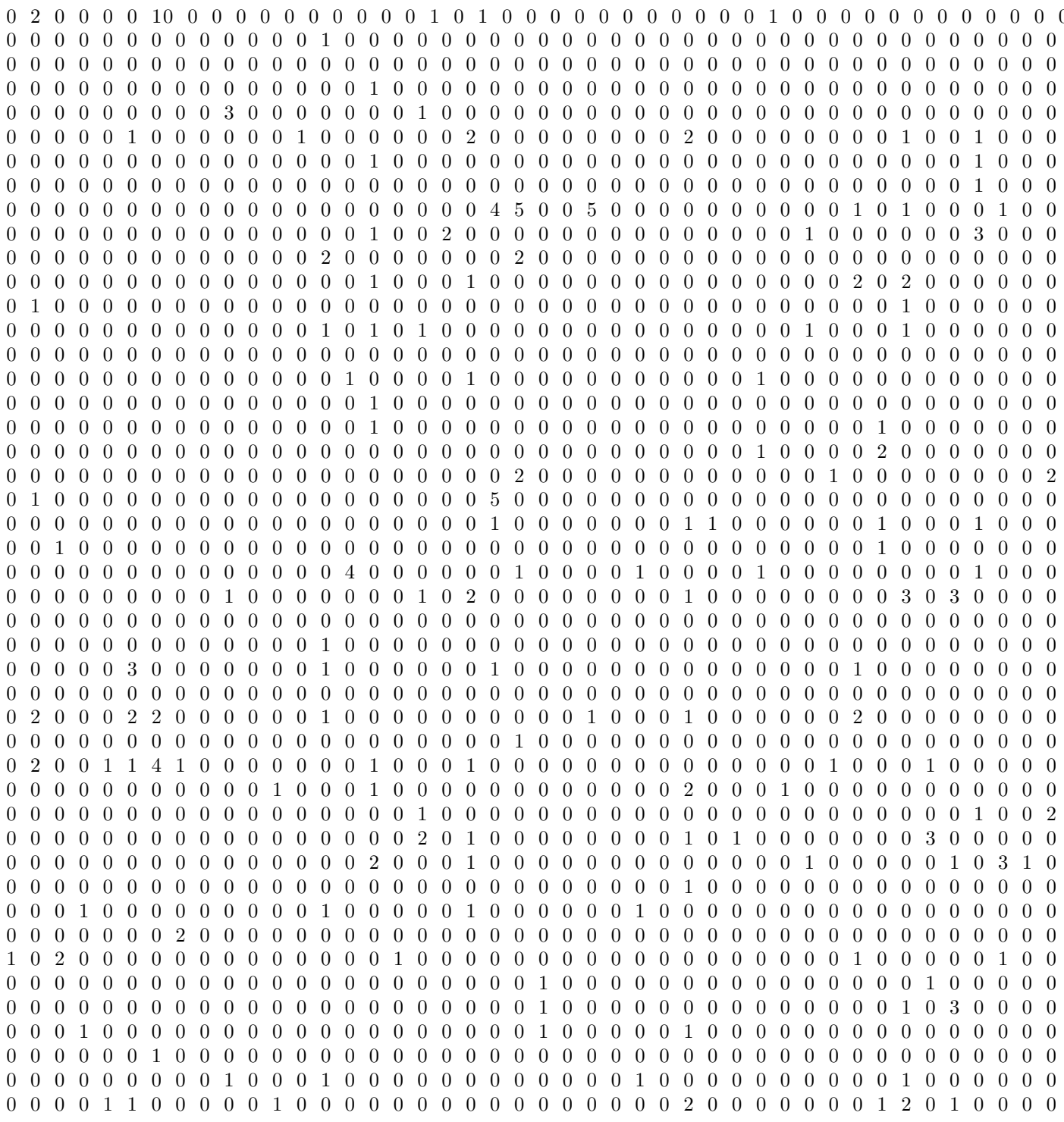
```
stop_file = open("../data/english.stop", "r")

stop_words_list = stop_file.readlines()
stop_words = {x.strip() for x in stop_words_list}


sorted_vocab = [x for x in sorted_vocab if x[0] not in stop_words]
vocab = []
for i in range(256):
    vocab.append(sorted_vocab[i][0])

vocab = sorted(vocab)
#for v in vocab:
#    print v


#generate document matrix from docs

final_vocab_set = {x for x in vocab}

doc_matrix = []

for i in range(256):
    doc_dict = defaultdict(int)

    d_words = docs[i].split(" ")
    d_list = []

    for w in d_words:
        if w in final_vocab_set:
            doc_dict[w] += 1

    for v in vocab:
        if v in doc_dict:
            d_list.append(doc_dict[v])

        else:
            d_list.append(0)

    doc_matrix.append(d_list)

print("256")
for d in doc_matrix:
    print " ".join(str(e) for e in d)
```

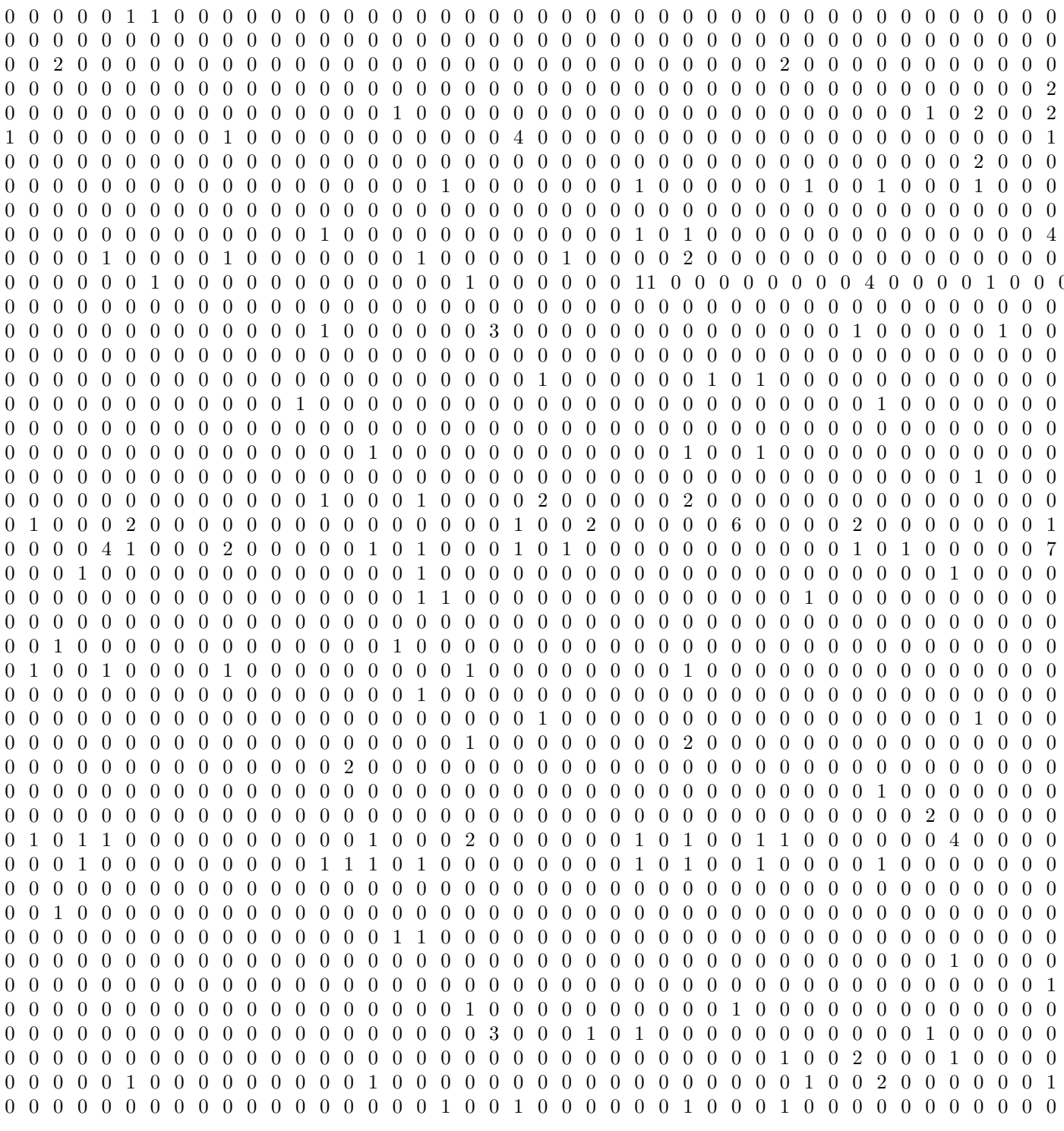
256



0 0 0 2 0 0 0 0 0 0 0 0 0 1 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 3 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 3 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 2 2 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 5
1 2 0 1 0 2 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 1 0 3 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 2 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 2
0 8 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 4 0 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 1 1 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0







```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 3 0 0 0 0 2 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 2
0 0 0 0 0 0 0 0 0 1 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 4 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 3 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 2
0 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 2 0 0 2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 4 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 4 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0
5
-1.41421356237   -0.707106781187  -0.707106781187  0.0         0.0
0.0         -1.58113883008    -0.316227766017  0.0         0.0
0.0         0.0         -1.54919333848    -0.645497224368  0.0
0.0         0.0         0.0         1.8929694486      0.528270543795
0.0         0.0         0.0         -0.0        0.84907610528
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <math.h>
#include "jacobi.h"

void generalized_print_matrix(double* m, int rows, int cols) {
    int i, j;
    for (i = 0; i < rows; i++) {
```

```
            printf("\n| ");
            for (j = 0; j < cols; j++) {
                printf("%7.4f | ", m[(i * cols) + j]);
            }
            //printf("\n");
        }

        printf("\n");
}

void gen_diag_matrix(double* vector, double*matrix, int dim_x, int dim_y) {


        int i, j;
        for (i = 0; i < dim_y; i++) {
            for (j = 0; j < dim_x; j++) {
                if (i == j) { //diagonal element
                    matrix[i * dim_x + j] = vector[i];
                }

                else {
                    matrix[i * dim_x + j] = 0.0;
                }
            }
        }

}


int copy_rows(double* in, double* out, int start_row, int end_row, int dim) {
        if (!in || !out || start_row >= end_row) {
            return -1;
        }

        int i, j;
        for (i = start_row; i < end_row; i++) {
            for (j = 0; j < dim; j++) {
                out[(i - start_row) * dim + j ] =  in [i * dim + j];
            }
        }
        return 0;
}

int copy_cols(double* in, double*out, int start_col, int end_col, int dim) {
        if (!in || !out || start_col >= end_col) {
            return -1;
```

```
    }

    int i, j;
    for (i = 0; i < dim; i++){
        for (j = start_col; j < end_col; j++) {
            // printf("DEBUG i : %d j : %d \n", i, j);
            out[i * (end_col - start_col)  + (j - start_col) ] =
in [i * dim + j];
        }
    }
    return 0;
}

int trunc_rows(double** in, int num_rows, int old_x_dim) {
    /* Copy the first num_rows rows from input matrix
     * Making a new matrix of num_rows rows and old_x_dim cols
     * Maybe I should have used realloc() here?
     * fuckit
     */

    double* new_matrix = (double*) malloc(sizeof(double) * num_rows * old_x_dim);
    if (!new_matrix) {
        return -1;
    }
    copy_rows(*in, new_matrix, 0, num_rows, old_x_dim);
    free(*in);
    *in = new_matrix;
    return 0;
}

int trunc_cols(double** in, int num_cols, int old_y_dim) {
    /* Copy the first num_cols cols from input matrix
     * Making a new matrix of old_y_dim rows and num_cols cols
     */

    double* new_matrix = (double*) malloc(sizeof(double) * num_cols * old_y_dim);
    if (!new_matrix) {
        return -1;
    }
    copy_cols(*in, new_matrix, 0, num_cols, old_y_dim);
    generalized_print_matrix(new_matrix, old_y_dim, num_cols);
    free(*in);
    *in = new_matrix;
    return 0;
    }
```

```
int lsa_transform(double* input_matrix, int input_dim, double* output_matrix, int output
    if (output_dim >= input_dim) {
        return -1;
    }

    //Anton's v returns Vt
    double *s, *u, *v;
    u = (double *) malloc (sizeof (double) * (input_dim * input_dim ));
    s = (double *) malloc (sizeof (double) * input_dim);
    v = (double *) malloc (sizeof (double) * (input_dim * input_dim));
    printf("DEBUG: Starting jacobi now\n");
    jacobi (input_matrix, input_dim, s, u, v);
    double* full_diag_s = (double*) malloc(sizeof(double) * input_dim * input_dim);
    gen_diag_matrix(s, full_diag_s, input_dim, input_dim);
    printf("Singular values matrix (diagonal) \n");
    printMatrix(full_diag_s, input_dim);

    printf("Matrix U\n");
    printMatrix(u, input_dim);

    printf("Matrix Vt\n");
    printMatrix(v, input_dim);


    int i;


    //Dimensionality reduction
    for (i = output_dim; i< input_dim; i++) {
        s[i] = 0.0;
    }

    //Reduce the size of the orthonormal vectors u v
    //TODO not sure what to return out - the lsa just says remove
    //lowest singular values - best output u' v' s' vs a' or vs' us'
    //depends on exactly what you are doing with it


    double * diag_s = (double *) malloc(sizeof(double) * (output_dim * output_dim));
    gen_diag_matrix(s, diag_s, output_dim, output_dim);
    printf("Truncated Matrix S' \n");
    printMatrix(diag_s, output_dim);

    trunc_rows(&u, output_dim, input_dim);
```

```
    printf("Truncated U'");

    generalized_print_matrix(u, output_dim, input_dim);


    printf("Truncated Vt'");
    trunc_cols(&v, output_dim, input_dim);

    free(s);
    free(v);
    free(u);
    return 0;
}

int read_from_file(char* filename,  double** arr) {

    printf("DEBUG: reading from file: %s\n", filename);
    FILE* in = fopen(filename, "r");
    int dim = 0;
    int i, j;
    char buf[4096];
    char* ptr;
    fgets(buf, sizeof(buf), in);
    if (sscanf(buf, "%d", &dim) != 1) {
        return -1;
    }

    if(!dim)
        return -1;
    printf("DEBUG: matrix dimension: %d\n", dim);
    *arr = (double*) calloc(dim * dim, sizeof(double));

    generalized_print_matrix(*arr, dim, dim);
    for (i = 0; i<dim; i++) {
        if(!fgets(buf, sizeof(buf), in)) {
            free(*arr);
            return -1;
        }

        printf("DEBUG: %s", buf);
        j = 0;
        ptr = strtok(buf, " ");

        while (ptr && j < dim) {
            //Hacky thing to make matrix real random and dense
            //TODO get rid of this
```

```
            (*arr)[i * dim + j++] = atof(ptr);
            ptr = strtok(NULL, " ");
        }

    }

    printf("Finished reading matrix\n");

    generalized_print_matrix(*arr, dim, dim);


    return 0;

}


int main(int argc, char** argv) {
    double* d = NULL;
    int dim = 128;
    int out_dim = 16;
    read_from_file(argv[1], &d);

    double* out = (double*) calloc(out_dim * out_dim, sizeof(double));

    lsa_transform(d, dim, out, out_dim);


    return 0;

}
1 0 1 0 0
1 1 0 0 0
0 1 0 0 0
0 1 1 0 0
0 0 0 1 0
0 0 1 1 0
0 0 0 1 0
0 0 0 1 1
# ------------------------------------------------------------------------- #
#
# Copyright (C) 1991-2014 Altera Corporation
# Your use of Altera Corporation's design tools, logic functions
# and other software and tools, and its AMPP partner logic
# functions, and any output files from any of the foregoing
# (including device programming or simulation files), and any
# associated documentation or information are expressly subject
```

