IKSwift

An Inverse Kinematics Accelerator

Yipeng Huang, Lianne Lairmore, Richard Townsend
{yipeng, lairmore, rtownsend}@cs.columbia.edu

May 14, 2014

Contents

1 Introduction

We present IKSwift, a specialized hardware design to tackle the inverse kinematics problem.

Inverse kinematics is widely used in robotics computing and in computer graphics. The problem takes as input a configuration of mechanical joints, which can be rotational or sliding, that are present in an arm or a leg. Then, it takes as input the limb's current shape and a target shape, solving for the required joint motions to get to the desired shape.

We have developed a configurable solver on an FPGA, in hopes of speeding up solutions when compared to running the same algorithm on a regular CPU.

1.1 Motivation

Computers are increasingly embedded in the real world, often permanently attached to sensors and actuators. One example of these systems are robots. Computer systems that must coordinate with sensors and actuators are distinct from general purpose computers, and the desired hardware to support its applications will also be different.

Specialty hardware such as GPUs can support hard-hitting *sensing* algorithms, especially those that involve image processing. Equally important, but less well studied, is how computer hardware should adapt to support *controlling* actuators. Actuator algorithms have yet to appear in well known computer architecture research workloads. Early studies of computer architecture support for robotics show that general purpose CPUs suffer when running typical robotic workloads [1].

Problems that arise when controlling actuators, such as kinematics, dynamics, obstacle avoidance, and collision detection, have been found to occupy a large portion of computer runtime in robotics. Live measurements show 33-66 percent dedicated to embedded computer on robot. In particular, the inverse kinematic problem is interesting because it has features of two distinct workload categories: sparse matrix math and graph traversal [2]. This project focuses on building hardware for inverse kinematics.

1.2 Inverse Kinematics Algorithm

We first present an algorithm that takes as input n homogenous transformation matrices T_i^{i-1} for $i = 1$ to n, and the current position of our end effector in three-dimensional space, which is represented as the vector s. The output of the algorithm is the Jacobian matrix for the current system configuration. Given matrices A and B, we use the notation AB for matrix multiplaction and $A \times B$ for the cross-product in the following pseudocode.

Let J be a 6 x n matrix, where each column corresponds to a joint. The top three scalars of a column represent the position of the joint in 3-space, while the bottom three represent the orientation of the joint in 3-space.
Let z be the z-axis of the coordinate frame at the base of our robot appendage
for $i = 1$ *to* n **do**
  Let R_i be the 3x3 rotation block derived from T_i^0
  Let $v_i = zR_i$ be the axis of rotation or translation for joint i
  Let p_i be a column vector composed of the top three scalars in the last column of T_i^0. This is the current position of joint i
  if *joint* i *is rotational* **then**
    Set column i of J to be $\begin{bmatrix}(v_i \times (s - p_i)) & v_i\end{bmatrix}^T$
  else
    Set column i of J to be $\begin{bmatrix}v_i & 0 & 0 & 0\end{bmatrix}^T$
  end
end
Return J

Algorithm 1: Jacobian($T_1^0, T_2^1, \ldots, T_n^{n-1}, s$)

The following algorithm describes the Jacobian Damped Least-Squares method of solving the inverse kinematics problem. As input we are given a set of D-H parameters that fully define the initial positions of the joints of our robot appendage, as well as a three-dimensional target position for our end effector. Formally, for each joint $i = 1 \ldots n$ we have D-H parameters $\theta_i, d_i, a_i, \alpha_i$, and we call our target vector t. The final output of our algorithm is a set of updated D-H parameters that fully define the required position of our joints such that the end effector position is sufficiently close to the target. Although the algorithm given applies to a general $n-$jointed robot, we only consider robots with 6 joints in our system, leading to a 6x6 Jacobian matrix.

Let ϵ be the desired accuracy of our final results;
for $i = 1$ *to* n **do**
    Calculate the homogenous transformation matrix T_i^{i-1} for joint i using the given D-H parameters;
end
Let $T_n^0 = \prod_{i=1}^{n} T_i^{i-1}$ be the full homogenous transformation matrix for the system;
Let s be a column vector composed of the top three scalars in the last column of T_n^0. This is the current position of our end effector;
Let $e = t - s$ be the desired change in the position of our end effector;
Set $e = \begin{bmatrix} e & 0 & 0 & 0 \end{bmatrix}^T$ so we can use it in our Jacobian equations, which deal with 6 x 6 matrices;
Let $J = JACOBIAN(T_1^0, T_2^1, \ldots, T_n^{n-1}, s)$ be the Jacobian matrix for the current system configuration;
while *the* l^2 *norm of* e *is greater than* ϵ **do**
    Let J^T be the transpose of J;
    Let λ be a small positive constant;
    Let I be the identity matrix;
    Use row operations to determine the vector f that satisfies the equation $(JJ^T + \lambda^2 I)f = e$;
    Let $\Delta\theta = J^T f$ be a vector whose ith component is a change in joint i's angle. Note that if joint i is translational along the unit vector v_i, then the joint "angle" measures the distance moved in the direction v_i and the ith component in $\Delta\theta$ will be a change in d_i;
    for $i = 1$ *to* n **do**
        if *joint* i *is translational* **then**
            Set $d_i = d_i + \Delta\theta[i]$;
        else
            Set $\theta_i = \theta_i + \Delta\theta[i]$;
        end
        Recalculate T_i^{i-1};
    end
    Recalculate $T_n^0, s, e,$ and J using our updated homogenous transformation matrices;
end
Return the final set of D-H parameters currently specifying the positions of our joints;

Algorithm 2: Least-Squares($\{\theta_1, d_1, a_1, \alpha_1\}, \ldots, \{\theta_n, d_n, a_n, \alpha_n\}, t$)

2 Software Prototype

Our software prototyping has two goals. The first goal is to verify we understand and can translate the algorithm to hardware. The second purpose of our prototype is a way of verifying results from our hardware. We found an open source C++ project on github.com which computes the incremental angle movements for a given robot to reach a target position given a beginning position using three different algorithms [3]. The algorithms being used in the open source project were all ones we had been exploring to implement in hardware. The code was organized and very clean. It was perfect for us to grasp how the Jacobian Transpose, Jacobian Pseudo Inverse, and Damped Least Squares algorithms worked in practice since our only source had been higher level papers. This project also covers our second need, for a way to verify results from our hardware. The fact that the project wasn't implemented by us gives it credibility; it would be easy to implement the algorithm incorrectly in software and then in hardware and not realize the original software was wrong.

Along with using this open source project we implemented a smaller prototype that reflects the structures in our hardware and only uses the one algorithm we are using. By creating this second software project we were able to play around with structures to determine what would work best. For example, we were able to modify the code and see how fixed point or integer math worked instead of floating point. We also used this project to test different computational methods for sin and cos. It was important to test how the accuracy of our algorithm changed if we changed computational methods like the ones mentioned above. Editing was much easier to do in software and less time consuming.

Since parsing XML documents in software isn't really an important part of learning how to write embedded systems we decided to use part of the open source C++ project to parse the XML robot configuration files to retrieve the original joint positions and the joint types. The software from the cpp-inverse-kinematics-library parses a given XML file and gives the resulting joint information to software we have created. Our software then obtains a target position from the user and, given the target information and the joint configurations, it computes the next angle positions using our FPGA hardware component.

3 Architecture

For our project we use the FPGA as an accelerator for inverse kinematics computations. Software running on the ARM processor on the SoCKit board drives the FPGA and displays the incremental joint positions on a robot arm displayed with an openGL program. The user is allowed to specify a robot design via an XML file, which contains the Denavit-Hartenberg parameters for the robot. The software supplies target Cartesian coordinates to the accelerator, which returns updated joint configurations that move the robot arm's end effector towards the given target coordinates. The software uses the joint configuration to update an image on the monitor. The resulting image results in an animation of an appendage moving towards a target position.

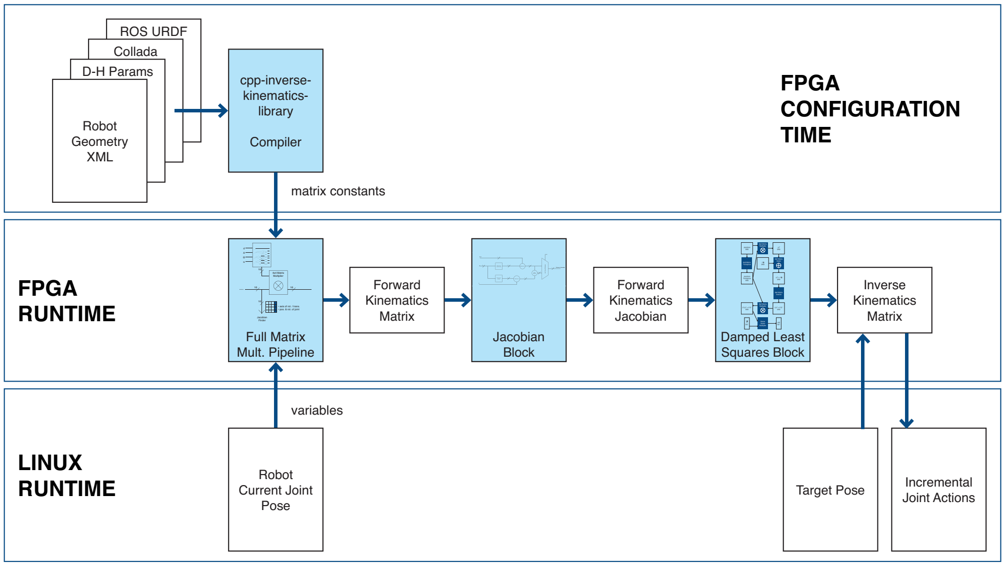


Figure 1: An architecture view of the software and hardware tools we will use for this design.

3.1 Software Driver

We designed a "joint" peripheral through which the software and hardware components of our system interact. The driver for the joint peripheral provides an ioctl that copies a struct to and from the user with the following components:

Field	Comments
`unsigned int start_signal`	signal for HW to run an iteration of the algorithm
`unsigned int done_signal`	signal for SW to run an iteration of the openGL display loop
`signed int target[3]`	The x,y,z, coordinates of the target position for our end-effector
`signed char joint`	Indicate which joint we're getting/setting
`unsigned char joint_type`	The ith bit is 1 if the ith joint is rotational, 0 for translational
`unsigned int magnitude`	

After displaying the most recent configuration of the robot appendage, the software will set `start_signal` to 1 to notify the hardware to run another iteration of the algorithm. The software will then spin until `done_signal` is set to 1, indicating that the D-H parameters have been updated and the appendage needs to be redisplayed. The `joint` field represents which joint we're referring to (we have a total of `MAX_JOINT` joints), the `joint_type` field keeps track of the type of every joint in system, and the `magnitude` field holds the value of the theta parameter for the joint in question (all other parameters are hard coded into the hardware). We also use the `joint` field to indicate when we're setting non-joint values. If `joint` $= -1$, then we're setting the `target` value, and if `joint` $= -2$, then we're setting the `start_signal` value.

3.2 Hardware Register Set

The registers used by the device driver are represented with the following struct, following the example given in lab 3:

```
struct joint_dev{
resource_size_t start; /* Address of start of registers */
resource_size_t size; /* Size of registers */
void __iomem *virtbase; /* Pointer to registers */
} dev;
```

We chose to have a fairly bare struct, as all of our interactions with the registers occur with ioreads and iowrites that interact directly with memory.

3.3 Top-Level Hardware Interface

```
module ikswift_top (
input logic clk, rst, en;
input logic [5:0] joint_type;
input logic target_0 [15:0];
input logic target_1 [15:0];
input logic target_2 [15:0];
input logic target_3 [15:0];
input logic target_4 [15:0];
input logic target_5 [15:0];
input logic dh_param_0 [3:0] [15:0];
input logic dh_param_1 [3:0] [15:0];
input logic dh_param_2 [3:0] [15:0];
input logic dh_param_3 [3:0] [15:0];
input logic dh_param_4 [3:0] [15:0];
input logic dh_param_5 [3:0] [15:0];
output logic done;
output logic output_0 [15:0];
output logic output_1 [15:0];
output logic output_2 [15:0];
output logic output_3 [15:0];
output logic output_4 [15:0];
output logic output_5 [15:0];
);
```

4 Microarchitecture

4.1 Hardware Submodules

In the following subsections, we describe the submodules of the accelerator. Then, we describe the custom functional units we have to build in order to assemble our submodules. We assemble these custom functional units using IP designs generated by Altera MegaFunctions.

We will represent numbers as 27-bit fixed point numbers in our hardware.

We are limited in the number of digital signal processors and lookup tables available in the FPGA. We pay attention to reusing units that use a lot of area, and time multiplex their use so they are used multiple times in the algorithm.

4.1.1 D-H Parameter Homogeneous Transformation Block

Frames are a set of axes and coordinates that describe 3D space. A local frame would be useful in describing the x, y, z positions of an object in space, along with the orientation (direction it is pointing) in space. Each link in a robot appendage has a frame associated with it.

A moving joint that connects two links results in a change in reference frames between the two links preceding and following the joint. We can transform from one frame to the next using homogeneous transforms, which are described as 4 by 4 matrices. For background information on homogeneous transforms refer to [4].

If we use the standard D-H parameters to describe the joint, this change in frames is a homogeneous transformation shown in Figure 2, which when multiplied out in full is the matrix shown in Figure 3.

$$T_i^{i-1} = Rot(Z, \theta_i)\, Trans(Z, d_i)\, Trans(X, a_i)\, Rot(X, \alpha_i)$$

Figure 2: The transformation between two frames linked by a joint, using D-H parameter variables.

Recall that revolute joints are represented as a rotation of joint angle θ about the Z axis, and prismatic joints are translations by link offset d along the Z axis. To align the two coordinates frames, we translate along the X axis by the link length a, and rotate by the X axis by the link twist α.

$$\begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3: The full matrix describing the transformation between two frames linked by a joint.

We can calculate this homogenous transformation using a dedicated hardware block in the FPGA. Figure 4 is the dataflow diagram for a hardware block that calculates all the elements in the transformation matrix. This submodule uses two instances of the sine cosine functional unit, which we describe later.

4.1.2 Full Matrix Multiplication Pipeline

A series of joints connected together by links in an appendage form a kinematic chain. To solve the inverse kinematics problem, we must first have the forward kinematic description

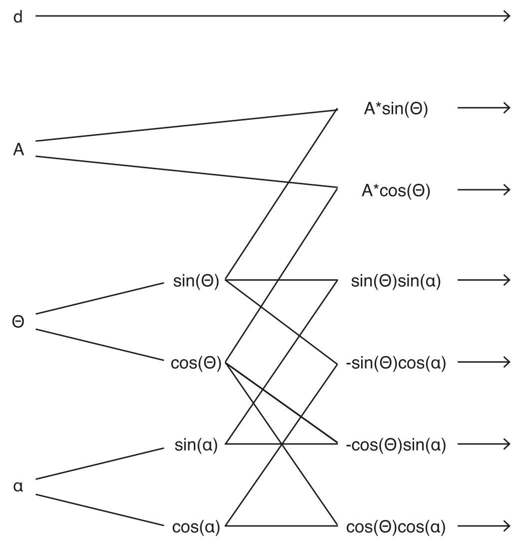


Figure 4: The dataflow diagram for a hardware block that calculates the elements in a homogenous transform matrix.

of the robot. We can describe the location and orientation of the end of the appendage with yet another 4 by matrix. This full forward kinematics matrix is simply the product of the matrices that describe each joint. We can calculate this in hardware using the pipeline shown in Figure 5.

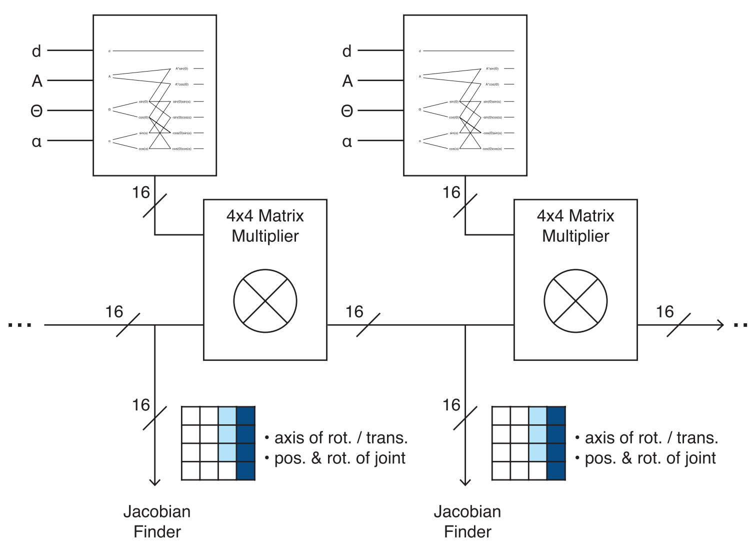


Figure 5: The homogenous transform matrix blocks are chained together to calculate the full transformation matrix of the forward kinematic chain.

4.1.3 Jacobian Block

The Jacobian matrix relates the differential motion of joints to differential motion in cartesian space. This matrix describes the velocity relationship between joints and the end of the actuator [5].

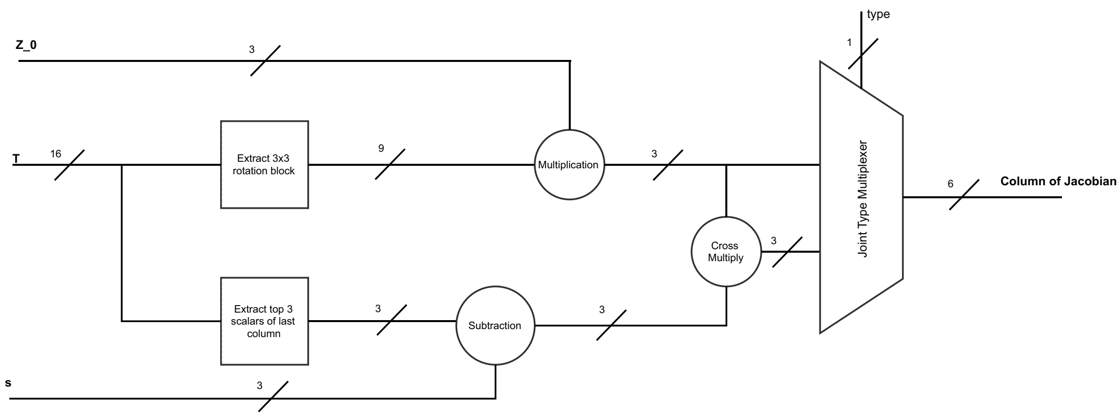


Figure 6: The dataflow diagram for a hardware block that calculates the ith column of the Jacobian matrix.

4.1.4 Damped Least Squares Block

The Jacobian matrix describes the velocity of the manipulator end point as a function of joint velocities. The inverse kinematics problem is solved if we can find the matrix inverse of the Jacobian matrix, which would describe the requisite joint velocities to obtain the desired velocity of the manipulator.

Finding the inverse Jacobian matrix is not possible in practice. A robot that has fewer than six degrees of freedom (six joints) would not have full control of translation and orientation of its hand, resulting in a non-square, and therefore non-invertible, Jacobian matrix. A robot that is at extreme points in its range of motion may also have a Jacobian matrix that does not have full row rank, and therefore have no inverse Jacobian matrix.

Instead, we will find the Jacobian matrix inverse in the least squared sense by solving the normalized matrix equation. We do this by multiplying both sides of the matrix equation with the Jacobian matrix transpose.

Furthermore, a square matrix may not be invertible when two or more rows cancel out, leading to a matrix that does not have full row rank. Running any matrix inversion algorithm on such a matrix would result in a divide by zero exception. We eliminate this possibility by adding a small bias constant along the diagonal of the Jacobian matrix, preventing the matrix from losing a row.

The pipeline for the damped least squares inverse kinematics algorithm is shown in Figure 7.

4.2 Custom Functional Units

In this section we describe the building blocks of the accelerator—our custom functional units—which we assemble to create the submodules described in previous sections. These custom functional units are assembled from IP designs generated by Altera MegaFunctions.

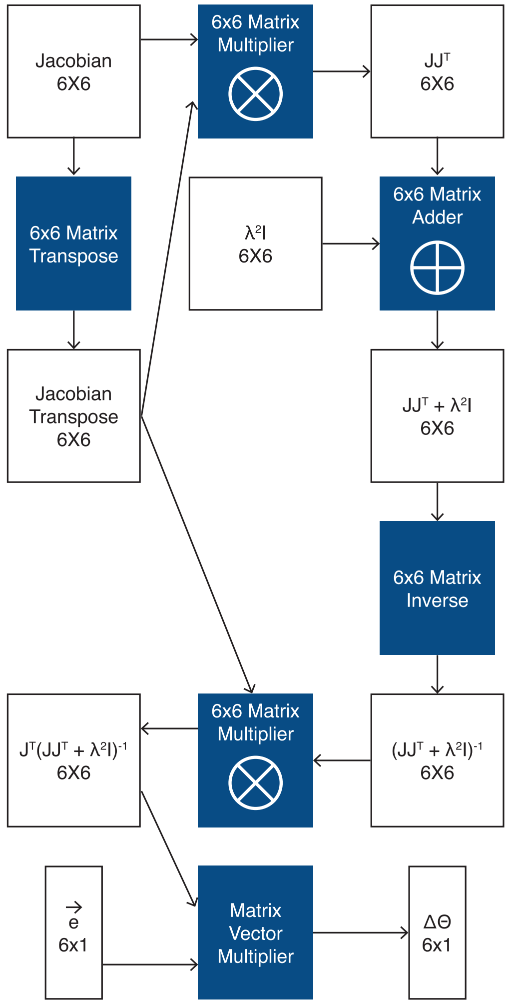


Figure 7: The pipeline for calculating joint movements using the damped least squares algorithm.

4.2.1 21-Bit Fixed Point Sine and Cosine

We use a quartic curve (a fourth degree polynomial of the input angle) to approximate the sine and cosine functions for input values of $-pi$ to pi. An example software code for such an approximation can be found at [6].

Within the sine cosine estimation, intermediate values do not get very large, so a narrow dynamic range for the variables is acceptable. On the other hand, we need high precision so minimal numerical errors are introduced into the rest of the algorithm. We decided on using a 21-bit estimator, with 16-bits of fractional values (right of the decimal point).

4.2.2 4x4 Matrix-Matrix Multiplication

The full transformation matrix pipeline needs to multiply D-H transformation blocks. Instead of instantiating a costly, dedicated 4x4 matrix multiplication functional unit, we use the 6x6 matrix multiplication functional unit needed for the damped least squared algorithm, which would otherwise be idle while the accelerator is finding the Jacobian matrix.

When using the 6x6 matrix multiplier for multiplying 4x4 matrices, the additional pair of rows and columns that pad the 4x4 matrices will be zero.

4.2.3 6x6 Matrix-Matrix Multiplication

Matrix-matrix multiplication is highly parallel—multiplying 6x6 matrices requires 6^3 multiplications that may occur in parallel. We cannot instantiate 216 multipliers on the FPGA, so instead we do 36 multiplies at once and accumulate the products for the result. These multipliers are pipelined so a new multiplication is always in flight.

4.2.4 27-Bit Fixed Point Multipliers

In all, we instantiated 59 multipliers in the design. Eight of them go to finding the second and fourth power of *theta* and *alpha* joint angles for both the sine and cosine pipelines. 36 of them are arranged for matrix multiplication, which can also be repurposed as parallel multipliers, useful for array-matrix multiplication and cross products. 15 of the multipliers are for low-latency parallel multiplications needed in matrix inversion.

We explored using higher bit precision multipliers so the accelerator could operate on a larger range of input values. However, the digital signal processors on the Cyclone V FPGA have a natural bitwidth of 27-bit. More precise multipliers would incur an immense area cost.

4.3 Timing Design

4.3.1 Submodule Timing Design

The FPGA has a limited number of DSPs which are used to implement multipliers, so we must time multiplex their use. We schedule the use of our functional units so that parts of the inverse kinematics algorithm can run in parallel, and so that functional units can be reused in different parts of the algorithm.

Figure 8 shows the timing design of the accelerator. The diagram lists the functional unit hardware resources along the vertical axis. These include the sine and cosine unit, array multiplier, matrix multiplier, array divider, and square root units. The clock cycles each of these resources are active are shown along the horizontal axis. The top half of the diagram shows the first 110 clock cycles of the algorithm, which is dedicated to finding the forward kinematics Jacobian matrix. The bottom half of the diagram is the second 140 clock cycles of the algorithm, which carries out the damped least squares algorithm.

While it may appear that there is minimal parallelism in the damped least squares algorithm, we point out that the matrix multiplier, divider, and adder units are all parallel, SIMD-style functional units. The matrix multiplier in particular does 36 parallel multiplications at once.

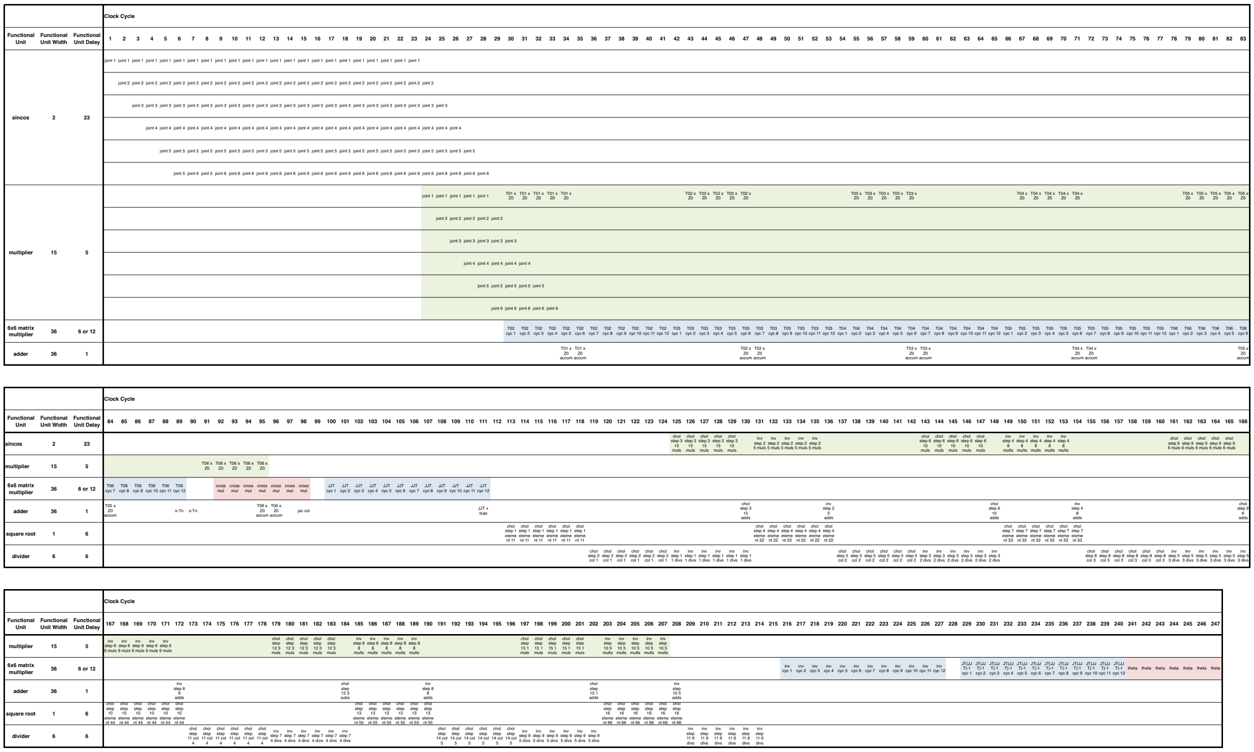


Figure 8: The timing design of the accelerator. The diagram lists the functional unit hardware resources along the vertical axis, and displays which clock cycles those resources are active along the horizontal axis. In each cell we provide a brief note on which part of the algorithm occurs in that cycle.

4.3.2 Control Signal Timing Design

Figure 9 shows the control and data signal timing for the accelerator. The host computer writes the target and initial joint angles to the accelerator, followed by the enable signal that launches calculation. Once updated joint angles are available, the accelerator asserts the done signal. The host computer can then read out the new angles and prepare for the next iteration.

Figure 9: The control and data signal timing for the accelerator.

Binaries produced by IKFast claim to finish in as fast as 4 microseconds on a fast OoO processor. Given the 50 MHz base clock on the FPGA, our algorithm would have to finish in 200 cycles to match software implementations.

Planners require being able to process thousands of configurations per second. The closed-form code generated by ikfast can produce solutions on the order of 4 microseconds! As a comparison, most numerical solutions are on the order of 10 milliseconds (assuming good convergence).

5 Project Plan

5.1 Milestones

March 27

1. Design block diagram of our system
2. Find C code that implements various inverse kinematics algorithms
3. Determine how to represent input and output with respect to the user (textual input, graphical output)

April 3

1. Write our own implementation of the damped least-squares algorithm in C
2. Design top-level module describing the interface between the hardware and software sections of our system
3. Design our joint peripheral device driver
4. Determine how best to decrease the number of DSP blocks our system uses in the FPGA

April 15

1. Associate the different blocks in the diagram of our system with corresponding sections of C code in our implementation
2. Construct timing diagrams for each of our submodules
3. Begin coding the submodules of our system in SystemVerilog

April 29

1. Full implementation of our system
2. Develop testbenches for the different modules in our system

May 13

1. Finish testing our system both in simulation and on the FPGA
2. Write up our final report and prepare our final presentation

5.2 Team Member Roles

Yipeng served the role of system integrator for the hardware. He created the sine and cosine hardware, along with the hardware for finding the forward kinematics matrix. He validated the hardware as a whole and its subcomponents individually against software models. Yipeng tuned the hardware timing and precision, in order to guarantee the solution converges while meeting FPGA area requirements.

Lianne created the skeleton of the C implementation of our algorithm used for understanding the algorithm and how components would fit together. She produced the template OpenGL file which was used to demonstrate that the IKSwift accelerator was working. Lianne also looked into matrix inverse algorithms and determined the best method for inverting matrices in our program. She implemented the two step process of inverting matrices in SystemVerilog so that it could be interfaced with the rest of the hardware files. After our hardware was built she worked with Qsys to develop the files required to interface the IKSwift accelerator with the device driver running on the ARM processor.

Richard designed the skeleton for the SystemVerilog module that calculates the Jacobian for a given configuration. Other than that, all of Richard's tangible contributions were on the software end of the project. He designed the device driver to communicate with the hardware and modified Lianne's openGL template to display an actual robot arm and interact with the device driver. In terms of deliverables, Richard was in charge of maintaining accurate milestones over the course of the semester and developed the pseudocode to describe the Jacobian and damped least-squares algorithms.

5.3 Lessons Learned

Yipeng Good software engineering practices are also good for hardware. We wrote models and tests first to convince ourself the algorithm would work, and then exhaustively show each module gives acceptably accurate results. The last bugs will be hiding in code the team hasn't tested.

Minimize the known unknowns so there's time left to deal with unknown unknowns. We aimed to make the design as clear as possible to the team, if not on paper too. This allowed us to estimate timing and area costs even before coding, which helped us minimize revisions.

Lianne The most important lesson I learned for programming embedded systems or any hardware design is that planning first is absolutely necessary for not just a good design but one that works.

By writing out detailed timing diagrams and just matching up what step of an algorithm has to happen when is crucial to identifying what resources can be shared and what can happen in parallel.

Richard My strongest suggestion would be to have weekly meetings that all members of your group attend. At these meetings we talked about any progress we made or obstacles we ran into over the past week. We would then discuss what the next logical step of our work would entail, and assign portions of this work to the different group members. Furthermore, we would make sure that our current project status correctly coincided with the milestones that should have been completed by that point. Luckily, we were always a bit ahead of the milestones we gave for ourselves, so that by the end of the semester we had some leeway and didn't feel stressed. This is not an endorsement for trivial milestones, but instead a suggestion that groups always try to stay a step ahead of their own expectations; it will make the project much easier in the end.

6 Appendix: Source Listings

6.1 Software

6.1.1 User Space Software

```
CC=g++
LFLAGS=-lGL -lGLU -lglut
CFLAGS=-c -Wall -Wextra -pg
LDFLAGS=-Wall -Wextra -pg
SOURCES=xml_help/pugixml.cpp lexer.cpp CConfigLoader.cpp robot_arm.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=robot_arm

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
  $(CC) $(LDFLAGS) $(OBJECTS) -o $@ $(LFLAGS)

.cpp.o:
  $(CC) $(CFLAGS) $< -o $@

clean:
  rm *.o  robot_arm
```

../../software/Makefile

```
<?xml version="1.0"?>
<Robot name="KUKA robot KR5" targetx="3" targety="-1.28" targetz="3.2">
  <Joints>
  <joint name="joint1" type="r" alphai="-90" ai=".75" di="3.35" theta="0" />
  <joint name="joint2" type="r" alphai="0" ai="2.70" di="0" theta="47" />
  <joint name="joint3" type="r" alphai="90" ai=".90" di="0" theta="0" />
  <joint name="joint4" type="r" alphai="-90" ai="0" di="-2.95" theta="45.2" />
  <joint name="joint5" type="r" alphai="90" ai="0" di="0" theta="0" />
  <joint name="joint6" type="r" alphai="180" ai="0" di="-.80" theta="-179" />
  </Joints>
</Robot>
```

../../software/robots/robot.xml

```
#ifndef __CCONFIGLOADER__
#define __CCONFIGLOADER__

#include "xml_help/pugixml.hpp"
#include "SharedTypes.h"

class CConfigLoader
{
    std::string fname;
    full_robot    for_load;
```

```
public:
    CConfigLoader(std::string & xml_name):fname(xml_name){}
    bool LoadXml();
    OUT full_robot & GetTable();
};

#endif
```

../../software/CConfigLoader.h

```
#include "CConfigLoader.h"
#include <iostream>
#include "lexer.h"


full_robot & CConfigLoader::GetTable()
{
    return for_load;
}

bool CConfigLoader::LoadXml()
{
    dh_parametrs temp_struct;
    int joint_num = 0;

    pugi::xml_document doc;

    doc.load_file(fname.c_str());

    for_load.targetx = doc.child("Robot").attribute("targetx").as_float();
    for_load.targety = doc.child("Robot").attribute("targety").as_float();
    for_load.targetz = doc.child("Robot").attribute("targetz").as_float();


    pugi::xml_node tools = doc.child("Robot").child("Joints");

    for (pugi::xml_node tool = tools.child("joint"); tool; tool = tool.next_sibling("joint")
    )
    {
        temp_struct.joint_name  = tool.attribute("name").value();
        temp_struct.alpha       = tool.attribute("alphai").as_float();
        temp_struct.a           = tool.attribute("ai").as_float();
        temp_struct.d = tool.attribute("di").as_float();
        temp_struct.theta = tool.attribute("theta").as_float();
        temp_struct.z_joint_type = (char *)tool.attribute("type").value();

        //Add dh params for this joint to our full robot struct
        for_load.params[joint_num++] = temp_struct;
    }

    return true;
}
```

../../software/CConfigLoader.cpp

```
#ifndef __SHAREDTYPES__
#define __SHAREDTYPES__


#include <stdlib.h>
#include <string>
#include "ik_driver.h"


#define IN
#define OUT
```

```
enum AxisT  {AxisX , AxisY , AxisZ};

struct dh_parametrs
{
    float a;             //Length of common normal
    float alpha;         //Angle between zi and zi-1 along xi
    float d;             //distance between xi and xi-1 along zi (variable for prismatic)
    float theta;         //Angle between xi and xi-1 along zi    (variavle for revolute)
    char* z_joint_type;//Joint type at z-1 axis
    std::string joint_name;
};

struct full_robot
{
  dh_parametrs params[MAX_JOINT]; //Robot is fully defined by a set of params for each joint
  float targetx, targety, targetz;//Target for end effector
};


#endif
```

../../software/SharedTypes.h

```
/*
 * This program displays the robot arm position and interacts
 * with the device driver to display the intermediate positions
 * over the course of our inverse kinematics algorithm
 *
 * Richard Townsend
 * Lianne Lairmore
 * Yipeng Huang
 *
 */
#include <cstdio>
#include <GL/glut.h>
#include <iostream>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "CConfigLoader.h"
#include "ik_driver.h"


float rotateZ[2] = {0, 0};
full_robot robot;
int count = 0;
int ik_driver_fd;
bool start_program;

/*
 * Convert a floating point number to our fixed-point representation
 */
int float_to_fixed(float num){
  float frac = num - (int)num;
  int decimal = ((int)num) << PRECISION; //Decimal part of number
  int fraction = (1 << PRECISION) * frac;
  //Check if we need to round up
  if (frac >= .5 && frac <= 1.0)
    fraction += 1;
  return decimal + fraction;
}
```

```
//Write the target for the end effector to the device
void write_target(float targetx, float targety, float targetz)
{
  ik_driver_arg_t vla;
  vla.joint = (char)-1;
  vla.target[0] = float_to_fixed(targetx);
  vla.target[1] = float_to_fixed(targety);
  vla.target[2] = float_to_fixed(targetz);
  if (ioctl(ik_driver_fd, IK_DRIVER_WRITE_PARAM, &vla)) {
    perror("ioctl(IK_DRIVER_WRITE_PARAM) failed");
    return;
  }
}

//Tell hardware we're ready for an iteration and wait until hardware is done
void notify_hardware(){
  ik_driver_arg_t vla;
  vla.start_signal = 1;
  vla.joint = (char)-2;//Signals that we're sending a start signal
  if (ioctl(ik_driver_fd, IK_DRIVER_WRITE_PARAM, &vla)) {
    perror("ioctl(IK_DRIVER_WRITE_PARAM) failed");
    return;
  }
  //Wait for hardware to be finished
  while (1){
    if (ioctl(ik_driver_fd, IK_DRIVER_READ_PARAM, &vla)) {
      perror("ioctl(IK_DRIVER_READ_PARAM) failed");
      return;
    }
    if (vla.done_signal == 1){
      vla.start_signal = 0;
      vla.joint = (char)-2;
      if (ioctl(ik_driver_fd, IK_DRIVER_WRITE_PARAM, &vla)) {
        perror("ioctl(IK_DRIVER_WRITE_PARAM) failed");
        return;
      }
      return;
    }
  }
}

//Write a THETA param for a specific joint to the device, converting from degrees to radians
void write_param(int joint, float magnitude)
{
  ik_driver_arg_t vla;
  vla.joint = (char)joint;

  //Do this error checking here so we don't use floats in the kernel
  if (magnitude < -180 || magnitude > 180){
    perror("Magnitude of parameter is outside acceptable range");
    exit(1);
  }
  else{
    magnitude = magnitude * (float)M_PI / 180.0;//Convert from degrees to radians
  }
  vla.magnitude = float_to_fixed(magnitude);
  if (ioctl(ik_driver_fd, IK_DRIVER_WRITE_PARAM, &vla)) {
    perror("ioctl(IK_DRIVER_WRITE_PARAM) failed");
    return;
  }
}

//Read a theta for a specific joint from the hardware
float read_param(int joint){
  ik_driver_arg_t vla;
  float value;
```

```
  //Get param
  vla.joint = (char)joint;
  if (ioctl(ik_driver_fd, IK_DRIVER_READ_PARAM, &vla)) {
    perror("ioctl(IK_DRIVER_READ_PARAM) failed");
    return 0;
  }

  //Convert from fixed to float
  value = vla.magnitude / pow(2,PRECISION);

  //Perform wraparound if necessary
  while (value > M_PI)
    value -= 2 * M_PI;
  while (value < -M_PI)
    value += 2 * M_PI;

  //convert from radians to degrees
  value = value * 180 / M_PI;

  write_param(joint, value);

  return value;
}

void arm(int index){

  float a,d,theta,alpha;

  //Draw a base for the whole arm so we can rotate our viewing of the arm
  glRotatef(rotateZ[index], 0, 0, 1);

  //Draw the target for the end effector
  glBegin(GL_LINES);
    glVertex3f(0,0,0);
    glVertex3f(robot.targetx, robot.targety, robot.targetz);
  glEnd();


  //Draw the links of the robot arm
  for (int i = 0; i < MAX_JOINT; i ++){
    glColor3f(.2*i+.5, .2*i+.1, .2*i);

    //These dh params won't change over the course of the algorithm
    d = robot.params[i].d;
    a = robot.params[i].a;
    alpha = robot.params[i].alpha;

    if (start_program){
      theta = robot.params[i].theta;
    }
    else{
       theta = read_param(i);
    }

    glRotatef(theta, 0, 0, 1);

    glBegin(GL_LINE_STRIP);
      glVertex3f(0, 0, 0);
      glVertex3f(a, 0, d);
    glEnd();

    glTranslatef(a, 0, d);
    glRotatef(alpha, 1, 0, 0);

  }
  glutPostRedisplay();
  if (start_program){
    start_program = false;
```

```
  }
  //Ready for next iteration
  usleep(5000);
  count++;
  notify_hardware();

  //Get new target after 80 cycles of algorithm
  if (count == 80){
    srand(time(NULL));
    robot.targetx = rand() % 7 - 4;
    robot.targety = rand() % 7 - 4;
    robot.targetz = rand() % 7 - 4;

    //Write target to hardware
    write_target(robot.targetx, robot.targety, robot.targetz);
    count = 0;
  }

}

void display(void) {
  // Before we do anything we need to clear the screen
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // Leave the projection matrix alone. In this function we are constructing M_{wc,vc}
  glMatrixMode(GL_MODELVIEW);

  // Start over with a new ModelView Matrix
  glLoadIdentity();

  // Perform the camera viewing transformation
  gluLookAt(0.0f, 0.0f, -10.0f,  0.0f, 0.0f, 0.0f,  0.0f, 1.0f, 0.0f);

  glPushMatrix();
  glRotatef(-90, 1, 0, 0);//Now z-axis points straight up, x points to left, and y is pointing into screen
  glScalef(.9, .9, .9);

  //Draw the arm
  arm(0);
  glPopMatrix();

  // Tell the API that we are done with this frame
  glFlush();
  glutSwapBuffers();
}

void init(void) {
  // We want a smooth interpolation of color between vertices
  glShadeModel(GL_SMOOTH);

  // Set the background color to clear to black
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

  // Initilize the viewing transformation; in this program we never change it
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(45, 1, 1, 100);

  // Enable z-buffering
  glEnable(GL_DEPTH_TEST);
}

void keyboard(unsigned char key, int x, int y) {
  switch(key) {

  case 'q':
  case 'Q':
```

```
    exit(0);
    break;
  case 'z':
    rotateZ[0]+=5;
    break;
  case 'Z':
    rotateZ[0]-=5;
    break;
  default:
    break;
  }

  // Force the screen to be redrawn with new parameters.
  // If this wasn't called, you might have to wait for the user
  // to cover and then uncover the window.
  glutPostRedisplay();
}

int main(int argc, char** argv) {

  static const char filename[] = "/dev/ik_driver";

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

  //Get dh params from xml file
  std::string str = "robots/robot.xml";
  CConfigLoader cfg(str);
  if(!cfg.LoadXml()) return 1;
  robot = cfg.GetTable();

  //Write random target to hardware
  srand(time(NULL));
  robot.targetx = rand() % 8 - 4;
  robot.targety = rand() % 8 - 4;
  robot.targetz = rand() % 8 - 4;

  write_target(robot.targetx, robot.targety, robot.targetz);

  //Inform hardware of initial theta configuration
  for (int i = 0; i < MAX_JOINT; i ++){
    write_param(i, robot.params[i].theta);
  }

  start_program = true;

  glutInit(&argc, argv);
  glutInitDisplayMode( GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(100, 100);

  // We need to store this in case we do something behind GLUI's back
  int GlutWindowID = glutCreateWindow("IKSwift Robot Simulator");

  // Call our init function
  init();

  // Register our callbacks with GLUT
  // Note that these are actually pointers to the functions declared above.
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);


  // Call GLUT's main loop, which never ends
  glutMainLoop();
```

```
  // Our program will never reach here
  return EXIT_SUCCESS;
}
```