```
# to the terms and conditions of the Altera Program License
# Subscription Agreement, Altera MegaCore Function License
# Agreement, or other applicable license agreement, including,
# without limitation, that your use is for the sole purpose of
# programming logic devices manufactured by Altera and sold by
# Altera or its authorized distributors.  Please refer to the
# applicable agreement for further details.
#
# ------------------------------------------------------------------------- #
#
# Quartus II 32-bit
# Version 13.1.2 Build 173 01/15/2014 SJ Full Version
# Date created = 21:21:35  February 18, 2014
#
# ------------------------------------------------------------------------- #
#
# Notes:
#
# 1) The default values for assignments are stored in the file:
#             SoCKit_Top_assignment_defaults.qdf
#    If this file doesn't exist, see file:
#             assignment_defaults.qdf
#
# 2) Altera recommends that you do not modify this file. This
#    file is updated automatically by the Quartus II software
#    and any changes you make may be lost or overwritten.
#
# ------------------------------------------------------------------------- #


set_global_assignment -name FAMILY "Cyclone V"
set_global_assignment -name DEVICE 5CSXFC6D6F31C8ES
set_global_assignment -name TOP_LEVEL_ENTITY SoCKit_Top
set_global_assignment -name ORIGINAL_QUARTUS_VERSION 13.1
set_global_assignment -name PROJECT_CREATION_TIME_DATE "21:21:35
FEBRUARY 18, 2014"
set_global_assignment -name LAST_QUARTUS_VERSION 13.1
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 256
set_global_assignment -name EDA_SIMULATION_TOOL "ModelSim-Altera (VHDL)"
set_global_assignment -name EDA_OUTPUT_DATA_FORMAT VHDL -section_id eda_simulation
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL PLACEMENT_AND_ROUTING -s
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
```

```
set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM HEAT SINK WITH 200 LFPM
set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_oct_rzqin -tag __
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[0] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[1] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[2] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[3] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[4] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[5] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[6] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[7] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[8] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[9] -tag __
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[10] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[11] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[12] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[13] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[14] -tag _
```

```
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[15] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[16] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[17] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[18] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[19] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[20] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[21] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[22] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[23] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[24] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[25] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[26] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[27] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[28] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[29] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
```

```
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[30] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dq[31] -tag _
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name INPUT_TERMINATION "PARALLEL 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITHOUT CALIBRATION" -to
set_instance_assignment -name D5_DELAY 2 -to memory_mem_ck -tag __hps_sdram_p0
set_instance_assignment -name IO_STANDARD "DIFFERENTIAL 1.5-V SSTL CLASS I" -to memory_m
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITHOUT CALIBRATION" -to
set_instance_assignment -name D5_DELAY 2 -to memory_mem_ck_n -tag __hps_sdram_p0
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[0] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[0]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[10] -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[10
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[11] -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[11
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[12] -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[12
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[13] -tag __
```

```
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[13
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[14] -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[14
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[1] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[1]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[2] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[2]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[3] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[3]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[4] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[4]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[5] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[5]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[6] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[6]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[7] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[7]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[8] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[8]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_a[9] -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_a[9]
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_ba[0] -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_ba[0
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_ba[1] -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_ba[1
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_ba[2] -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_ba[2
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_cas_n -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_cas_
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_cke -tag __hp
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_cke
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_cs_n -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_cs_n
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_odt -tag __hp
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_odt
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_ras_n -tag __
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_ras_
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_we_n -tag __h
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_we_n
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_reset_n -tag
set_instance_assignment -name CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to memory_mem_res
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dm[0] -tag __
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dm[1] -tag __
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to me
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dm[2] -tag __
```

```
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name IO_STANDARD "SSTL-15 CLASS I" -to memory_mem_dm[3] -tag __
set_instance_assignment -name OUTPUT_TERMINATION "SERIES 50 OHM WITH CALIBRATION" -to m
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[0] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[1] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[2] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[3] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[4] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[5] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[6] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[7] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[8] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[9] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[10] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[11] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[12] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[13] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[14] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[15] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[16] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[17] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[18] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[19] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[20] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[21] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[22] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[23] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[24] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[25] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[26] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[27] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[28] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[29] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[30] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dq[31] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dm[0] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dm[1] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dm[2] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dm[3] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs[0] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs[1] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs[2] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs[3] -tag _
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs_n[0] -tag
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs_n[1] -tag
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs_n[2] -tag
```

```
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_dqs_n[3] -tag
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[0] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[10] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[11] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[12] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[13] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[14] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[1] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[2] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[3] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[4] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[5] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[6] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[7] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[8] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_a[9] -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_ba[0] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_ba[1] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_ba[2] -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_cas_n -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_cke -tag __hp
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_cs_n -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_odt -tag __hp
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_ras_n -tag __
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_we_n -tag __h
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_reset_n -tag
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_ck -tag __hps
set_instance_assignment -name PACKAGE_SKEW_COMPENSATION OFF -to memory_mem_ck_n -tag __h
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name GLOBAL_SIGNAL OFF -to u0|hps_0|hps_io|border|hps_sdram_ins
set_instance_assignment -name ENABLE_BENEFICIAL_SKEW_OPTIMIZATION_FOR_NON_GLOBAL_CLOCKS
set_instance_assignment -name PLL_COMPENSATION_MODE DIRECT -to u0|hps_0|hps_io|border|h
set_global_assignment -name USE_DLL_FREQUENCY_FOR_DQS_DELAY_CHAIN ON
set_global_assignment -name UNIPHY_SEQUENCER_DQS_CONFIG_ENABLE ON
```

```
set_global_assignment -name OPTIMIZE_MULTI_CORNER_TIMING ON
set_global_assignment -name ECO_REGENERATE_REPORT ON
set_location_assignment PIN_AC27 -to AUD_ADCDAT
set_location_assignment PIN_AG30 -to AUD_ADCLRCK
set_location_assignment PIN_AE7 -to AUD_BCLK
set_location_assignment PIN_AG3 -to AUD_DACDAT
set_location_assignment PIN_AH4 -to AUD_DACLRCK
set_location_assignment PIN_AH30 -to AUD_I2C_SCLK
set_location_assignment PIN_AF30 -to AUD_I2C_SDAT
set_location_assignment PIN_AD26 -to AUD_MUTE
set_location_assignment PIN_AC9 -to AUD_XCK
set_location_assignment PIN_AJ14 -to DDR3_A[0]
set_location_assignment PIN_AK14 -to DDR3_A[1]
set_location_assignment PIN_AH12 -to DDR3_A[2]
set_location_assignment PIN_AJ12 -to DDR3_A[3]
set_location_assignment PIN_AG15 -to DDR3_A[4]
set_location_assignment PIN_AH15 -to DDR3_A[5]
set_location_assignment PIN_AK12 -to DDR3_A[6]
set_location_assignment PIN_AK13 -to DDR3_A[7]
set_location_assignment PIN_AH13 -to DDR3_A[8]
set_location_assignment PIN_AH14 -to DDR3_A[9]
set_location_assignment PIN_AJ9 -to DDR3_A[10]
set_location_assignment PIN_AK9 -to DDR3_A[11]
set_location_assignment PIN_AK7 -to DDR3_A[12]
set_location_assignment PIN_AK8 -to DDR3_A[13]
set_location_assignment PIN_AG12 -to DDR3_A[14]
set_location_assignment PIN_AH10 -to DDR3_BA[0]
set_location_assignment PIN_AJ11 -to DDR3_BA[1]
set_location_assignment PIN_AK11 -to DDR3_BA[2]
set_location_assignment PIN_AH7 -to DDR3_CAS_n
set_location_assignment PIN_AJ21 -to DDR3_CKE
set_location_assignment PIN_AA15 -to DDR3_CK_n
set_location_assignment PIN_AA14 -to DDR3_CK_p
set_location_assignment PIN_AB15 -to DDR3_CS_n
set_location_assignment PIN_AH17 -to DDR3_DM[0]
set_location_assignment PIN_AG23 -to DDR3_DM[1]
set_location_assignment PIN_AK23 -to DDR3_DM[2]
set_location_assignment PIN_AJ27 -to DDR3_DM[3]
set_location_assignment PIN_AF18 -to DDR3_DQ[0]
set_location_assignment PIN_AE17 -to DDR3_DQ[1]
set_location_assignment PIN_AG16 -to DDR3_DQ[2]
set_location_assignment PIN_AF16 -to DDR3_DQ[3]
set_location_assignment PIN_AH20 -to DDR3_DQ[4]
set_location_assignment PIN_AG21 -to DDR3_DQ[5]
set_location_assignment PIN_AJ16 -to DDR3_DQ[6]
set_location_assignment PIN_AH18 -to DDR3_DQ[7]
```

```
set_location_assignment PIN_AK18 -to DDR3_DQ[8]
set_location_assignment PIN_AJ17 -to DDR3_DQ[9]
set_location_assignment PIN_AG18 -to DDR3_DQ[10]
set_location_assignment PIN_AK19 -to DDR3_DQ[11]
set_location_assignment PIN_AG20 -to DDR3_DQ[12]
set_location_assignment PIN_AF19 -to DDR3_DQ[13]
set_location_assignment PIN_AJ20 -to DDR3_DQ[14]
set_location_assignment PIN_AH24 -to DDR3_DQ[15]
set_location_assignment PIN_AE19 -to DDR3_DQ[16]
set_location_assignment PIN_AE18 -to DDR3_DQ[17]
set_location_assignment PIN_AG22 -to DDR3_DQ[18]
set_location_assignment PIN_AK22 -to DDR3_DQ[19]
set_location_assignment PIN_AF21 -to DDR3_DQ[20]
set_location_assignment PIN_AF20 -to DDR3_DQ[21]
set_location_assignment PIN_AH23 -to DDR3_DQ[22]
set_location_assignment PIN_AK24 -to DDR3_DQ[23]
set_location_assignment PIN_AF24 -to DDR3_DQ[24]
set_location_assignment PIN_AF23 -to DDR3_DQ[25]
set_location_assignment PIN_AJ24 -to DDR3_DQ[26]
set_location_assignment PIN_AK26 -to DDR3_DQ[27]
set_location_assignment PIN_AE23 -to DDR3_DQ[28]
set_location_assignment PIN_AE22 -to DDR3_DQ[29]
set_location_assignment PIN_AG25 -to DDR3_DQ[30]
set_location_assignment PIN_AK27 -to DDR3_DQ[31]
set_location_assignment PIN_W16 -to DDR3_DQS_n[0]
set_location_assignment PIN_W17 -to DDR3_DQS_n[1]
set_location_assignment PIN_AA18 -to DDR3_DQS_n[2]
set_location_assignment PIN_AD19 -to DDR3_DQS_n[3]
set_location_assignment PIN_V16 -to DDR3_DQS_p[0]
set_location_assignment PIN_V17 -to DDR3_DQS_p[1]
set_location_assignment PIN_Y17 -to DDR3_DQS_p[2]
set_location_assignment PIN_AC20 -to DDR3_DQS_p[3]
set_location_assignment PIN_AE16 -to DDR3_ODT
set_location_assignment PIN_AH8 -to DDR3_RAS_n
set_location_assignment PIN_AK21 -to DDR3_RESET_n
set_location_assignment PIN_AG17 -to DDR3_RZQ
set_location_assignment PIN_AJ6 -to DDR3_WE_n
set_location_assignment PIN_AG27 -to FAN_CTRL
set_location_assignment PIN_AB27 -to HSMC_CLKIN_n[1]
set_location_assignment PIN_G15 -to HSMC_CLKIN_n[2]
set_location_assignment PIN_AA26 -to HSMC_CLKIN_p[1]
set_location_assignment PIN_H15 -to HSMC_CLKIN_p[2]
set_location_assignment PIN_E6 -to HSMC_CLKOUT_n[1]
set_location_assignment PIN_A10 -to HSMC_CLKOUT_n[2]
set_location_assignment PIN_E7 -to HSMC_CLKOUT_p[1]
set_location_assignment PIN_A11 -to HSMC_CLKOUT_p[2]
```

```
set_location_assignment PIN_J14 -to HSMC_CLK_IN0
set_location_assignment PIN_AD29 -to HSMC_CLK_OUT0
set_location_assignment PIN_C10 -to HSMC_D[0]
set_location_assignment PIN_H13 -to HSMC_D[1]
set_location_assignment PIN_C9 -to HSMC_D[2]
set_location_assignment PIN_H12 -to HSMC_D[3]
set_location_assignment PIN_AE2 -to HSMC_GXB_RX_p[0]
set_location_assignment PIN_AC2 -to HSMC_GXB_RX_p[1]
set_location_assignment PIN_AA2 -to HSMC_GXB_RX_p[2]
set_location_assignment PIN_W2 -to HSMC_GXB_RX_p[3]
set_location_assignment PIN_U2 -to HSMC_GXB_RX_p[4]
set_location_assignment PIN_R2 -to HSMC_GXB_RX_p[5]
set_location_assignment PIN_N2 -to HSMC_GXB_RX_p[6]
set_location_assignment PIN_J2 -to HSMC_GXB_RX_p[7]
set_location_assignment PIN_AD4 -to HSMC_GXB_TX_p[0]
set_location_assignment PIN_AB4 -to HSMC_GXB_TX_p[1]
set_location_assignment PIN_Y4 -to HSMC_GXB_TX_p[2]
set_location_assignment PIN_V4 -to HSMC_GXB_TX_p[3]
set_location_assignment PIN_T4 -to HSMC_GXB_TX_p[4]
set_location_assignment PIN_P4 -to HSMC_GXB_TX_p[5]
set_location_assignment PIN_M4 -to HSMC_GXB_TX_p[6]
set_location_assignment PIN_H4 -to HSMC_GXB_TX_p[7]
set_location_assignment PIN_P9 -to HSMC_REF_CLK_p
set_location_assignment PIN_G11 -to HSMC_RX_n[0]
set_location_assignment PIN_J12 -to HSMC_RX_n[1]
set_location_assignment PIN_F10 -to HSMC_RX_n[2]
set_location_assignment PIN_J9 -to HSMC_RX_n[3]
set_location_assignment PIN_K8 -to HSMC_RX_n[4]
set_location_assignment PIN_H7 -to HSMC_RX_n[5]
set_location_assignment PIN_G8 -to HSMC_RX_n[6]
set_location_assignment PIN_F8 -to HSMC_RX_n[7]
set_location_assignment PIN_E11 -to HSMC_RX_n[8]
set_location_assignment PIN_B5 -to HSMC_RX_n[9]
set_location_assignment PIN_D9 -to HSMC_RX_n[10]
set_location_assignment PIN_D12 -to HSMC_RX_n[11]
set_location_assignment PIN_D10 -to HSMC_RX_n[12]
set_location_assignment PIN_B12 -to HSMC_RX_n[13]
set_location_assignment PIN_E13 -to HSMC_RX_n[14]
set_location_assignment PIN_G13 -to HSMC_RX_n[15]
set_location_assignment PIN_F14 -to HSMC_RX_n[16]
set_location_assignment PIN_G12 -to HSMC_RX_p[0]
set_location_assignment PIN_K12 -to HSMC_RX_p[1]
set_location_assignment PIN_G10 -to HSMC_RX_p[2]
set_location_assignment PIN_J10 -to HSMC_RX_p[3]
set_location_assignment PIN_K7 -to HSMC_RX_p[4]
set_location_assignment PIN_J7 -to HSMC_RX_p[5]
```

```
set_location_assignment PIN_H8 -to HSMC_RX_p[6]
set_location_assignment PIN_F9 -to HSMC_RX_p[7]
set_location_assignment PIN_F11 -to HSMC_RX_p[8]
set_location_assignment PIN_B6 -to HSMC_RX_p[9]
set_location_assignment PIN_E9 -to HSMC_RX_p[10]
set_location_assignment PIN_E12 -to HSMC_RX_p[11]
set_location_assignment PIN_D11 -to HSMC_RX_p[12]
set_location_assignment PIN_C13 -to HSMC_RX_p[13]
set_location_assignment PIN_F13 -to HSMC_RX_p[14]
set_location_assignment PIN_H14 -to HSMC_RX_p[15]
set_location_assignment PIN_F15 -to HSMC_RX_p[16]
set_location_assignment PIN_AA28 -to HSMC_SCL
set_location_assignment PIN_AE29 -to HSMC_SDA
set_location_assignment PIN_A8 -to HSMC_TX_n[0]
set_location_assignment PIN_D7 -to HSMC_TX_n[1]
set_location_assignment PIN_F6 -to HSMC_TX_n[2]
set_location_assignment PIN_C5 -to HSMC_TX_n[3]
set_location_assignment PIN_C4 -to HSMC_TX_n[4]
set_location_assignment PIN_E2 -to HSMC_TX_n[5]
set_location_assignment PIN_D4 -to HSMC_TX_n[6]
set_location_assignment PIN_B3 -to HSMC_TX_n[7]
set_location_assignment PIN_D1 -to HSMC_TX_n[8]
set_location_assignment PIN_C2 -to HSMC_TX_n[9]
set_location_assignment PIN_B1 -to HSMC_TX_n[10]
set_location_assignment PIN_A3 -to HSMC_TX_n[11]
set_location_assignment PIN_A5 -to HSMC_TX_n[12]
set_location_assignment PIN_B7 -to HSMC_TX_n[13]
set_location_assignment PIN_B8 -to HSMC_TX_n[14]
set_location_assignment PIN_B11 -to HSMC_TX_n[15]
set_location_assignment PIN_A13 -to HSMC_TX_n[16]
set_location_assignment PIN_A9 -to HSMC_TX_p[0]
set_location_assignment PIN_E8 -to HSMC_TX_p[1]
set_location_assignment PIN_G7 -to HSMC_TX_p[2]
set_location_assignment PIN_D6 -to HSMC_TX_p[3]
set_location_assignment PIN_D5 -to HSMC_TX_p[4]
set_location_assignment PIN_E3 -to HSMC_TX_p[5]
set_location_assignment PIN_E4 -to HSMC_TX_p[6]
set_location_assignment PIN_C3 -to HSMC_TX_p[7]
set_location_assignment PIN_E1 -to HSMC_TX_p[8]
set_location_assignment PIN_D2 -to HSMC_TX_p[9]
set_location_assignment PIN_B2 -to HSMC_TX_p[10]
set_location_assignment PIN_A4 -to HSMC_TX_p[11]
set_location_assignment PIN_A6 -to HSMC_TX_p[12]
set_location_assignment PIN_C7 -to HSMC_TX_p[13]
set_location_assignment PIN_C8 -to HSMC_TX_p[14]
set_location_assignment PIN_C12 -to HSMC_TX_p[15]
```

```
set_location_assignment PIN_B13 -to HSMC_TX_p[16]
set_location_assignment PIN_AH2 -to IRDA_RXD
set_location_assignment PIN_AE9 -to KEY[0]
set_location_assignment PIN_AE12 -to KEY[1]
set_location_assignment PIN_AD9 -to KEY[2]
set_location_assignment PIN_AD11 -to KEY[3]
set_location_assignment PIN_AF10 -to LED[0]
set_location_assignment PIN_AD10 -to LED[1]
set_location_assignment PIN_AE11 -to LED[2]
set_location_assignment PIN_AD7 -to LED[3]
set_location_assignment PIN_AF14 -to OSC_50_B3B
set_location_assignment PIN_AA16 -to OSC_50_B4A
set_location_assignment PIN_Y26 -to OSC_50_B5B
set_location_assignment PIN_K14 -to OSC_50_B8A
set_location_assignment PIN_W22 -to PCIE_PERST_n
set_location_assignment PIN_W21 -to PCIE_WAKE_n
set_location_assignment PIN_AD27 -to RESET_n
set_location_assignment PIN_AE26 -to SI5338_SCL
set_location_assignment PIN_AJ29 -to SI5338_SDA
set_location_assignment PIN_W25 -to SW[0]
set_location_assignment PIN_V25 -to SW[1]
set_location_assignment PIN_AC28 -to SW[2]
set_location_assignment PIN_AC29 -to SW[3]
set_location_assignment PIN_AF8 -to TEMP_CS_n
set_location_assignment PIN_AG7 -to TEMP_DIN
set_location_assignment PIN_AG1 -to TEMP_DOUT
set_location_assignment PIN_AF9 -to TEMP_SCLK
set_location_assignment PIN_AF13 -to USB_B2_CLK
set_location_assignment PIN_AK28 -to USB_B2_DATA[0]
set_location_assignment PIN_AD20 -to USB_B2_DATA[1]
set_location_assignment PIN_AD21 -to USB_B2_DATA[2]
set_location_assignment PIN_Y19 -to USB_B2_DATA[3]
set_location_assignment PIN_AA20 -to USB_B2_DATA[4]
set_location_assignment PIN_AH27 -to USB_B2_DATA[5]
set_location_assignment PIN_AF25 -to USB_B2_DATA[6]
set_location_assignment PIN_AC22 -to USB_B2_DATA[7]
set_location_assignment PIN_AJ4 -to USB_EMPTY
set_location_assignment PIN_AK3 -to USB_FULL
set_location_assignment PIN_AE14 -to USB_OE_n
set_location_assignment PIN_AJ5 -to USB_RD_n
set_location_assignment PIN_AD14 -to USB_RESET_n
set_location_assignment PIN_AK4 -to USB_SCL
set_location_assignment PIN_AE13 -to USB_SDA
set_location_assignment PIN_AK6 -to USB_WR_n
set_location_assignment PIN_AE28 -to VGA_B[0]
set_location_assignment PIN_Y23 -to VGA_B[1]
```

```
set_location_assignment PIN_Y24 -to VGA_B[2]
set_location_assignment PIN_AG28 -to VGA_B[3]
set_location_assignment PIN_AF28 -to VGA_B[4]
set_location_assignment PIN_V23 -to VGA_B[5]
set_location_assignment PIN_W24 -to VGA_B[6]
set_location_assignment PIN_AF29 -to VGA_B[7]
set_location_assignment PIN_AH3 -to VGA_BLANK_n
set_location_assignment PIN_W20 -to VGA_CLK
set_location_assignment PIN_Y21 -to VGA_G[0]
set_location_assignment PIN_AA25 -to VGA_G[1]
set_location_assignment PIN_AB26 -to VGA_G[2]
set_location_assignment PIN_AB22 -to VGA_G[3]
set_location_assignment PIN_AB23 -to VGA_G[4]
set_location_assignment PIN_AA24 -to VGA_G[5]
set_location_assignment PIN_AB25 -to VGA_G[6]
set_location_assignment PIN_AE27 -to VGA_G[7]
set_location_assignment PIN_AD12 -to VGA_HS
set_location_assignment PIN_AG5 -to VGA_R[0]
set_location_assignment PIN_AA12 -to VGA_R[1]
set_location_assignment PIN_AB12 -to VGA_R[2]
set_location_assignment PIN_AF6 -to VGA_R[3]
set_location_assignment PIN_AG6 -to VGA_R[4]
set_location_assignment PIN_AJ2 -to VGA_R[5]
set_location_assignment PIN_AH5 -to VGA_R[6]
set_location_assignment PIN_AJ1 -to VGA_R[7]
set_location_assignment PIN_AG2 -to VGA_SYNC_n
set_location_assignment PIN_AC12 -to VGA_VS
set_global_assignment -name QIP_FILE tyrion/synthesis/tyrion.qip
set_global_assignment -name VERILOG_FILE SoCKit_top.v
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id Top//
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera Development
//   Kits made by Terasic.  Other use of this code, including the selling
//   ,duplication, or modification of any portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design reference
//   which illustrates how these types of functions can be implemented.
//   It is the user's responsibility to verify their design for
//   consistency and functionality through the use of formal
```

```
//    verification methods.  Terasic provides no warranty regarding the use
//    or functionality of this code.