../../software/robot˙arm.cpp

6.1.2 Kernel Space Software

```
#ifndef _IK_DRIVER_H
#define _IK_DRIVER_H

#include <linux/ioctl.h>

//PI
# define M_PI   3.14159265358979323846

//How many joints we have
#define MAX_JOINT 6

//Except for joint_type, all values will be stored
//in 32 bit (4 byte) registers
#define REG_SIZE 4

//Memory offset where we store the start_signal
#define START_OFFSET 4

//Memory offset where we start to store dh-params
#define PARAM_OFFSET 64

//All dh-params for a single joint take up 32 bytes
#define JOINT_OFFSET 32

//Our fractional precision in our fixed-point representation
#define PRECISION 16

//Max and min values for any coordinates in our system in fixed-point
#define MAX_COORD 262144
#define MIN_COORD -262144

/* DH Parameters */
#define THETA 0       //theta_i
#define L_OFFSET 1    //d_i
#define L_LENGTH 2    //a_i
#define ALPHA 3       //alpha_i
#define NUM_PARAMS 4

typedef struct {
  unsigned int start_signal; /* Tell hardware when we're ready for an iteration of the
    algorithm */
  unsigned int done_signal; /* Tell hardware when we're ready for an iteration of the
    algorithm */
  signed char joint; /* Indicate which joint we're getting/setting; -1 indicates that we're
    setting the target */
  unsigned char joint_type; /* The ith bit is 1 if ith joint is rotational; translational
    otherwise */
  signed int target[3]; /* (x,y,z) coordinates of target position */
  signed int magnitude;
} ik_driver_arg_t;

/* ioctls and their arguments */
#define IK_DRIVER_WRITE_PARAM _IOW('q', 1, ik_driver_arg_t *)
#define IK_DRIVER_READ_PARAM  _IOR('q', 2, ik_driver_arg_t *)

#endif
```

../../software/ik˙driver.h

```
/*
 * Device driver for the Inverse Kinematics Solver
 *
 * A Platform device implemented using the misc subsystem
 *
 * Richard Townsend
 * Yipeng Huang
 * Lianne Lairmore
 *
 * "make" to build
 * insmod ik_driver.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree ik_driver.c
 */


#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 "ik_driver.h"

#define DRIVER_NAME "ik_driver"

/*
 * Information about our device
 */
struct joint_dev {
  resource_size_t start; /* Address of start of registers */
  resource_size_t size;
  void __iomem *virtbase; /* Where registers can be accessed in memory */
} dev;




/*
 * Write target position of the end effector and the bit vector for the joint types
 * Assumes target position is in range and the device information has been set up
 */
static void write_target(u32 target[3])
{
  int i;
  u32 curtarget;

  for (i = 1; i < 4; i++){
    curtarget = target[i-1];
    //Write 4 MSB (need to multiply i by 2 to skip over first 64 bits of mem)
    iowrite32(curtarget, dev.virtbase+(i*2)*REG_SIZE);
  }
}

/*
```

```
 * Write parameter for a given joint
 * Assumes joint is in range and the device information has been set up
 */
static void write_parameter(u8 joint,  u32 magnitude){
  u32 mag = magnitude;
  iowrite32(mag, dev.virtbase+PARAM_OFFSET+(JOINT_OFFSET * joint));
}

//Inform hardware that it can do an iteration of the algorithm
static void write_start(u32 start){
  iowrite32(start, dev.virtbase+START_OFFSET);
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the coordinates.
 * Note extensive error checking of arguments
 */
static long ik_driver_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
  ik_driver_arg_t vla;

  switch (cmd) {
  case IK_DRIVER_WRITE_PARAM:
    if (copy_from_user(&vla, (ik_driver_arg_t *) arg,
           sizeof(ik_driver_arg_t)))
      return -EACCES;
    //Distributed checks over a bunch of if statements to avoid one huge conditional
    if (vla.joint < -3 || vla.joint > MAX_JOINT)
      return -EINVAL;
    if (vla.joint == -1 && ((vla.target[0] < MIN_COORD || vla.target[0] > MAX_COORD) ||
                           (vla.target[1] < MIN_COORD || vla.target[1] > MAX_COORD) ||
                           (vla.target[2] < MIN_COORD || vla.target[2] > MAX_COORD)))
      return -EINVAL;
    if (vla.joint == -2 && vla.start_signal != 1 && vla.start_signal != 0)
      return -EINVAL;
    if (vla.joint == -1)
      write_target(vla.target);
    else if (vla.joint == -2)
      write_start(vla.start_signal);
    else
      write_parameter(vla.joint,  vla.magnitude);
    break;

  case IK_DRIVER_READ_PARAM:

    if (copy_from_user(&vla, (ik_driver_arg_t *) arg,
           sizeof(ik_driver_arg_t)))
      return -EACCES;
    if (vla.joint < -3 || vla.joint > MAX_JOINT)
      return -EINVAL;
    vla.magnitude = ioread32(dev.virtbase + PARAM_OFFSET + (JOINT_OFFSET * vla.joint));
    vla.done_signal = ioread32(dev.virtbase+START_OFFSET);
    if (copy_to_user((ik_driver_arg_t *) arg, &vla,
         sizeof(ik_driver_arg_t)))
      return -EACCES;
    break;

  default:
    return -EINVAL;
  }

  return 0;
}

/* The operations our device knows how to do */
static const struct file_operations ik_driver_fops = {
  .owner     = THIS_MODULE,
```

```c
  .unlocked_ioctl = ik_driver_ioctl,
};

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

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init ik_driver_probe(struct platform_device *pdev)
{
  int ret;
  struct resource res;


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

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

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

  dev.start = res.start;
  dev.size = resource_size(&res);

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

  return 0;

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

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

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id ik_driver_of_match[] = {
```

```
  { .compatible = "altr,ik_driver" },
  {},
};
MODULE_DEVICE_TABLE(of, ik_driver_of_match);
#endif

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

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

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

module_init(ik_driver_init);
module_exit(ik_driver_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Richard Townsend, Yipeng Huang, Lianne Lairmore");
MODULE_DESCRIPTION("IK Swift Interface");
```

../../software/ik˙driver.c

```
#ifndef _ALTERA_HPS_0_H_
#define _ALTERA_HPS_0_H_

/*
 * This file was automatically generated by the swinfo2header utility.
 *
 * Created from SOPC Builder system 'lab3' in
 * file './lab3.sopcinfo'.
 */

/*
 * This file contains macros for module 'hps_0' and devices
 * connected to the following masters:
 *   h2f_axi_master
 *   h2f_lw_axi_master
 *
 * Do not include this header file and another header file created for a
 * different module or master group at the same time.
 * Doing so may result in duplicate macro names.
 * Instead, use the system header file which has macros with unique names.
 */

/*
 * Macros for device 'ik_driver_0', class 'ik_driver'
 * The macros are prefixed with 'IK_DRIVER_0_'.
 * The prefix is the slave descriptor.
 */
#define IK_DRIVER_0_COMPONENT_TYPE ik_driver
```

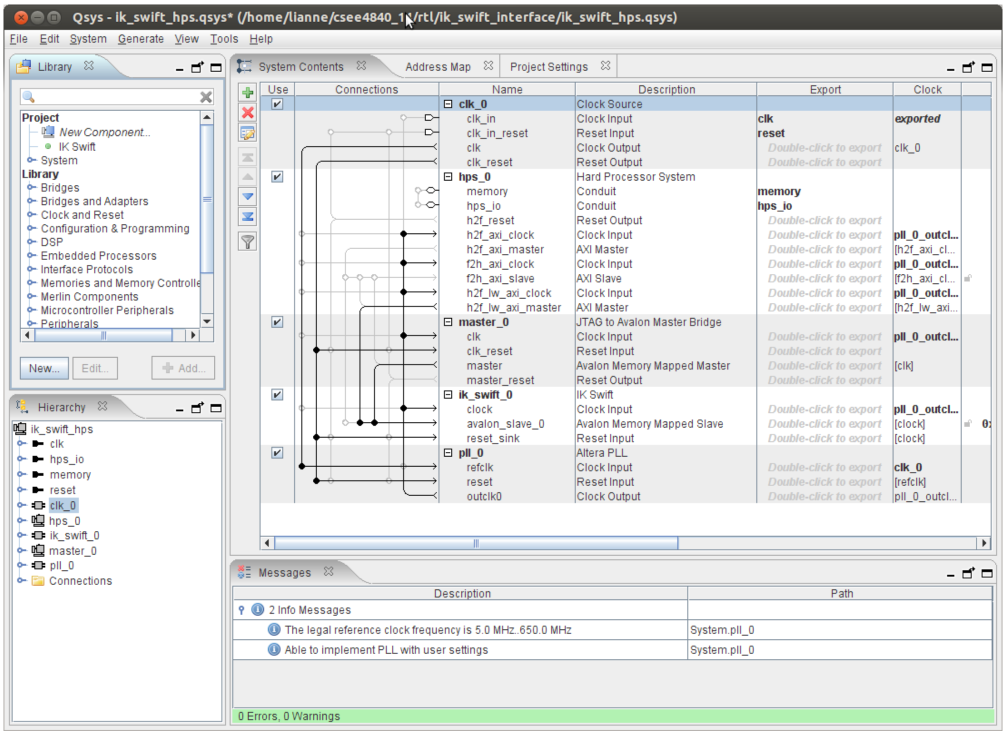
```
#define IK_DRIVER_0_COMPONENT_NAME ik_driver_0
#define IK_DRIVER_0_BASE 0x0
#define IK_DRIVER_0_SPAN 8
#define IK_DRIVER_0_END 0x7


#endif /* _ALTERA_HPS_0_H_ */
```

../../software/hps˙0.h

6.2 Hardware

6.2.1 Qsys Configuration



6.2.2 Top Level and Memory Map

```
/*
 * Memory interface for IKSwift
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

`timescale 1ns/1ps

`include "../ik_swift_32/ik_swift_interface.sv"
`include "../ik_swift_32/ik_swift.sv"

`include "../ik_swift_32/full_jacobian/full_jacobian_interface.sv"
`include "../ik_swift_32/full_jacobian/full_jacobian.sv"
`include "../ik_swift_32/full_jacobian/jacobian/jacobian_interface.sv"
`include "../ik_swift_32/full_jacobian/jacobian/jacobian.sv"
`include "../ik_swift_32/full_jacobian/full_mat/full_mat_interface.sv"
`include "../ik_swift_32/full_jacobian/full_mat/full_mat.sv"
```

```
`include "../ik_swift_32/full_jacobian/full_mat/t_block/t_block_interface.sv"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/t_block.sv"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/sincos_interface.sv"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/sincos.sv"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/sin.sv"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/cos.sv"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21_coeff_26561/mult_21_coeff_26561.v"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21_coeff_83443/mult_21_coeff_83443.v"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21_coeff_14746/mult_21_coeff_14746.v"
`include "../ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21/mult_21.v"

`include "../ik_swift_32/inverse/inverse_interface.sv"
`include "../ik_swift_32/inverse/inverse.sv"
`include "../ik_swift_32/inverse/cholesky_block/cholesky_block_interface.sv"
`include "../ik_swift_32/inverse/cholesky_block/cholesky_block.sv"
`include "../ik_swift_32/inverse/cholesky_block/sqrt_43/sqrt_43_interface.sv"
`include "../ik_swift_32/inverse/cholesky_block/sqrt_43/sqrt_43.v"
`include "../ik_swift_32/inverse/lt_block/lt_block_interface.sv"
`include "../ik_swift_32/inverse/lt_block/lt_block.sv"
`include "../ik_swift_32/inverse/array_div/array_div_interface.sv"
`include "../ik_swift_32/inverse/array_div/array_div.sv"
`include "../ik_swift_32/inverse/array_div/div_43/div_43.v"

`include "../ik_swift_32/mat_mult/mat_mult_interface.sv"
`include "../ik_swift_32/mat_mult/mat_mult.sv"
`include "../ik_swift_32/mat_mult/mult_array.sv"
`include "../ik_swift_32/array_mult/array_mult_interface.sv"
`include "../ik_swift_32/array_mult/array_mult.sv"
`include "../ik_swift_32/mult_27/mult_27.v"

// `include "../ik_swift_32/sim_models/lpm_mult.v"
// `include "../ik_swift_32/sim_models/mult_block.v"
// `include "../ik_swift_32/sim_models/addsub_block.v"
// `include "../ik_swift_32/sim_models/pipeline_internal_fv.v"
// `include "../ik_swift_32/sim_models/dffep.v"
// `include "../ik_swift_32/sim_models/altera_mf.v"
// `include "../ik_swift_32/sim_models/220model.v"

parameter MAX_JOINT = 6;

module ik_swift_interface (
  input logic clk,
  input logic reset,

  // inputs
  input logic chipselect,
  input logic write,
  input logic [5:0] address,
  input logic [31:0] writedata,

  // outputs
  output logic [31:0] readdata
);

  // REGISTERS
  logic [2:0] [26:0] target; // (x,y,z) coordinates and orientation of target position

  // INSTANTIATE IK_FAST TOP MODULE
  ifc_ik_swift ifc_ik_swift (clk);
  // INPUTS
  assign ifc_ik_swift.rst = reset;
  // base joint's axis of rotation/translation
  assign ifc_ik_swift.z = { 18'd65536, 18'd0, 18'd0 }; // unit vector in z direction
  // bit vector describing type of each joint
  assign ifc_ik_swift.joint_type = 6'b111111;
```

```
  // target coordinates
  assign ifc_ik_swift.target = {{81'b0}, target};
  ik_swift ik_swift (ifc_ik_swift.ik_swift);

  always_ff @(posedge clk) begin
    if (reset) begin
      target <= {3{27'b0}};
      ifc_ik_swift.en <= 1'b0;
      ifc_ik_swift.dh_dyn_in <= {6{21'b0}};
    end else if ( chipselect && write ) begin
      case (address)

        // 6'd00 : joint_type <= writedata[5:0]; // joint type vector
        6'd01 : ifc_ik_swift.en <= writedata[0]; // start signal

        6'd02 : target[0] <= writedata[26:0]; // target[0] x
        6'd04 : target[1] <= writedata[26:0]; // target[1] y
        6'd06 : target[2] <= writedata[26:0]; // target[2] z

        6'd16 : ifc_ik_swift.dh_dyn_in[0]/*[THETA]*/ <= writedata[20:0];
        6'd24 : ifc_ik_swift.dh_dyn_in[1]/*[THETA]*/ <= writedata[20:0];
        6'd32 : ifc_ik_swift.dh_dyn_in[2]/*[THETA]*/ <= writedata[20:0];
        6'd40 : ifc_ik_swift.dh_dyn_in[3]/*[THETA]*/ <= writedata[20:0];
        6'd48 : ifc_ik_swift.dh_dyn_in[4]/*[THETA]*/ <= writedata[20:0];
        6'd56 : ifc_ik_swift.dh_dyn_in[5]/*[THETA]*/ <= writedata[20:0];

      endcase
    end
  end

  // OUTPUTS
  // deltas for joint parameters
  always_ff @(posedge clk) begin
    if (reset) begin
      readdata <= {32'b0};
    end else if ( chipselect ) begin
      case (address)

        6'd00 : readdata <= {26'b0, ifc_ik_swift.joint_type};
        6'd01 : readdata <= {31'b0, ifc_ik_swift.done};

        6'd02 : readdata <= {{5{target[0][26]}}, target[0]};
        6'd04 : readdata <= {{5{target[1][26]}}, target[1]};
        6'd06 : readdata <= {{5{target[2][26]}}, target[2]};

        6'd16 : readdata <= {{11{ifc_ik_swift.dh_dyn_out[0][20]}}, ifc_ik_swift.dh_dyn_out
    [0]/*[THETA]*/};
        6'd24 : readdata <= {{11{ifc_ik_swift.dh_dyn_out[1][20]}}, ifc_ik_swift.dh_dyn_out
    [1]/*[THETA]*/};
        6'd32 : readdata <= {{11{ifc_ik_swift.dh_dyn_out[2][20]}}, ifc_ik_swift.dh_dyn_out
    [2]/*[THETA]*/};
        6'd40 : readdata <= {{11{ifc_ik_swift.dh_dyn_out[3][20]}}, ifc_ik_swift.dh_dyn_out
    [3]/*[THETA]*/};
        6'd48 : readdata <= {{11{ifc_ik_swift.dh_dyn_out[4][20]}}, ifc_ik_swift.dh_dyn_out
    [4]/*[THETA]*/};
        6'd56 : readdata <= {{11{ifc_ik_swift.dh_dyn_out[5][20]}}, ifc_ik_swift.dh_dyn_out
    [5]/*[THETA]*/};

      endcase
    end
  end

endmodule
```

../../rtl/ik_swift_interface/ik_swift_interface.sv

```
/*
```

```
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_ik_swift (
  input logic clk
);

logic en, done, rst;

// INPUTS
// base joint's axis of rotation/translation
logic [2:0] [17:0] z;
// bit vector describing type of each joint
logic [5:0] joint_type;
// dh joint parameters
logic [5:0] [20:0] dh_dyn_in;
// target coordinates
logic [5:0] [26:0] target;

// TEST OUTPUTS
// jacobian
logic [5:0] [5:0] [26:0] jacobian_matrix;
// jacobian * jacobian transpose + bias
logic [5:0] [5:0] [26:0] jjt_bias;
// LT decomposition of given matrix
logic [5:0] [5:0] [26:0] lt;
// inverse of LT matrix
logic [5:0] [5:0] [26:0] lt_inverse;
// inverse of given matrix
logic [5:0] [5:0] [26:0] inverse;
// damped least squares matrix
logic [5:0] [5:0] [26:0] dls;
// deltas for joint parameters
logic [5:0] [26:0] delta;

// OUTPUTS
// dh joint parameters
logic [5:0] [20:0] dh_dyn_out;

// clocking cb @(posedge clk);
//  output en;
//  output rst;
//  output z;
//  output joint_type;
//  output dh_dyn_in;
//  output target;
//
//  input jacobian_matrix;
//  input jjt_bias;
//  input lt;
//  input lt_inverse;
//  input inverse;
//  input dls;
//  input delta;
//  input done;
//  input dh_dyn_out;
// endclocking
//
// modport ik_swift_tb (clocking cb);

// restrict directions
modport ik_swift (
  input clk, en, rst,

  input z,
  input joint_type,
  input target,
```

```
  input dh_dyn_in,

  output jacobian_matrix,
  output jjt_bias,
  output lt,
  output lt_inverse,
  output inverse,
  output dls,
  output delta,
  output done,
  output dh_dyn_out
);

endinterface
```

../../rtl/ik_swift_32/ik_swift_interface.sv

```
// the timescale directive tells the compiler the clock period and the
// precision that needs to be displayed in the VCD dump file

`timescale 1ns/1ps

parameter THETA = 0;
parameter A_PARAM = 1;
parameter D_PARAM = 2;
parameter ALPHA = 3;

module ik_swift (
  ifc_ik_swift.ik_swift i
);

  // LOGIC GOVERNING COUNT
  logic [7:0] count;
  parameter MAX = 250;
  always_ff @(posedge i.clk) begin
    if ( i.rst ) begin
      count <= 8'b0;
    end else if ( i.en && !i.done ) begin
      if ( count==MAX-1'b1 ) begin
        count <= 8'b0;
      end else begin
        count <= count + 1'b1;
      end
    end
  end

  // LOGIC GOVERNING DONE
  always_ff @(posedge i.clk) begin
    if ( i.rst || !i.en ) begin // at begin, clear done
      i.done <= 1'b0;
    end else if ( i.en && count==8'd248 ) begin
      i.done <= 1'b1;
    end
  end

  // INSTANTIATE FULL JACOBIAN BLOCK
  ifc_full_jacobian i_jac (i.clk);
  // inputs
  assign i_jac.en = i.en && !i.done;
  assign i_jac.rst = i.rst;
  assign i_jac.count = count;
  assign i_jac.z = i.z;
  assign i_jac.joint_type = i.joint_type;
  // GENERATE DH_PARAMS
  assign i_jac.dh_param = {
    {
      21'd205887,   // trans.dh_data[5][ALPHA] = 3.14159265359;
```

```
    -21'd52429,   // trans.dh_data[5][D_PARAM] = -0.8;
    21'd0,       // trans.dh_data[5][A_PARAM] = 0.0;
    i.dh_dyn_in[5]  // trans.dh_data[5][THETA] = 0.0;
  },
  {
    21'd102944,   // trans.dh_data[4][ALPHA] = 3.14159265359/2;
    21'd0,       // trans.dh_data[4][D_PARAM] = 0.0;
    21'd0,       // trans.dh_data[4][A_PARAM] = 0.0;
    i.dh_dyn_in[4]  // trans.dh_data[4][THETA] = 0.0;
  },
  {
    -21'd102944,  // trans.dh_data[3][ALPHA] = -3.14159265359/2;
    -21'd193331,  // trans.dh_data[3][D_PARAM] = -2.95;
    21'd0,       // trans.dh_data[3][A_PARAM] = 0.0;
    i.dh_dyn_in[3]  // trans.dh_data[3][THETA] = 0.0;
  },
  {
    21'd102944,   // trans.dh_data[2][ALPHA] = 3.14159265359/2;
    21'd0,       // trans.dh_data[2][D_PARAM] = 0.0;
    21'd58982,    // trans.dh_data[2][A_PARAM] = 0.9;
    i.dh_dyn_in[2]  // trans.dh_data[2][THETA] = 0.0;
  },
  {
    21'd0,       // trans.dh_data[1][ALPHA] = 0.0;
    21'd0,       // trans.dh_data[1][D_PARAM] = 0.0;
    21'd176947,   // trans.dh_data[1][A_PARAM] = 2.7;
    i.dh_dyn_in[1]  // trans.dh_data[1][THETA] = 0.0;
  },
  {
    -21'd102944,  // trans.dh_data[0][ALPHA] = -3.14159265359/2;
    21'd219546,   // trans.dh_data[0][D_PARAM] = 3.35;
    21'd49152,    // trans.dh_data[0][A_PARAM] = 0.75;
    i.dh_dyn_in[0]  // trans.dh_data[0][THETA] = 0.0;
  }
};
full_jacobian full_jacobian (i_jac.full_jacobian);
// outputs
assign i.jacobian_matrix = i_jac.jacobian_matrix;
assign i.jjt_bias = i_jac.jjt_bias;

// INSTANTIATE FULL INVERSE BLOCK
ifc_inverse ifc_inverse (i.clk);
// inputs
assign ifc_inverse.en = i.en && !i.done;
assign ifc_inverse.rst = i.rst;
assign ifc_inverse.count = count;
assign ifc_inverse.matrix = i_jac.jjt_bias;
inverse inverse (ifc_inverse.inverse_dut);
// ouptuts
assign i.lt = ifc_inverse.lt;
assign i.lt_inverse = ifc_inverse.lt_inverse;
assign i.inverse = ifc_inverse.inverse;

// MATRIX MULTIPLY FOR JT * INVERSE
logic [5:0] [5:0] [26:0] dls_mat_mult_dataa;
logic [5:0] [5:0] [26:0] dls_mat_mult_datab;

// shared multipliers
// INSTANTIATE MAT MULT
ifc_mat_mult ifc_mat_mult (i.clk);
assign ifc_mat_mult.en = i.en && !i.done;
// delay rst for mat_mult by four
always_ff @(posedge i.clk)
  if (i.en && !i.done)
    ifc_mat_mult.rst <= count==8'd28 || count==8'd98 || count==8'd214 || count==8'd227;

// two periods mat_mult in parallel mode
```

```
assign ifc_mat_mult.mat_mode = (8'd91<=count&&count<8'd99)||(8'd240<=count&&count<8'd248)
  ? 1'b0 : 1'b1;
// Output to matrix multipliers
assign ifc_mat_mult.dataa = i_jac.mat_mult_dataa | ifc_inverse.mat_mult_dataa |
  dls_mat_mult_dataa;
assign ifc_mat_mult.datab = i_jac.mat_mult_datab | ifc_inverse.mat_mult_datab |
  dls_mat_mult_datab;
mat_mult mat_mult (ifc_mat_mult.mat_mult);
assign i_jac.mat_mult_result = ifc_mat_mult.result;
assign ifc_inverse.mat_mult_result = ifc_mat_mult.result;

// INSTANTIATE ARRAY MULT
ifc_array_mult ifc_array_mult (i.clk);
assign ifc_array_mult.en = i.en && !i.done;
assign ifc_array_mult.rst = i.rst;
// Output to array multipliers
assign ifc_array_mult.dataa = { {6{27'b0}}, i_jac.array_mult_dataa } | ifc_inverse.
  array_mult_dataa;
assign ifc_array_mult.datab = { {6{27'b0}}, i_jac.array_mult_datab } | ifc_inverse.
  array_mult_datab;
assign i_jac.array_mult_result = ifc_array_mult.result[8:0];
assign ifc_inverse.array_mult_result = ifc_array_mult.result;
array_mult array_mult (ifc_array_mult.array_mult);

// MATRIX MULTIPLY FOR JT * INVERSE
// MAT_MULT INPUTS
always_ff @(posedge i.clk)
  if (i.en && !i.done)
    case (count)
      8'd0: begin
        dls_mat_mult_dataa <= {36{27'b0}};
        dls_mat_mult_datab <= {36{27'b0}};
      end
      8'd228: begin
        dls_mat_mult_dataa <= {
          { i_jac.jacobian_matrix[5][5], i_jac.jacobian_matrix[4][5], i_jac.
  jacobian_matrix[3][5], i_jac.jacobian_matrix[2][5], i_jac.jacobian_matrix[1][5], i_jac.
  jacobian_matrix[0][5] },
          { i_jac.jacobian_matrix[5][4], i_jac.jacobian_matrix[4][4], i_jac.
  jacobian_matrix[3][4], i_jac.jacobian_matrix[2][4], i_jac.jacobian_matrix[1][4], i_jac.
  jacobian_matrix[0][4] },
          { i_jac.jacobian_matrix[5][3], i_jac.jacobian_matrix[4][3], i_jac.
  jacobian_matrix[3][3], i_jac.jacobian_matrix[2][3], i_jac.jacobian_matrix[1][3], i_jac.
  jacobian_matrix[0][3] },
          { i_jac.jacobian_matrix[5][2], i_jac.jacobian_matrix[4][2], i_jac.
  jacobian_matrix[3][2], i_jac.jacobian_matrix[2][2], i_jac.jacobian_matrix[1][2], i_jac.
  jacobian_matrix[0][2] },
          { i_jac.jacobian_matrix[5][1], i_jac.jacobian_matrix[4][1], i_jac.
  jacobian_matrix[3][1], i_jac.jacobian_matrix[2][1], i_jac.jacobian_matrix[1][1], i_jac.
  jacobian_matrix[0][1] },
          { i_jac.jacobian_matrix[5][0], i_jac.jacobian_matrix[4][0], i_jac.
  jacobian_matrix[3][0], i_jac.jacobian_matrix[2][0], i_jac.jacobian_matrix[1][0], i_jac.
  jacobian_matrix[0][0] }
        };
        dls_mat_mult_datab <= ifc_inverse.inverse;
      end
      8'd240: begin
        // dls matrix * error vector
        dls_mat_mult_dataa <= ifc_mat_mult.result; // DLS matrix
        dls_mat_mult_datab <= {6{
          // axis of rotation / translation for joints 1...6
          i.target[5] - i_jac.axis[6][2], // k unit vector
          i.target[4] - i_jac.axis[6][1], // j unit vector
          i.target[3] - i_jac.axis[6][0], // i unit vector
          // multiplied results of transformation matrices
          i.target[2] - i_jac.full_matrix[5][2][3], // z coordinate
          i.target[1] - i_jac.full_matrix[5][1][3], // y coordinate
          i.target[0] - i_jac.full_matrix[5][0][3] // x coordinate
```

```
          }};
        end
        8'd247: begin
          dls_mat_mult_dataa <= {36{27'b0}};
          dls_mat_mult_datab <= {36{27'b0}};
        end
        default: begin
          dls_mat_mult_dataa <= dls_mat_mult_dataa;
          dls_mat_mult_datab <= dls_mat_mult_datab;
        end
      endcase

  // MAT_MULT OUTPUTS
  always_ff @(posedge i.clk)
    if (i.en && !i.done)
      case (count)
        8'd240: i.dls <= ifc_mat_mult.result;
        8'd247: i.delta <= {
          ifc_mat_mult.result[5][5] + ifc_mat_mult.result[5][4] + ifc_mat_mult.result[5][3]
    + ifc_mat_mult.result[5][2] + ifc_mat_mult.result[5][1] + ifc_mat_mult.result[5][0],
          ifc_mat_mult.result[4][5] + ifc_mat_mult.result[4][4] + ifc_mat_mult.result[4][3]
    + ifc_mat_mult.result[4][2] + ifc_mat_mult.result[4][1] + ifc_mat_mult.result[4][0],
          ifc_mat_mult.result[3][5] + ifc_mat_mult.result[3][4] + ifc_mat_mult.result[3][3]
    + ifc_mat_mult.result[3][2] + ifc_mat_mult.result[3][1] + ifc_mat_mult.result[3][0],
          ifc_mat_mult.result[2][5] + ifc_mat_mult.result[2][4] + ifc_mat_mult.result[2][3]
    + ifc_mat_mult.result[2][2] + ifc_mat_mult.result[2][1] + ifc_mat_mult.result[2][0],
          ifc_mat_mult.result[1][5] + ifc_mat_mult.result[1][4] + ifc_mat_mult.result[1][3]
    + ifc_mat_mult.result[1][2] + ifc_mat_mult.result[1][1] + ifc_mat_mult.result[1][0],
          ifc_mat_mult.result[0][5] + ifc_mat_mult.result[0][4] + ifc_mat_mult.result[0][3]
    + ifc_mat_mult.result[0][2] + ifc_mat_mult.result[0][1] + ifc_mat_mult.result[0][0]
        };
      endcase

  // ADD BACK TO DH PARAMS
  genvar joint;
  generate
    for ( joint=0 ; joint<6 ; joint++ ) begin: add_dh_param
      always_ff @(posedge i.clk) begin
        if (i.en && !i.done) begin
          case (count)
            8'd0: i.dh_dyn_out[joint] <= i.dh_dyn_in[joint];
            8'd248: begin
              case (i.joint_type[joint])
                1'b0: begin // translational
                  i.dh_dyn_out[joint]/*[D_PARAM]*/ <= i.dh_dyn_in[joint]/*[D_PARAM]*/ + i.
    delta[joint][20:0];
                end
                1'b1: begin // rotational
                  i.dh_dyn_out[joint]/*[THETA]*/ <= i.dh_dyn_in[joint]/*[THETA]*/ + i.delta[
    joint][20:0];
                end
              endcase
            end
          endcase
        end
      end
    end
  endgenerate

endmodule
```

../../rtl/ik_swift_32/ik_swift.sv

6.2.3 Top Level Randomized Validation Harness

```
`timescale 1ns/1ps
`include "ik_swift_test.sv"

class ik_swift_transaction;

  rand logic [3][30:0] z_increment;
  real z_fraction [3];
  real z_data [3];

  logic [5:0] joint_type = 6'b111111;

  rand logic [6][30:0] target_increment;
  real target_fraction [6];
  real target_data [6];

  rand logic [6]/*[4]*/[30:0] dh_increment;
  real dh_fraction [6]/*[4]*/;
  real dh_data [6][4];

endclass

class ik_swift_env;
  int robots = 1000;
  int convergence = 100;
endclass

program ik_swift_tb (ifc_ik_swift.ik_swift_tb ds);

  ik_swift_transaction trans;
  ik_swift_env env;
  ik_swift_test test;

  initial begin
    trans = new();
    test = new();
    env = new();

    @(ds.cb);
    ds.cb.rst <= 1'b1;
    @(ds.cb);
    ds.cb.rst <= 1'b0;
    ds.cb.en <= 1'b0;

    repeat (env.robots) begin

      trans.randomize();

      // GENERATE JOINT TYPE
      $display("joint type = %b", trans.joint_type);
      ds.cb.joint_type <= trans.joint_type;

      // RANDOMIZE Z BASIS VECTOR
      trans.z_data[0] = 0.0;
      trans.z_data[1] = 0.0;
      trans.z_data[2] = 1.0;
      for ( int z=0 ; z<3 ; z++ ) begin // z index
        // trans.z_fraction[z] = real'(trans.z_increment[z]) / 2147483648.0;
        // trans.z_data[z] = -4.0 + trans.z_fraction[z] * 2 * 4.0;
        $display("z = %d", z);
        $display("data = %f", trans.z_data[z]);
        ds.cb.z[z] <= int'(trans.z_data[z] * 65536.0);
      end

      // RANDOMIZE TARGET COORDINATE VECTOR
      // <Robot name="KUKA robot KR5" targetx="3" targety="-1.28" targetz="3.2">
      for ( int index=0 ; index<6 ; index++ ) begin // index
        trans.target_fraction[index] = real'(trans.target_increment[index]) / 2147483648.0;
```

```
      trans.target_data[index] = -3.0 + trans.target_fraction[index] * 2 * 3.0;
      trans.target_data[3] = 0.0;
      trans.target_data[4] = 0.0;
      trans.target_data[5] = 0.0;
      $display("target coordinate = %d", index);
      $display("data = %f", trans.target_data[index]);
      ds.cb.target[index] <= int'(trans.target_data[index] * 65536.0);
    end

    // RANDOMIZE DH_PARAMS
    // GENERATE DH_PARAMS
    trans.dh_data[0][THETA] = 0.0;
    trans.dh_data[0][A_PARAM] = 0.75;
    trans.dh_data[0][D_PARAM] = 3.35;
    trans.dh_data[0][ALPHA] = -3.14159265359/2;

    trans.dh_data[1][THETA] = 0.0;
    trans.dh_data[1][A_PARAM] = 2.7;
    trans.dh_data[1][D_PARAM] = 0.0;
    trans.dh_data[1][ALPHA] = 0.0;

    trans.dh_data[2][THETA] = 0.0;
    trans.dh_data[2][A_PARAM] = 0.9;
    trans.dh_data[2][D_PARAM] = 0.0;
    trans.dh_data[2][ALPHA] = 3.14159265359/2;

    trans.dh_data[3][THETA] = 0.0;
    trans.dh_data[3][A_PARAM] = 0.0;
    trans.dh_data[3][D_PARAM] = -2.95;
    trans.dh_data[3][ALPHA] = -3.14159265359/2;

    trans.dh_data[4][THETA] = 0.0;
    trans.dh_data[4][A_PARAM] = 0.0;
    trans.dh_data[4][D_PARAM] = 0.0;
    trans.dh_data[4][ALPHA] = 3.14159265359/2;

    trans.dh_data[5][THETA] = 0.0;
    trans.dh_data[5][A_PARAM] = 0.0;
    trans.dh_data[5][D_PARAM] = -0.8;
    trans.dh_data[5][ALPHA] = 3.14159265359;
    for ( int joint=0 ; joint<6 ; joint++ ) begin // joint index
      // for ( int param=0 ; param<4 ; param++ ) begin // dh param
      //  trans.dh_fraction[joint][param] = real'(trans.dh_increment[joint][param]) / 2147483648.0;
      // end
      // trans.dh_data[joint][THETA] = -3.1415926535897932384626433832795028841971 + trans.dh_fraction[joint][THETA] * 2 * 3.1415926535897932384626433832795028841971;
      // trans.dh_data[joint][A_PARAM] = -4.0 + trans.dh_fraction[joint][A_PARAM] * 2 * 4.0;
      // trans.dh_data[joint][D_PARAM] = -4.0 + trans.dh_fraction[joint][D_PARAM] * 2 * 4.0;
      // trans.dh_data[joint][ALPHA] = -3.1415926535897932384626433832795028841971 + trans.dh_fraction[joint][ALPHA] * 2 * 3.1415926535897932384626433832795028841971;
      $display("joint index = %d", joint);
      $display("theta = %f", trans.dh_data[joint][THETA]);
      $display("a = %f", trans.dh_data[joint][A_PARAM]);
      $display("d = %f", trans.dh_data[joint][D_PARAM]);
      $display("alpha = %f", trans.dh_data[joint][ALPHA]);
      // for ( int param=0 ; param<4 ; param++ ) begin // dh param
      ds.cb.dh_dyn_in[joint]/*[param]*/ <= int'(trans.dh_data[joint][THETA] * 65536.0);
      // end
    end

    // CONVERGENCE TESTING
    repeat (env.convergence) begin

      ds.cb.en <= 1'b1;
      @(ds.cb);
```

```
          test.update_ik_swift (
            trans.z_data,
            trans.joint_type,
            trans.dh_data,
            trans.target_data
          );

          repeat (360) @(ds.cb);

          test.check_ik_swift (
            ds.cb.jacobian_matrix,
            ds.cb.jjt_bias,
            ds.cb.lt,
            ds.cb.lt_inverse,
            ds.cb.inverse,
            ds.cb.dls,
            ds.cb.delta,
            ds.cb.done,
            ds.cb.dh_dyn_out
          );

          ds.cb.en <= 1'b0;
          repeat (120) @(ds.cb);

          for ( int joint=0 ; joint<6 ; joint++ ) begin // joint index
            // for ( int param=0 ; param<4 ; param++ ) begin // dh param
            trans.dh_data[joint][THETA] = real'(int'({{11{ds.cb.dh_dyn_out[joint]/*[param]*/
    [20]}}, ds.cb.dh_dyn_out[joint]/*[param]*/}))/65536.0;
            // if (param==ALPHA || param==THETA) begin
            while (trans.dh_data[joint][THETA]>3.141592653589793238462643383279502884197)
              trans.dh_data[joint][THETA] = trans.dh_data[joint][THETA] -
    2*3.141592653589793238462643383279502884197;
            while (trans.dh_data[joint][THETA]<-3.141592653589793238462643383279502884197)
              trans.dh_data[joint][THETA] = trans.dh_data[joint][THETA] +
    2*3.141592653589793238462643383279502884197;
            // end
            ds.cb.dh_dyn_in[joint]/*[param]*/ <= int'(trans.dh_data[joint][THETA] * 65536.0)
    ;
            // end
          end

        // end // end one solution cycle
      end // end convergence loop
    end // end randomized robots loop
  end // end initial
endprogram
```