//
// ====================================================================
//
//  Terasic Technologies Inc
//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
//
// ====================================================================
// ====================================================================
//
// Major Functions:      SoCKit_Default
//
// ====================================================================
// Revision History :
// ====================================================================
//    Ver  :| Author            :| Mod. Date :| Changes Made:
//    V1.0 :| xinxian           :| 04/02/13  :| Initial Revision
// ====================================================================

//`define ENABLE_DDR3
//`define ENABLE_HPS
//`define ENABLE_HSMC_XCVR

module SoCKit_Top(

                    ///////////AUD//////////
                    AUD_ADCDAT,
                    AUD_ADCLRCK,
                    AUD_BCLK,
                    AUD_DACDAT,
                    AUD_DACLRCK,
                    AUD_I2C_SCLK,
                    AUD_I2C_SDAT,
                    AUD_MUTE,
                    AUD_XCK,

`ifdef ENABLE_DDR3
                    ////////DDR3////////
                    DDR3_A,
                    DDR3_BA,
                    DDR3_CAS_n,
```

```
                        DDR3_CKE,
                        DDR3_CK_n,
                        DDR3_CK_p,
                        DDR3_CS_n,
                        DDR3_DM,
                        DDR3_DQ,
                        DDR3_DQS_n,
                        DDR3_DQS_p,
                        DDR3_ODT,
                        DDR3_RAS_n,
                        DDR3_RESET_n,
                        DDR3_RZQ,
                        DDR3_WE_n,
`endif /*ENABLE_DDR3*/

                        /////////FAN/////////
                        FAN_CTRL,

`ifdef ENABLE_HPS
                        /////////HPS/////////
                        HPS_CLOCK_25,
                        HPS_CLOCK_50,
                        HPS_CONV_USB_n,
                        HPS_DDR3_A,
                        HPS_DDR3_BA,
                        HPS_DDR3_CAS_n,
                        HPS_DDR3_CKE,
                        HPS_DDR3_CK_n,
                        HPS_DDR3_CK_p,
                        HPS_DDR3_CS_n,
                        HPS_DDR3_DM,
                        HPS_DDR3_DQ,
                        HPS_DDR3_DQS_n,
                        HPS_DDR3_DQS_p,
                        HPS_DDR3_ODT,
                        HPS_DDR3_RAS_n,
                        HPS_DDR3_RESET_n,
                        HPS_DDR3_RZQ,
                        HPS_DDR3_WE_n,
                        HPS_ENET_GTX_CLK,
                        HPS_ENET_INT_n,
                        HPS_ENET_MDC,
                        HPS_ENET_MDIO,
                        HPS_ENET_RESET_n,
                        HPS_ENET_RX_CLK,
                        HPS_ENET_RX_DATA,
```

```
                        HPS_ENET_RX_DV,
                        HPS_ENET_TX_DATA,
                        HPS_ENET_TX_EN,
                        HPS_FLASH_DATA,
                        HPS_FLASH_DCLK,
                        HPS_FLASH_NCSO,
                        HPS_GSENSOR_INT,
                        HPS_I2C_CLK,
                        HPS_I2C_SDA,
                        HPS_KEY,
                        HPS_LCM_D_C,
                        HPS_LCM_RST_N,
                        HPS_LCM_SPIM_CLK,
                        HPS_LCM_SPIM_MISO,
                        HPS_LCM_SPIM_MOSI,
                        HPS_LCM_SPIM_SS,
                        HPS_LED,
                        HPS_LTC_GPIO,
                        HPS_RESET_n,
                        HPS_SD_CLK,
                        HPS_SD_CMD,
                        HPS_SD_DATA,
                        HPS_SPIM_CLK,
                        HPS_SPIM_MISO,
                        HPS_SPIM_MOSI,
                        HPS_SPIM_SS,
                        HPS_SW,
                        HPS_UART_RX,
                        HPS_UART_TX,
                        HPS_USB_CLKOUT,
                        HPS_USB_DATA,
                        HPS_USB_DIR,
                        HPS_USB_NXT,
                        HPS_USB_RESET_PHY,
                        HPS_USB_STP,
                        HPS_WARM_RST_n,
`endif /*ENABLE_HPS*/

                        /////////HSMC/////////
                        HSMC_CLKIN_n,
                        HSMC_CLKIN_p,
                        HSMC_CLKOUT_n,
                        HSMC_CLKOUT_p,
                        HSMC_CLK_IN0,
                        HSMC_CLK_OUT0,
                        HSMC_D,
```

```
`ifdef ENABLE_HSMC_XCVR

                        HSMC_GXB_RX_p,
                        HSMC_GXB_TX_p,
                        HSMC_REF_CLK_p,
`endif
                        HSMC_RX_n,
                        HSMC_RX_p,
                        HSMC_SCL,
                        HSMC_SDA,
                        HSMC_TX_n,
                        HSMC_TX_p,

                        ////////IRDA////////
                        IRDA_RXD,

                        ////////KEY////////
                        KEY,

                        ////////LED////////
                        LED,

                        ////////OSC////////
                        OSC_50_B3B,
                        OSC_50_B4A,
                        OSC_50_B5B,
                        OSC_50_B8A,

                        ////////PCIE////////
                        PCIE_PERST_n,
                        PCIE_WAKE_n,

                        ////////RESET////////
                        RESET_n,

                        ////////SI5338////////
                        SI5338_SCL,
                        SI5338_SDA,

                        ////////SW////////
                        SW,

                        ////////TEMP////////
                        TEMP_CS_n,
                        TEMP_DIN,
```

```
TEMP_DOUT,
TEMP_SCLK,

/////////USB/////////
USB_B2_CLK,
USB_B2_DATA,
USB_EMPTY,
USB_FULL,
USB_OE_n,
USB_RD_n,
USB_RESET_n,
USB_SCL,
USB_SDA,
USB_WR_n,

/////////VGA/////////
VGA_B,
VGA_BLANK_n,
VGA_CLK,
VGA_G,
VGA_HS,
VGA_R,
VGA_SYNC_n,
VGA_VS,
//////////hps//////////
memory_mem_a,
memory_mem_ba,
memory_mem_ck,
memory_mem_ck_n,
memory_mem_cke,
memory_mem_cs_n,
memory_mem_ras_n,
memory_mem_cas_n,
memory_mem_we_n,
memory_mem_reset_n,
memory_mem_dq,
memory_mem_dqs,
memory_mem_dqs_n,
memory_mem_odt,
memory_mem_dm,
memory_oct_rzqin,
hps_io_hps_io_emac1_inst_TX_CLK,
hps_io_hps_io_emac1_inst_TXD0,
hps_io_hps_io_emac1_inst_TXD1,
hps_io_hps_io_emac1_inst_TXD2,
hps_io_hps_io_emac1_inst_TXD3,
```

```
hps_io_hps_io_emac1_inst_RXD0 ,
hps_io_hps_io_emac1_inst_MDIO ,
hps_io_hps_io_emac1_inst_MDC ,
hps_io_hps_io_emac1_inst_RX_CTL ,
hps_io_hps_io_emac1_inst_TX_CTL ,
hps_io_hps_io_emac1_inst_RX_CLK ,
hps_io_hps_io_emac1_inst_RXD1 ,
hps_io_hps_io_emac1_inst_RXD2 ,
hps_io_hps_io_emac1_inst_RXD3 ,
hps_io_hps_io_qspi_inst_IO0 ,
hps_io_hps_io_qspi_inst_IO1 ,
hps_io_hps_io_qspi_inst_IO2 ,
hps_io_hps_io_qspi_inst_IO3 ,
hps_io_hps_io_qspi_inst_SS0 ,
hps_io_hps_io_qspi_inst_CLK ,
hps_io_hps_io_sdio_inst_CMD ,
hps_io_hps_io_sdio_inst_D0 ,
hps_io_hps_io_sdio_inst_D1 ,
hps_io_hps_io_sdio_inst_CLK ,
hps_io_hps_io_sdio_inst_D2 ,
hps_io_hps_io_sdio_inst_D3 ,
hps_io_hps_io_usb1_inst_D0 ,
hps_io_hps_io_usb1_inst_D1 ,
hps_io_hps_io_usb1_inst_D2 ,
hps_io_hps_io_usb1_inst_D3 ,
hps_io_hps_io_usb1_inst_D4 ,
hps_io_hps_io_usb1_inst_D5 ,
hps_io_hps_io_usb1_inst_D6 ,
hps_io_hps_io_usb1_inst_D7 ,
hps_io_hps_io_usb1_inst_CLK ,
hps_io_hps_io_usb1_inst_STP ,
hps_io_hps_io_usb1_inst_DIR ,
hps_io_hps_io_usb1_inst_NXT ,
hps_io_hps_io_spim0_inst_CLK ,
hps_io_hps_io_spim0_inst_MOSI ,
hps_io_hps_io_spim0_inst_MISO ,
hps_io_hps_io_spim0_inst_SS0 ,
hps_io_hps_io_spim1_inst_CLK ,
hps_io_hps_io_spim1_inst_MOSI ,
hps_io_hps_io_spim1_inst_MISO ,
hps_io_hps_io_spim1_inst_SS0 ,
hps_io_hps_io_uart0_inst_RX ,
hps_io_hps_io_uart0_inst_TX ,
hps_io_hps_io_i2c1_inst_SDA ,
hps_io_hps_io_i2c1_inst_SCL ,
hps_io_hps_io_gpio_inst_GPIO00
```

```
                    );

    //=======================================================
    //  PORT declarations
    //=======================================================

    ///////// AUD /////////
    input                                                   AUD_ADCDAT;
    inout                                                   AUD_ADCLRCK;
    inout                                                   AUD_BCLK;
    output                                                  AUD_DACDAT;
    inout                                                   AUD_DACLRCK;
    output                                                  AUD_I2C_SCLK;
    inout                                                   AUD_I2C_SDAT;
    output                                                  AUD_MUTE;
    output                                                  AUD_XCK;

`ifdef ENABLE_DDR3
    ///////// DDR3 /////////
    output [14:0]                                           DDR3_A;
    output [2:0]                                            DDR3_BA;
    output                                                  DDR3_CAS_n;
    output                                                  DDR3_CKE;
    output                                                  DDR3_CK_n;
    output                                                  DDR3_CK_p;
    output                                                  DDR3_CS_n;
    output [3:0]                                            DDR3_DM;
    inout [31:0]                                            DDR3_DQ;
    inout [3:0]                                             DDR3_DQS_n;
    inout [3:0]                                             DDR3_DQS_p;
    output                                                  DDR3_ODT;
    output                                                  DDR3_RAS_n;
    output                                                  DDR3_RESET_n;
    input                                                   DDR3_RZQ;
    output                                                  DDR3_WE_n;
`endif /*ENABLE_DDR3*/

    ///////// FAN /////////
    output                                                  FAN_CTRL;

`ifdef ENABLE_HPS
    ///////// HPS /////////
    input                                                   HPS_CLOCK_25;
    input                                                   HPS_CLOCK_50;
    input                                                   HPS_CONV_USB_n;
    output [14:0]                                           HPS_DDR3_A;
```

```
output  [2:0]   HPS_DDR3_BA;
output          HPS_DDR3_CAS_n;
output          HPS_DDR3_CKE;
output          HPS_DDR3_CK_n;
output          HPS_DDR3_CK_p;
output          HPS_DDR3_CS_n;
output  [3:0]   HPS_DDR3_DM;
inout   [31:0]  HPS_DDR3_DQ;
inout   [3:0]   HPS_DDR3_DQS_n;
inout   [3:0]   HPS_DDR3_DQS_p;
output          HPS_DDR3_ODT;
output          HPS_DDR3_RAS_n;
output          HPS_DDR3_RESET_n;
input           HPS_DDR3_RZQ;
output          HPS_DDR3_WE_n;
input           HPS_ENET_GTX_CLK;
input           HPS_ENET_INT_n;
output          HPS_ENET_MDC;
inout           HPS_ENET_MDIO;
output          HPS_ENET_RESET_n;
input           HPS_ENET_RX_CLK;
input   [3:0]   HPS_ENET_RX_DATA;
input           HPS_ENET_RX_DV;
output  [3:0]   HPS_ENET_TX_DATA;
output          HPS_ENET_TX_EN;
inout   [3:0]   HPS_FLASH_DATA;
output          HPS_FLASH_DCLK;
output          HPS_FLASH_NCSO;
input           HPS_GSENSOR_INT;
inout           HPS_I2C_CLK;
inout           HPS_I2C_SDA;
inout   [3:0]   HPS_KEY;
output          HPS_LCM_D_C;
output          HPS_LCM_RST_N;
input           HPS_LCM_SPIM_CLK;
inout           HPS_LCM_SPIM_MISO;
output          HPS_LCM_SPIM_MOSI;
output          HPS_LCM_SPIM_SS;
output  [3:0]   HPS_LED;
inout           HPS_LTC_GPIO;
input           HPS_RESET_n;
output          HPS_SD_CLK;
inout           HPS_SD_CMD;
inout   [3:0]   HPS_SD_DATA;
output          HPS_SPIM_CLK;
input           HPS_SPIM_MISO;
```

```
    output                                  HPS_SPIM_MOSI;
    output                                  HPS_SPIM_SS;
    input  [3:0]                            HPS_SW;
    input                                   HPS_UART_RX;
    output                                  HPS_UART_TX;
    input                                   HPS_USB_CLKOUT;
    inout  [7:0]                            HPS_USB_DATA;
    input                                   HPS_USB_DIR;
    input                                   HPS_USB_NXT;
    output                                  HPS_USB_RESET_PHY;
    output                                  HPS_USB_STP;
    input                                   HPS_WARM_RST_n;
`endif /*ENABLE_HPS*/

    ///////// HSMC /////////
    input  [2:1]                            HSMC_CLKIN_n;
    input  [2:1]                            HSMC_CLKIN_p;
    output [2:1]                            HSMC_CLKOUT_n;
    output [2:1]                            HSMC_CLKOUT_p;
    input                                   HSMC_CLK_IN0;
    output                                  HSMC_CLK_OUT0;
    inout  [3:0]                            HSMC_D;
`ifdef ENABLE_HSMC_XCVR
    input  [7:0]                            HSMC_GXB_RX_p;
    output [7:0]                            HSMC_GXB_TX_p;
    input                                   HSMC_REF_CLK_p;
`endif
    inout  [16:0]                           HSMC_RX_n;
    inout  [16:0]                           HSMC_RX_p;
    output                                  HSMC_SCL;
    inout                                   HSMC_SDA;
    inout  [16:0]                           HSMC_TX_n;
    inout  [16:0]                           HSMC_TX_p;

    ///////// IRDA /////////
    input                                   IRDA_RXD;

    ///////// KEY /////////
    input  [3:0]                            KEY;

    ///////// LED /////////
    output [3:0]                            LED;

    ///////// OSC /////////
    input                                   OSC_50_B3B;
    input                                   OSC_50_B4A;
```

```
input                                   OSC_50_B5B;
input                                   OSC_50_B8A;

//////// PCIE ////////
input                                   PCIE_PERST_n;
input                                   PCIE_WAKE_n;

//////// RESET ////////
input                                   RESET_n;

//////// SI5338 ////////
inout                                   SI5338_SCL;
inout                                   SI5338_SDA;

//////// SW ////////
input  [3:0]                            SW;

//////// TEMP ////////
output                                  TEMP_CS_n;
output                                  TEMP_DIN;
input                                   TEMP_DOUT;
output                                  TEMP_SCLK;

//////// USB ////////
input                                   USB_B2_CLK;
inout  [7:0]                            USB_B2_DATA;
output                                  USB_EMPTY;
output                                  USB_FULL;
input                                   USB_OE_n;
input                                   USB_RD_n;
input                                   USB_RESET_n;
inout                                   USB_SCL;
inout                                   USB_SDA;
input                                   USB_WR_n;

//////// VGA ////////
output [7:0]                            VGA_B;
output                                  VGA_BLANK_n;
output                                  VGA_CLK;
output [7:0]                            VGA_G;
output                                  VGA_HS;
output [7:0]                            VGA_R;
output                                  VGA_SYNC_n;
output                                  VGA_VS;

/////////hps pin///////
```

```
output  wire [14:0]  memory_mem_a;
output  wire [2:0]   memory_mem_ba;
output  wire         memory_mem_ck;
output  wire         memory_mem_ck_n;
output  wire         memory_mem_cke;
output  wire         memory_mem_cs_n;
output  wire         memory_mem_ras_n;
output  wire         memory_mem_cas_n;
output  wire         memory_mem_we_n;
output  wire         memory_mem_reset_n;
inout   wire [31:0]  memory_mem_dq;
inout   wire [3:0]   memory_mem_dqs;
inout   wire [3:0]   memory_mem_dqs_n;
output  wire         memory_mem_odt;
output  wire [3:0]   memory_mem_dm;
input   wire         memory_oct_rzqin;
output  wire         hps_io_hps_io_emac1_inst_TX_CLK;
output  wire         hps_io_hps_io_emac1_inst_TXD0;
output  wire         hps_io_hps_io_emac1_inst_TXD1;
output  wire         hps_io_hps_io_emac1_inst_TXD2;
output  wire         hps_io_hps_io_emac1_inst_TXD3;
input   wire         hps_io_hps_io_emac1_inst_RXD0;
inout   wire         hps_io_hps_io_emac1_inst_MDIO;
output  wire         hps_io_hps_io_emac1_inst_MDC;
input   wire         hps_io_hps_io_emac1_inst_RX_CTL;
output  wire         hps_io_hps_io_emac1_inst_TX_CTL;
input   wire         hps_io_hps_io_emac1_inst_RX_CLK;
input   wire         hps_io_hps_io_emac1_inst_RXD1;
input   wire         hps_io_hps_io_emac1_inst_RXD2;
input   wire         hps_io_hps_io_emac1_inst_RXD3;
inout   wire         hps_io_hps_io_qspi_inst_IO0;
inout   wire         hps_io_hps_io_qspi_inst_IO1;
inout   wire         hps_io_hps_io_qspi_inst_IO2;
inout   wire         hps_io_hps_io_qspi_inst_IO3;
output  wire         hps_io_hps_io_qspi_inst_SS0;
output  wire         hps_io_hps_io_qspi_inst_CLK;
inout   wire         hps_io_hps_io_sdio_inst_CMD;
inout   wire         hps_io_hps_io_sdio_inst_D0;
inout   wire         hps_io_hps_io_sdio_inst_D1;
output  wire         hps_io_hps_io_sdio_inst_CLK;
inout   wire         hps_io_hps_io_sdio_inst_D2;
inout   wire         hps_io_hps_io_sdio_inst_D3;
inout   wire         hps_io_hps_io_usb1_inst_D0;
inout   wire         hps_io_hps_io_usb1_inst_D1;
inout   wire         hps_io_hps_io_usb1_inst_D2;
inout   wire         hps_io_hps_io_usb1_inst_D3;
```

```
inout   wire                                    hps_io_hps_io_usb1_inst_D4;
inout   wire                                    hps_io_hps_io_usb1_inst_D5;
inout   wire                                    hps_io_hps_io_usb1_inst_D6;
inout   wire                                    hps_io_hps_io_usb1_inst_D7;
input   wire                                    hps_io_hps_io_usb1_inst_CLK;
output  wire                                    hps_io_hps_io_usb1_inst_STP;
input   wire                                    hps_io_hps_io_usb1_inst_DIR;
input   wire                                    hps_io_hps_io_usb1_inst_NXT;
output  wire                                    hps_io_hps_io_spim0_inst_CLK;
output  wire                                    hps_io_hps_io_spim0_inst_MOSI;
input   wire                                    hps_io_hps_io_spim0_inst_MISO;
output  wire                                    hps_io_hps_io_spim0_inst_SS0;
output  wire                                    hps_io_hps_io_spim1_inst_CLK;
output  wire                                    hps_io_hps_io_spim1_inst_MOSI;
input   wire                                    hps_io_hps_io_spim1_inst_MISO;
output  wire                                    hps_io_hps_io_spim1_inst_SS0;
input   wire                                    hps_io_hps_io_uart0_inst_RX;
output  wire                                    hps_io_hps_io_uart0_inst_TX;
inout   wire                                    hps_io_hps_io_i2c1_inst_SDA;
inout   wire                                    hps_io_hps_io_i2c1_inst_SCL;
inout   wire                                    hps_io_hps_io_gpio_inst_GPIO00;
//=======================================================
//  REG/WIRE declarations
//=======================================================

//    For Audio CODEC
wire                                            AUD_CTRL_CLK;     //        For Aud

reg [31:0]                                      Cont;
wire                                            VGA_CTRL_CLK;
wire [9:0]                                      mVGA_R;
wire [9:0]                                      mVGA_G;
wire [9:0]                                      mVGA_B;
wire [19:0]                                     mVGA_ADDR;
wire                                            DLY_RST;

//    For VGA Controller
wire                                            mVGA_CLK;
wire [9:0]                                      mRed;
wire [9:0]                                      mGreen;
wire [9:0]                                      mBlue;
wire                                            VGA_Read; //       VGA data request

wire [9:0]                                      recon_VGA_R;
wire [9:0]                                      recon_VGA_G;
wire [9:0]                                      recon_VGA_B;
```

```
    //    For Down Sample
    wire [3:0]                                      Remain;
    wire [9:0]                                      Quotient;

    wire                                            AUD_MUTE;

    // Drive the LEDs with the switches
    assign LED = SW;

    // Make the FPGA reset cause an HPS reset
    reg [19:0]                                      hps_reset_counter = 20'h0;
    reg                                             hps_fpga_reset_n = 0;

    always @(posedge OSC_50_B4A) begin
       if (hps_reset_counter == 20'h fffff) hps_fpga_reset_n <= 1;
       hps_reset_counter <= hps_reset_counter + 1;
    end

    tyrion u0 (
            .clk_clk                                (OSC_50_B4A),
//                  clk.clk
            .reset_reset_n                          (hps_fpga_reset_n),
//                  reset.reset_n
            .memory_mem_a                           (memory_mem_a),
//                  memory.mem_a
            .memory_mem_ba                          (memory_mem_ba),
//                  .mem_ba
            .memory_mem_ck                          (memory_mem_ck),
//                  .mem_ck
            .memory_mem_ck_n                        (memory_mem_ck_n),
//                  .mem_ck_n
            .memory_mem_cke                         (memory_mem_cke),
//                  .mem_cke
            .memory_mem_cs_n                        (memory_mem_cs_n),
//                  .mem_cs_n
            .memory_mem_ras_n                       (memory_mem_ras_n),
//                  .mem_ras_n
            .memory_mem_cas_n                       (memory_mem_cas_n),
//                  .mem_cas_n
            .memory_mem_we_n                        (memory_mem_we_n),
//                  .mem_we_n
            .memory_mem_reset_n                     (memory_mem_reset_n),
//                  .mem_reset_n
            .memory_mem_dq                          (memory_mem_dq),
//                  .mem_dq
```

```
                .memory_mem_dqs                 (memory_mem_dqs),
//                      .mem_dqs
                .memory_mem_dqs_n               (memory_mem_dqs_n),
//                      .mem_dqs_n
                .memory_mem_odt                 (memory_mem_odt),
//                      .mem_odt
                .memory_mem_dm                  (memory_mem_dm),
//                      .mem_dm
                .memory_oct_rzqin               (memory_oct_rzqin),
//                      .oct_rzqin
                .hps_io_hps_io_emac1_inst_TX_CLK
(hps_io_hps_io_emac1_inst_TX_CLK), //
.hps_0_hps_io.hps_io_emac1_inst_TX_CLK
                .hps_io_hps_io_emac1_inst_TXD0
(hps_io_hps_io_emac1_inst_TXD0),    //              .hps_io_emac1_inst_TXD0
                .hps_io_hps_io_emac1_inst_TXD1
(hps_io_hps_io_emac1_inst_TXD1),    //              .hps_io_emac1_inst_TXD1
                .hps_io_hps_io_emac1_inst_TXD2
(hps_io_hps_io_emac1_inst_TXD2),    //              .hps_io_emac1_inst_TXD2
                .hps_io_hps_io_emac1_inst_TXD3
(hps_io_hps_io_emac1_inst_TXD3),    //              .hps_io_emac1_inst_TXD3