../../rtl/ik_swift_32/bench.sv

```
// golden model class
class ik_swift_test;

  int n = 6;

  real m_jacobian [6][6];
  real m_jjt_bias [6][6];
  real m_lt [6][6];
  real m_lt_inv [6][6];
  real m_inverse [6][6];
  real m_dls [6][6];
  real m_delta [6];
  real m_dh_param [6][4];

  function real abs (real num);
    abs = (num<0) ? -num : num;
  endfunction
```

```
function void update_ik_swift (
  real z [3],
  logic [5:0] joint_type,
  real dh_param [6][4],
  real target [6]
);

  real full_matrix [6][4][4];
  real rotation [6][3][3];
  real axis [7][3];
  real position [7][3];
  real dist_to_end [6][3];
  real error [6];

  // COPY DH PARAMS
  // we add to these numbers later
  for ( int joint=0 ; joint<6 ; joint++ )
    for ( int param=0 ; param<4 ; param++ )
      m_dh_param[joint][param] = dh_param[joint][param];

  // GENERATE FULL MATRIX
  // iterate over joint index
  for ( int joint=0 ; joint<n ; joint++ ) begin
    real t_matrix [4][4];
    // generate local transformation matrix
    t_matrix[0][0] =  $cos(dh_param[joint][THETA]);
    t_matrix[0][1] = -$sin(dh_param[joint][THETA]) * $cos(dh_param[joint][ALPHA]);
    t_matrix[0][2] =  $sin(dh_param[joint][THETA]) * $sin(dh_param[joint][ALPHA]);
    t_matrix[0][3] =  $cos(dh_param[joint][THETA]) * dh_param[joint][A_PARAM];
    t_matrix[1][0] =  $sin(dh_param[joint][THETA]);
    t_matrix[1][1] =  $cos(dh_param[joint][THETA]) * $cos(dh_param[joint][ALPHA]);
    t_matrix[1][2] = -$cos(dh_param[joint][THETA]) * $sin(dh_param[joint][ALPHA]);
    t_matrix[1][3] =  $sin(dh_param[joint][THETA]) * dh_param[joint][A_PARAM];
    t_matrix[2][0] = 0.0;
    t_matrix[2][1] = $sin(dh_param[joint][ALPHA]);
    t_matrix[2][2] = $cos(dh_param[joint][ALPHA]);
    t_matrix[2][3] = dh_param[joint][D_PARAM];
    t_matrix[3][0] = 0.0;
    t_matrix[3][1] = 0.0;
    t_matrix[3][2] = 0.0;
    t_matrix[3][3] = 1.0;
    // full_matrix = copy * t_matrix;
    for ( int i=0 ; i<4 ; i++ ) begin // product row
      for ( int j=0 ; j<4 ; j++ ) begin // product column
      if ( joint==0 ) begin // first joint is just multiplied against identity; copy
  instead
          full_matrix[0][i][j] = t_matrix[i][j];
        end else begin
          real element = 0.0;
          for ( int k=0 ; k<4 ; k++ ) begin // inner term
            element += full_matrix[joint-1][i][k] * t_matrix[k][j];
          end
          full_matrix[joint][i][j] = element;
        end
      end
    end
  end

  // EXTRACT ROTATION FROM TRANSFORMATION MATRICES
  for ( int joint=0 ; joint<n ; joint++ )
    for ( int row=0 ; row<3 ; row++ )
      for ( int col=0 ; col<3 ; col++ )
        rotation[joint][row][col] = full_matrix[joint][row][col];

  // GENERATE AXES OF ROTATION
  axis[0] = z; // first joint just comes off of basis vector
  for ( int joint=1 ; joint<n+1 ; joint++ )
```

```
  for ( int row=0 ; row<3 ; row++ ) begin
    axis[joint][row] = 0.0; // clear data from last round
    for ( int col=0 ; col<3 ; col++ )
       axis[joint][row] += rotation[joint-1][row][col] * z[col];
  end

// EXTRACT POSITION FROM TRANSFORMATION MATRICES
position[0] = { 0.0, 0.0, 0.0 }; // first joint starts at origin
for ( int joint=1 ; joint<n+1 ; joint++ )
  for ( int row=0 ; row<3 ; row++ )
    position[joint][row] = full_matrix[joint-1][row][3];

$display("end effector x target, position: %f, %f", target[0], position[6][0]);
$display("end effector y target, position: %f, %f", target[1], position[6][1]);
$display("end effector z target, position: %f, %f", target[2], position[6][2]);
$display("end effector x target, rotation: %f, %f", target[3], axis[6][0]);
$display("end effector y target, rotation: %f, %f", target[4], axis[6][1]);
$display("end effector z target, rotation: %f, %f", target[5], axis[6][2]);

// CALCULATE VECTOR TO END OF EFFECTOR
for ( int joint=0 ; joint<n ; joint++ )
  for ( int row=0 ; row<3 ; row++ )
    dist_to_end[joint][row] = full_matrix[5][row][3] - position[joint][row];

// CREATE JACOBIAN COLUMN BY COLUMN
for ( int joint=0 ; joint<n ; joint++ ) begin
  if ( joint_type[joint]==1'b0 ) begin // translational
    m_jacobian[0][joint] = axis[joint][0];
    m_jacobian[1][joint] = axis[joint][1];
    m_jacobian[2][joint] = axis[joint][2];
    m_jacobian[3][joint] = 0.0;
    m_jacobian[4][joint] = 0.0;
    m_jacobian[5][joint] = 0.0;
  end else begin // rotational
    m_jacobian[0][joint] = axis[joint][1] * dist_to_end[joint][2] - axis[joint][2] *
dist_to_end[joint][1];
    m_jacobian[1][joint] = axis[joint][2] * dist_to_end[joint][0] - axis[joint][0] *
dist_to_end[joint][2];
    m_jacobian[2][joint] = axis[joint][0] * dist_to_end[joint][1] - axis[joint][1] *
dist_to_end[joint][0];
    m_jacobian[3][joint] = axis[joint][0];
    m_jacobian[4][joint] = axis[joint][1];
    m_jacobian[5][joint] = axis[joint][2];
  end
end

// jjt_bias = jacobian * jacobian transpose;
for ( int row=0 ; row<n ; row++ ) // product row
  for ( int col=0 ; col<n ; col++ ) begin // product column
    m_jjt_bias[row][col] = row==col ? 0.00001525878*4 : 0.0; // bias term
    for ( int k=0 ; k<n ; k++ ) // inner term
      m_jjt_bias[row][col] += m_jacobian[row][k] * m_jacobian[col][k];
  end

// ZERO OUT THE MODELS
for ( int row=0 ; row<n ; row++ ) begin
  for ( int col=0 ; col<n ; col++ ) begin
    m_lt[row][col] = 0.0;
    m_lt_inv[row][col] = 0.0;
    m_inverse[row][col] = 0.0;
    m_dls[row][col] = 0.0;
  end
  m_delta[row] = 0.0;
end

// CALCULATE LOWER TRIANGULAR MATRIX
for ( int row=0 ; row<n ; row++ ) begin
  for ( int col=0 ; col<(row+1) ; col++ ) begin
```

```
      real s = 0.0;
      for ( int index=0 ; index<col ; index++ )
        s += m_lt[row][index] * m_lt[col][index];
      m_lt[row][col] = (row==col) ? $sqrt(m_jjt_bias[row][col]-s) : (m_jjt_bias[row][col]-
s)/m_lt[col][col];
    end
  end

  // CALCULATE LOWER TRIANGULAR INVERSE MATRIX
  m_lt_inv[0][0] = 1.0 / m_lt[0][0];
  m_lt_inv[1][1] = 1.0 / m_lt[1][1];
  m_lt_inv[2][2] = 1.0 / m_lt[2][2];
  m_lt_inv[3][3] = 1.0 / m_lt[3][3];
  m_lt_inv[4][4] = 1.0 / m_lt[4][4];
  m_lt_inv[5][5] = 1.0 / m_lt[5][5];
  m_lt_inv[1][0] = -m_lt[1][0]*m_lt_inv[0][0] / m_lt[1][1];
  m_lt_inv[2][1] = -m_lt[2][1]*m_lt_inv[1][1] / m_lt[2][2];
  m_lt_inv[3][2] = -m_lt[3][2]*m_lt_inv[2][2] / m_lt[3][3];
  m_lt_inv[4][3] = -m_lt[4][3]*m_lt_inv[3][3] / m_lt[4][4];
  m_lt_inv[5][4] = -m_lt[5][4]*m_lt_inv[4][4] / m_lt[5][5];
  m_lt_inv[2][0] = ( -m_lt[2][1]*m_lt_inv[1][0] + -m_lt[2][0]*m_lt_inv[0][0] ) / m_lt
  [2][2];
  m_lt_inv[3][1] = ( -m_lt[3][2]*m_lt_inv[2][1] + -m_lt[3][1]*m_lt_inv[1][1] ) / m_lt
  [3][3];
  m_lt_inv[4][2] = ( -m_lt[4][3]*m_lt_inv[3][2] + -m_lt[4][2]*m_lt_inv[2][2] ) / m_lt
  [4][4];
  m_lt_inv[5][3] = ( -m_lt[5][4]*m_lt_inv[4][3] + -m_lt[5][3]*m_lt_inv[3][3] ) / m_lt
  [5][5];
  m_lt_inv[3][0] = ( -m_lt[3][2]*m_lt_inv[2][0] + -m_lt[3][1]*m_lt_inv[1][0] + -m_lt
  [3][0]*m_lt_inv[0][0] ) / m_lt[3][3];
  m_lt_inv[4][1] = ( -m_lt[4][3]*m_lt_inv[3][1] + -m_lt[4][2]*m_lt_inv[2][1] + -m_lt
  [4][1]*m_lt_inv[1][1] ) / m_lt[4][4];
  m_lt_inv[5][2] = ( -m_lt[5][4]*m_lt_inv[4][2] + -m_lt[5][3]*m_lt_inv[3][2] + -m_lt
  [5][2]*m_lt_inv[2][2] ) / m_lt[5][5];
  m_lt_inv[4][0] = ( -m_lt[4][3]*m_lt_inv[3][0] + -m_lt[4][2]*m_lt_inv[2][0] + -m_lt
  [4][1]*m_lt_inv[1][0] + -m_lt[4][0]*m_lt_inv[0][0] ) / m_lt[4][4];
  m_lt_inv[5][1] = ( -m_lt[5][4]*m_lt_inv[4][1] + -m_lt[5][3]*m_lt_inv[3][1] + -m_lt
  [5][2]*m_lt_inv[2][1] + -m_lt[5][1]*m_lt_inv[1][1] ) / m_lt[5][5];
  m_lt_inv[5][0] = ( -m_lt[5][4]*m_lt_inv[4][0] + -m_lt[5][3]*m_lt_inv[3][0] + -m_lt
  [5][2]*m_lt_inv[2][0] + -m_lt[5][1]*m_lt_inv[1][0] + -m_lt[5][0]*m_lt_inv[0][0] ) / m_lt
  [5][5];

  // MATRIX MULTIPLY TO GET INVERSE MATRIX
  // A^-1 = L^-T * L^-1
  for ( int row=0 ; row<n ; row++ )
    for ( int col=0 ; col<n ; col++ )
      for ( int index=0 ; index<n ; index++ )
        m_inverse[row][col] += m_lt_inv[index][row] * m_lt_inv[index][col];

  // MATRIX MULTIPLY TO GET DLS MATRIX
  // JT * INVERSE
  for ( int row=0 ; row<n ; row++ )
    for ( int col=0 ; col<n ; col++ )
      for ( int index=0 ; index<n ; index++ )
        m_dls[row][col] += m_jacobian[index][row] * m_inverse[index][col];

  // DETERMINE ERROR VECTOR
  for ( int row=0 ; row<3 ; row++ ) begin
    error[row] = target[row] - position[6][row];
    error[row+3] = target[row+3] - axis[6][row];
  end

  // MATRIX VECTOR MULTIPLY TO GET DELTA THETA
  for ( int row=0 ; row<n ; row++ )
    for ( int col=0 ; col<n ; col++ )
      m_delta[row] += m_dls[row][col] * error[col];

  // ADD DELTAS BACK TO DH PARAMS
```

```
    for ( int joint=0 ; joint<6 ; joint++ )
      if ( joint_type[joint] == 1'b0 ) // translational
        m_dh_param[joint][D_PARAM] = m_dh_param[joint][D_PARAM] + m_delta[joint];
      else
        m_dh_param[joint][THETA] = m_dh_param[joint][THETA] + m_delta[joint];

  endfunction

  function void check_ik_swift (
    logic [5:0] [5:0] [26:0] jacobian_matrix,
    logic [5:0] [5:0] [26:0] jjt_bias,
    logic [5:0] [5:0] [26:0] lt,
    logic [5:0] [5:0] [26:0] lt_inverse,
    logic [5:0] [5:0] [26:0] inverse,
    logic [5:0] [5:0] [26:0] dls,
    logic [5:0] [26:0] delta,
    logic done,
    logic [5:0] /*[3:0]*/ [20:0] dh_dyn_out
  );

    real abs_tol = 0.05;
    real rel_tol = 0.05;

    real jacobian_real[6][6];
    real jacobian_error[6][6];
    real jacobian_percent[6][6];

    real jjt_bias_real[6][6];
    real jjt_bias_error[6][6];
    real jjt_bias_percent[6][6];

    real lt_real[6][6];
    real lt_error[6][6];
    real lt_percent[6][6];

    real lt_inverse_real[6][6];
    real lt_inverse_error[6][6];
    real lt_inverse_percent[6][6];

    real inverse_real[6][6];
    real inverse_error[6][6];
    real inverse_percent[6][6];

    real dls_real[6][6];
    real dls_error[6][6];
    real dls_percent[6][6];

    real delta_real[6];
    real delta_error[6];
    real delta_percent[6];

    real dh_dyn_out_real[6]; //[4];
    real dh_dyn_out_error[6]; //[4];
    real dh_dyn_out_percent[6]; //[4];

    bit passed = 1'b1;

    // CHECK done
    if (done != 1'b1) begin
      $write("%t : fail done\n", $realtime);
      $exit();
    end

    // CHECK JACOBIAN
    for ( int i=0 ; i<n ; i++ ) begin // jacobian matrix row
      for ( int j=0 ; j<n ; j++ ) begin // jacobian matrix column
        jacobian_real[i][j] = real'(int'({{5{jacobian_matrix[i][j][26]}}, jacobian_matrix[i
    ][j]}))/65536.0;
```

```
        jacobian_error[i][j] = abs( jacobian_real[i][j] - m_jacobian[i][j] );
        jacobian_percent[i][j] = abs( jacobian_error[i][j] / m_jacobian[i][j] );
        if (jacobian_error[i][j]>abs_tol && jacobian_percent[i][j]>rel_tol) begin
          $write("%t : fail jacobian i=%d j=%d\n", $realtime, i, j);
          $write("m_jacobian=%f; dut_result=%f; jacobian_error=%f.\n", m_jacobian[i][j], jacobian_real[i][j], jacobian_error[i][j]);
          $write("m_jacobian=%f; dut_result=%f; jacobian_percent=%f.\n", m_jacobian[i][j], jacobian_real[i][j], jacobian_percent[i][j]);
          passed = 1'b0;
        end else begin
          // $write("%t : pass jacobian i=%d j=%d\n", $realtime, i, j);
        end
      end
    end

    // CHECK JJT
    for ( int i=0 ; i<n ; i++ ) begin // jjt row
      for ( int j=0 ; j<n ; j++ ) begin // jjt column
        jjt_bias_real[i][j] = real'(int'({{5{jjt_bias[i][j][26]}}, jjt_bias[i][j]}))/65536.0;
        jjt_bias_error[i][j] = abs( jjt_bias_real[i][j] - m_jjt_bias[i][j] );
        jjt_bias_percent[i][j] = abs( jjt_bias_error[i][j] / m_jjt_bias[i][j] );
        if (jjt_bias_error[i][j]>abs_tol && jjt_bias_percent[i][j]>rel_tol) begin
          $write("%t : fail jjt_bias i=%d j=%d\n", $realtime, i, j);
          $write("m_jjt_bias=%f; dut_result=%f; jjt_bias_error=%f.\n", m_jjt_bias[i][j], jjt_bias_real[i][j], jjt_bias_error[i][j]);
          $write("m_jjt_bias=%f; dut_result=%f; jjt_bias_percent=%f.\n", m_jjt_bias[i][j], jjt_bias_real[i][j], jjt_bias_percent[i][j]);
          passed = 1'b0;
        end else begin
          // $write("%t : pass jjt_bias i=%d j=%d\n", $realtime, i, j);
        end
      end
    end

    // CHECK CHOLESKY
    for ( int i=0 ; i<n ; i++ ) begin // cholesky row
      for ( int j=0 ; j<n ; j++ ) begin // cholesky column
        lt_real[i][j] = real'(int'({{5{lt[i][j][26]}}, lt[i][j]}))/65536.0;
        lt_error[i][j] = abs( lt_real[i][j] - m_lt[i][j] );
        lt_percent[i][j] = abs( lt_error[i][j] / m_lt[i][j] );
        if (lt_error[i][j]>abs_tol && lt_percent[i][j]>rel_tol) begin
          $write("%t : fail cholesky i=%d j=%d\n", $realtime, i, j);
          $write("m_lt=%f; dut_result=%f; lt_error=%f.\n", m_lt[i][j], lt_real[i][j], lt_error[i][j]);
          $write("m_lt=%f; dut_result=%f; lt_percent=%f.\n", m_lt[i][j], lt_real[i][j], lt_percent[i][j]);
          passed = 1'b0;
        end else begin
          // $write("%t : pass cholesky i=%d j=%d\n", $realtime, i, j);
        end
      end
    end

    // CHECK LT_INVERSE
    for ( int i=0 ; i<n ; i++ ) begin // lt_inverse row
      for ( int j=0 ; j<n ; j++ ) begin // lt_inverse column
        lt_inverse_real[i][j] = real'(int'({{5{lt_inverse[i][j][26]}}, lt_inverse[i][j]}))/65536.0;
        lt_inverse_error[i][j] = abs( lt_inverse_real[i][j] - m_lt_inv[i][j] );
        lt_inverse_percent[i][j] = abs( lt_inverse_error[i][j] / m_lt_inv[i][j] );
        if (lt_inverse_error[i][j]>abs_tol && lt_inverse_percent[i][j]>rel_tol) begin
          $write("%t : fail lt_inverse i=%d j=%d\n", $realtime, i, j);
          $write("m_lt_inv=%f; dut_result=%f; lt_inverse_error=%f.\n", m_lt_inv[i][j], lt_inverse_real[i][j], lt_inverse_error[i][j]);
          $write("m_lt_inv=%f; dut_result=%f; lt_inverse_percent=%f.\n", m_lt_inv[i][j], lt_inverse_real[i][j], lt_inverse_percent[i][j]);
          passed = 1'b0;
```

```
    end else begin
      // $write("%t : pass lt_inverse i=%d j=%d\n", $realtime, i, j);
    end
  end
end

// CHECK INVERSE
for ( int i=0 ; i<n ; i++ ) begin // inverse row
  for ( int j=0 ; j<n ; j++ ) begin // inverse column
    inverse_real[i][j] = real'(int'({{5{inverse[i][j][26]}}, inverse[i][j]}))/65536.0;
    inverse_error[i][j] = abs( inverse_real[i][j] - m_inverse[i][j] );
    inverse_percent[i][j] = abs( inverse_error[i][j] / m_inverse[i][j] );
    if (inverse_error[i][j]>abs_tol && inverse_percent[i][j]>rel_tol) begin
      $write("%t : fail inverse i=%d j=%d\n", $realtime, i, j);
      $write("m_inverse=%f; dut_result=%f; inverse_error=%f.\n", m_inverse[i][j],
inverse_real[i][j], inverse_error[i][j]);
      $write("m_inverse=%f; dut_result=%f; inverse_percent=%f.\n", m_inverse[i][j],
inverse_real[i][j], inverse_percent[i][j]);
      passed = 1'b0;
    end else begin
      // $write("%t : pass inverse i=%d j=%d\n", $realtime, i, j);
    end
  end
end

// CHECK DLS
for ( int i=0 ; i<n ; i++ ) begin // dls row
  for ( int j=0 ; j<n ; j++ ) begin // dls column
    dls_real[i][j] = real'(int'({{5{dls[i][j][26]}}, dls[i][j]}))/65536.0;
    dls_error[i][j] = abs( dls_real[i][j] - m_dls[i][j] );
    dls_percent[i][j] = abs( dls_error[i][j] / m_dls[i][j] );
    if (dls_error[i][j]>abs_tol && dls_percent[i][j]>rel_tol) begin
      $write("%t : fail dls i=%d j=%d\n", $realtime, i, j);
      $write("m_dls=%f; dut_result=%f; dls_error=%f.\n", m_dls[i][j], dls_real[i][j],
dls_error[i][j]);
      $write("m_dls=%f; dut_result=%f; dls_percent=%f.\n", m_dls[i][j], dls_real[i][j],
dls_percent[i][j]);
      passed = 1'b0;
    end else begin
      // $write("%t : pass dls i=%d j=%d\n", $realtime, i, j);
    end
  end
end

// CHECK DELTA
for ( int i=0 ; i<n ; i++ ) begin // delta row
  delta_real[i] = real'(int'({{5{delta[i][26]}}, delta[i]}))/65536.0;
  delta_error[i] = abs( delta_real[i] - m_delta[i] );
  delta_percent[i] = abs( delta_error[i] / m_delta[i] );
  if (delta_error[i]>abs_tol && delta_percent[i]>rel_tol) begin
    $write("%t : fail delta i=%d\n", $realtime, i);
    $write("m_delta=%f; dut_result=%f; delta_error=%f.\n", m_delta[i], delta_real[i],
delta_error[i]);
    $write("m_delta=%f; dut_result=%f; delta_percent=%f.\n", m_delta[i], delta_real[i],
delta_percent[i]);
    passed = 1'b0;
  end else begin
    // $write("%t : pass delta i=%d\n", $realtime, i);
  end
end

// CHECK DH_DYN_OUT
for ( int joint=0 ; joint<n ; joint++ ) begin // param joint
  // for ( int param=0 ; param<4 ; param++ ) begin // dh param
  dh_dyn_out_real[joint]/*[param]*/ = real'(int'({{11{dh_dyn_out[joint]/*[param]*/
[20]}}, dh_dyn_out[joint]/*[param]*/}))/65536.0;
  dh_dyn_out_error[joint]/*[param]*/ = abs( dh_dyn_out_real[joint]/*[param]*/ -
m_dh_param[joint][THETA] );
```

```
      dh_dyn_out_percent[joint]/*[param]*/ = abs( dh_dyn_out_error[joint]/*[param]*/ /
    m_dh_param[joint][THETA] );
      if (dh_dyn_out_error[joint]/*[param]*/>abs_tol && dh_dyn_out_percent[joint]/*[param]*/
    >rel_tol) begin
        $write("%t : fail dh_dyn_out joint=%d\n", $realtime, joint);
        passed = 1'b0;
      end else begin
        $write("%t : pass dh_dyn_out joint=%d\n", $realtime, joint);
      end
      $write("m_dh_param=%f; dut_result=%f; dh_dyn_out_error=%f.\n", m_dh_param[joint][THETA
    ], dh_dyn_out_real[joint]/*[param]*/, dh_dyn_out_error[joint]/*[param]*/);
      $write("m_dh_param=%f; dut_result=%f; dh_dyn_out_percent=%f.\n", m_dh_param[joint][
    THETA], dh_dyn_out_real[joint]/*[param]*/, dh_dyn_out_percent[joint]/*[param]*/);
      // end
    end

    if (passed) begin
      $display("%t : pass \n", $realtime);
    end else begin
      // $exit();
    end

  endfunction
endclass
```

../../rtl/ik_swift_32/ik_swift_test.sv

6.2.4 Jacobian Finder Hardware

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_full_jacobian (
  input logic clk
);

logic en, rst;
logic [7:0] count;

// base joint's axis of rotation/translation
logic [2:0] [17:0] z;

// bit vector describing type of each joint
logic [5:0] joint_type;

// dh joint parameters
logic [5:0] [3:0] [20:0] dh_param;

// shared multipliers
logic [8:0] [26:0] array_mult_result;
logic [5:0] [5:0] [26:0] mat_mult_result;

// shared multipliers
logic [8:0] [26:0] array_mult_dataa;
logic [8:0] [26:0] array_mult_datab;
logic [5:0] [5:0] [26:0] mat_mult_dataa;
logic [5:0] [5:0] [26:0] mat_mult_datab;

// multiplied results of transformation matrices
logic [5:0] [3:0] [3:0] [26:0] full_matrix;

// axis of rotation / translation for joints 1...6
logic [6:0] [2:0] [26:0] axis;
```

```
// location of joints 1...6
logic [5:0] [2:0] [26:0] dist_to_end;

// jacobian
logic [5:0] [5:0] [26:0] jacobian_matrix;

// jacobian * jacobian transpose + bias
logic [5:0] [5:0] [26:0] jjt_bias;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output z;
//   output joint_type;
//   output dh_param;
//
//   input full_matrix;
//   input axis;
//   input dist_to_end;
//   input jacobian_matrix;
//   input jjt_bias;
// endclocking
//
// modport full_jacobian_tb (clocking cb);

// restrict directions
modport full_jacobian (
  input clk,
  input en,
  input rst,
  input count,

  input z,
  input joint_type,
  input dh_param,

  input array_mult_result,
  input mat_mult_result,

  output array_mult_dataa,
  output array_mult_datab,
  output mat_mult_dataa,
  output mat_mult_datab,

  output full_matrix,
  output axis,
  output dist_to_end,
  output jacobian_matrix,

  output jjt_bias
);

endinterface
```

../../rtl/ik_swift_32/full_jacobian/full_jacobian_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module full_jacobian (
  ifc_full_jacobian.full_jacobian i
);

  // INSTANTIATE FULL MATRIX BLOCK
```

```
ifc_full_mat ifc_full_mat (i.clk);
// inputs
assign ifc_full_mat.en = i.en;
assign ifc_full_mat.rst = i.rst;
assign ifc_full_mat.count = i.count;
assign ifc_full_mat.dh_param = i.dh_param;
// shared multipliers
assign ifc_full_mat.array_mult_result = i.array_mult_result[5:0];
assign ifc_full_mat.mat_mult_result = i.mat_mult_result;
full_mat full_mat (ifc_full_mat.full_mat);
// outputs
assign i.full_matrix = ifc_full_mat.full_matrix;

// INSTANTIATE JACOBIAN BLOCK
ifc_jacobian ifc_jacobian (i.clk);
// inputs
assign ifc_jacobian.en = i.en;
assign ifc_jacobian.rst = i.rst;
assign ifc_jacobian.count = i.count;
assign ifc_jacobian.z = i.z;
assign ifc_jacobian.joint_type = i.joint_type;
// transformation matrices from full matrix block
assign ifc_jacobian.full_matrix = ifc_full_mat.full_matrix;
// shared mulitpliers
assign ifc_jacobian.array_mult_result = i.array_mult_result;
assign ifc_jacobian.mat_mult_result = i.mat_mult_result;
jacobian jacobian (ifc_jacobian.jacobian);
// outputs
assign i.axis = ifc_jacobian.axis;
assign i.dist_to_end = ifc_jacobian.dist_to_end;
assign i.jacobian_matrix = ifc_jacobian.jacobian_matrix;

// MATRIX MULTIPLY FOR JJT JACOBIAN * JACOBIAN TRANSPOSE
logic [5:0] [5:0] [26:0] jjt_dataa;
logic [5:0] [5:0] [26:0] jjt_datab;

// MAT_MULT INPUTS
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd0: begin
        jjt_dataa <= {36{27'b0}};
        jjt_datab <= {36{27'b0}};
      end
      8'd99: begin
        jjt_dataa <= i.jacobian_matrix;
        jjt_datab <= {
          { i.jacobian_matrix[5][5], i.jacobian_matrix[4][5], i.jacobian_matrix[3][5], i.
  jacobian_matrix[2][5], i.jacobian_matrix[1][5], i.jacobian_matrix[0][5] },
          { i.jacobian_matrix[5][4], i.jacobian_matrix[4][4], i.jacobian_matrix[3][4], i.
  jacobian_matrix[2][4], i.jacobian_matrix[1][4], i.jacobian_matrix[0][4] },
          { i.jacobian_matrix[5][3], i.jacobian_matrix[4][3], i.jacobian_matrix[3][3], i.
  jacobian_matrix[2][3], i.jacobian_matrix[1][3], i.jacobian_matrix[0][3] },
          { i.jacobian_matrix[5][2], i.jacobian_matrix[4][2], i.jacobian_matrix[3][2], i.
  jacobian_matrix[2][2], i.jacobian_matrix[1][2], i.jacobian_matrix[0][2] },
          { i.jacobian_matrix[5][1], i.jacobian_matrix[4][1], i.jacobian_matrix[3][1], i.
  jacobian_matrix[2][1], i.jacobian_matrix[1][1], i.jacobian_matrix[0][1] },
          { i.jacobian_matrix[5][0], i.jacobian_matrix[4][0], i.jacobian_matrix[3][0], i.
  jacobian_matrix[2][0], i.jacobian_matrix[1][0], i.jacobian_matrix[0][0] }
        };
      end
      8'd111: begin
        jjt_dataa <= {36{27'b0}};
        jjt_datab <= {36{27'b0}};
      end
      default: begin
        jjt_dataa <= jjt_dataa;
        jjt_datab <= jjt_datab;
```

```
        end
      endcase

  // MAT_MULT OUTPUTS
  always_ff @(posedge i.clk)
    if (i.en)
      if ( i.count==8'd111 )
        i.jjt_bias <= i.mat_mult_result + {
        { 27'd4, 27'b0, 27'b0, 27'b0, 27'b0, 27'b0 },
        { 27'b0, 27'd4, 27'b0, 27'b0, 27'b0, 27'b0 },
        { 27'b0, 27'b0, 27'd4, 27'b0, 27'b0, 27'b0 },
        { 27'b0, 27'b0, 27'b0, 27'd4, 27'b0, 27'b0 },
        { 27'b0, 27'b0, 27'b0, 27'b0, 27'd4, 27'b0 },
        { 27'b0, 27'b0, 27'b0, 27'b0, 27'b0, 27'd4 }};

  // timing design prevents module outputs to shared multipliers colliding
  assign i.array_mult_dataa = {81'b0,ifc_full_mat.array_mult_dataa} | ifc_jacobian.
    array_mult_dataa;
  assign i.array_mult_datab = {81'b0,ifc_full_mat.array_mult_datab} | ifc_jacobian.
    array_mult_datab;
  assign i.mat_mult_dataa = ifc_full_mat.mat_mult_dataa | ifc_jacobian.mat_mult_dataa |
    jjt_dataa;
  assign i.mat_mult_datab = ifc_full_mat.mat_mult_datab | ifc_jacobian.mat_mult_datab |
    jjt_datab;

endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_jacobian.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_jacobian (
  input logic clk
);

logic en, rst;

// Global clock cycle counter
logic [7:0] count;

// Base joint's axis of rotation/translation
logic [2:0] [17:0] z;

// Bit vector describing type of each joint
logic [5:0] joint_type;

// T blocks
logic [5:0] [3:0] [3:0] [26:0] full_matrix;

// Axis of rotation / translation for joints 1...6
logic [6:0] [2:0] [26:0] axis;

// Location of joints 1...6
logic [5:0] [2:0] [26:0] dist_to_end;

// Jacobian
logic [5:0] [5:0] [26:0] jacobian_matrix;

// Output to array multipliers
logic [8:0] [26:0] array_mult_dataa;
logic [8:0] [26:0] array_mult_datab;
// Input from array multipliers
logic [8:0] [26:0] array_mult_result;
// Output to matrix multipliers
```

```
logic [5:0] [5:0] [26:0] mat_mult_dataa;
logic [5:0] [5:0] [26:0] mat_mult_datab;
// Input from matrix multipliers
logic [5:0] [5:0] [26:0] mat_mult_result;

// clocking cb @(posedge clk);
//   output en, rst;
//   output z;
//   output joint_type;
//   output full_matrix;
//
//   input axis;
//   input dist_to_end;
//   input jacobian_matrix;
// endclocking
//
// modport jacobian_tb (clocking cb);

// restrict directions
modport jacobian (
  input clk, en, rst,
  input count,

  input z,
  input joint_type,
  input full_matrix,

  input array_mult_result,
  input mat_mult_result,

  output array_mult_dataa,
  output array_mult_datab,
  output mat_mult_dataa,
  output mat_mult_datab,

  output axis,
  output dist_to_end,
  output jacobian_matrix
);

endinterface
```

../../rtl/ik_swift_32/full_jacobian/jacobian/jacobian_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module jacobian (
  ifc_jacobian.jacobian i
);

  // Axis of rotation / translation for joints 1...6
  // logic [5:0] [2:0] [26:0] i.axis;

  // Location of joints 1...6
  // logic [5:0] [2:0] [26:0] i.dist_to_end;

  // LOGIC GOVERNING JOINT COUNT FOR AXIS
  //Which joint we're on
  logic [2:0] joint;
  always_ff @(posedge i.clk)
    if (i.en)
      case (i.count)
        8'd28: joint <= 3'd0;
        8'd41: joint <= 3'd1;
```

```
      8'd53: joint <= 3'd2;
      8'd65: joint <= 3'd3;
      8'd77: joint <= 3'd4;
      8'd89: joint <= 3'd5;
      default: joint <= joint;
    endcase

// LOGIC GOVERNING ARRAY MULT INPUT
always_ff @(posedge i.clk)
  if (i.en)
    if (
      i.count==8'd29 || i.count==8'd30 ||
      i.count==8'd42 || i.count==8'd43 ||
      i.count==8'd54 || i.count==8'd55 ||
      i.count==8'd66 || i.count==8'd67 ||
      i.count==8'd78 || i.count==8'd79 ||
      i.count==8'd90 || i.count==8'd91 ) begin
      // || i.count==8'd90
      i.array_mult_dataa[0] <= i.full_matrix[joint][0][0];
      i.array_mult_dataa[1] <= i.full_matrix[joint][0][1];
      i.array_mult_dataa[2] <= i.full_matrix[joint][0][2];
      i.array_mult_dataa[3] <= i.full_matrix[joint][1][0];
      i.array_mult_dataa[4] <= i.full_matrix[joint][1][1];
      i.array_mult_dataa[5] <= i.full_matrix[joint][1][2];
      i.array_mult_dataa[6] <= i.full_matrix[joint][2][0];
      i.array_mult_dataa[7] <= i.full_matrix[joint][2][1];
      i.array_mult_dataa[8] <= i.full_matrix[joint][2][2];

      i.array_mult_datab[0] <= {{9{i.z[0][17]}}, i.z[0]};
      i.array_mult_datab[1] <= {{9{i.z[1][17]}}, i.z[1]};
      i.array_mult_datab[2] <= {{9{i.z[2][17]}}, i.z[2]};
      i.array_mult_datab[3] <= {{9{i.z[0][17]}}, i.z[0]};
      i.array_mult_datab[4] <= {{9{i.z[1][17]}}, i.z[1]};
      i.array_mult_datab[5] <= {{9{i.z[2][17]}}, i.z[2]};
      i.array_mult_datab[6] <= {{9{i.z[0][17]}}, i.z[0]};
      i.array_mult_datab[7] <= {{9{i.z[1][17]}}, i.z[1]};
      i.array_mult_datab[8] <= {{9{i.z[2][17]}}, i.z[2]};
    end else begin
      i.array_mult_dataa <= {9{27'b0}};
      i.array_mult_datab <= {9{27'b0}};
    end

// LOGIC GOVERNING ARRAY MULT OUTPUT
// axis[5:0] (axis of rotation/translation)
always_ff @(posedge i.clk) begin
  if (i.en)
    if ( i.count==8'd34 || i.count==8'd47 || i.count==8'd59 || i.count==8'd71 || i.count
  ==8'd83 || i.count==8'd95 ) begin
      // || i.count==8'd95
      i.axis[joint+1][0] <= i.array_mult_result[0] + i.array_mult_result[1];
      i.axis[joint+1][1] <= i.array_mult_result[3] + i.array_mult_result[4];
      i.axis[joint+1][2] <= i.array_mult_result[6] + i.array_mult_result[7];
    end else if ( i.count==8'd35 || i.count==8'd48 || i.count==8'd60 || i.count==8'd72 ||
  i.count==8'd84 || i.count==8'd96 ) begin
      // || i.count==8'd96
      i.axis[joint+1][0] <= i.axis[joint+1][0] + i.array_mult_result[2];
      i.axis[joint+1][1] <= i.axis[joint+1][1] + i.array_mult_result[5];
      i.axis[joint+1][2] <= i.axis[joint+1][2] + i.array_mult_result[8];
    end
  i.axis[0] <= {
    {{9{i.z[2][17]}}, i.z[2]},
    {{9{i.z[1][17]}}, i.z[1]},
    {{9{i.z[0][17]}}, i.z[0]}
  };

end

// LOGIC GOVERNING dist_to_end[0]/2/3/4/5/6 (dist_to_end of joint)
```

```
always_ff @(posedge i.clk)
  if (i.en)
    if ( i.count==8'd90 ) begin
      i.dist_to_end[1][0] <= i.full_matrix[5][0][3] - i.full_matrix[0][0][3];
      i.dist_to_end[1][1] <= i.full_matrix[5][1][3] - i.full_matrix[0][1][3];
      i.dist_to_end[1][2] <= i.full_matrix[5][2][3] - i.full_matrix[0][2][3];
      i.dist_to_end[2][0] <= i.full_matrix[5][0][3] - i.full_matrix[1][0][3];
      i.dist_to_end[2][1] <= i.full_matrix[5][1][3] - i.full_matrix[1][1][3];
      i.dist_to_end[2][2] <= i.full_matrix[5][2][3] - i.full_matrix[1][2][3];
      i.dist_to_end[3][0] <= i.full_matrix[5][0][3] - i.full_matrix[2][0][3];
      i.dist_to_end[3][1] <= i.full_matrix[5][1][3] - i.full_matrix[2][1][3];
      i.dist_to_end[3][2] <= i.full_matrix[5][2][3] - i.full_matrix[2][2][3];
      i.dist_to_end[4][0] <= i.full_matrix[5][0][3] - i.full_matrix[3][0][3];
      i.dist_to_end[4][1] <= i.full_matrix[5][1][3] - i.full_matrix[3][1][3];
      i.dist_to_end[4][2] <= i.full_matrix[5][2][3] - i.full_matrix[3][2][3];
      i.dist_to_end[5][0] <= i.full_matrix[5][0][3] - i.full_matrix[4][0][3];
      i.dist_to_end[5][1] <= i.full_matrix[5][1][3] - i.full_matrix[4][1][3];
      i.dist_to_end[5][2] <= i.full_matrix[5][2][3] - i.full_matrix[4][2][3];
    end
assign i.dist_to_end[0] = {
  i.full_matrix[5][2][3],
  i.full_matrix[5][1][3],
  i.full_matrix[5][0][3]
};

// LOGIC GOVERNING MAT MULT INPUT
// LOGIC GOVERNING dataa/datab (multiplications for cross-products)
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd0: begin
        i.mat_mult_dataa <= {36{27'b0}};
        i.mat_mult_datab <= {36{27'b0}};
      end
      8'd91: begin
        i.mat_mult_dataa <= {
          i.axis[0][1],i.axis[0][2],i.axis[0][2],i.axis[0][0],i.axis[0][0],i.axis[0][1],
          i.axis[1][1],i.axis[1][2],i.axis[1][2],i.axis[1][0],i.axis[1][0],i.axis[1][1],
          i.axis[2][1],i.axis[2][2],i.axis[2][2],i.axis[2][0],i.axis[2][0],i.axis[2][1],
          i.axis[3][1],i.axis[3][2],i.axis[3][2],i.axis[3][0],i.axis[3][0],i.axis[3][1],
          i.axis[4][1],i.axis[4][2],i.axis[4][2],i.axis[4][0],i.axis[4][0],i.axis[4][1],
          i.axis[5][1],i.axis[5][2],i.axis[5][2],i.axis[5][0],i.axis[5][0],i.axis[5][1]
        };
        i.mat_mult_datab <= {
          i.dist_to_end[0][2],i.dist_to_end[0][1],i.dist_to_end[0][0],i.dist_to_end[0][2],
  i.dist_to_end[0][1],i.dist_to_end[0][0],
          i.dist_to_end[1][2],i.dist_to_end[1][1],i.dist_to_end[1][0],i.dist_to_end[1][2],
  i.dist_to_end[1][1],i.dist_to_end[1][0],
          i.dist_to_end[2][2],i.dist_to_end[2][1],i.dist_to_end[2][0],i.dist_to_end[2][2],
  i.dist_to_end[2][1],i.dist_to_end[2][0],
          i.dist_to_end[3][2],i.dist_to_end[3][1],i.dist_to_end[3][0],i.dist_to_end[3][2],
  i.dist_to_end[3][1],i.dist_to_end[3][0],
          i.dist_to_end[4][2],i.dist_to_end[4][1],i.dist_to_end[4][0],i.dist_to_end[4][2],
  i.dist_to_end[4][1],i.dist_to_end[4][0],
          i.dist_to_end[5][2],i.dist_to_end[5][1],i.dist_to_end[5][0],i.dist_to_end[5][2],
  i.dist_to_end[5][1],i.dist_to_end[5][0]
        };
      end
      8'd98: begin // clear
        i.mat_mult_dataa <= {36{27'b0}};
        i.mat_mult_datab <= {36{27'b0}};
      end
      default: begin
        i.mat_mult_dataa <= i.mat_mult_dataa;
        i.mat_mult_datab <= i.mat_mult_datab;
      end
    endcase
```