                .hps_io_hps_io_emac1_inst_RXD0
(hps_io_hps_io_emac1_inst_RXD0),    //              .hps_io_emac1_inst_RXD0
                .hps_io_hps_io_emac1_inst_MDIO
(hps_io_hps_io_emac1_inst_MDIO),    //              .hps_io_emac1_inst_MDIO
                .hps_io_hps_io_emac1_inst_MDC
(hps_io_hps_io_emac1_inst_MDC),     //              .hps_io_emac1_inst_MDC
                .hps_io_hps_io_emac1_inst_RX_CTL
(hps_io_hps_io_emac1_inst_RX_CTL), //              .hps_io_emac1_inst_RX_CTL
                .hps_io_hps_io_emac1_inst_TX_CTL
(hps_io_hps_io_emac1_inst_TX_CTL), //              .hps_io_emac1_inst_TX_CTL
                .hps_io_hps_io_emac1_inst_RX_CLK
(hps_io_hps_io_emac1_inst_RX_CLK), //              .hps_io_emac1_inst_RX_CLK
                .hps_io_hps_io_emac1_inst_RXD1
(hps_io_hps_io_emac1_inst_RXD1),    //              .hps_io_emac1_inst_RXD1
                .hps_io_hps_io_emac1_inst_RXD2
(hps_io_hps_io_emac1_inst_RXD2),    //              .hps_io_emac1_inst_RXD2
                .hps_io_hps_io_emac1_inst_RXD3
(hps_io_hps_io_emac1_inst_RXD3),    //              .hps_io_emac1_inst_RXD3
                .hps_io_hps_io_qspi_inst_IO0
(hps_io_hps_io_qspi_inst_IO0),      //              .hps_io_qspi_inst_IO0
                .hps_io_hps_io_qspi_inst_IO1
(hps_io_hps_io_qspi_inst_IO1),      //              .hps_io_qspi_inst_IO1
                .hps_io_hps_io_qspi_inst_IO2
(hps_io_hps_io_qspi_inst_IO2),      //              .hps_io_qspi_inst_IO2
                .hps_io_hps_io_qspi_inst_IO3
```

```
(hps_io_hps_io_qspi_inst_IO3),      //                .hps_io_qspi_inst_IO3
            .hps_io_hps_io_qspi_inst_SS0
(hps_io_hps_io_qspi_inst_SS0),      //                .hps_io_qspi_inst_SS0
            .hps_io_hps_io_qspi_inst_CLK
(hps_io_hps_io_qspi_inst_CLK),      //                .hps_io_qspi_inst_CLK
            .hps_io_hps_io_sdio_inst_CMD
(hps_io_hps_io_sdio_inst_CMD),      //                .hps_io_sdio_inst_CMD
            .hps_io_hps_io_sdio_inst_D0
(hps_io_hps_io_sdio_inst_D0),       //                .hps_io_sdio_inst_D0
            .hps_io_hps_io_sdio_inst_D1
(hps_io_hps_io_sdio_inst_D1),       //                .hps_io_sdio_inst_D1
            .hps_io_hps_io_sdio_inst_CLK
(hps_io_hps_io_sdio_inst_CLK),      //                .hps_io_sdio_inst_CLK
            .hps_io_hps_io_sdio_inst_D2
(hps_io_hps_io_sdio_inst_D2),       //                .hps_io_sdio_inst_D2
            .hps_io_hps_io_sdio_inst_D3
(hps_io_hps_io_sdio_inst_D3),       //                .hps_io_sdio_inst_D3
            .hps_io_hps_io_usb1_inst_D0
(hps_io_hps_io_usb1_inst_D0),       //                .hps_io_usb1_inst_D0
            .hps_io_hps_io_usb1_inst_D1
(hps_io_hps_io_usb1_inst_D1),       //                .hps_io_usb1_inst_D1
            .hps_io_hps_io_usb1_inst_D2
(hps_io_hps_io_usb1_inst_D2),       //                .hps_io_usb1_inst_D2
            .hps_io_hps_io_usb1_inst_D3
(hps_io_hps_io_usb1_inst_D3),       //                .hps_io_usb1_inst_D3
            .hps_io_hps_io_usb1_inst_D4
(hps_io_hps_io_usb1_inst_D4),       //                .hps_io_usb1_inst_D4
            .hps_io_hps_io_usb1_inst_D5
(hps_io_hps_io_usb1_inst_D5),       //                .hps_io_usb1_inst_D5
            .hps_io_hps_io_usb1_inst_D6
(hps_io_hps_io_usb1_inst_D6),       //                .hps_io_usb1_inst_D6
            .hps_io_hps_io_usb1_inst_D7
(hps_io_hps_io_usb1_inst_D7),       //                .hps_io_usb1_inst_D7
            .hps_io_hps_io_usb1_inst_CLK
(hps_io_hps_io_usb1_inst_CLK),      //                .hps_io_usb1_inst_CLK
            .hps_io_hps_io_usb1_inst_STP
(hps_io_hps_io_usb1_inst_STP),      //                .hps_io_usb1_inst_STP
            .hps_io_hps_io_usb1_inst_DIR
(hps_io_hps_io_usb1_inst_DIR),      //                .hps_io_usb1_inst_DIR
            .hps_io_hps_io_usb1_inst_NXT
(hps_io_hps_io_usb1_inst_NXT),      //                .hps_io_usb1_inst_NXT
            .hps_io_hps_io_spim0_inst_CLK
(hps_io_hps_io_spim0_inst_CLK),     //                .hps_io_spim0_inst_CLK
            .hps_io_hps_io_spim0_inst_MOSI
(hps_io_hps_io_spim0_inst_MOSI),    //                .hps_io_spim0_inst_MOSI
            .hps_io_hps_io_spim0_inst_MISO
```

```
(hps_io_hps_io_spim0_inst_MISO),    //                .hps_io_spim0_inst_MISO
            .hps_io_hps_io_spim0_inst_SS0
(hps_io_hps_io_spim0_inst_SS0),     //                .hps_io_spim0_inst_SS0
            .hps_io_hps_io_spim1_inst_CLK
(hps_io_hps_io_spim1_inst_CLK),     //                .hps_io_spim1_inst_CLK
            .hps_io_hps_io_spim1_inst_MOSI
(hps_io_hps_io_spim1_inst_MOSI),    //                .hps_io_spim1_inst_MOSI
            .hps_io_hps_io_spim1_inst_MISO
(hps_io_hps_io_spim1_inst_MISO),    //                .hps_io_spim1_inst_MISO
            .hps_io_hps_io_spim1_inst_SS0
(hps_io_hps_io_spim1_inst_SS0),     //                .hps_io_spim1_inst_SS0
            .hps_io_hps_io_uart0_inst_RX
(hps_io_hps_io_uart0_inst_RX),      //                .hps_io_uart0_inst_RX
            .hps_io_hps_io_uart0_inst_TX
(hps_io_hps_io_uart0_inst_TX),      //                .hps_io_uart0_inst_TX
            .hps_io_hps_io_i2c1_inst_SDA
(hps_io_hps_io_i2c1_inst_SDA),      //                .hps_io_i2c1_inst_SDA
            .hps_io_hps_io_i2c1_inst_SCL
(hps_io_hps_io_i2c1_inst_SCL)       //                .hps_io_i2c1_inst_SCL
            );

endmodule
# TCL File Generated by Component Editor 13.1.1
# Thu May 14 15:03:18 EDT 2015
# DO NOT MODIFY


#
# tyrion_dev "TYRION" v1.2
#  2015.05.14.15:03:18
#
#

#
# request TCL package from ACDS 13.1
#
package require -exact qsys 13.1


#
# module tyrion_dev
#
set_module_property DESCRIPTION ""
set_module_property NAME tyrion_dev
set_module_property VERSION 1.2
set_module_property INTERNAL false
```

```
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME TYRION
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property ANALYZE_HDL AUTO
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false


#
# file sets
#
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL svd_wrapper_rtl
set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
add_fileset_file tyrion_rtl.v VERILOG PATH ../syn/tyrion/tyrion_rtl.v TOP_LEVEL_FILE


#
# parameters
#


#
# display items
#


#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true
set_interface_property clock EXPORT_OF ""
set_interface_property clock PORT_NAME_MAP ""
set_interface_property clock CMSIS_SVD_VARIABLES ""
set_interface_property clock SVD_ADDRESS_GROUP ""

add_interface_port clock clk clk Input 1


#
# connection point reset_sink
#
```

```
add_interface reset_sink reset end
set_interface_property reset_sink associatedClock clock
set_interface_property reset_sink synchronousEdges DEASSERT
set_interface_property reset_sink ENABLED true
set_interface_property reset_sink EXPORT_OF ""
set_interface_property reset_sink PORT_NAME_MAP ""
set_interface_property reset_sink CMSIS_SVD_VARIABLES ""
set_interface_property reset_sink SVD_ADDRESS_GROUP ""

add_interface_port reset_sink rst_in reset Input 1


#
# connection point avalon_streaming_sink
#
add_interface avalon_streaming_sink avalon_streaming end
set_interface_property avalon_streaming_sink associatedClock clock
set_interface_property avalon_streaming_sink associatedReset reset_sink
set_interface_property avalon_streaming_sink dataBitsPerSymbol 32
set_interface_property avalon_streaming_sink errorDescriptor ""
set_interface_property avalon_streaming_sink firstSymbolInHighOrderBits true
set_interface_property avalon_streaming_sink maxChannel 0
set_interface_property avalon_streaming_sink readyLatency 0
set_interface_property avalon_streaming_sink ENABLED true
set_interface_property avalon_streaming_sink EXPORT_OF ""
set_interface_property avalon_streaming_sink PORT_NAME_MAP ""
set_interface_property avalon_streaming_sink CMSIS_SVD_VARIABLES ""
set_interface_property avalon_streaming_sink SVD_ADDRESS_GROUP ""

add_interface_port avalon_streaming_sink data_in_valid valid Input 1
add_interface_port avalon_streaming_sink data_in_data data Input 32
add_interface_port avalon_streaming_sink data_in_ready ready Output 1


#
# connection point avalon_streaming_source
#
add_interface avalon_streaming_source avalon_streaming start
set_interface_property avalon_streaming_source associatedClock clock
set_interface_property avalon_streaming_source associatedReset reset_sink
set_interface_property avalon_streaming_source dataBitsPerSymbol 32
set_interface_property avalon_streaming_source errorDescriptor ""
set_interface_property avalon_streaming_source firstSymbolInHighOrderBits true
set_interface_property avalon_streaming_source maxChannel 0
set_interface_property avalon_streaming_source readyLatency 0
set_interface_property avalon_streaming_source ENABLED true
```

```
set_interface_property avalon_streaming_source EXPORT_OF ""
set_interface_property avalon_streaming_source PORT_NAME_MAP ""
set_interface_property avalon_streaming_source CMSIS_SVD_VARIABLES ""
set_interface_property avalon_streaming_source SVD_ADDRESS_GROUP ""

add_interface_port avalon_streaming_source data_out_ready ready Input 1
add_interface_port avalon_streaming_source data_out_valid valid Output 1
add_interface_port avalon_streaming_source data_out_data data Output 32

# -------------------------------------------------------------------------- #
#
# Copyright (C) 1991-2014 Altera Corporation
# Your use of Altera Corporation's design tools, logic functions
# and other software and tools, and its AMPP partner logic
# functions, and any output files from any of the foregoing
# (including device programming or simulation files), and any
# associated documentation or information are expressly subject
# to the terms and conditions of the Altera Program License
# Subscription Agreement, Altera MegaCore Function License
# Agreement, or other applicable license agreement, including,
# without limitation, that your use is for the sole purpose of
# programming logic devices manufactured by Altera and sold by
# Altera or its authorized distributors.  Please refer to the
# applicable agreement for further details.
#
# -------------------------------------------------------------------------- #
#
# Quartus II 32-bit
# Version 13.1.2 Build 173 01/15/2014 SJ Full Version
# Date created = 21:21:35  February 18, 2014
#
# -------------------------------------------------------------------------- #

QUARTUS_VERSION = "13.1"
DATE = "21:21:35  February 18, 2014"

# Revisions

PROJECT_REVISION = "SoCKit_Top"
<?xml version="1.0" encoding="UTF-8"?>
<system name="$${FILENAME}">
 <component
    name="$${FILENAME}"
    displayName="$${FILENAME}"
    version="1.0"
    description=""
```

```
   tags=""
   categories="System" />
 <parameter name="bonusData"><![CDATA[bonusData
{
   element $${FILENAME}
   {
   }
   element clk_0
   {
      datum _sortIndex
      {
         value = "0";
         type = "int";
      }
   }
   element hps_0.f2h_axi_slave
   {
      datum baseAddress
      {
         value = "4294967296";
         type = "String";
      }
   }
   element fifo_0
   {
      datum _sortIndex
      {
         value = "3";
         type = "int";
      }
   }
   element fifo_1
   {
      datum _sortIndex
      {
         value = "4";
         type = "int";
      }
   }
   element hps_0
   {
      datum _sortIndex
      {
         value = "1";
         type = "int";
      }
```

```
}
element fifo_0.in
{
   datum baseAddress
   {
      value = "8";
      type = "String";
   }
}
element fifo_0.in_csr
{
   datum baseAddress
   {
      value = "64";
      type = "String";
   }
}
element fifo_1.in_csr
{
   datum baseAddress
   {
      value = "32";
      type = "String";
   }
}
element master_0
{
   datum _sortIndex
   {
      value = "2";
      type = "int";
   }
}
element fifo_1.out
{
   datum baseAddress
   {
      value = "0";
      type = "String";
   }
}
element tyrion_dev_0
{
   datum _sortIndex
   {
      value = "5";
```

```
            type = "int";
        }
    }
}
]]></parameter>
 <parameter name="clockCrossingAdapter" value="HANDSHAKE" />
 <parameter name="device" value="5CSXFC6D6F31C8ES" />
 <parameter name="deviceFamily" value="Cyclone V" />
 <parameter name="deviceSpeedGrade" value="8_H6" />
 <parameter name="fabricMode" value="QSYS" />
 <parameter name="generateLegacySim" value="false" />
 <parameter name="generationId" value="0" />
 <parameter name="globalResetBus" value="false" />
 <parameter name="hdlLanguage" value="VERILOG" />
 <parameter name="maxAdditionalLatency" value="1" />
 <parameter name="projectName" value="tyrion.qpf" />
 <parameter name="sopcBorderPoints" value="false" />
 <parameter name="systemHash" value="0" />
 <parameter name="timeStamp" value="0" />
 <parameter name="useTestBenchNamingPattern" value="false" />
 <instanceScript></instanceScript>
 <interface name="clk" internal="clk_0.clk_in" type="clock" dir="end" />
 <interface name="reset" internal="clk_0.clk_in_reset" type="reset" dir="end" />
 <interface name="memory" internal="hps_0.memory" type="conduit" dir="end" />
 <interface name="hps_io" internal="hps_0.hps_io" type="conduit" dir="end" />
 <module kind="clock_source" version="13.1" enabled="1" name="clk_0">
  <parameter name="clockFrequency" value="50000000" />
  <parameter name="clockFrequencyKnown" value="true" />
  <parameter name="inputClockFrequency" value="0" />
  <parameter name="resetSynchronousEdges" value="NONE" />
 </module>
 <module kind="altera_hps" version="13.1.1" enabled="1" name="hps_0">
  <parameter name="MEM_VENDOR" value="Micron" />
  <parameter name="MEM_FORMAT" value="DISCRETE" />
  <parameter name="RDIMM_CONFIG" value="0000000000000000" />
  <parameter name="LRDIMM_EXTENDED_CONFIG">0x000000000000000000</parameter>
  <parameter name="DISCRETE_FLY_BY" value="true" />
  <parameter name="DEVICE_DEPTH" value="1" />
  <parameter name="MEM_MIRROR_ADDRESSING" value="0" />
  <parameter name="MEM_CLK_FREQ_MAX" value="800.0" />
  <parameter name="MEM_ROW_ADDR_WIDTH" value="15" />
  <parameter name="MEM_COL_ADDR_WIDTH" value="10" />
  <parameter name="MEM_DQ_WIDTH" value="32" />
  <parameter name="MEM_DQ_PER_DQS" value="8" />
  <parameter name="MEM_BANKADDR_WIDTH" value="3" />
  <parameter name="MEM_IF_DM_PINS_EN" value="true" />
```

```
<parameter name="MEM_IF_DQSN_EN" value="true" />
<parameter name="MEM_NUMBER_OF_DIMMS" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DIMM" value="1" />
<parameter name="MEM_NUMBER_OF_RANKS_PER_DEVICE" value="1" />
<parameter name="MEM_RANK_MULTIPLICATION_FACTOR" value="1" />
<parameter name="MEM_CK_WIDTH" value="1" />
<parameter name="MEM_CS_WIDTH" value="1" />
<parameter name="MEM_CLK_EN_WIDTH" value="1" />
<parameter name="ALTMEMPHY_COMPATIBLE_MODE" value="false" />
<parameter name="NEXTGEN" value="true" />
<parameter name="MEM_IF_BOARD_BASE_DELAY" value="10" />
<parameter name="MEM_IF_SIM_VALID_WINDOW" value="0" />
<parameter name="MEM_GUARANTEED_WRITE_INIT" value="false" />
<parameter name="MEM_VERBOSE" value="true" />
<parameter name="PINGPONGPHY_EN" value="false" />
<parameter name="REFRESH_BURST_VALIDATION" value="false" />
<parameter name="MEM_BL" value="OTF" />
<parameter name="MEM_BT" value="Sequential" />
<parameter name="MEM_ASR" value="Manual" />
<parameter name="MEM_SRT" value="Normal" />
<parameter name="MEM_PD" value="DLL off" />
<parameter name="MEM_DRV_STR" value="RZQ/7" />
<parameter name="MEM_DLL_EN" value="true" />
<parameter name="MEM_RTT_NOM" value="RZQ/4" />
<parameter name="MEM_RTT_WR" value="RZQ/4" />
<parameter name="MEM_WTCL" value="8" />
<parameter name="MEM_ATCL" value="Disabled" />
<parameter name="MEM_TCL" value="11" />
<parameter name="MEM_AUTO_LEVELING_MODE" value="true" />
<parameter name="MEM_USER_LEVELING_MODE" value="Leveling" />
<parameter name="MEM_INIT_EN" value="false" />
<parameter name="MEM_INIT_FILE" value="" />
<parameter name="DAT_DATA_WIDTH" value="32" />
<parameter name="TIMING_TIS" value="180" />
<parameter name="TIMING_TIH" value="140" />
<parameter name="TIMING_TDS" value="30" />
<parameter name="TIMING_TDH" value="65" />
<parameter name="TIMING_TDQSQ" value="125" />
<parameter name="TIMING_TQHS" value="300" />
<parameter name="TIMING_TQH" value="0.38" />
<parameter name="TIMING_TDQSCK" value="255" />
<parameter name="TIMING_TDQSCKDS" value="450" />
<parameter name="TIMING_TDQSCKDM" value="900" />
<parameter name="TIMING_TDQSCKDL" value="1200" />
<parameter name="TIMING_TDQSS" value="0.25" />
<parameter name="TIMING_TDQSH" value="0.35" />
```

```
<parameter name="TIMING_TQSH" value="0.4" />
<parameter name="TIMING_TDSH" value="0.2" />
<parameter name="TIMING_TDSS" value="0.2" />
<parameter name="MEM_TINIT_US" value="500" />
<parameter name="MEM_TMRD_CK" value="4" />
<parameter name="MEM_TRAS_NS" value="35.0" />
<parameter name="MEM_TRCD_NS" value="13.75" />
<parameter name="MEM_TRP_NS" value="13.75" />
<parameter name="MEM_TREFI_US" value="7.8" />
<parameter name="MEM_TRFC_NS" value="260.0" />
<parameter name="CFG_TCCD_NS" value="2.5" />
<parameter name="MEM_TWR_NS" value="15.0" />
<parameter name="MEM_TWTR" value="4" />
<parameter name="MEM_TFAW_NS" value="30.0" />
<parameter name="MEM_TRRD_NS" value="7.5" />
<parameter name="MEM_TRTP_NS" value="7.5" />
<parameter name="POWER_OF_TWO_BUS" value="false" />
<parameter name="SOPC_COMPAT_RESET" value="false" />
<parameter name="AVL_MAX_SIZE" value="4" />
<parameter name="BYTE_ENABLE" value="true" />
<parameter name="ENABLE_CTRL_AVALON_INTERFACE" value="true" />
<parameter name="CTL_DEEP_POWERDN_EN" value="false" />
<parameter name="CTL_SELF_REFRESH_EN" value="false" />
<parameter name="AUTO_POWERDN_EN" value="false" />
<parameter name="AUTO_PD_CYCLES" value="0" />
<parameter name="CTL_USR_REFRESH_EN" value="false" />
<parameter name="CTL_AUTOPCH_EN" value="false" />
<parameter name="CTL_ZQCAL_EN" value="false" />
<parameter name="ADDR_ORDER" value="0" />
<parameter name="CTL_LOOK_AHEAD_DEPTH" value="4" />
<parameter name="CONTROLLER_LATENCY" value="5" />
<parameter name="CFG_REORDER_DATA" value="true" />
<parameter name="STARVE_LIMIT" value="10" />
<parameter name="CTL_CSR_ENABLED" value="false" />
<parameter name="CTL_CSR_CONNECTION" value="INTERNAL_JTAG" />
<parameter name="CTL_ECC_ENABLED" value="false" />
<parameter name="CTL_HRB_ENABLED" value="false" />
<parameter name="CTL_ECC_AUTO_CORRECTION_ENABLED" value="false" />
<parameter name="MULTICAST_EN" value="false" />
<parameter name="CTL_DYNAMIC_BANK_ALLOCATION" value="false" />
<parameter name="CTL_DYNAMIC_BANK_NUM" value="4" />
<parameter name="DEBUG_MODE" value="false" />
<parameter name="ENABLE_BURST_MERGE" value="false" />
<parameter name="CTL_ENABLE_BURST_INTERRUPT" value="false" />
<parameter name="CTL_ENABLE_BURST_TERMINATE" value="false" />
<parameter name="LOCAL_ID_WIDTH" value="8" />
```

```
<parameter name="WRBUFFER_ADDR_WIDTH" value="6" />
<parameter name="MAX_PENDING_WR_CMD" value="16" />
<parameter name="MAX_PENDING_RD_CMD" value="32" />
<parameter name="USE_MM_ADAPTOR" value="true" />
<parameter name="USE_AXI_ADAPTOR" value="false" />
<parameter name="HCX_COMPAT_MODE" value="false" />
<parameter name="CTL_CMD_QUEUE_DEPTH" value="8" />
<parameter name="CTL_CSR_READ_ONLY" value="1" />
<parameter name="CFG_DATA_REORDERING_TYPE" value="INTER_BANK" />
<parameter name="NUM_OF_PORTS" value="1" />
<parameter name="ENABLE_BONDING" value="false" />
<parameter name="ENABLE_USER_ECC" value="false" />
<parameter name="AVL_DATA_WIDTH_PORT" value="32,32,32,32,32,32" />
<parameter name="PRIORITY_PORT" value="1,1,1,1,1,1" />
<parameter name="WEIGHT_PORT" value="0,0,0,0,0,0" />
<parameter name="CPORT_TYPE_PORT">Bidirectional,Bidirectional,Bidirectional,Bidirectio
<parameter name="ENABLE_EMIT_BFM_MASTER" value="false" />
<parameter name="FORCE_SEQUENCER_TCL_DEBUG_MODE" value="false" />
<parameter name="ENABLE_SEQUENCER_MARGINING_ON_BY_DEFAULT" value="false" />
<parameter name="REF_CLK_FREQ" value="25.0" />
<parameter name="REF_CLK_FREQ_PARAM_VALID" value="false" />
<parameter name="REF_CLK_FREQ_MIN_PARAM" value="0.0" />
<parameter name="REF_CLK_FREQ_MAX_PARAM" value="0.0" />
<parameter name="PLL_DR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_DR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_DR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_DR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_DR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_MEM_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_MEM_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_MEM_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_MEM_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_WRITE_CLK_MULT_PARAM" value="0" />
```

```
<parameter name="PLL_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_ADDR_CMD_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_ADDR_CMD_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_ADDR_CMD_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_HALF_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_HALF_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_HALF_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_NIOS_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_NIOS_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_NIOS_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_CONFIG_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_CONFIG_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_CONFIG_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_P2C_READ_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_P2C_READ_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_P2C_READ_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_C2P_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_C2P_WRITE_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_C2P_WRITE_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_HR_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_HR_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_PHASE_PS_PARAM" value="0" />
<parameter name="PLL_HR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_HR_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_HR_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_PARAM" value="0.0" />
<parameter name="PLL_AFI_PHY_CLK_FREQ_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_PARAM" value="0" />
```

```
<parameter name="PLL_AFI_PHY_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
<parameter name="PLL_AFI_PHY_CLK_MULT_PARAM" value="0" />
<parameter name="PLL_AFI_PHY_CLK_DIV_PARAM" value="0" />
<parameter name="PLL_CLK_PARAM_VALID" value="false" />
<parameter name="ENABLE_EXTRA_REPORTING" value="false" />
<parameter name="NUM_EXTRA_REPORT_PATH" value="10" />
<parameter name="ENABLE_ISS_PROBES" value="false" />
<parameter name="CALIB_REG_WIDTH" value="8" />
<parameter name="USE_SEQUENCER_BFM" value="false" />
<parameter name="DEFAULT_FAST_SIM_MODEL" value="true" />
<parameter name="PLL_SHARING_MODE" value="None" />
<parameter name="NUM_PLL_SHARING_INTERFACES" value="1" />
<parameter name="EXPORT_AFI_HALF_CLK" value="false" />
<parameter name="ABSTRACT_REAL_COMPARE_TEST" value="false" />
<parameter name="INCLUDE_BOARD_DELAY_MODEL" value="false" />
<parameter name="INCLUDE_MULTIRANK_BOARD_DELAY_MODEL" value="false" />
<parameter name="USE_FAKE_PHY" value="false" />
<parameter name="FORCE_MAX_LATENCY_COUNT_WIDTH" value="0" />
<parameter name="ENABLE_NON_DESTRUCTIVE_CALIB" value="false" />
<parameter name="TRACKING_ERROR_TEST" value="false" />
<parameter name="TRACKING_WATCH_TEST" value="false" />
<parameter name="MARGIN_VARIATION_TEST" value="false" />
<parameter name="EXTRA_SETTINGS" value="" />
<parameter name="MEM_DEVICE" value="MISSING_MODEL" />
<parameter name="FORCE_SYNTHESIS_LANGUAGE" value="" />
<parameter name="FORCED_NUM_WRITE_FR_CYCLE_SHIFTS" value="0" />
<parameter name="SEQUENCER_TYPE" value="NIOS" />