```
  // LOGIC GOVERNING RESULT (Jacobian matrix)
  parameter MAX = 6;
  genvar col;
  generate
    for ( col=0 ; col<MAX ; col++ ) begin: jacobian_col
      always_ff @(posedge i.clk)
        if (i.en)
          if ( i.count==8'd98 )
            if ( i.joint_type[col]==1'b1 ) begin // rotational
              i.jacobian_matrix[0][col] <= i.mat_mult_result[5-col][5] - i.mat_mult_result[5-col][4];
              i.jacobian_matrix[1][col] <= i.mat_mult_result[5-col][3] - i.mat_mult_result[5-col][2];
              i.jacobian_matrix[2][col] <= i.mat_mult_result[5-col][1] - i.mat_mult_result[5-col][0];
              i.jacobian_matrix[3][col] <= i.axis[col][0];
              i.jacobian_matrix[4][col] <= i.axis[col][1];
              i.jacobian_matrix[5][col] <= i.axis[col][2];
            end else if ( i.joint_type[col]==1'b0 ) begin
              i.jacobian_matrix[0][col] <= i.axis[col][0];
              i.jacobian_matrix[1][col] <= i.axis[col][1];
              i.jacobian_matrix[2][col] <= i.axis[col][2];
              i.jacobian_matrix[3][col] <= 27'b0;
              i.jacobian_matrix[4][col] <= 27'b0;
              i.jacobian_matrix[5][col] <= 27'b0;
            end
    end // end col loop
  endgenerate
endmodule
```

../../rtl/ik_swift_32/full_jacobian/jacobian/jacobian.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_full_mat (
  input logic clk
);

logic en, rst;
logic [7:0] count;

logic [5:0] [3:0] [20:0] dh_param;

// shared multipliers
logic [5:0] [26:0] array_mult_result;
logic [5:0] [5:0] [26:0] mat_mult_result;

// shared multipliers
logic [5:0] [26:0] array_mult_dataa;
logic [5:0] [26:0] array_mult_datab;
logic [5:0] [5:0] [26:0] mat_mult_dataa;
logic [5:0] [5:0] [26:0] mat_mult_datab;

// multiplied results of transformation matrices
logic [5:0] [3:0] [3:0] [26:0] full_matrix;

// clocking cb @(posedge clk);
//  output en;
//  output rst;
//  output dh_param;
//
//  input full_matrix;
// endclocking
//
```

```
// modport full_mat_tb (clocking cb);

// restrict directions
modport full_mat (
  input clk,
  input en,
  input rst,
  input count,
  input dh_param,

  input array_mult_result,
  input mat_mult_result,

  output array_mult_dataa,
  output array_mult_datab,
  output mat_mult_dataa,
  output mat_mult_datab,

  output full_matrix
);

endinterface
```

../../rtl/ik_swift_32/full_jacobian/full_mat/full_mat_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module full_mat (
  ifc_full_mat.full_mat i
);

  parameter THETA = 0;
  parameter A_PARAM = 1;
  parameter D_PARAM = 2;
  parameter ALPHA = 3;

  // each transformation matrix
  logic [5:0] [3:0] [3:0] [26:0] t_matrix_array;

  // instantiate t_block
  ifc_t_block ifc_t_block (i.clk);
  assign ifc_t_block.en = i.en;
  assign ifc_t_block.rst = i.rst;
  assign ifc_t_block.count = i.count;
  assign ifc_t_block.array_mult_result = i.array_mult_result;
  t_block t_block (ifc_t_block.t_block);
  assign i.array_mult_dataa = ifc_t_block.array_mult_dataa;
  assign i.array_mult_datab = ifc_t_block.array_mult_datab;

  // LOGIC GOVERNING T_BLOCK INPUTS
  always_ff @(posedge i.clk)
    if (i.en)
      if ( 8'd0 <= i.count && i.count < 8'd6 ) begin
        ifc_t_block.a <= i.dh_param[i.count][A_PARAM];
        ifc_t_block.d <= i.dh_param[i.count][D_PARAM];
        ifc_t_block.alpha <= i.dh_param[i.count][ALPHA];
        ifc_t_block.theta <= i.dh_param[i.count][THETA];
      end else begin
        ifc_t_block.a <= 21'b0;
        ifc_t_block.d <= 21'b0;
        ifc_t_block.alpha <= 21'b0;
        ifc_t_block.theta <= 21'b0;
      end
```

```
// LOGIC GOVERNING T_BLOCK OUTPUTS
always_ff @(posedge i.clk)
  if (i.en)
    if ( 8'd28 <= i.count && i.count < 8'd34 ) begin
      t_matrix_array[i.count-8'd28] <= ifc_t_block.t_matrix;
    end else begin
      // do nothing
    end

// LOGIC GOVERNING MAT_MULT INPUTS
always_ff @(posedge i.clk)
  if (i.en)
    case(i.count)
      8'd29: begin // t_02
        i.mat_mult_dataa <=
        {
          {6{27'b0}},
          {27'b0,t_matrix_array[0][3],27'b0},
          {27'b0,t_matrix_array[0][2],27'b0},
          {27'b0,t_matrix_array[0][1],27'b0},
          {27'b0,t_matrix_array[0][0],27'b0},
          {6{27'b0}}
        };
        i.mat_mult_datab <= // FAST FORWARD
        {
          {6{27'b0}},
          {27'b0,ifc_t_block.t_matrix[3],27'b0},
          {27'b0,ifc_t_block.t_matrix[2],27'b0},
          {27'b0,ifc_t_block.t_matrix[1],27'b0},
          {27'b0,ifc_t_block.t_matrix[0],27'b0},
          {6{27'b0}}
        };
      end
      8'd41: begin // t_03
        i.mat_mult_dataa <= i.mat_mult_result;
        i.mat_mult_datab <=
        {
          {6{27'b0}},
          {27'b0,t_matrix_array[2][3],27'b0},
          {27'b0,t_matrix_array[2][2],27'b0},
          {27'b0,t_matrix_array[2][1],27'b0},
          {27'b0,t_matrix_array[2][0],27'b0},
          {6{27'b0}}
        };
      end
      8'd53: begin // t_04
        i.mat_mult_dataa <= i.mat_mult_result;
        i.mat_mult_datab <=
        {
          {6{27'b0}},
          {27'b0,t_matrix_array[3][3],27'b0},
          {27'b0,t_matrix_array[3][2],27'b0},
          {27'b0,t_matrix_array[3][1],27'b0},
          {27'b0,t_matrix_array[3][0],27'b0},
          {6{27'b0}}
        };
      end
      8'd65: begin // t_05
        i.mat_mult_dataa <= i.mat_mult_result;
        i.mat_mult_datab <=
        {
          {6{27'b0}},
          {27'b0,t_matrix_array[4][3],27'b0},
          {27'b0,t_matrix_array[4][2],27'b0},
          {27'b0,t_matrix_array[4][1],27'b0},
          {27'b0,t_matrix_array[4][0],27'b0},
          {6{27'b0}}
        };
```

```
      end
      8'd77: begin // t_06
        i.mat_mult_dataa <= i.mat_mult_result;
        i.mat_mult_datab <=
        {
          {6{27'b0}},
          {27'b0,t_matrix_array[5][3],27'b0},
          {27'b0,t_matrix_array[5][2],27'b0},
          {27'b0,t_matrix_array[5][1],27'b0},
          {27'b0,t_matrix_array[5][0],27'b0},
          {6{27'b0}}
        };
      end
      8'd89: begin // clear
        i.mat_mult_dataa <= {36{27'b0}};
        i.mat_mult_datab <= {36{27'b0}};
      end
      default: begin
        i.mat_mult_dataa <= i.mat_mult_dataa;
        i.mat_mult_datab <= i.mat_mult_datab;
      end
    endcase

  // LOGIC GOVERNING MAT_MULT OUTPUTS
  always_ff @(posedge i.clk)
    if (i.en)
      case(i.count)
        8'd28: begin // t_01
          i.full_matrix[0] <= ifc_t_block.t_matrix;
        end
        8'd41: begin // t_02
          i.full_matrix[1][3] <= i.mat_mult_result[4][4:1];
          i.full_matrix[1][2] <= i.mat_mult_result[3][4:1];
          i.full_matrix[1][1] <= i.mat_mult_result[2][4:1];
          i.full_matrix[1][0] <= i.mat_mult_result[1][4:1];
        end
        8'd53: begin // t_03
          i.full_matrix[2][3] <= i.mat_mult_result[4][4:1];
          i.full_matrix[2][2] <= i.mat_mult_result[3][4:1];
          i.full_matrix[2][1] <= i.mat_mult_result[2][4:1];
          i.full_matrix[2][0] <= i.mat_mult_result[1][4:1];
        end
        8'd65: begin // t_04
          i.full_matrix[3][3] <= i.mat_mult_result[4][4:1];
          i.full_matrix[3][2] <= i.mat_mult_result[3][4:1];
          i.full_matrix[3][1] <= i.mat_mult_result[2][4:1];
          i.full_matrix[3][0] <= i.mat_mult_result[1][4:1];
        end
        8'd77: begin // t_05
          i.full_matrix[4][3] <= i.mat_mult_result[4][4:1];
          i.full_matrix[4][2] <= i.mat_mult_result[3][4:1];
          i.full_matrix[4][1] <= i.mat_mult_result[2][4:1];
          i.full_matrix[4][0] <= i.mat_mult_result[1][4:1];
        end
        8'd89: begin // t_06
          i.full_matrix[5][3] <= i.mat_mult_result[4][4:1];
          i.full_matrix[5][2] <= i.mat_mult_result[3][4:1];
          i.full_matrix[5][1] <= i.mat_mult_result[2][4:1];
          i.full_matrix[5][0] <= i.mat_mult_result[1][4:1];
        end
        default: begin
          // do nothing
        end
      endcase

endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/full_mat.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_t_block (
  input logic clk
);

logic en, rst;

logic [7:0] count;

logic [20:0] a;
logic [20:0] d;
logic [20:0] alpha;
logic [20:0] theta;

logic [5:0] [26:0] array_mult_result;
logic [5:0] [26:0] array_mult_dataa;
logic [5:0] [26:0] array_mult_datab;

logic [3:0] [3:0] [26:0] t_matrix;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output count;
//   output a;
//   output d;
//   output alpha;
//   output theta;
//
//   input t_matrix;
// endclocking
//
// modport t_block_tb (clocking cb);

// restrict directions
modport t_block (
  input clk,
  input en,
  input rst,
  input count,

  input a,
  input d,
  input alpha,
  input theta,

  input array_mult_result,
  output array_mult_dataa,
  output array_mult_datab,

  output t_matrix
);

endinterface
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/t_block_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
```

```
 * Columbia University
 */

module t_block (
  ifc_t_block.t_block i
);

  ifc_sincos i_alpha (i.clk);
  assign i_alpha.angle = i.alpha;
  assign i_alpha.en = i.en;
  assign i_alpha.rst = i.rst;
  sincos sincos_alpha (i_alpha.sincos);

  ifc_sincos i_theta (i.clk);
  assign i_theta.angle = i.theta;
  assign i_theta.en = i.en;
  assign i_theta.rst = i.rst;
  sincos sincos_theta (i_theta.sincos);

  // delay a by 22
  logic [22:0] [20:0] a_delay;
  assign a_delay[0] = i.a;
  always_ff @(posedge i.clk)
    if (i.en)
      a_delay[22:1] <= a_delay[21:0];

  // delay d by 26
  logic [27:0] [20:0] d_delay;
  assign d_delay[0] = i.d;
  always_ff @(posedge i.clk)
    if (i.en)
      d_delay[27:1] <= d_delay[26:0];

  logic [20:0] neg_sin_theta;
  assign neg_sin_theta = -i_theta.sin;
  logic [20:0] neg_sin_alpha;
  assign neg_sin_alpha = -i_alpha.sin;

  // LOGIC GOVERNING ARRAY MULT INPUTS
  always_ff @(posedge i.clk)
    if (i.en)
      if ( 8'd23<=i.count && i.count<8'd29 ) begin
        i.array_mult_dataa[0] <= {{6{neg_sin_theta[20]}}, neg_sin_theta};
        i.array_mult_datab[0] <= {{6{i_alpha.cos[20]}}, i_alpha.cos};
        i.array_mult_dataa[1] <= {{6{i_theta.sin[20]}}, i_theta.sin};
        i.array_mult_datab[1] <= {{6{i_alpha.sin[20]}}, i_alpha.sin};
        i.array_mult_dataa[2] <= {{6{a_delay[22][20]}}, a_delay[22]};
        i.array_mult_datab[2] <= {{6{i_theta.cos[20]}}, i_theta.cos};
        i.array_mult_dataa[3] <= {{6{i_theta.cos[20]}}, i_theta.cos};
        i.array_mult_datab[3] <= {{6{i_alpha.cos[20]}}, i_alpha.cos};
        i.array_mult_dataa[4] <= {{6{i_theta.cos[20]}}, i_theta.cos};
        i.array_mult_datab[4] <= {{6{neg_sin_alpha[20]}}, neg_sin_alpha};
        i.array_mult_dataa[5] <= {{6{a_delay[22][20]}}, a_delay[22]};
        i.array_mult_datab[5] <= {{6{i_theta.sin[20]}}, i_theta.sin};
      end else begin
        i.array_mult_dataa[5:0] <= {5{27'b0}};
        i.array_mult_datab[5:0] <= {5{27'b0}};
      end

  // delay cos(theta) by 4
  logic [5:0] [20:0] cos_theta_delay;
  assign cos_theta_delay[0] = i_theta.cos;
  always_ff @(posedge i.clk)
    if (i.en)
      cos_theta_delay[5:1] <= cos_theta_delay[4:0];

  // delay sin(theta) by 4
  logic [5:0] [20:0] sin_theta_delay;
```

```
  assign sin_theta_delay[0] = i_theta.sin;
  always_ff @(posedge i.clk)
    if (i.en)
      sin_theta_delay[5:1] <= sin_theta_delay[4:0];

  // delay cos(alpha) by 4
  logic [5:0] [20:0] cos_alpha_delay;
  assign cos_alpha_delay[0] = i_alpha.cos;
  always_ff @(posedge i.clk)
    if (i.en)
      cos_alpha_delay[5:1] <= cos_alpha_delay[4:0];

  // delay sin(alpha) by 4
  logic [5:0] [20:0] sin_alpha_delay;
  assign sin_alpha_delay[0] = i_alpha.sin;
  always_ff @(posedge i.clk)
    if (i.en)
      sin_alpha_delay[5:1] <= sin_alpha_delay[4:0];

  assign i.t_matrix[0][0] = {{6{cos_theta_delay[5][20]}}, cos_theta_delay[5]};
  assign i.t_matrix[0][1] = i.array_mult_result[0];
  assign i.t_matrix[0][2] = i.array_mult_result[1];
  assign i.t_matrix[0][3] = i.array_mult_result[2];

  assign i.t_matrix[1][0] = {{6{sin_theta_delay[5][20]}}, sin_theta_delay[5]};
  assign i.t_matrix[1][1] = i.array_mult_result[3];
  assign i.t_matrix[1][2] = i.array_mult_result[4];
  assign i.t_matrix[1][3] = i.array_mult_result[5];

  assign i.t_matrix[2][0] = 27'b0;
  assign i.t_matrix[2][1] = {{6{sin_alpha_delay[5][20]}}, sin_alpha_delay[5]};
  assign i.t_matrix[2][2] = {{6{cos_alpha_delay[5][20]}}, cos_alpha_delay[5]};
  assign i.t_matrix[2][3] = {{6{d_delay[27][20]}}, d_delay[27]};

  assign i.t_matrix[3][0] = 27'b0;
  assign i.t_matrix[3][1] = 27'b0;
  assign i.t_matrix[3][2] = 27'b0;
  assign i.t_matrix[3][3] = 27'd65536;

endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/t_block.sv

6.2.5 Matrix Inverter Hardware

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_inverse (
  input logic clk
);

logic en, rst;

// Global clock cycle counter
logic [7:0] count;

// Matrix that needs to be inversed
logic [5:0] [5:0] [26:0] matrix;

// LT decomposition of given matrix
logic [5:0] [5:0] [26:0] lt;
// LT inverse
```

```
logic [5:0] [5:0] [26:0] lt_inverse;
// Inverse of given matrix
logic [5:0] [5:0] [26:0] inverse;

// Output to matrix multipliers
logic [5:0] [5:0] [26:0] mat_mult_dataa;
logic [5:0] [5:0] [26:0] mat_mult_datab;
// Input from matrix multipliers
logic [5:0] [5:0] [26:0] mat_mult_result;

// Output to array multipliers
logic [14:0] [26:0] array_mult_dataa;
logic [14:0] [26:0] array_mult_datab;
// Input from array multipliers
logic [14:0] [26:0] array_mult_result;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output matrix;
//
//   input lt;
//   input lt_inverse;
//   input inverse;
// endclocking
//
// modport inverse_tb (clocking cb);

modport inverse_dut (
  input clk, en, rst,
  input count,

  input matrix,

  input array_mult_result,
  input mat_mult_result,
  output array_mult_dataa,
  output array_mult_datab,
  output mat_mult_dataa,
  output mat_mult_datab,

  output lt,
  output lt_inverse,
  output inverse
);

endinterface
```

../../rtl/ik_swift_32/inverse/inverse_interface.sv

```
// the timescale directive tells the compiler the clock period and the
// precision that needs to be displayed in the VCD dump file

`timescale 1ns/1ps

module inverse (
  ifc_inverse.inverse_dut i
);

  // INSTANTIATE CHOLESKY BLOCK
  ifc_cholesky_block ifc_cholesky_block (i.clk);
  // inputs
  assign ifc_cholesky_block.en = i.en;
  assign ifc_cholesky_block.rst = i.rst;
  assign ifc_cholesky_block.count = i.count;
  assign ifc_cholesky_block.matrix = i.matrix;
  cholesky_block cholesky_block (ifc_cholesky_block.cholesky_block);
```

```
// outputs
assign i.lt = ifc_cholesky_block.lt;

// INSTANTIATE LT INVERSE BLOCK
ifc_lt_block ifc_lt_block (i.clk);
// inputs
assign ifc_lt_block.en = i.en;
assign ifc_lt_block.rst = i.rst;
assign ifc_lt_block.count = i.count;
// lower triangular matrix from cholesky block
assign ifc_lt_block.lt = ifc_cholesky_block.lt;
lt_block lt_block (ifc_lt_block.lt_block_dut);
// outputs
assign i.lt_inverse = ifc_lt_block.lt_inverse;

// INSTANTIATE SHARED ARRAY DIV
ifc_array_div ifc_array_div (i.clk);
// inputs
assign ifc_array_div.en = i.en;
assign ifc_array_div.rst = i.rst;
// timing design prevents module outputs to shared dividers colliding
assign ifc_array_div.dividends = ifc_cholesky_block.dividends | ifc_lt_block.dividends;
assign ifc_array_div.divisor = ifc_cholesky_block.divisor;
array_div array_div (ifc_array_div.array_div);
assign ifc_cholesky_block.quotients = ifc_array_div.quotients;
assign ifc_lt_block.quotients = ifc_array_div.quotients;

// SHARED ARRAY MULT
// timing design prevents module outputs to shared multipliers colliding
assign i.array_mult_dataa =
  ifc_cholesky_block.array_mult_dataa |
  ifc_lt_block.array_mult_dataa;
assign i.array_mult_datab =
  ifc_cholesky_block.array_mult_datab |
  ifc_lt_block.array_mult_datab;
assign ifc_cholesky_block.array_mult_result = i.array_mult_result;
assign ifc_lt_block.array_mult_result = i.array_mult_result;

// MATRIX MULTIPLY FOR L^-T * L^-1
// MAT_MULT INPUTS
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd0: begin
        i.mat_mult_dataa <= {36{27'b0}};
        i.mat_mult_datab <= {36{27'b0}};
      end
      8'd215: begin
        i.mat_mult_dataa <= {
          { i.lt_inverse[5][5], i.lt_inverse[4][5], i.lt_inverse[3][5], i.lt_inverse
  [2][5], i.lt_inverse[1][5], i.lt_inverse[0][5] },
          { i.lt_inverse[5][4], i.lt_inverse[4][4], i.lt_inverse[3][4], i.lt_inverse
  [2][4], i.lt_inverse[1][4], i.lt_inverse[0][4] },
          { i.lt_inverse[5][3], i.lt_inverse[4][3], i.lt_inverse[3][3], i.lt_inverse
  [2][3], i.lt_inverse[1][3], i.lt_inverse[0][3] },
          { i.lt_inverse[5][2], i.lt_inverse[4][2], i.lt_inverse[3][2], i.lt_inverse
  [2][2], i.lt_inverse[1][2], i.lt_inverse[0][2] },
          { i.lt_inverse[5][1], i.lt_inverse[4][1], i.lt_inverse[3][1], i.lt_inverse
  [2][1], i.lt_inverse[1][1], i.lt_inverse[0][1] },
          { i.lt_inverse[5][0], i.lt_inverse[4][0], i.lt_inverse[3][0], i.lt_inverse
  [2][0], i.lt_inverse[1][0], i.lt_inverse[0][0] }
        };
        i.mat_mult_datab <= i.lt_inverse;
      end
      8'd227: begin
        i.mat_mult_dataa <= {36{27'b0}};
        i.mat_mult_datab <= {36{27'b0}};
      end
```

```
        default: begin
          i.mat_mult_dataa <= i.mat_mult_dataa;
          i.mat_mult_datab <= i.mat_mult_datab;
        end
      endcase