<parameter name="ADVERTIZE_SEQUENCER_SW_BUILD_FILES" value="false" />
<parameter name="FORCED_NON_LDC_ADDR_CMD_MEM_CK_INVERT" value="false" />
<parameter name="PHY_ONLY" value="false" />
<parameter name="SEQ_MODE" value="0" />
<parameter name="ADVANCED_CK_PHASES" value="false" />
<parameter name="COMMAND_PHASE" value="0.0" />
<parameter name="MEM_CK_PHASE" value="0.0" />
<parameter name="P2C_READ_CLOCK_ADD_PHASE" value="0.0" />
<parameter name="C2P_WRITE_CLOCK_ADD_PHASE" value="0.0" />
<parameter name="ACV_PHY_CLK_ADD_FR_PHASE" value="0.0" />
<parameter name="MEM_VOLTAGE" value="1.5V DDR3" />
<parameter name="PLL_LOCATION" value="Top_Bottom" />
<parameter name="SKIP_MEM_INIT" value="true" />
<parameter name="READ_DQ_DQS_CLOCK_SOURCE" value="INVERTED_DQS_BUS" />
<parameter name="DQ_INPUT_REG_USE_CLKN" value="false" />
<parameter name="DQS_DQSN_MODE" value="DIFFERENTIAL" />
<parameter name="AFI_DEBUG_INFO_WIDTH" value="32" />
<parameter name="CALIBRATION_MODE" value="Skip" />
<parameter name="NIOS_ROM_DATA_WIDTH" value="32" />
```

```
<parameter name="READ_FIFO_SIZE" value="8" />
<parameter name="PHY_CSR_ENABLED" value="false" />
<parameter name="PHY_CSR_CONNECTION" value="INTERNAL_JTAG" />
<parameter name="USER_DEBUG_LEVEL" value="1" />
<parameter name="TIMING_BOARD_DERATE_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_CK_CKN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_AC_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_DQS_DQSN_SLEW_RATE" value="2.0" />
<parameter name="TIMING_BOARD_DQ_SLEW_RATE" value="1.0" />
<parameter name="TIMING_BOARD_TIS" value="0.0" />
<parameter name="TIMING_BOARD_TIH" value="0.0" />
<parameter name="TIMING_BOARD_TDS" value="0.0" />
<parameter name="TIMING_BOARD_TDH" value="0.0" />
<parameter name="TIMING_BOARD_ISI_METHOD" value="AUTO" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_SU" value="0.0" />
<parameter name="TIMING_BOARD_AC_EYE_REDUCTION_H" value="0.0" />
<parameter name="TIMING_BOARD_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_DELTA_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="TIMING_BOARD_READ_DQ_EYE_REDUCTION" value="0.0" />
<parameter name="TIMING_BOARD_DELTA_READ_DQS_ARRIVAL_TIME" value="0.0" />
<parameter name="PACKAGE_DESKEW" value="false" />
<parameter name="AC_PACKAGE_DESKEW" value="false" />
<parameter name="TIMING_BOARD_MAX_CK_DELAY" value="0.03" />
<parameter name="TIMING_BOARD_MAX_DQS_DELAY" value="0.02" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MIN" value="0.09" />
<parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MAX" value="0.16" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DIMMS" value="0.05" />
<parameter name="TIMING_BOARD_SKEW_WITHIN_DQS" value="0.01" />
<parameter name="TIMING_BOARD_SKEW_BETWEEN_DQS" value="0.08" />
<parameter name="TIMING_BOARD_DQ_TO_DQS_SKEW" value="0.0" />
<parameter name="TIMING_BOARD_AC_SKEW" value="0.03" />
<parameter name="TIMING_BOARD_AC_TO_CK_SKEW" value="0.0" />
<parameter name="RATE" value="Full" />
<parameter name="MEM_CLK_FREQ" value="400.0" />
<parameter name="USE_MEM_CLK_FREQ" value="false" />
<parameter name="FORCE_DQS_TRACKING" value="AUTO" />
<parameter name="FORCE_SHADOW_REGS" value="AUTO" />
<parameter name="MRS_MIRROR_PING_PONG_ATSO" value="false" />
<parameter name="SYS_INFO_DEVICE_FAMILY" value="Cyclone V" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM_VALID" value="false" />
<parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM" value="" />
<parameter name="DEVICE_FAMILY_PARAM" value="" />
<parameter name="SPEED_GRADE" value="7" />
<parameter name="IS_ES_DEVICE" value="false" />
<parameter name="DISABLE_CHILD_MESSAGING" value="false" />
<parameter name="HARD_EMIF" value="true" />
```

```
<parameter name="HHP_HPS" value="true" />
<parameter name="HHP_HPS_VERIFICATION" value="false" />
<parameter name="HHP_HPS_SIMULATION" value="false" />
<parameter name="HPS_PROTOCOL" value="DDR3" />
<parameter name="CUT_NEW_FAMILY_TIMING" value="true" />
<parameter name="ENABLE_EXPORT_SEQ_DEBUG_BRIDGE" value="false" />
<parameter name="CORE_DEBUG_CONNECTION" value="EXPORT" />
<parameter name="ADD_EXTERNAL_SEQ_DEBUG_NIOS" value="false" />
<parameter name="ED_EXPORT_SEQ_DEBUG" value="false" />
<parameter name="ADD_EFFICIENCY_MONITOR" value="false" />
<parameter name="ENABLE_ABS_RAM_MEM_INIT" value="false" />
<parameter name="ABS_RAM_MEM_INIT_FILENAME" value="meminit" />
<parameter name="DLL_SHARING_MODE" value="None" />
<parameter name="NUM_DLL_SHARING_INTERFACES" value="1" />
<parameter name="OCT_SHARING_MODE" value="None" />
<parameter name="NUM_OCT_SHARING_INTERFACES" value="1" />
<parameter name="MPU_EVENTS_Enable" value="false" />
<parameter name="GP_Enable" value="false" />
<parameter name="DEBUGAPB_Enable" value="false" />
<parameter name="STM_Enable" value="false" />
<parameter name="CTI_Enable" value="false" />
<parameter name="TPIUFPGA_Enable" value="false" />
<parameter name="BOOTFROMFPGA_Enable" value="false" />
<parameter name="TEST_Enable" value="false" />
<parameter name="HLGPI_Enable" value="false" />
<parameter name="BSEL_EN" value="false" />
<parameter name="BSEL" value="1" />
<parameter name="CSEL_EN" value="false" />
<parameter name="CSEL" value="0" />
<parameter name="F2S_Width" value="2" />
<parameter name="S2F_Width" value="2" />
<parameter name="LWH2F_Enable" value="true" />
<parameter name="F2SDRAM_Type" value="" />
<parameter name="F2SDRAM_Width" value="" />
<parameter name="BONDING_OUT_ENABLED" value="false" />
<parameter name="S2FCLK_COLDRST_Enable" value="false" />
<parameter name="S2FCLK_PENDINGRST_Enable" value="false" />
<parameter name="F2SCLK_DBGRST_Enable" value="false" />
<parameter name="F2SCLK_WARMRST_Enable" value="false" />
<parameter name="F2SCLK_COLDRST_Enable" value="false" />
<parameter name="DMA_Enable">No,No,No,No,No,No,No,No</parameter>
<parameter name="F2SINTERRUPT_Enable" value="true" />
<parameter name="S2FINTERRUPT_CAN_Enable" value="false" />
<parameter name="S2FINTERRUPT_CLOCKPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_CTI_Enable" value="false" />
<parameter name="S2FINTERRUPT_DMA_Enable" value="false" />
```

```
<parameter name="S2FINTERRUPT_EMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_FPGAMANAGER_Enable" value="false" />
<parameter name="S2FINTERRUPT_GPIO_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CEMAC_Enable" value="false" />
<parameter name="S2FINTERRUPT_I2CPERIPHERAL_Enable" value="false" />
<parameter name="S2FINTERRUPT_L4TIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_NAND_Enable" value="false" />
<parameter name="S2FINTERRUPT_OSCTIMER_Enable" value="false" />
<parameter name="S2FINTERRUPT_QSPI_Enable" value="false" />
<parameter name="S2FINTERRUPT_SDMMC_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPIMASTER_Enable" value="false" />
<parameter name="S2FINTERRUPT_SPISLAVE_Enable" value="false" />
<parameter name="S2FINTERRUPT_UART_Enable" value="false" />
<parameter name="S2FINTERRUPT_USB_Enable" value="false" />
<parameter name="S2FINTERRUPT_WATCHDOG_Enable" value="false" />
<parameter name="EMAC0_PinMuxing" value="Unused" />
<parameter name="EMAC0_Mode" value="N/A" />
<parameter name="EMAC1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="EMAC1_Mode" value="RGMII" />
<parameter name="NAND_PinMuxing" value="Unused" />
<parameter name="NAND_Mode" value="N/A" />
<parameter name="QSPI_PinMuxing" value="HPS I/O Set 0" />
<parameter name="QSPI_Mode" value="1 SS" />
<parameter name="SDIO_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SDIO_Mode" value="4-bit Data" />
<parameter name="USB0_PinMuxing" value="Unused" />
<parameter name="USB0_Mode" value="N/A" />
<parameter name="USB1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="USB1_Mode" value="SDR" />
<parameter name="SPIM0_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SPIM0_Mode" value="Single Slave Select" />
<parameter name="SPIM1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="SPIM1_Mode" value="Single Slave Select" />
<parameter name="SPIS0_PinMuxing" value="Unused" />
<parameter name="SPIS0_Mode" value="N/A" />
<parameter name="SPIS1_PinMuxing" value="Unused" />
<parameter name="SPIS1_Mode" value="N/A" />
<parameter name="UART0_PinMuxing" value="HPS I/O Set 0" />
<parameter name="UART0_Mode" value="No Flow Control" />
<parameter name="UART1_PinMuxing" value="Unused" />
<parameter name="UART1_Mode" value="N/A" />
<parameter name="I2C0_PinMuxing" value="Unused" />
<parameter name="I2C0_Mode" value="N/A" />
<parameter name="I2C1_PinMuxing" value="HPS I/O Set 0" />
<parameter name="I2C1_Mode" value="I2C" />
<parameter name="I2C2_PinMuxing" value="Unused" />
```

```
<parameter name=”I2C2_Mode” value=”N/A” />
<parameter name=”I2C3_PinMuxing” value=”Unused” />
<parameter name=”I2C3_Mode” value=”N/A” />
<parameter name=”CAN0_PinMuxing” value=”Unused” />
<parameter name=”CAN0_Mode” value=”N/A” />
<parameter name=”CAN1_PinMuxing” value=”Unused” />
<parameter name=”CAN1_Mode” value=”N/A” />
<parameter name=”TRACE_PinMuxing” value=”Unused” />
<parameter name=”TRACE_Mode” value=”N/A” />
<parameter name=”GPIO_Enable”>No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,N
<parameter name=”LOANIO_Enable”>No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,No,
<parameter name=”S2FCLK_USER0CLK_Enable” value=”false” />
<parameter name=”S2FCLK_USER0CLK_FREQ” value=”100” />
<parameter name=”S2FCLK_USER1CLK_Enable” value=”false” />
<parameter name=”S2FCLK_USER1CLK_FREQ” value=”100” />
<parameter name=”S2FCLK_USER2CLK_Enable” value=”false” />
<parameter name=”S2FCLK_USER2CLK_FREQ” value=”100” />
<parameter name=”F2SCLK_PERIPHCLK_Enable” value=”false” />
<parameter name=”F2SCLK_PERIPHCLK_FREQ” value=”100” />
<parameter name=”F2SCLK_SDRAMCLK_Enable” value=”false” />
<parameter name=”F2SCLK_SDRAMCLK_FREQ” value=”100” />
<parameter name=”F2H_AXI_CLOCK_FREQ” value=”50000000” />
<parameter name=”H2F_AXI_CLOCK_FREQ” value=”50000000” />
<parameter name=”H2F_LW_AXI_CLOCK_FREQ” value=”50000000” />
<parameter name=”F2H_SDRAM0_CLOCK_FREQ” value=”100” />
<parameter name=”F2H_SDRAM1_CLOCK_FREQ” value=”100” />
<parameter name=”F2H_SDRAM2_CLOCK_FREQ” value=”100” />
<parameter name=”F2H_SDRAM3_CLOCK_FREQ” value=”100” />
<parameter name=”F2H_SDRAM4_CLOCK_FREQ” value=”100” />
<parameter name=”F2H_SDRAM5_CLOCK_FREQ” value=”100” />
<parameter name=”H2F_CTI_CLOCK_FREQ” value=”100” />
<parameter name=”H2F_TPIU_CLOCK_IN_FREQ” value=”100” />
<parameter name=”H2F_DEBUG_APB_CLOCK_FREQ” value=”100” />
<parameter
   name=”FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC_PTP_REF_CLOCK”
   value=”100” />
<parameter name=”FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_RX_CLK_IN” value=”100” />
<parameter name=”FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC0_TX_CLK_IN” value=”100” />
<parameter name=”FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_MD_CLK” value=”100” />
<parameter name=”FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC0_GTX_CLK” value=”100” />
<parameter name=”FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_RX_CLK_IN” value=”100” />
<parameter name=”FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_TX_CLK_IN” value=”100” />
<parameter name=”FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_MD_CLK” value=”100” />
<parameter name=”FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_EMAC1_GTX_CLK” value=”100” />
<parameter name=”FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_QSPI_SCLK_OUT” value=”100” />
<parameter name=”FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SDIO_CLK_IN” value=”100” />
```

```
 <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SDIO_CCLK" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB0_CLK_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_USB1_CLK_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM0_SCLK_OUT" value="100" />
 <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_SPIM1_SCLK_OUT" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS0_SCLK_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_SPIS1_SCLK_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C0_SCL_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C0_CLK" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C1_SCL_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C1_CLK" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C2_SCL_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C2_CLK" value="100" />
 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_I2C3_SCL_IN" value="100" />
 <parameter name="FPGA_PERIPHERAL_OUTPUT_CLOCK_FREQ_I2C3_CLK" value="100" />
 <parameter name="device_name" value="5CSXFC6D6F31C8ES" />
 <parameter
    name="quartus_ini_hps_ip_enable_all_peripheral_fpga_interfaces"
    value="false" />
 <parameter
    name="quartus_ini_hps_ip_enable_emac0_peripheral_fpga_interface"
    value="false" />
 <parameter name="quartus_ini_hps_ip_enable_test_interface" value="false" />
 <parameter name="quartus_ini_hps_ip_fast_f2sdram_sim_model" value="false" />
 <parameter name="quartus_ini_hps_ip_suppress_sdram_synth" value="false" />
 <parameter
    name="quartus_ini_hps_ip_enable_low_speed_serial_fpga_interfaces"
    value="false" />
 <parameter name="quartus_ini_hps_ip_enable_bsel_csel" value="false" />
 <parameter name="quartus_ini_hps_ip_f2sdram_bonding_out" value="false" />
</module>
<module
  kind="altera_jtag_avalon_master"
  version="13.1"
  enabled="1"
  name="master_0">
 <parameter name="USE_PLI" value="0" />
 <parameter name="PLI_PORT" value="50000" />
 <parameter name="COMPONENT_CLOCK" value="0" />
 <parameter name="FAST_VER" value="0" />
 <parameter name="FIFO_DEPTHS" value="2" />
 <parameter name="AUTO_DEVICE_FAMILY" value="Cyclone V" />
 <parameter name="AUTO_DEVICE" value="5CSXFC6D6F31C8ES" />
</module>
<module kind="altera_avalon_fifo" version="13.1" enabled="1" name="fifo_0">
 <parameter name="avalonMMAvalonMMDataWidth" value="32" />
```

```
 <parameter name="avalonMMAvalonSTDataWidth" value="32" />
 <parameter name="bitsPerSymbol" value="32" />
 <parameter name="channelWidth" value="0" />
 <parameter name="errorWidth" value="0" />
 <parameter name="fifoDepth" value="16" />
 <parameter name="fifoInputInterfaceOptions" value="AVALONMM_WRITE" />
 <parameter name="fifoOutputInterfaceOptions" value="AVALONST_SOURCE" />
 <parameter name="showHiddenFeatures" value="false" />
 <parameter name="singleClockMode" value="true" />
 <parameter name="singleResetMode" value="false" />
 <parameter name="symbolsPerBeat" value="1" />
 <parameter name="useBackpressure" value="true" />
 <parameter name="useIRQ" value="false" />
 <parameter name="usePacket" value="false" />
 <parameter name="useReadControl" value="false" />
 <parameter name="useRegister" value="false" />
 <parameter name="useWriteControl" value="true" />
 <parameter name="deviceFamilyString" value="Cyclone V" />
</module>
<module kind="altera_avalon_fifo" version="13.1" enabled="1" name="fifo_1">
 <parameter name="avalonMMAvalonMMDataWidth" value="32" />
 <parameter name="avalonMMAvalonSTDataWidth" value="32" />
 <parameter name="bitsPerSymbol" value="32" />
 <parameter name="channelWidth" value="0" />
 <parameter name="errorWidth" value="0" />
 <parameter name="fifoDepth" value="16" />
 <parameter name="fifoInputInterfaceOptions" value="AVALONST_SINK" />
 <parameter name="fifoOutputInterfaceOptions" value="AVALONMM_READ" />
 <parameter name="showHiddenFeatures" value="false" />
 <parameter name="singleClockMode" value="true" />
 <parameter name="singleResetMode" value="false" />
 <parameter name="symbolsPerBeat" value="1" />
 <parameter name="useBackpressure" value="true" />
 <parameter name="useIRQ" value="false" />
 <parameter name="usePacket" value="false" />
 <parameter name="useReadControl" value="false" />
 <parameter name="useRegister" value="false" />
 <parameter name="useWriteControl" value="true" />
 <parameter name="deviceFamilyString" value="Cyclone V" />
</module>
<module kind="tyrion_dev" version="1.2" enabled="1" name="tyrion_dev_0" />
<connection
   kind="clock"
   version="13.1"
   start="clk_0.clk"
   end="hps_0.h2f_axi_clock" />
```

```
<connection
   kind="clock"
   version="13.1"
   start="clk_0.clk"
   end="hps_0.f2h_axi_clock" />
<connection
   kind="clock"
   version="13.1"
   start="clk_0.clk"
   end="hps_0.h2f_lw_axi_clock" />
<connection kind="clock" version="13.1" start="clk_0.clk" end="master_0.clk" />
<connection
   kind="reset"
   version="13.1"
   start="clk_0.clk_reset"
   end="master_0.clk_reset" />
<connection kind="clock" version="13.1" start="clk_0.clk" end="fifo_0.clk_in" />
<connection kind="clock" version="13.1" start="clk_0.clk" end="fifo_1.clk_in" />
<connection
   kind="reset"
   version="13.1"
   start="clk_0.clk_reset"
   end="fifo_1.reset_in" />
<connection
   kind="reset"
   version="13.1"
   start="clk_0.clk_reset"
   end="fifo_0.reset_in" />
<connection
   kind="avalon"
   version="13.1"
   start="hps_0.h2f_lw_axi_master"
   end="fifo_0.in">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0008" />
 <parameter name="defaultConnection" value="false" />
</connection>
<connection
   kind="avalon"
   version="13.1"
   start="hps_0.h2f_lw_axi_master"
   end="fifo_1.out">
 <parameter name="arbitrationPriority" value="1" />
 <parameter name="baseAddress" value="0x0000" />
 <parameter name="defaultConnection" value="false" />
</connection>
```

```
<connection
   kind="avalon"
   version="13.1"
   start="master_0.master"
   end="fifo_0.in_csr">
  <parameter name="arbitrationPriority" value="1" />
  <parameter name="baseAddress" value="0x0040" />
  <parameter name="defaultConnection" value="false" />
</connection>
<connection kind="avalon" version="13.1" start="master_0.master" end="fifo_0.in">
  <parameter name="arbitrationPriority" value="1" />
  <parameter name="baseAddress" value="0x0008" />
  <parameter name="defaultConnection" value="false" />
</connection>
<connection kind="avalon" version="13.1" start="master_0.master" end="fifo_1.out">
  <parameter name="arbitrationPriority" value="1" />
  <parameter name="baseAddress" value="0x0000" />
  <parameter name="defaultConnection" value="false" />
</connection>
<connection
   kind="avalon"
   version="13.1"
   start="master_0.master"
   end="fifo_1.in_csr">
  <parameter name="arbitrationPriority" value="1" />
  <parameter name="baseAddress" value="0x0020" />
  <parameter name="defaultConnection" value="false" />
</connection>
<connection
   kind="avalon_streaming"
   version="13.1"
   start="tyrion_dev_0.avalon_streaming_source"
   end="fifo_1.in" />
<connection
   kind="avalon_streaming"
   version="13.1"
   start="fifo_0.out"
   end="tyrion_dev_0.avalon_streaming_sink" />
<connection
   kind="clock"
   version="13.1"
   start="clk_0.clk"
   end="tyrion_dev_0.clock" />
<connection
   kind="reset"
   version="13.1"
```

```
    start="clk_0.clk_reset"
    end="tyrion_dev_0.reset_sink" />
 <interconnectRequirement for="$system" name="qsys_mm.clockCrossingAdapter" value="HANDS
 <interconnectRequirement for="$system" name="qsys_mm.maxAdditionalLatency" value="1" />
 <interconnectRequirement for="$system" name="qsys_mm.insertDefaultSlave" value="false"
</system>
SYSTEMC ?= /opt/lib/systemc-2.2.0
TLM ?= /opt/lib/TLM-2009-07-15
TARGET_ARCH := linux64
WAVE_VIEWER := simvision
CC := g++

CFLAGS ?=
include Makefile.params # after CFLAGS defined

PARAMS_FILE=.params

LIBS    := -lsystemc
LIBDIR = -L. -L$(SYSTEMC)/lib-$(TARGET_ARCH)
CTOS_ROOT := $(CTOS_PATH)
CTOS_INCLUDES := -I$(CTOS_ROOT)/share/ctos/include \
                 -I$(CTOS_ROOT)/share/ctos/include/ctos_fx \
                 -I$(CTOS_ROOT)/share/ctos/include/ctos_tlm \
                 -I$(CTOS_ROOT)/share/ctos/include/ctos_flex_channels
MATH = /opt/zynq-math/src/

INCDIR = -I../src -I../tb -I$(SYSTEMC)/include -I$(TLM)/include/tlm \
-I$(DRAMSIM) $(CTOS_INCLUDES) -I$(MATH) -I/tools/ctos141/share/ctos/include

TARGET := svd
VPATH := ../src:../tb
SRCS := svd_wrapper.cpp svd.cpp svd_tb.cpp sc_main.cpp gm_jacobi.cpp
OBJS = $(SRCS:.cpp=.o)
HDRS := svd_data.h svd_wrapper.h svd.h svd_ctos_funcs.h svd_tb.h gm_jacobi.h

.SUFFIXES: .cpp .cc .o
$(TARGET): $(OBJS)
        $(CC) ${INCDIR} $(CFLAGS) ${LIBDIR} -o $@ $(OBJS) $(C_OBJS) ${LIBS}

$(OBJS): $(HDRS) $(PARAMS_FILE)

.PHONY: force
$(PARAMS_FILE): force
        @echo '$(CFLAGS)' | cmp -s $@ || echo '$(CFLAGS)' > $@

.PHONY: clean distclean run wave
```

```
.cpp.o:
        $(CC) ${INCDIR} $(CFLAGS) -c $<

clean:
        rm -f *.o $(TARGET) *.dsn *.trn *.vcd $(PARAMS_FILE)
        if [ -d .simvision ]; then rm -r .simvision; fi

distclean: clean
        rm -rf results
        rm -rf *.log

run: $(TARGET)
        rm -f core
        ./$(TARGET)

wave:
        $(WAVE_VIEWER) $(TARGET).vcd
#include "svd.h"

#ifdef CTOS_SC_FIXED_POINT
#include "svd_ctos_funcs.h"
#endif

SVD_CELL_TYPE inline fp_abs(SVD_CELL_TYPE in) {
#ifdef CTOS_SC_FIXED_POINT
        return sld::abs(in);
#else
        if (in >= 0)
                return in;
        else
                return -in;
#endif
}

void svd::identify (SVD_CELL_TYPE *matrix, int dimension) {
        int row, col;
IDENTITY_OUTER:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;
IDENTITY_INNER:
                for (col = 0; col < MAX_SIZE; col++) {
                        if (col == dimension) break;
                        // Elements on diagonal =1
                        if (col == row) {matrix [(row * dimension) + col] = 1.0;}
                        // ... all others 0
```

```
                        else {matrix [(row * dimension) + col] = 0.0;}
                }
        }
        return;
}

void svd::copyMatrix (SVD_CELL_TYPE *a, SVD_CELL_TYPE *b, int dimension) {
        int row, col;

COPY_MATRIX_OUTER:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;

COPY_MATRIX_INNER:
                for (col = 0; col < MAX_SIZE; col++) {
                        if (col == dimension) break;
                        b [(row * dimension) + col] = a [(row * dimension) + col];
                }
        }
        return;
}

void svd::multiply (SVD_CELL_TYPE *left, SVD_CELL_TYPE *right, SVD_CELL_TYPE *result, in
        int leftRow, rightRow; // row numbers of the left and right matrix, respectively
        int leftCol, rightCol; // same as above but for columns
        SVD_CELL_TYPE tempResult = 0;

MULTIPLY_MATRIX_OUTER:
        for (leftRow = 0; leftRow < MAX_SIZE; leftRow++) {
                if (leftRow == dimension) break;

MULTIPLY_MATRIX_MID:
                for (rightCol = 0; rightCol < MAX_SIZE; rightCol++) {
                        if (rightCol == dimension) break;

MULTIPLY_MATRIX_INNER:
                        for (leftCol = rightRow = 0; leftCol < MAX_SIZE;
                                        leftCol++, rightRow++) {
                                if (leftCol == dimension) break;
                                /* TODO reevaluate best way to handle this re: waits */
                                SVD_CELL_TYPE tmp, tmp2;
                                tmp = left [(leftRow * dimension) + leftCol];
                                wait();
                                tmp2 = right [(rightRow * dimension) + rightCol];
                                tempResult += tmp * tmp2;
                                wait();
```

```
                        }
                        result [(leftRow * dimension) + rightCol] = tempResult;
                        tempResult = 0;
                        wait();
                }
        }
        return;
}

void svd::jacobi (SVD_CELL_TYPE *a, int n, SVD_CELL_TYPE *s, SVD_CELL_TYPE *u, SVD_CELL_
        // Arrays that contain the coordinates of the elements of the 2x2
        // sub matrix formed by the largest off-diagonal element. First entry
        // is the row number and second entry is the column number.
        int a11[2];
        int a12[2];
        int a21[2];
        int a22[2];
        LargestElement le; // largest element of matrix a
        le.value = le.rowNum = le.colNum = -1;
        int i, j;

        // I could have statically allocated these but I would not have been
        // able to pass them as double pointers: this is a known problem in C

        identify (u, n);
        identify (v, n);

        if (n == 1) {   // 1x1 matrix is already in SVD
                SVD_CELL_TYPE tmp = s[0];
                wait();
                s[0] = tmp;
                return;
        }

        le = findLargestElement (a, n, a11, a12, a21, a22);

        int count = 0;
        SVD_CELL_TYPE old_value = 0;
        /* FIXME This loop may not be synthesizable */

CONVERGENCE_LOOP:
        while (fp_abs (le.value) >
#ifndef REAL_FLOAT
                        SVD_CELL_TYPE(SVD_PRECISION)
#else
                        SVD_PRECISION
```

```
#endif
                        && fp_abs(le.value - old_value) > SVD_CELL_TYPE(MIN_MOVEMENT)

            && count < MAX_ITERATIONS
            ) {
                old_value = le.value;
                count++;
                rotate (a, n, u, v, a11, a12, a21, a22);
                le = findLargestElement (a, n, a11, a12, a21, a22);
#ifndef REAL_FLOAT
                if (!(count % 10))
                cerr << "current iteration: " << count << "; " << le.value.to_double() <
                        old_value.to_double() << endl;
#endif

        }
#ifndef REAL_FLOAT
                cerr << "current iteration: " << count << "; " << le.value.to_double() <
                        old_value.to_double() << endl;
#endif

        cout << "loop count: " << count << endl;

        reorder (a, n, u, v);
        wait();

        // Copy over the singular values in a to s
COPY_SINGULAR_VALUES:
        for (i = 1; i < MAX_SIZE; i++) {
                if (i == n) break;
                SVD_CELL_TYPE tmp;
                tmp = s[(i * n) + i];
                wait();
                s[i] = tmp;
                wait();
                s[(i * n) + i] = 0;
                wait();
        }

        return;
}

void svd::rotate (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *v, i
        SVD_CELL_TYPE a11, a12, a21, a22; // elements of the sub-matrix
        SVD_CELL_TYPE alpha, beta; // angles used in rotations; alpha is angle of left r
        SVD_CELL_TYPE cosA, sinA, cosB, sinB;
```

```
        SVD_CELL_TYPE X, Y; // temporary values used in calculating angles

        // Assign elements of sub-matrix to the their actual values from a
        a11 = a [x11[0] * dimension + x11[1]];
        wait();
        a12 = a [x12[0] * dimension + x12[1]];
        wait();
        a21 = a [x21[0] * dimension + x21[1]];
        wait();
        a22 = a [x22[0] * dimension + x22[1]];
        wait();

        // Calculate angles and sin and cos of those angles using the closed
        // formulas found.
#ifdef CTOS_SC_FIXED_POINT
        X = sld::atan2_cordic_func<CORDIC_ITER, WL, IWL>((SVD_CELL_TYPE) (a11+a22), (SVI
        Y = sld::atan2_cordic_func<CORDIC_ITER, WL, IWL>((SVD_CELL_TYPE) (a11-a22), (SVI
        //cout << "X, Y: " << X << "; " << Y << "; " << (X + Y) << "; " << (Y - X) << en

        alpha = (X + Y) * SVD_CELL_TYPE(0.5);//sld::div_func<WL, IWL, WL, IWL, WL, IWL>
        beta = (Y - X) * SVD_CELL_TYPE(0.5);//sld::div_func<WL, IWL, WL, IWL, WL, IWL>(
        //cout << "a, b: " << alpha << "; " << beta << endl;

        sld::cos_sin_cordic_func<CORDIC_ITER, WL, IWL>(alpha, cosA, sinA);
        //cosA = sld::cos_cordic_func<CORDIC_ITER, WL, IWL>(alpha);
        //sinA = sld::sin_cordic_func<CORDIC_ITER, WL, IWL>(alpha);
        sld::cos_sin_cordic_func<CORDIC_ITER, WL, IWL>(beta, cosB, sinB);
        //cosB = sld::cos_cordic_func<CORDIC_ITER, WL, IWL>(beta);
        //sinB = sld::sin_cordic_func<CORDIC_ITER, WL, IWL>(beta);

        //cout << "rotate stats: " << cosA << "; " << sinA << "; " << cosB << "; " << si
#else
        X = atan((a21 - a12) / (a11 + a22));
        Y = atan((a21 + a12) / (a11 - a22));
        alpha = 0.5 * (X + Y);
        beta = 0.5 * (Y - X);

        cosA = cos (alpha); sinA = sin (alpha);
        cosB = cos (beta); sinB = sin (beta);
#endif

        // Create left rotation matrix, namely U_i which looks like this
        //
        //              | cosA  sinA |
        //              | -sinA cosA |
        //
```

```
        identify (Ui, dimension);
        Ui [x11[0] * dimension + x11[1]] = Ui [x22[0] * dimension + x22[1]] = cosA;
        Ui [x12[0] * dimension + x12[1]] = sinA;
        Ui [x21[0] * dimension + x21[1]] = -sinA;
        // Rotate a
    // First on the left with Ui
    multiply (Ui, a, tempMatrix, dimension);
    copyMatrix (tempMatrix, a, dimension);
        // Apply rotation matrix Ui to U
    multiply (Ui, u, tempMatrix, dimension);
    copyMatrix (tempMatrix, u, dimension);
    // Create the right rotation matrix, namely V_i which looks like this
    // Note that I'm using Ui as Vi
    //          | cosB  -sinB |
    //          | sinB   cosB |
    //
    identify (Ui, dimension);
    Ui [x11[0] * dimension + x11[1]] = Ui [x22[0] * dimension + x22[1]] = cosB;
    Ui [x12[0] * dimension + x12[1]] = -sinB;
    Ui [x21[0] * dimension + x21[1]] = sinB;

    // Then on the right with Vi
    multiply (a, Ui, tempMatrix, dimension);
    copyMatrix (tempMatrix, a, dimension);
    multiply (v, Ui, tempMatrix, dimension);
    copyMatrix (tempMatrix, v, dimension);



/*

    // Create the right rotation matrix, namely V_i which looks like this
        //
        //              | cosB  -sinB |
        //              | sinB   cosB |
        //
        identify (Vi, dimension);
        Vi [x11[0] * dimension + x11[1]] = Vi [x22[0] * dimension + x22[1]] = cosB;
        Vi [x12[0] * dimension + x12[1]] = -sinB;
        Vi [x21[0] * dimension + x21[1]] = sinB;

        // Rotate a
        // First on the left with Ui
        multiply (Ui, a, tempResult, dimension);
        copyMatrix (tempResult, a, dimension);
        // Then on the right with Vi
```

```
        multiply (a, Vi, tempResult, dimension);
        copyMatrix (tempResult, a, dimension);

        // Apply rotation matrix Ui to U
        multiply (Ui, u, tempResult, dimension);
        copyMatrix (tempResult, u, dimension);
        // Apply rotation matrix Vi to V
        multiply (v, Vi, tempResult, dimension);
        copyMatrix (tempResult, v, dimension);
*/
        return;
}

LargestElement svd::findLargestElement (SVD_CELL_TYPE *matrix, int dimension, int *a11,
        LargestElement leTemp;
        int i, j;

        leTemp.value = 0;
        leTemp.rowNum = leTemp.colNum = -1;

        // Populate leArray such that the entry at index i contains information
        // about the largest element of row i
FLE_POPULATE_OUTER:
        for (i = 0; i < MAX_SIZE; i++) {
                if (i == dimension) break;

FLE_POPULATE_INNER:
                for (j = 0; j < MAX_SIZE; j++) {
                        if (j == dimension) break;
                        // We are looking for the largest OFF-DIAGONAL element
                        if (j == i)
                                continue;
                        if (fp_abs (matrix [(i * dimension) + j]) > fp_abs (leTemp.value
                                leTemp.value = matrix[(i * dimension) + j];
                                leTemp.rowNum = i;
                                leTemp.colNum = j;
                        }
                }
                //leArray[i].value = leTemp.value;
                //leArray[i].rowNum = leTemp.rowNum;
                //leArray[i].colNum = leTemp.colNum;
                //leTemp.value = 0;
        }
        //leTemp.value = 0;
        //leTemp.rowNum = leTemp.colNum = -1;
        // Iterate over leArray and find the largest element in the matrix as a
```

```
        // whole
/*
FLE_ITERATE_LOOP:
        for (i = 0; i < MAX_SIZE; i++) {
                if (i == dimension) break;
                if (fp_abs (leArray [i].value) > fp_abs (leTemp.value)) {
                        leTemp.value = leArray [i].value;
                        leTemp.rowNum = leArray [i].rowNum;
                        leTemp.colNum = leArray [i].colNum;
                }
        }
*/
        // Determining coordinates of the 2x2 sub matrix formed by this largest
        // element
        if (leTemp.rowNum < leTemp.colNum) { // largest element is above diagonal
                (a11) [0] = (a11) [1] = (a12) [0] = (a21) [1] = leTemp.rowNum;
                (a21) [0] = (a22) [0] = (a22) [1] = (a12) [1] = leTemp.colNum;
        }
        else { // below diagonal
                a22[0] = leTemp.rowNum;
                a22[1] = leTemp.rowNum;
                a21[0] = leTemp.rowNum;
                a12[1] = leTemp.rowNum;

                a11[0] = leTemp.colNum;
                a11[1] = leTemp.colNum;
                a21[1] = leTemp.colNum;
                a12[0] = leTemp.colNum;
        }

        return leTemp;
}

void svd::transpose (SVD_CELL_TYPE *matrix, int dimension) {
        SVD_CELL_TYPE temp, temp2;
        int row, col;

TRANSPOSE_OUTER:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;

TRANSPOSE_INNER:
                for (col = (row + 1); col < MAX_SIZE; col++) {
                        /* TODO rework indexing */
                        if (col == dimension) break;
                        if (col == row) {continue;} // skip diagonal elements
```

```
                        temp = matrix [(row * dimension) + col];
                        wait();
                        temp2 = matrix [(col * dimension) + row];
                        wait();
                        matrix [(row * dimension) + col] = temp2;
                        wait();
                        matrix [(col * dimension) + row] = temp;
                        wait();
                }
        }
        return;
}

void svd::reorder (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *v)
        int row, col, x, largestElementRow;
        SVD_CELL_TYPE temp; // stores the largest singular value
    //Replace p with Ui because why not?



REORDER_OUTER:
        for (x = 0; x < MAX_SIZE; x++) {
                if (x == dimension) break;
                temp = 0.0;
                identify (Ui, dimension);

REORDER_INNER:
                for (row = x; row < MAX_SIZE; row++) {
                        if (row == dimension) break;
                        col = row;
                        if (fp_abs (a [row * dimension + col]) > fp_abs (temp)) {
                                temp = a [row * dimension + col];
                                largestElementRow = row;
                        }
                }
                swapRows (Ui, x, largestElementRow, dimension);
                // Reorder a
                multiply (Ui, a, tempMatrix, dimension); copyMatrix (tempMatrix, a, dime
                multiply (a, Ui, tempMatrix, dimension); copyMatrix (tempMatrix, a, dime
                // Reorder u
                multiply (Ui, u, tempMatrix, dimension); copyMatrix (tempMatrix, u, dime
                // Reorder v
                multiply (v, Ui, tempMatrix, dimension); copyMatrix (tempMatrix, v, dime
        }

        transpose (u, dimension);
```

```
        transpose (v, dimension);
        identify (Ui, dimension);
REORDER_FIND_NEG:
        for (row = 0; row < MAX_SIZE; row++) {
                if (row == dimension) break;
                col = row;
                if (a [row * dimension + col] < 0)
                        Ui[row * dimension + col] = -1.0;
        }
        multiply (Ui, a, tempMatrix, dimension);
        copyMatrix (tempMatrix, a, dimension);
        multiply (u, Ui, tempMatrix, dimension);
        copyMatrix (tempMatrix, u, dimension);
/* Old version
REORDER_OUTER:
for (x = 0; x < MAX_SIZE; x++) {
    if (x == dimension) break;
    temp = 0.0;
    identify (p, dimension);

REORDER_INNER:
    for (row = x; row < MAX_SIZE; row++) {
        if (row == dimension) break;
        col = row;
        if (fp_abs (a [row * dimension + col]) > fp_abs (temp)) {
            temp = a [row * dimension + col];
            largestElementRow = row;
        }
    }
    swapRows (p, x, largestElementRow, dimension);
    // Reorder a
    multiply (p, a, tempMatrix, dimension); copyMatrix (tempMatrix, a, dimension);
    multiply (a, p, tempMatrix, dimension); copyMatrix (tempMatrix, a, dimension);
    // Reorder u
    multiply (p, u, tempMatrix, dimension); copyMatrix (tempMatrix, u, dimension);
    // Reorder v
    multiply (v, p, tempMatrix, dimension); copyMatrix (tempMatrix, v, dimension);
}

transpose (u, dimension);
transpose (v, dimension);
identify (p, dimension);
REORDER_FIND_NEG:
for (row = 0; row < MAX_SIZE; row++) {
    if (row == dimension) break;
    col = row;
```

```
    if (a[row * dimension + col] < 0)
        p[row * dimension + col] = -1.0;
}
multiply (p, a, tempMatrix, dimension);
copyMatrix (tempMatrix, a, dimension);
multiply (u, p, tempMatrix, dimension);
copyMatrix (tempMatrix, u, dimension);
*/
    return;
}

// Swap row a with row b of matrix m
void svd::swapRows (SVD_CELL_TYPE *m, int a, int b, int dimension) {
        SVD_CELL_TYPE temp;

SWAP_ROW:
        for (int col = 0; col < MAX_SIZE; col++) {
                if (col == dimension) break;
                temp = m[a * dimension + col];
                m[a * dimension + col] = m[b * dimension + col];
                m[b * dimension + col] = temp;
        }
        return;
}


void svd::config_svd()
{
        // Initialization
        size.write(0);
        init_done.write(false);

        wait();

        // Read configuration until done
        bool done = false;
CONFIG_REGISTER_WHILE:
        do {
                wait();
                done = conf_done.read();
                int a = conf_size.read();
                size.write(a);
        } while (!done);

        // Let other threads run then do nothing
        init_done.write(true);
```

```
        while (true) {
                wait();
        }
}

void svd::load_input()
{
RESET_LOAD:
        bufdin.reset_get();

        rd_index.write(0);
        rd_length.write(0);
        rd_request.write(false);

        input_done.write(false);

        do {wait();}
        while(!init_done.read());

        const int rows = size.read();
        int index = 0;

LOAD_INPUT_WHILE:
        while(true) {
                if (index == rows * rows)
                        // Input complete; wait for reset
                        do { wait(); } while (true);

                int length = rows * rows;

                rd_index.write(index);
                rd_length.write(length);
                index += length;

                // 4-phase handshake
                rd_request.write(true);
                do { wait(); } while(!rd_grant.read());
                rd_request.write(false);
                do { wait(); } while(rd_grant.read());

LOAD_OUTER:
                for (int i = 0; i < MAX_SIZE; ++i) {
                        if (i == rows) break;
LOAD_INNER:
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                if (j == rows) break;
```

```
                                SVD_CELL_TYPE cell = bufdin.get();
                                matrix_in[i * rows + j] = cell;
#ifdef VERBOSE
                                cout << "DUT GET: (index, length, i, j, val)" << index <
                                        << " " << i << " " << j << " " << cell << endl;
#endif
                                wait();
                        }
                }

                // 4-phase handshake
                input_done.write(true);
                do { wait(); } while (!process_start.read());
                input_done.write(false);
                do { wait(); } while (process_start.read());
        }
}

void svd::store_output()
{
RESET_STORE:
        bufdout.reset_put();

        wr_index.write(0);
        wr_request.write(false);
        wr_length.write(0);

        output_start.write(false);
        svd_done.write(false);

        do {wait();}
        while(!init_done.read());

        const int rows = size.read();
        int length = rows * rows;
        int index = 0;

STORE_OUTPUT_WHILE:
        while(true) {

                if (index == NUM_OUTPUT_MATRIX * rows * rows) {
                        // DEBAYER Done (need a reset)
                        svd_done.write(true);
                        do { wait(); } while(true);
                }
```

```
                // 4-phase handshake
                do { wait(); }
                while (!process_done.read());
                output_start.write(true);
                do { wait(); }
                while (process_done.read());
                output_start.write(false);

                /* S matrix */
                // Send DMA request
                wr_index.write(index);
                wr_length.write(length);
                index += length;

                wr_request.write(true);
                do { wait(); } while(!wr_grant.read());
                wr_request.write(false);
                do { wait(); } while(wr_grant.read());

OUTPUT_S_OUTER:
                for (int i = 0; i < MAX_SIZE; ++i) {
                        if (i == rows) break;
OUTPUT_S_INNER:
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                if (j == rows) break;
                                SVD_CELL_TYPE cell = s[i * size + j];
                                wait();
                                bufdout.put(cell);
#ifdef VERBOSE
                                cout << "DUT PUT: (index, length, i, j, val)" << index <
                                        << " " << i << " " << j << " " << cell << endl;
#endif
                                wait();
                        }
                }

                /* U matrix */
                // Send DMA request
                wr_index.write(index);
                wr_length.write(length);
                index += length;

                wr_request.write(true);
                do { wait(); } while(!wr_grant.read());
                wr_request.write(false);
                do { wait(); } while(wr_grant.read());
```

```
OUTPUT_U_OUTER:
                for (int i = 0; i < MAX_SIZE; ++i) {
                        if (i == rows) break;
OUTPUT_U_INNER:
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                if (j == rows) break;
                                SVD_CELL_TYPE cell = u[i * size + j];
                                bufdout.put(cell);
#ifdef VERBOSE
                                cout << "DUT PUT: (index, length, i, j, val)" << index <
                                        << " " << i << " " << j << " " << cell << endl;
#endif
                                wait();
                        }
                }


                /* V matrix */
                // Send DMA request
                wr_index.write(index);
                wr_length.write(length);
                index += length;

                wr_request.write(true);
                do { wait(); } while(!wr_grant.read());
                wr_request.write(false);
                do { wait(); } while(wr_grant.read());

OUTPUT_V_OUTER:
                for (int i = 0; i < MAX_SIZE; ++i) {
                        if (i == rows) break;
OUTPUT_V_INNER:
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                if (j == rows) break;
                                SVD_CELL_TYPE cell = v[i * size + j];
                                bufdout.put(cell);
#ifdef VERBOSE
                                cout << "DUT PUT: (index, length, i, j, val)" << index <
                                        << " " << i << " " << j << " " << cell << endl;
#endif
                                wait();
                        }
                }
        }
}
```

```
void svd::process_svd()
{
        // Reset
        process_start.write(false);
        process_done.write(false);

        // Wait for configuratin
        do { wait(); } while (!init_done.read());

        const int rows = size.read();
        int svd_row = 0;

DEBAYER_WHILE:
        while (true) {

                if (svd_row == rows)
                        // Wait for reset
                        do { wait(); } while(true);

                // 4-phase handshake
                do { wait(); }
                while (!input_done.read());
                process_start.write(true);
                do { wait(); }
                while (input_done.read());
                process_start.write(false);

                for (int i = 0; i < MAX_SIZE; ++i) {
                        for (int j = 0; j < MAX_SIZE; ++j) {
                                SVD_CELL_TYPE tmp;
                                tmp = matrix_in[i * rows + j];
                                wait();
                                s[i * rows + j] = tmp;
                                wait();
                        }
                }

                jacobi(s, rows, s, u, v);

                // 4-phase handshake
                process_done.write(true);
                do { wait(); }
                while (!output_start.read());
                process_done.write(false);
                do { wait(); }
```

```
                while (output_start.read());

                svd_row += rows;
        }
}
#ifdef __CTOS__
SC_MODULE_EXPORT(svd)
#endif
#ifndef __SVD_CTOS_FUNCS__
#define __SVD_CTOS_FUNCS__

#ifdef CTOS_SC_FIXED_POINT
#include "math/sysc_math.h"
#include "math/div.h"
#include "cordic/atan2_cordic.h"
#include "cordic/cos_sin_cordic.h"
#include "cordic/cos_cordic.h"
#include "cordic/sin_cordic.h"
#endif

#endif
#ifndef _MYDATA_H_
#define _MYDATA_H_

#define SC_INCLUDE_FX
#include <systemc.h>

//#define REAL_FLOAT
//#define SC_FIXED_POINT_FAST
//#define SC_FIXED_POINT
#define CTOS_SC_FIXED_POINT

#ifdef __CTOS__
#define MAX_SIZE     64
#else
#define MAX_SIZE     20
#endif

#define MAX_ITERATIONS  1000000

#if defined(REAL_FLOAT)
#define SVD_CELL_TYPE double
#elif defined(SC_FIXED_POINT_FAST)
#define SC_FIXED_TYPE sc_dt::sc_fixed_fast
#elif defined(SC_FIXED_POINT)
#define SC_FIXED_TYPE sc_dt::sc_fixed
```

```
#elif defined(CTOS_SC_FIXED_POINT)
#include <ctos_fx.h>
#define SC_FIXED_TYPE ctos_sc_dt::sc_fixed
#else
#error "specify fixed or floating point type"
#endif

#ifndef REAL_FLOAT
#define WL  64
#define IWL 24
#define SVD_CELL_TYPE SC_FIXED_TYPE<WL,IWL>
#endif

#define SVD_INPUT_SIZE(__sz) (__sz * __sz)
#define SVD_OUTPUT_SIZE(__sz) (3 * __sz * __sz)
#define SVD_GET_S(__ptr, __sz) (__ptr)
#define SVD_GET_U(__ptr, __sz) (__ptr + __sz * __sz)
#define SVD_GET_V(__ptr, __sz) (__ptr + 2 * __sz * __sz)

#endif
#ifndef _SVD_H_
#define _SVD_H_

#define SC_INCLUDE_FX
#include "systemc.h"
#include <ctos_flex_channels.h>
//#include <flex_channels.hpp>

#include "svd_data.h"

#define NUM_OUTPUT_MATRIX 3

#define SVD_PRECISION (0.000000000001)
#define MIN_MOVEMENT  (0.000000000001)
#define CORDIC_ITER 80

//#define VERBOSE

typedef struct {
        SVD_CELL_TYPE value;
        char rowNum;
        char colNum;
} LargestElement;

SC_MODULE(svd) {
        sc_in<bool> clk;
```

```
sc_in<bool> rst;

// DMA requests interface from memory to device
sc_out<unsigned> rd_index;   // array index (offset from base address)
sc_out<unsigned> rd_length;  // burst size
sc_out<bool>     rd_request; // transaction request
sc_in<bool>      rd_grant;   // transaction grant

// DMA requests interface from device to memory
sc_out<unsigned> wr_index;   // array index (offset from base address)
sc_out<unsigned> wr_length;  // burst size
sc_out<bool>     wr_request; // transaction request
sc_in<bool>      wr_grant;   // transaction grant

// input data read by load_input
get_initiator<SVD_CELL_TYPE> bufdin;
// output data written by store output
put_initiator<SVD_CELL_TYPE> bufdout;

sc_in<unsigned> conf_size;
sc_in<bool>     conf_done;

// computation complete
sc_out<bool>    svd_done;

void config_svd(void);
void load_input(void);
void store_output(void);
void process_svd(void);

SC_CTOR(svd) {
        SC_CTHREAD(config_svd, clk.pos());
        reset_signal_is(rst, false);
        SC_CTHREAD(load_input, clk.pos());
        reset_signal_is(rst, false);
        SC_CTHREAD(store_output, clk.pos());
        reset_signal_is(rst, false);
        SC_CTHREAD(process_svd, clk.pos());
        reset_signal_is(rst, false);

        bufdin.clk_rst(clk,rst);
        bufdout.clk_rst(clk,rst);
}

private:
//Written by config_debayer
```

```
        sc_signal<unsigned> size;
        sc_signal<bool>     init_done;

        //Written by load_input
        sc_signal<bool>     input_done;

        //Written by process_debayer
        sc_signal<bool>     process_start;
        sc_signal<bool>     process_done;

        //Written by store_output
        sc_signal<bool>     output_start;

        /* SVD functions */
        void multiply (SVD_CELL_TYPE *left, SVD_CELL_TYPE *right, SVD_CELL_TYPE *result,
        void jacobi (SVD_CELL_TYPE *a, int n, SVD_CELL_TYPE *s, SVD_CELL_TYPE *u, SVD_CE
        LargestElement findLargestElement (SVD_CELL_TYPE *matrix, int dimension, int *a1
        void copyMatrix (SVD_CELL_TYPE *a, SVD_CELL_TYPE *b, int dimension);
        void identify (SVD_CELL_TYPE *matrix, int dimension);
        void rotate (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *v
        void transpose (SVD_CELL_TYPE *matrix, int dimension);
        void reorder (SVD_CELL_TYPE *a, int dimension, SVD_CELL_TYPE *u, SVD_CELL_TYPE *
        void swapRows (SVD_CELL_TYPE *m, int a, int b, int dimension);

        /* scratchpad matrices */
        SVD_CELL_TYPE Ui[MAX_SIZE * MAX_SIZE];
        //SVD_CELL_TYPE Vi[MAX_SIZE * MAX_SIZE];
        //SVD_CELL_TYPE tempResult[MAX_SIZE * MAX_SIZE];
        SVD_CELL_TYPE tempMatrix[MAX_SIZE * MAX_SIZE];
        //SVD_CELL_TYPE p[MAX_SIZE * MAX_SIZE];
        //LargestElement leArray[MAX_SIZE];

        SVD_CELL_TYPE matrix_in[MAX_SIZE * MAX_SIZE];

        SVD_CELL_TYPE s[MAX_SIZE * MAX_SIZE];
        SVD_CELL_TYPE u[MAX_SIZE * MAX_SIZE];
        SVD_CELL_TYPE v[MAX_SIZE * MAX_SIZE];
};

#endif
#include "svd_wrapper.h"

void svd_wrapper::config_svd() {
        conf_done.write(false);
        wait();
```

```
        conf_size.write(MAX_SIZE);
        conf_done.write(true);

        do {wait();}
        while (true);
}

void svd_wrapper::handle_irq() {
        svd_done_irq.write(false);
        dut_rst.write(false);
        wait();
        dut_rst.write(true);

        do {wait();}
        while(true);
        /* TODO reset self properly */

        while (true) {
                do {wait();}
                while (!svd_done.read());

                /* TODO handle irq properly */
                /* XXX don't need this functionality in hardware */
                svd_done_irq.write(true);
                wait(); wait(); wait(); wait();
                wait(); wait(); wait(); wait();
                svd_done_irq.write(false);

                dut_rst.write(false);
                wait();
                dut_rst.write(true);
        }
}

void svd_wrapper::handle_input() {
        SVD_CELL_TYPE tmp;
        sc_uint<32> tmp_sc_uint;
        uint64_t whole;
        uint32_t lower_half, upper_half;
        unsigned length, index;

        data_in.reset_get();
        data_to_dut.reset_put();
        wait();

        while (true) {
```

```
                do { wait(); }
                while (!rd_request.read());
                rd_grant.write(true);
                length = rd_length.read();
                index = rd_index.read();
                do { wait(); }
                while (rd_request.read());
                rd_grant.write(false);

WRAPPER_INPUT_LOOP:
                for (int i = 0; i < length; ++i) {
                        tmp_sc_uint = data_in.get();
                        lower_half = tmp_sc_uint;
                        wait();

                        tmp_sc_uint = data_in.get();
                        upper_half = tmp_sc_uint;
                        wait();

                        whole = ((uint64_t) upper_half) << 32 + lower_half;
                        CTOS_FX_ASSIGN_RANGE(tmp, whole);

                        data_to_dut.put(tmp);
                        wait();
                }
        }
}

void svd_wrapper::handle_output() {
        SVD_CELL_TYPE tmp;
        sc_uint<64> tmp_sc_uint;
        uint64_t whole;
        uint32_t lower_half, upper_half;
        unsigned length, index;

        data_out.reset_put();
        data_from_dut.reset_get();
        wait();

        while (true) {
                do { wait(); }
                while (!wr_request.read());
                wr_grant.write(true);
                length = wr_length.read();
                index = wr_index.read();
                do { wait(); }
```

```
                while (wr_request.read());
                wr_grant.write(false);

WRAPPER_OUTPUT_LOOP:
                for (int i = 0; i < length; ++i) {
                        tmp = data_from_dut.get();

                        /* TODO check this does what I expect */
                        tmp_sc_uint = tmp.range();
                        whole = tmp_sc_uint;

                        lower_half = (uint32_t) whole;
                        upper_half = (uint32_t) (whole >> 32);

                        data_out.put(lower_half);
                        wait();
                        data_out.put(upper_half);
                        wait();
                }
        }
}
#ifdef __CTOS__
SC_MODULE_EXPORT(svd_wrapper)
#endif
#ifndef _SVD_WRAPPER_H_
#define _SVD_WRAPPER_H_

#define SC_INCLUDE_FX
#include "systemc.h"
#include <ctos_flex_channels.h>

#include <stdint.h>

#include "svd.h"
#include "svd_data.h"

SC_MODULE(svd_wrapper) {
        sc_in<bool> clk;
        sc_in<bool> rst;

        get_initiator<uint32_t> data_in;
        put_initiator<uint32_t> data_out;

        //sc_out<bool> svd_done_irq;

        void config_svd(void);
```

```
        void handle_irq(void);
        void handle_input(void);
        void handle_output(void);

        SC_CTOR(svd_wrapper):
                dut("svd")
        {
                /* processes */
                SC_CTHREAD(config_svd, clk.pos());
                reset_signal_is(rst, false);
                SC_CTHREAD(handle_irq, clk.pos());
                reset_signal_is(rst, false);
                SC_CTHREAD(handle_input, clk.pos());
                reset_signal_is(rst, false);
                SC_CTHREAD(handle_output, clk.pos());
                reset_signal_is(rst, false);

                /* external interface */
                data_in.clk_rst(clk, rst);
                data_out.clk_rst(clk, rst);

                /* this -> signals */
                data_to_dut(bufdin);
                data_from_dut(bufdout);
                data_to_dut.clk_rst(clk, rst);
                data_from_dut.clk_rst(clk, rst);

                /* signals -> dut */
                dut.clk(clk);
                dut.rst(dut_rst);
                dut.rd_index(rd_index);
                dut.rd_length(rd_length);
                dut.rd_request(rd_request);
                dut.rd_grant(rd_grant);
                dut.wr_index(wr_index);
                dut.wr_length(wr_length);
                dut.wr_request(wr_request);
                dut.wr_grant(wr_grant);
                dut.bufdin(bufdin);
                dut.bufdout(bufdout);
                dut.conf_size(conf_size);
                dut.conf_done(conf_done);
                dut.svd_done(svd_done);
        }

private:
```

```
        svd dut;
        sc_signal<bool> dut_rst;
        sc_signal<unsigned> rd_index;
        sc_signal<unsigned> rd_length;
        sc_signal<bool> rd_request;
        sc_signal<bool> rd_grant;
        sc_signal<unsigned> wr_index;
        sc_signal<unsigned> wr_length;
        sc_signal<bool> wr_request;
        sc_signal<bool> wr_grant;
        put_get_channel<SVD_CELL_TYPE> bufdin;
        put_get_channel<SVD_CELL_TYPE> bufdout;
        sc_signal<unsigned> conf_size;
        sc_signal<bool> conf_done;
        sc_signal<bool> svd_done;

        put_initiator<SVD_CELL_TYPE> data_to_dut;
        get_initiator<SVD_CELL_TYPE> data_from_dut;

        sc_signal<bool> svd_done_irq;
};


#endif
if { [file exist setup_locals.tcl] } {
        source setup_locals.tcl
} else {
        puts "Cannot find \"setup_locals.tcl\"."
        exit
}

if {[get_design] != "" } then {
        close_design
}

new_design tyrion
set_attr design_dir "" /designs/tyrion
set_attr auto_write_models "true" /designs/tyrion
define_sim_config -model_dir "./tyrion" /designs/tyrion
set_attr source_files [list ../src/svd.cpp ../src/svd_wrapper.cpp] /designs/tyrion
set_attr header_files [list ../src/svd.h ../src/svd_wrapper.h ../src/svd_ctos_funcs.h ..
set_attr compile_flags " -w -I../src/ -I/opt/zynq-math/src/" /designs/tyrion
set_attr top_module_path "svd_wrapper" /designs/tyrion
set_attr build_flat "true" /designs/tyrion
define_clock -name clk -period 20000 -rise 0 -fall 10000
define_sim_config -makefile_name ../ctos_sim_unisim/Makefile -model_dir tyrion -simulat
```

```
define_synth_config -run_dir "run_synth_gates" -standard_flow "default_synthesis_flow" -

# implementation
#set_attr implementation_target FPGA [get_design]
#set_attr fpga_install_path /opt/altera/quartus/14.1/quartus/linux64/quartus_sta [get_de
#set_attr fpga_target [list Altera Cyclone2 EP2C35F672C6] [get_design]
set_attr implementation_target FPGA [get_design]
set_attr fpga_install_path /tools/xilinx/14.6/ISE_DS/ISE/bin/lin64/xst [get_design]
set_attr fpga_target [list Xilinx virtex7 xc7vx485t-2-ffg1761] [get_design]
set_attr verilog_use_indexed_part_select false [get_design]
set_attr reset_registers internal_and_outputs [get_design]

build
write_sim_makefile -overwrite

set_attr default_scheduling_effort high [get_design]

# loops
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/copyMatrix/nodes/
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/cordic_func_80__6
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/findLargestElemen
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/identify/nodes/ID
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper_dut_l
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/multiply/nodes/M
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper_dut_s
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper_dut_s
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper_dut_s
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/reorder/nodes/RE
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/reorder/nodes/RE
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/swapRows/nodes/SW
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/transpose/nodes/T
break_combinational_loop /designs/tyrion/modules/svd_wrapper/behaviors/transpose/nodes/T

# initial memory
flatten_array  /designs/tyrion/modules/svd_wrapper/arrays/a22 /designs/tyrion/modules/sv

# functions
inline /designs/tyrion/modules/svd_wrapper/behaviors/transpose /designs/tyrion/modules/s


# memory
#allocate_prototype_memory -interface_types {rw r } -clock /designs/tyrion/modules/svd_w
#allocate_prototype_memory -interface_types {rw } -clock /designs/tyrion/modules/svd_wra
#allocate_prototype_memory -interface_types {r w } -clock /designs/tyrion/modules/svd_wr
#allocate_prototype_memory -interface_types {rw r } -clock /designs/tyrion/modules/svd_w
#allocate_prototype_memory -interface_types {rw } -clock /designs/tyrion/modules/svd_wra
```

```
#allocate_prototype_memory -interface_types {rw r } -clock /designs/tyrion/modules/svd_w

allocate_prototype_memory /designs/tyrion/modules/svd_wrapper/arrays/dut_v_m_mant
allocate_prototype_memory /designs/tyrion/modules/svd_wrapper/arrays/dut_Ui_m_mant
allocate_prototype_memory /designs/tyrion/modules/svd_wrapper/arrays/dut_matrix_in_m_man
allocate_prototype_memory /designs/tyrion/modules/svd_wrapper/arrays/dut_s_m_mant
allocate_prototype_memory /designs/tyrion/modules/svd_wrapper/arrays/dut_tempMatrix_m_ma
allocate_prototype_memory /designs/tyrion/modules/svd_wrapper/arrays/dut_u_m_mant

# schedule
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
set_attr relax_latency "true" /designs/tyrion/modules/svd_wrapper/behaviors/svd_wrapper
schedule -passes 200 -verbose /designs/tyrion/modules/svd_wrapper

allocate_registers /designs/tyrion/modules/svd_wrapper

write_sim -type verilog -suffix _final -dir ./tyrion /designs/tyrion/modules/svd_wrapper
write_rtl -file ./tyrion/tyrion_rtl.v /designs/tyrion/modules/svd_wrapper

report_timing > timing.txt
report_area > area.txt
```

```
// CPSC 445 - Problem Set 1
// Anton Petrov 22-09-11

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "gm_jacobi.h"

void gm_identify (double *matrix, int dimension) {
        int row, col;
        for (row = 0; row < dimension; row++) {
                for (col = 0; col < dimension; col++) {
                        // Elements on diagonal =1
                        if (col == row) {matrix [(row * dimension) + col] = 1.0;}
                        // ... all others 0
                        else {matrix [(row * dimension) + col] = 0.0;}
                }
        }
```

```
        return;
}

void gm_copyMatrix (double *a, double *b, int dimension) {
        int row, col;
        for (row = 0; row < dimension; row++) {
                for (col = 0; col < dimension; col++)
                        b [(row * dimension) + col] = a [(row * dimension) + col];
        }
        return;
}

void gm_multiply (double *left, double *right, double *result, int dimension) {
        int leftRow, rightRow; // row numbers of the left and right matrix, respectively
        int leftCol, rightCol; // same as above but for columns
        double tempResult = 0;

        for (leftRow = 0; leftRow < dimension; leftRow++) {
                for (rightCol = 0; rightCol < dimension; rightCol++) {
                        for (leftCol = rightRow = 0; leftCol < dimension; leftCol++, rig
                                tempResult += left[(leftRow * dimension) + leftCol] * ri
                        result [(leftRow * dimension) + rightCol] = tempResult;
                        tempResult = 0;
                }
        }
        return;
}

void gm_jacobi (double *a, int n, double *s, double *u, double *v) {
        // Arrays that contain the coordinates of the elements of the 2x2
        // sub matrix formed by the largest off-diagonal element. First entry
        // is the row number and second entry is the column number.
        int *a11, *a12, *a21, *a22;
        LargestElement le; // largest element of matrix a
        le.value = le.rowNum = le.colNum = -1;
        int i, j;

        // I could have statically allocated these but I would not have been
        // able to pass them as double pointers: this is a known problem in C
        a11=(int*)malloc(sizeof(int)*2); a12=(int*)malloc(sizeof(int)*2);
        a21=(int*)malloc(sizeof(int)*2); a22=(int*)malloc(sizeof(int)*2);

        gm_identify (u, n);
        gm_identify (v, n);

        if (n == 1) {   // 1x1 matrix is already in SVD
```

```
                s[0] = a[0];
                return;
        }

        le = gm_findLargestElement (a, n, &a11, &a12, &a21, &a22);

        int count = 0;
        while (fabs (le.value) > 0.00000000000000000001) {
                count++;
                gm_rotate (a, n, u, v, a11, a12, a21, a22);
                le = gm_findLargestElement (a, n, &a11, &a12, &a21, &a22);
        }

        gm_reorder (a, n, u, v);

        // Copy over the singular values in a to s
        for (i = 0; i < n; i++) {
                j = i;
                s [i] = a [(i * n) + j];
        }

        free (a11); free (a12); free (a21); free (a22);
        return;
}

void gm_rotate (double *a, int dimension, double *u, double *v, int *x11, int *x12, int
        double a11, a12, a21, a22; // elements of the sub-matrix
        double alpha, beta; // angles used in rotations; alpha is angle of left rotation
        double cosA, sinA, cosB, sinB;
        double *Ui, *Vi; // left and right rotation matrices, respectively
        double *tempResult;
        double X, Y; // temporary values used in calculating angles

        // Assign elements of sub-matrix to the their actual values from a
        a11 = a [x11[0] * dimension + x11[1]]; a12 = a [x12[0] * dimension + x12[1]];
        a21 = a [x21[0] * dimension + x21[1]]; a22 = a [x22[0] * dimension + x22[1]];

        // Calculate angles and sin and cos of those angles using the closed
        // formulas found.
        X = atan ((a21 - a12) / (a11 + a22)); Y = atan ((a21 + a12) / (a11 - a22));
        alpha = 0.5 * (X + Y);
        beta = 0.5 * (Y - X);
        cosA = cos (alpha); sinA = sin (alpha);
        cosB = cos (beta); sinB = sin (beta);

        // Create left rotation matrix, namely U_i which looks like this
```

```
        //
        //              | cosA  sinA |
        //              | -sinA cosA |
        //
        Ui = (double *) malloc (sizeof (double) * (dimension * dimension));
        gm_identify (Ui, dimension);
        Ui [x11[0] * dimension + x11[1]] = Ui [x22[0] * dimension + x22[1]] = cosA;
        Ui [x12[0] * dimension + x12[1]] = sinA;
        Ui [x21[0] * dimension + x21[1]] = (-1.0) * sinA;

        // Create the right rotation matrix, namely V_i which looks like this
        //
        //              | cosB  -sinB |
        //              | sinB   cosB |
        //
        Vi = (double *) malloc (sizeof (double) * (dimension * dimension));
        gm_identify (Vi, dimension);
        Vi [x11[0] * dimension + x11[1]] = Vi [x22[0] * dimension + x22[1]] = cosB;
        Vi [x12[0] * dimension + x12[1]] = (-1.0) * sinB;
        Vi [x21[0] * dimension + x21[1]] = sinB;

        // Rotate a
        // First on the left with Ui
        tempResult = (double *) malloc (sizeof (double) * (dimension * dimension));
        gm_multiply (Ui, a, tempResult, dimension);
        gm_copyMatrix (tempResult, a, dimension);
        // Then on the right with Vi
        gm_multiply (a, Vi, tempResult, dimension);
        gm_copyMatrix (tempResult, a, dimension);

        // Apply rotation matrix Ui to U
        gm_multiply (Ui, u, tempResult, dimension);
        gm_copyMatrix (tempResult, u, dimension);
        // Apply rotation matrix Vi to V
        gm_multiply (v, Vi, tempResult, dimension);
        gm_copyMatrix (tempResult, v, dimension);

        // Free rotation matrices
        free (Ui); free (Vi); free (tempResult);
        return;
}

LargestElement gm_findLargestElement (double *matrix, int dimension, int **a11, int **a1
        LargestElement *leArray;
        LargestElement leTemp;
        int i, j;
```

```
leTemp.value = 0;
leTemp.rowNum = leTemp.colNum = -1;

leArray = (LargestElement *) malloc (sizeof (LargestElement) * dimension);

// Populate leArray such that the entry at index i contains information
// about the largest element of row i
for (i = 0; i < dimension; i++) {
        for (j = 0; j < dimension; j++) {
                // We are looking for the largest OFF-DIAGONAL element
                if (j == i)
                        continue;
                if (fabs (matrix [(i * dimension) + j]) > fabs (leTemp.value)) {
                        leTemp.value = matrix[(i * dimension) + j];
                        leTemp.rowNum = i;
                        leTemp.colNum = j;
                }
        }
        leArray[i].value = leTemp.value;
        leArray[i].rowNum = leTemp.rowNum;
        leArray[i].colNum = leTemp.colNum;
        leTemp.value = 0;
}
leTemp.value = 0;
leTemp.rowNum = leTemp.colNum = -1;
// Iterate over leArray and find the largest element in the matrix as a
// whole
for (i = 0; i < dimension; i++) {
        if (fabs (leArray [i].value) > fabs (leTemp.value)) {
                leTemp.value = leArray [i].value;
                leTemp.rowNum = leArray [i].rowNum;
                leTemp.colNum = leArray [i].colNum;
        }
}

free (leArray);

// Determining coordinates of the 2x2 sub matrix formed by this largest
// element
if (leTemp.rowNum < leTemp.colNum) { // largest element is above diagonal
        (*a11) [0] = (*a11) [1] = (*a12) [0] = (*a21) [1] = leTemp.rowNum;
        (*a21) [0] = (*a22) [0] = (*a22) [1] = (*a12) [1] = leTemp.colNum;
}
else { // below diagonal
        (*a22) [0] = (*a22) [1] = (*a21) [0] = (*a12) [1] = leTemp.rowNum;
```

```
                (*a11) [0] = (*a11) [1] = (*a21) [1] = (*a12) [0] = leTemp.colNum;
        }

        return leTemp;
}

void gm_transpose (double *matrix, int dimension) {
        double temp;
        int row, col;
        for (row = 0; row < dimension; row++) {
                for (col = (row + 1); col < dimension; col++) {
                        if (col == row) {continue;} // skip diagonal elements
                        temp = matrix [(row * dimension) + col];
                        matrix [(row * dimension) + col] = matrix [(col * dimension) + r
                        matrix [(col * dimension) + row] = temp;
                }
        }
        return;
}

void gm_reorder (double *a, int dimension, double *u, double *v) {
        int row, col, x, largestElementRow;
        double temp; // stores the largest singular value
        double *p; // permutation matrix
        double *tempMatrix;

        p = (double *) malloc (sizeof (double) * (dimension * dimension));
        tempMatrix = (double *) malloc (sizeof (double) * (dimension * dimension));

        for (x = 0; x < dimension; x++) {
                temp = 0.0;
                gm_identify (p, dimension);
                for (row = x; row < dimension; row++) {
                        col = row;
                        if (fabs (a [row * dimension + col]) > fabs (temp)) {
                                temp = a [row * dimension + col];
                                largestElementRow = row;
                        }
                }
                gm_swapRows (p, x, largestElementRow, dimension);
                // Reorder a
                gm_multiply (p, a, tempMatrix, dimension); gm_copyMatrix (tempMatrix, a,
                gm_multiply (a, p, tempMatrix, dimension); gm_copyMatrix (tempMatrix, a,
                // Reorder u
                gm_multiply (p, u, tempMatrix, dimension); gm_copyMatrix (tempMatrix, u,
                // Reorder v
```

```
                gm_multiply (v, p, tempMatrix, dimension); gm_copyMatrix (tempMatrix, v,
        }
        gm_transpose (u, dimension);
        gm_transpose (v, dimension);
        gm_identify (p, dimension);
        for (row = 0; row < dimension; row++) {
                col = row;
                if (a [row * dimension + col] < 0)
                        p [row * dimension + col] = -1.0;
        }
        gm_multiply (p, a, tempMatrix, dimension);
        gm_copyMatrix (tempMatrix, a, dimension);
        gm_multiply (u, p, tempMatrix, dimension);
        gm_copyMatrix (tempMatrix, u, dimension);
        free (p); free (tempMatrix);
        return;
}

// Swap row a with row b of matrix m
void gm_swapRows (double *m, int a, int b, int dimension) {
        double temp;
        int col;
        for (col = 0; col < dimension; col++) {
                temp = m [a * dimension + col];
                m [a * dimension + col] = m [b * dimension + col];
                m [b * dimension + col] = temp;
        }
        return;
}
// Anton Petrov
// jacobi.h - contains function declarations

typedef struct {
        double value;
        int rowNum;
        int colNum;
} LargestElement;

// Self-explanatory
void gm_multiply (double *left, double *right, double *result, int dimension);

// Function that performs a number of jacobi rotations on the input matrix a
// and returns the SVD of that matrix in the u, s and v matrices
void gm_jacobi (double *a, int n, double *s, double *u, double *v);

// Find largest off-diagonal element in input matrix. Uses the LargestElement
```

```
// struct to store the row, col and value of the element.
LargestElement gm_findLargestElement (double *matrix, int dimension, int **a11, int **a1

// Self-explanatory
void gm_copyMatrix (double *a, double *b, int dimension);

// Loop over matrix and transform it into the identity matrix
void gm_identify (double *matrix, int dimension);

// Perform a single rotation
void gm_rotate (double *a, int dimension, double *u, double *v, int *x11, int *x12, int

// Calculate the transpose of the input matrix in place
void gm_transpose (double *matrix, int dimension);

// Apply a series of permutation matrices to a so that the singular values
// are ordered in decreasing order. Apply those same permutations in the
// correct order to u and v.
void gm_reorder (double *a, int dimension, double *u, double *v);

// Swap two rows of a matric in place.
void gm_swapRows (double *m, int a, int b, int dimension);
#define SC_INCLUDE_FX
#include "systemc.h"
#include <ctos_flex_channels.h>

#include <stdint.h>

#include "svd_wrapper.h"
//#include "svd.h"
#include "svd_tb.h"
#include "svd_data.h"

int sc_main(int, char**) {
        sc_report_handler::set_actions("/IEEE_Std_1666/deprecated",
                        SC_DO_NOTHING);

        int clk_in = 20;

        sc_clock clk("clk", clk_in, SC_NS);
        sc_signal<bool> rst;
        sc_signal<bool> rst_dut;

#if 0
        /* from TB to DUT */
        sc_signal<bool> rd_request;
```

```
        sc_signal<bool> rd_grant;
        sc_signal<svd_token> rd_data;

        /* from DUT to TB */
        sc_signal<bool> wr_request;
        sc_signal<bool> wr_grant;
        sc_signal<svd_token> wr_data;
#endif

        // DMA requests interface from memory to device
        sc_signal<unsigned> rd_index;   // array index (offset from base address)
        sc_signal<unsigned> rd_length;  // burst size
        sc_signal<bool>     rd_request; // transaction request
        sc_signal<bool>      rd_grant;   // transaction grant

        // DMA requests signalterface from device to memory
        sc_signal<unsigned> wr_index;   // array index (offset from base address)
        sc_signal<unsigned> wr_length;  // burst size
        sc_signal<bool>     wr_request; // transaction request
        sc_signal<bool>      wr_grant;   // transaction grant

        // input data read by load_input
        put_get_channel<SVD_CELL_TYPE> bufdin("bufdin-yolo");
        // output data written by store output
        put_get_channel<SVD_CELL_TYPE> bufdout("bufdout-yolo");

        sc_signal<unsigned> conf_size;
        sc_signal<bool>     conf_done;

        // computation complete
        //sc_signal<bool>    svd_done;


#if 0
        svd dut("dut");
#endif
        svd_tb tb("tb");

#if 0
        dut.clk(clk);
        dut.rst(rst_dut);
        dut.rd_index(rd_index);
        dut.rd_length(rd_length);
        dut.rd_request(rd_request);
        dut.rd_grant(rd_grant);
        dut.wr_index(wr_index);
```

```
        dut.wr_length(wr_length);
        dut.wr_request(wr_request);
        dut.wr_grant(wr_grant);
        dut.bufdin(bufdin);
        dut.bufdout(bufdout);
        dut.conf_size(conf_size);
        dut.conf_done(conf_done);
        dut.svd_done(svd_done);
#endif

        put_get_channel<uint32_t> data_in("din");
        put_get_channel<uint32_t> data_out("dout");

        svd_wrapper wrapper("wrapper");
        wrapper.clk(clk);
        wrapper.rst(rst_dut);
        wrapper.data_in(data_in);
        wrapper.data_out(data_out);

        tb.clk(clk);
        tb.rst(rst);
        tb.rst_dut(rst_dut);
#if 0
        tb.data_in(rd_data);
        tb.req_in(rd_request);
        tb.grant_in(rd_grant);
        tb.data_out(wr_data);
        tb.req_out(wr_request);
        tb.grant_out(wr_grant);
#endif

#if 0
        tb.rd_index(rd_index);
        tb.rd_length(rd_length);
        tb.rd_request(rd_request);
        tb.rd_grant(rd_grant);
        tb.wr_index(wr_index);
        tb.wr_length(wr_length);
        tb.wr_request(wr_request);
        tb.wr_grant(wr_grant);
        tb.bufdin(bufdin);
        tb.bufdout(bufdout);
        tb.conf_size(conf_size);
        tb.conf_done(conf_done);
        tb.svd_done(svd_done);
#endif
```

```
        tb.data_to_dut(data_in);
        tb.data_from_dut(data_out);


        /* simulation */
        rst.write(false);
        sc_start(clk_in*2, SC_NS);
        rst.write(true);

        sc_start();
}
#include "svd_tb.h"
#include "gm_jacobi.h"

void svd_tb::fill_buf() {
        //Not sure what headers I can use here so
        //real basic matrix
        int i = 0;
        for(; i < SVD_INPUT_SIZE(mat_size); i++) {
                int tmp = rand() % 8192;
                input_matrix[i] = tmp;
                golden_input_matrix[i] = tmp;
        }
}

void svd_tb::compute_golden_model(void) {
        gm_jacobi(golden_input_matrix, mat_size,
                        SVD_GET_S(golden_matrix, mat_size),
                        SVD_GET_U(golden_matrix, mat_size),
                        SVD_GET_V(golden_matrix, mat_size));
}

void print_matrix(double *mat, int size) {
        for (int i = 0; i < size; ++i) {
                for (int j = 0; j < size; ++j) {
                        printf("%f ", mat[i * size + j]);
                }
                printf("\n");
        }
}

#ifndef REAL_FLOAT
void print_matrix(SVD_CELL_TYPE *mat, int size) {
        for (int i = 0; i < size; ++i) {
                for (int j = 0; j < size; ++j) {
```

```
                printf("%f ", mat[i * size + j].to_double());
            }
            printf("\n");
        }
}
#endif

void svd_tb::dmac(void) {
        // RESET
        srand(time(NULL));
        rst_dut.write(0);
#if 0
        rd_grant.write(0);
        wr_grant.write(0);
        conf_done.write(0);
        bufdin.reset_put();
        bufdout.reset_get();
#endif
        data_to_dut.reset_put();
        data_from_dut.reset_get();

        wait();
        mat_size = MAX_SIZE;
        fill_buf();
        compute_golden_model();

        rst_dut.write(1);
#if 0
        conf_size.write(mat_size);
        conf_done.write(1);
#endif

#if 0
        while (true) {
            do { wait(); }
            while (!rd_request.read() && !wr_request.read() && !svd_done.read());

            if (svd_done.read()) {
                cout << "TB: SVD done @ " << sc_time_stamp() << endl;
                // Reset DUT
                rst_dut.write(0);

                for (int i = 0; i < SVD_OUTPUT_SIZE(mat_size); ++i) {
                    double diff =
#ifdef REAL_FLOAT
                                abs(output_matrix[i])
```

```
#else
                                        abs(output_matrix[i].to_double())
#endif
                                        - abs(golden_matrix[i]);
                                if (diff < 0) {
                                        diff *= -1;
                                }

                                mismatches += !(diff < MAX_ERROR);
                        }

                        /* exit loop now */
                        break;
                }

                if (rd_request.read()) {
                        unsigned index = rd_index.read();
                        unsigned length = rd_length.read();

                        rd_grant.write(true);
                        do { wait(); }
                        while (rd_request.read());
                        rd_grant.write(false);
                        wait();

                        // DMA from memory to DUT
                        int i;
                        for (i = index; i < index + length; ++i) {
                                bufdin.put(input_matrix[i]);
#ifdef VERBOSE
                                cout << "TB PUT: (index, length, i, val)" << index << "
                                        << " " << i << " " << input_matrix[i] << endl;
#endif
                                wait();
                        }
                } else if (wr_request.read()) {
                        unsigned index = wr_index.read();
                        unsigned length = wr_length.read();

                        wr_grant.write(true);
                        do { wait(); }
                        while (wr_request.read());
                        wr_grant.write(false);
                        wait();

                        // DMA from device to memory
```

```
                        int i;
                        for (i = index; i < index + length; ++i) {
                                output_matrix[i] = bufdout.get();
#ifdef VERBOSE
                                cout << "TB GET: (index, length, i, val)" << index << "
                                        << " " << i << " " << input_matrix[i] << endl;
#endif
                                wait();
                        }
                }
        }
#endif
top:
        /* put data to DUT */
        for (int i = 0; i < SVD_INPUT_SIZE(MAX_SIZE); ++i) {
                SVD_CELL_TYPE tmp;
                sc_uint<64> tmp_sc_uint;
                uint64_t whole;
                uint32_t lower_half, upper_half;

                tmp = input_matrix[i];

                /* TODO check this does what I expect */
                tmp_sc_uint = tmp.range();
                whole = tmp_sc_uint;

                lower_half = whole;
                upper_half = whole >> 32;

                data_to_dut.put(lower_half);
                wait();
                data_to_dut.put(upper_half);
                wait();
        }

        /* get back result */
        for (int i = 0; i < SVD_OUTPUT_SIZE(MAX_SIZE); ++i) {
                SVD_CELL_TYPE tmp;
                sc_uint<32> tmp_sc_uint;
                uint64_t whole;
                uint32_t lower_half, upper_half;

                tmp_sc_uint = data_from_dut.get();
                lower_half = tmp_sc_uint;
                wait();
```

```
                tmp_sc_uint = data_from_dut.get();
                upper_half = tmp_sc_uint;
                wait();

                whole = ((uint64_t) upper_half) << 32 + lower_half;
                CTOS_FX_ASSIGN_RANGE(tmp, whole);
                output_matrix[i] = tmp;
        }

#if 0
        static int one = 1;
        if (!--one)
                goto top;
#endif

        // Stop simulation
        sc_stop();

#ifdef VERBOSE
        /* print matrices */
        cout << "Input: " << endl;
        print_matrix(input_matrix, mat_size);
        cout << endl;

        cout << "Software: " << endl;
        print_matrix(SVD_GET_S(golden_matrix, mat_size), mat_size);
        print_matrix(SVD_GET_U(golden_matrix, mat_size), mat_size);
        print_matrix(SVD_GET_V(golden_matrix, mat_size), mat_size);
        cout << endl;

        cout << "Hardware: " << endl;
        print_matrix(SVD_GET_S(output_matrix, mat_size), mat_size);
        print_matrix(SVD_GET_U(output_matrix, mat_size), mat_size);
        print_matrix(SVD_GET_V(output_matrix, mat_size), mat_size);
        cout << endl << endl;
#endif

        int errors = get_mismatches();      // show the testing summary
        if (errors > 0)
                cout << "Simulation with " << errors << " mismatches." << endl;
        else
                cout << "Simulation Successful! @ " << sc_time_stamp() << endl;
}

#ifndef __SVDTB_H__
#define __SVDTB_H__
```

```
#define SC_INCLUDE_FX
#include "systemc.h"
#include <ctos_flex_channels.h>

#include <iostream>
#include <iomanip>
#include <ctime>
#include <ostream>
#include <stdint.h>
#include "svd_data.h"

//#define VERBOSE
#define MAX_ERROR (0.0001)

SC_MODULE(svd_tb) {
        sc_in<bool> clk;
        sc_in<bool> rst;
        sc_out<bool> rst_dut;

#if 0
        // DMA requests interface from memory to device
        sc_in<unsigned>   rd_index;     // array index (offset from starting address)
        sc_in<unsigned>   rd_length;    // burst size (in words)
        sc_in<bool>       rd_request;   // transaction request
        sc_out<bool>      rd_grant;     // transaction grant

        // DMA requests interface from device to memory
        sc_in<unsigned>   wr_index;   // array index (offset from starting address)
        sc_in<unsigned>   wr_length;  // burst size (in words)
        sc_in<bool>       wr_request; // transaction request
        sc_out<bool>      wr_grant;   // transaction grant

        // input data readen by load_input
        put_initiator<SVD_CELL_TYPE> bufdin;
        // output data written by store output
        get_initiator<SVD_CELL_TYPE> bufdout;

        sc_out<unsigned> conf_size;
        sc_out<bool>     conf_done;
#endif
        put_initiator<uint32_t> data_to_dut;
        get_initiator<uint32_t> data_from_dut;

        // computation complete. Written by store_output
        //sc_in<bool> svd_done;
```

```
        void dmac(void);

        SC_CTOR(svd_tb) {
                SC_CTHREAD(dmac, clk.pos());
                reset_signal_is(rst, false);

#if 0
                bufdin.clk_rst(clk,rst);
                bufdout.clk_rst(clk,rst);
#endif
                data_to_dut.clk_rst(clk,rst);
                data_from_dut.clk_rst(clk,rst);
        }

        int get_mismatches() {return mismatches;}

        private:
        void fill_buf(void);
        void compute_golden_model(void);
        int mismatches;
        int mat_size;

        SVD_CELL_TYPE input_matrix[SVD_INPUT_SIZE(MAX_SIZE)];
        SVD_CELL_TYPE output_matrix[SVD_OUTPUT_SIZE(MAX_SIZE)];

        double golden_input_matrix[SVD_INPUT_SIZE(MAX_SIZE)];
        double golden_matrix[SVD_OUTPUT_SIZE(MAX_SIZE)];
};
#endif
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

#include <Eigen/Core>
#include <Eigen/SVD>

using namespace std;
using namespace Eigen;

void test_svd(MatrixXd m) {
        JacobiSVD<MatrixXd> svd(m, ComputeThinU | ComputeThinV);
#if 0
        cout << "Here is the matrix m:" << endl << m << endl;
#endif
```

```
        cout << svd.singularValues() << endl;
#if 0
        cout << "Its left singular vectors are the columns of the thin U matrix:" <<
                endl << svd.matrixU() << endl;
        cout << "Its right singular vectors are the columns of the thin V matrix:"
                << endl << svd.matrixV() << endl;
#endif
}

int main() {
        int n;

        typedef vector<vector<double> > Rows;
        Rows rows;
        ifstream input("input-matrix.txt");
        char const row_delim = '\n';
        char const field_delim = '\t';

        input >> n;

        double **r;
        int i, j;


        r = (double **) malloc(sizeof *r * n);
        for (i = 0; i < n; ++i)
                r[i] = (double *) malloc(sizeof **r * n);

        i = -1; j = 0;

        for (string row; getline(input, row, row_delim); ++i) {
                rows.push_back(Rows::value_type());
                istringstream ss(row);
                j = 0;
                for (string field; getline(ss, field, field_delim);++j ) {
                        double tmp = stod(field);
                        r[i][j] = tmp;
                        rows.back().push_back(tmp);
                }
        }

        MatrixXd m(n,n);

        for (i = 0; i < n; ++i) {
                for (j = 0; j < n; ++j) {
                        m(i,j) = r[i][j];
```

```
                }
        }

        test_svd(m);
        return 0;
}
EIGEN_PATH ?= ../install_eigen/include/eigen3/

CXXFLAGS=-I$(EIGEN_PATH) -std=c++11

EXE=eigen_test

default: $(EXE)

run: $(EXE)
        ./$(EXE)
import random
import sys

def random_matrix(n):
    print("{}".format(n))
    for x in range (0, n):
        s = ""
        for y in range (0, n):
            s += "{}\t".format(random.uniform(-10000, 10000))
        print(s)


random_matrix(int(sys.argv[1]))

#include <iostream>
#include <fstream>
#include <vector>
#include <string>

#include <Eigen/Core>
#include <Eigen/SVD>

using namespace std;
using namespace Eigen;

void test_svd(MatrixXd m) {
        JacobiSVD<MatrixXd> svd(m, ComputeThinU | ComputeThinV);
#if 0
        cout << "Here is the matrix m:" << endl << m << endl;
#endif
```

```
        cout << svd.singularValues() << endl;
#if 0
        cout << "Its left singular vectors are the columns of the thin U matrix:" <<
                endl << svd.matrixU() << endl;
        cout << "Its right singular vectors are the columns of the thin V matrix:"
                << endl << svd.matrixV() << endl;
#endif
}

int main() {
        int n;

        typedef vector<vector<double> > Rows;
        Rows rows;
        ifstream input("input-matrix.txt");
        char const row_delim = '\n';
        char const field_delim = '\t';

        input >> n;

        double **r;
        int i, j;


        r = (double **) malloc(sizeof *r * n);
        for (i = 0; i < n; ++i)
                r[i] = (double *) malloc(sizeof **r * n);

        i = -1; j = 0;

        for (string row; getline(input, row, row_delim); ++i) {
                rows.push_back(Rows::value_type());
                istringstream ss(row);
                j = 0;
                for (string field; getline(ss, field, field_delim);++j ) {
                        double tmp = stod(field);
                        r[i][j] = tmp;
                        rows.back().push_back(tmp);
                }
        }

        MatrixXd m(n,n);

        for (i = 0; i < n; ++i) {
                for (j = 0; j < n; ++j) {
                        m(i,j) = r[i][j];
```

```
                }
        }

        test_svd(m);
        return 0;
}
EIGEN_PATH ?= ../install_eigen/include/eigen3/

CXXFLAGS=-I$(EIGEN_PATH) -std=c++11

EXE=eigen_test

default: $(EXE)

run: $(EXE)
        ./$(EXE)
import random
import sys

def random_matrix(n):
    print("{}".format(n))
    for x in range (0, n):
        s = ""
        for y in range (0, n):
            s += "{}\t".format(random.uniform(-10000, 10000))
        print(s)


random_matrix(int(sys.argv[1]))
```