  // MAT_MULT OUTPUTS
  always_ff @(posedge i.clk)
    if (i.en)
      if ( i.count==8'd227 )
        i.inverse <= i.mat_mult_result;

endmodule
```

../../rtl/ik_swift_32/inverse/inverse.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_cholesky_block (
  input logic clk
);

logic en, rst;
logic [7:0] count;

logic [5:0] [5:0] [26:0] matrix;
logic [5:0] [5:0] [26:0] lt;

// shared array_mult
logic [14:0] [26:0] array_mult_dataa;
logic [14:0] [26:0] array_mult_datab;
logic [14:0] [26:0] array_mult_result;

// shared array_div
logic [5:0] [26:0] dividends;
logic [26:0] divisor;
logic [5:0] [26:0] quotients;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output count;
//
//   output matrix;
//   input lt;
// endclocking
//
// modport cholesky_block_tb (clocking cb);

// restrict directions
modport cholesky_block(

  input clk, en, rst,
  input count,

  input matrix,

  input array_mult_result,
  input quotients,
  output array_mult_dataa,
  output array_mult_datab,
  output dividends,
  output divisor,
```

```
  output lt
);

endinterface
```

../../rtl/ik_swift_32/inverse/cholesky_block/cholesky_block_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module cholesky_block (
  ifc_cholesky_block.cholesky_block i
);

  // LOGIC GOVERNING ROW COUNTER
  logic [2:0] row;
  always_ff @(posedge i.clk)
    if (i.en)
      case (i.count)
        8'd111: row <= 3'd0;
        8'd129: row <= 3'd1;
        8'd147: row <= 3'd2;
        8'd165: row <= 3'd3;
        8'd183: row <= 3'd4;
        8'd201: row <= 3'd5;
        default: row <= row;
      endcase

  // LOWER TEMP MATRIX
  logic [5:0] [5:0] [26:0] temp;

  // INSTANTITATE THE SQRT
  ifc_sqrt_43 ifc_sqrt_43 (i.clk);
  assign ifc_sqrt_43.en = i.en;
  assign ifc_sqrt_43.rst = i.rst;
  assign ifc_sqrt_43.q[26:22] = 5'b0;
  logic [22:0] remainder;
  sqrt_43 sqrt_43_inst (
    .clk ( ifc_sqrt_43.clk ),
    .ena ( ifc_sqrt_43.en ),
    .radical ( { ifc_sqrt_43.radical, 16'b0 } ),
    .q ( ifc_sqrt_43.q[21:0] ),
    .remainder ( remainder )
  );

  // LOGIC GOVERNING SQRT RADICAL
  // L_xx = sqrt(A_xx)
  always_ff @(posedge i.clk)
    if (i.en)
      if ( i.count==8'd112 )
        ifc_sqrt_43.radical <= i.matrix[row][row];
      else if (
        i.count==8'd130 || i.count==8'd148 || i.count==8'd166 || i.count==8'd184 || i.count
    ==8'd202
      )
        ifc_sqrt_43.radical <= temp[row][row];
      else
        ifc_sqrt_43.radical <= 27'b0;

  // LOGIC GOVERNING SQRT Q
  // L_xx = sqrt(A_xx)
  // LOGIC GOVERNING ARRAY DIV QUOTIENTS
  always_ff @(posedge i.clk)
    if (i.en)
```

```
    case (i.count)
      8'd112: begin
        // INITIALIZE LOWER TRIANGULAR MATRIX
        i.lt <= {36{27'b0}};
      end
      8'd118: begin
        i.lt[0][0] <= ifc_sqrt_43.q;
      end
      8'd124: begin
        i.lt[1][0] <= i.quotients[1]; // L_10 = A_10 / L_00
        i.lt[2][0] <= i.quotients[2]; // L_20 = A_20 / L_00
        i.lt[3][0] <= i.quotients[3]; // L_30 = A_30 / L_00
        i.lt[4][0] <= i.quotients[4]; // L_40 = A_40 / L_00
        i.lt[5][0] <= i.quotients[5]; // L_50 = A_50 / L_00
      end
      8'd136: begin
        i.lt[1][1] <= ifc_sqrt_43.q;
      end
      8'd142: begin
        i.lt[2][1] <= i.quotients[2]; // L_21 = A_21 / L_11
        i.lt[3][1] <= i.quotients[3]; // L_31 = A_31 / L_11
        i.lt[4][1] <= i.quotients[4]; // L_41 = A_41 / L_11
        i.lt[5][1] <= i.quotients[5]; // L_51 = A_51 / L_11
      end
      8'd154: begin
        i.lt[2][2] <= ifc_sqrt_43.q;
      end
      8'd160: begin
        i.lt[3][2] <= i.quotients[3]; // L_32 = A_32 / L_22
        i.lt[4][2] <= i.quotients[4]; // L_42 = A_42 / L_22
        i.lt[5][2] <= i.quotients[5]; // L_52 = A_52 / L_22
      end
      8'd172: begin
        i.lt[3][3] <= ifc_sqrt_43.q;
      end
      8'd178: begin
        i.lt[4][3] <= i.quotients[4]; // L_43 = A_43 / L_33
        i.lt[5][3] <= i.quotients[5]; // L_53 = A_53 / L_33
      end
      8'd190: begin
        i.lt[4][4] <= ifc_sqrt_43.q;
      end
      8'd196: begin
        i.lt[5][4] <= i.quotients[5]; // L_54 = A_54 / L_44
      end
      8'd208: begin
        i.lt[5][5] <= ifc_sqrt_43.q;
      end
      default: begin
      end
    endcase

// LOGIC GOVERNING ARRAY DIV DIVIDENDS
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd118: begin
        i.dividends[0] <= 27'b0;
        i.dividends[1] <= temp[1][0]; // L_10 = A_10 / L_00
        i.dividends[2] <= temp[2][0]; // L_20 = A_20 / L_00
        i.dividends[3] <= temp[3][0]; // L_30 = A_30 / L_00
        i.dividends[4] <= temp[4][0]; // L_40 = A_40 / L_00
        i.dividends[5] <= temp[5][0]; // L_50 = A_50 / L_00
      end
      8'd136: begin
        i.dividends[0] <= 27'b0;
        i.dividends[1] <= 27'b0;
        i.dividends[2] <= temp[2][1]; // L_21 = A_21 / L_11
```

```
        i.dividends[3] <= temp[3][1]; // L_31 = A_31 / L_11
        i.dividends[4] <= temp[4][1]; // L_41 = A_41 / L_11
        i.dividends[5] <= temp[5][1]; // L_51 = A_51 / L_11
      end
      8'd154: begin
        i.dividends[0] <= 27'b0;
        i.dividends[1] <= 27'b0;
        i.dividends[2] <= 27'b0;
        i.dividends[3] <= temp[3][2]; // L_32 = A_32 / L_22
        i.dividends[4] <= temp[4][2]; // L_42 = A_42 / L_22
        i.dividends[5] <= temp[5][2]; // L_52 = A_52 / L_22
      end
      8'd172: begin
        i.dividends[0] <= 27'b0;
        i.dividends[1] <= 27'b0;
        i.dividends[2] <= 27'b0;
        i.dividends[3] <= 27'b0;
        i.dividends[4] <= temp[4][3]; // L_43 = A_43 / L_33
        i.dividends[5] <= temp[5][3]; // L_53 = A_53 / L_33
      end
      8'd190: begin
        i.dividends[0] <= 27'b0;
        i.dividends[1] <= 27'b0;
        i.dividends[2] <= 27'b0;
        i.dividends[3] <= 27'b0;
        i.dividends[4] <= 27'b0;
        i.dividends[5] <= temp[5][4]; // L_54 = A_54 / L_44
      end
      default: begin
        i.dividends[0] <= 27'b0;
        i.dividends[1] <= 27'b0;
        i.dividends[2] <= 27'b0;
        i.dividends[3] <= 27'b0;
        i.dividends[4] <= 27'b0;
        i.dividends[5] <= 27'b0;
      end
    endcase

// LOGIC GOVERNING ARRAY DIV DIVISOR
// fast forwarding from sqrt
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd118: i.divisor <= ifc_sqrt_43.q; // i.lt[row][row]; // L_00
      8'd136: i.divisor <= ifc_sqrt_43.q; // i.lt[row][row]; // L_11
      8'd154: i.divisor <= ifc_sqrt_43.q; // i.lt[row][row]; // L_22
      8'd172: i.divisor <= ifc_sqrt_43.q; // i.lt[row][row]; // L_33
      8'd190: i.divisor <= ifc_sqrt_43.q; // i.lt[row][row]; // L_44
      8'd208: i.divisor <= ifc_sqrt_43.q; // i.lt[row][row]; // L_55
      default: i.divisor <= 27'b0;
    endcase

// LOGIC GOVERNING ARRAY MULT DATAA
// fast forwarding from array div
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd124: begin
        i.array_mult_dataa[0] <= i.quotients[1]; // A_11 = A_11 - L_10 * L_01 = A_11 -
  L_10 * L_10
        i.array_mult_dataa[1] <= i.quotients[2]; // A_21 = A_21 - L_20 * L_01 = A_21 -
  L_20 * L_10
        i.array_mult_dataa[2] <= i.quotients[2]; // A_22 = A_22 - L_20 * L_02 = A_22 -
  L_20 * L_20
        i.array_mult_dataa[3] <= i.quotients[3]; // A_31 = A_31 - L_30 * L_01 = A_31 -
  L_30 * L_10
        i.array_mult_dataa[4] <= i.quotients[3]; // A_32 = A_32 - L_30 * L_02 = A_32 -
  L_30 * L_20
```

```
        i.array_mult_dataa[5] <= i.quotients[3]; // A_33 = A_33 - L_30 * L_03 = A_33 -
L_30 * L_30
        i.array_mult_dataa[6] <= i.quotients[4]; // A_41 = A_41 - L_40 * L_01 = A_41 -
L_40 * L_10
        i.array_mult_dataa[7] <= i.quotients[4]; // A_42 = A_42 - L_40 * L_02 = A_42 -
L_40 * L_20
        i.array_mult_dataa[8] <= i.quotients[4]; // A_43 = A_43 - L_40 * L_03 = A_43 -
L_40 * L_30
        i.array_mult_dataa[9] <= i.quotients[4]; // A_44 = A_44 - L_40 * L_04 = A_44 -
L_40 * L_40
        i.array_mult_dataa[10] <= i.quotients[5]; // A_51 = A_51 - L_50 * L_01 = A_51 -
L_50 * L_10
        i.array_mult_dataa[11] <= i.quotients[5]; // A_52 = A_52 - L_50 * L_02 = A_52 -
L_50 * L_20
        i.array_mult_dataa[12] <= i.quotients[5]; // A_53 = A_53 - L_50 * L_03 = A_53 -
L_50 * L_30
        i.array_mult_dataa[13] <= i.quotients[5]; // A_54 = A_54 - L_50 * L_04 = A_54 -
L_50 * L_40
        i.array_mult_dataa[14] <= i.quotients[5]; // A_55 = A_55 - L_50 * L_05 = A_55 -
L_50 * L_50
      end
      8'd142: begin
        i.array_mult_dataa[0] <= i.quotients[2]; // A_22 = A_22 - L_21 * L_12 = A_22 -
L_21 * L_21
        i.array_mult_dataa[1] <= i.quotients[3]; // A_32 = A_32 - L_31 * L_12 = A_32 -
L_31 * L_21
        i.array_mult_dataa[2] <= i.quotients[3]; // A_33 = A_33 - L_31 * L_13 = A_33 -
L_31 * L_31
        i.array_mult_dataa[3] <= i.quotients[4]; // A_42 = A_42 - L_41 * L_12 = A_42 -
L_41 * L_21
        i.array_mult_dataa[4] <= i.quotients[4]; // A_43 = A_43 - L_41 * L_13 = A_43 -
L_41 * L_31
        i.array_mult_dataa[5] <= i.quotients[4]; // A_44 = A_44 - L_41 * L_14 = A_44 -
L_41 * L_41
        i.array_mult_dataa[6] <= i.quotients[5]; // A_52 = A_52 - L_51 * L_12 = A_52 -
L_51 * L_21
        i.array_mult_dataa[7] <= i.quotients[5]; // A_53 = A_53 - L_51 * L_13 = A_53 -
L_51 * L_31
        i.array_mult_dataa[8] <= i.quotients[5]; // A_54 = A_54 - L_51 * L_14 = A_54 -
L_51 * L_41
        i.array_mult_dataa[9] <= i.quotients[5]; // A_55 = A_55 - L_51 * L_15 = A_55 -
L_51 * L_51
      end
      8'd160: begin
        i.array_mult_dataa[0] <= i.quotients[3]; // A_33 = A_33 - L_32 * L_23 = A_33 -
L_32 * L_32
        i.array_mult_dataa[1] <= i.quotients[4]; // A_43 = A_43 - L_42 * L_23 = A_43 -
L_42 * L_32
        i.array_mult_dataa[2] <= i.quotients[4]; // A_44 = A_44 - L_42 * L_24 = A_44 -
L_42 * L_42
        i.array_mult_dataa[3] <= i.quotients[5]; // A_53 = A_53 - L_52 * L_23 = A_53 -
L_52 * L_32
        i.array_mult_dataa[4] <= i.quotients[5]; // A_54 = A_54 - L_52 * L_24 = A_54 -
L_52 * L_42
        i.array_mult_dataa[5] <= i.quotients[5]; // A_55 = A_55 - L_52 * L_25 = A_55 -
L_52 * L_52
      end
      8'd178: begin
        i.array_mult_dataa[0] <= i.quotients[4]; // A_44 = A_44 - L_43 * L_34 = A_44 -
L_43 * L_43
        i.array_mult_dataa[1] <= i.quotients[5]; // A_54 = A_54 - L_53 * L_34 = A_54 -
L_53 * L_43
        i.array_mult_dataa[2] <= i.quotients[5]; // A_55 = A_55 - L_53 * L_35 = A_55 -
L_53 * L_53
      end
      8'd196: begin
        i.array_mult_dataa[0] <= i.quotients[5]; // A_55 = A_55 - L_54 * L_45 = A_55 -
L_54 * L_54
```

```
      end
      default: begin
        i.array_mult_dataa <= {15{27'b0}};
      end
    endcase

// LOGIC GOVERNING ARRAY MULT DATAB
// fast forwarding from array div
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd124: begin
        i.array_mult_datab[0] <= i.quotients[1]; // A_11 = A_11 - L_10 * L_01 = A_11 - L_10 * L_10
        i.array_mult_datab[1] <= i.quotients[1]; // A_21 = A_21 - L_20 * L_01 = A_21 - L_20 * L_10
        i.array_mult_datab[2] <= i.quotients[2]; // A_22 = A_22 - L_20 * L_02 = A_22 - L_20 * L_20
        i.array_mult_datab[3] <= i.quotients[1]; // A_31 = A_31 - L_30 * L_01 = A_31 - L_30 * L_10
        i.array_mult_datab[4] <= i.quotients[2]; // A_32 = A_32 - L_30 * L_02 = A_32 - L_30 * L_20
        i.array_mult_datab[5] <= i.quotients[3]; // A_33 = A_33 - L_30 * L_03 = A_33 - L_30 * L_30
        i.array_mult_datab[6] <= i.quotients[1]; // A_41 = A_41 - L_40 * L_01 = A_41 - L_40 * L_10
        i.array_mult_datab[7] <= i.quotients[2]; // A_42 = A_42 - L_40 * L_02 = A_42 - L_40 * L_20
        i.array_mult_datab[8] <= i.quotients[3]; // A_43 = A_43 - L_40 * L_03 = A_43 - L_40 * L_30
        i.array_mult_datab[9] <= i.quotients[4]; // A_44 = A_44 - L_40 * L_04 = A_44 - L_40 * L_40
        i.array_mult_datab[10] <= i.quotients[1]; // A_51 = A_51 - L_50 * L_01 = A_51 - L_50 * L_10
        i.array_mult_datab[11] <= i.quotients[2]; // A_52 = A_52 - L_50 * L_02 = A_52 - L_50 * L_20
        i.array_mult_datab[12] <= i.quotients[3]; // A_53 = A_53 - L_50 * L_03 = A_53 - L_50 * L_30
        i.array_mult_datab[13] <= i.quotients[4]; // A_54 = A_54 - L_50 * L_04 = A_54 - L_50 * L_40
        i.array_mult_datab[14] <= i.quotients[5]; // A_55 = A_55 - L_50 * L_05 = A_55 - L_50 * L_50
      end
      8'd142: begin
        i.array_mult_datab[0] <= i.quotients[2]; // A_22 = A_22 - L_21 * L_12 = A_22 - L_21 * L_21
        i.array_mult_datab[1] <= i.quotients[2]; // A_32 = A_32 - L_31 * L_12 = A_32 - L_31 * L_21
        i.array_mult_datab[2] <= i.quotients[3]; // A_33 = A_33 - L_31 * L_13 = A_33 - L_31 * L_31
        i.array_mult_datab[3] <= i.quotients[2]; // A_42 = A_42 - L_41 * L_12 = A_42 - L_41 * L_21
        i.array_mult_datab[4] <= i.quotients[3]; // A_43 = A_43 - L_41 * L_13 = A_43 - L_41 * L_31
        i.array_mult_datab[5] <= i.quotients[4]; // A_44 = A_44 - L_41 * L_14 = A_44 - L_41 * L_41
        i.array_mult_datab[6] <= i.quotients[2]; // A_52 = A_52 - L_51 * L_12 = A_52 - L_51 * L_21
        i.array_mult_datab[7] <= i.quotients[3]; // A_53 = A_53 - L_51 * L_13 = A_53 - L_51 * L_31
        i.array_mult_datab[8] <= i.quotients[4]; // A_54 = A_54 - L_51 * L_14 = A_54 - L_51 * L_41
        i.array_mult_datab[9] <= i.quotients[5]; // A_55 = A_55 - L_51 * L_15 = A_55 - L_51 * L_51
      end
      8'd160: begin
        i.array_mult_datab[0] <= i.quotients[3]; // A_33 = A_33 - L_32 * L_23 = A_33 - L_32 * L_32
```

```
          i.array_mult_datab[1] <= i.quotients[3]; // A_43 = A_43 - L_42 * L_23 = A_43 - L_42 * L_32
          i.array_mult_datab[2] <= i.quotients[4]; // A_44 = A_44 - L_42 * L_24 = A_44 - L_42 * L_42
          i.array_mult_datab[3] <= i.quotients[3]; // A_53 = A_53 - L_52 * L_23 = A_53 - L_52 * L_32
          i.array_mult_datab[4] <= i.quotients[4]; // A_54 = A_54 - L_52 * L_24 = A_54 - L_52 * L_42
          i.array_mult_datab[5] <= i.quotients[5]; // A_55 = A_55 - L_52 * L_25 = A_55 - L_52 * L_52
        end
        8'd178: begin
          i.array_mult_datab[0] <= i.quotients[4]; // A_44 = A_44 - L_43 * L_34 = A_44 - L_43 * L_43
          i.array_mult_datab[1] <= i.quotients[4]; // A_54 = A_54 - L_53 * L_34 = A_54 - L_53 * L_43
          i.array_mult_datab[2] <= i.quotients[5]; // A_55 = A_55 - L_53 * L_35 = A_55 - L_53 * L_53
        end
        8'd196: begin
          i.array_mult_datab[0] <= i.quotients[5]; // A_55 = A_55 - L_54 * L_45 = A_55 - L_54 * L_54
        end
        default: begin
          i.array_mult_datab <= {15{27'b0}};
        end
      endcase

  // LOGIC GOVERNING ARRAY MULT RESULT
  always_ff @(posedge i.clk)
    if (i.en)
      case (i.count)
        8'd112: begin
          // INITIALIZE LOWER TEMP MATRIX
          temp <= i.matrix;
        end
        8'd129: begin
          temp[1][1] <= temp[1][1] - i.array_mult_result[0]; // A_11 = A_11 - L_10 * L_01 = A_11 - L_10 * L_10
          temp[2][1] <= temp[2][1] - i.array_mult_result[1]; // A_21 = A_21 - L_20 * L_01 = A_21 - L_20 * L_10
          temp[2][2] <= temp[2][2] - i.array_mult_result[2]; // A_22 = A_22 - L_20 * L_02 = A_22 - L_20 * L_20
          temp[3][1] <= temp[3][1] - i.array_mult_result[3]; // A_31 = A_31 - L_30 * L_01 = A_31 - L_30 * L_10
          temp[3][2] <= temp[3][2] - i.array_mult_result[4]; // A_32 = A_32 - L_30 * L_02 = A_32 - L_30 * L_20
          temp[3][3] <= temp[3][3] - i.array_mult_result[5]; // A_33 = A_33 - L_30 * L_03 = A_33 - L_30 * L_30
          temp[4][1] <= temp[4][1] - i.array_mult_result[6]; // A_41 = A_41 - L_40 * L_01 = A_41 - L_40 * L_10
          temp[4][2] <= temp[4][2] - i.array_mult_result[7]; // A_42 = A_42 - L_40 * L_02 = A_42 - L_40 * L_20
          temp[4][3] <= temp[4][3] - i.array_mult_result[8]; // A_43 = A_43 - L_40 * L_03 = A_43 - L_40 * L_30
          temp[4][4] <= temp[4][4] - i.array_mult_result[9]; // A_44 = A_44 - L_40 * L_04 = A_44 - L_40 * L_40
          temp[5][1] <= temp[5][1] - i.array_mult_result[10]; // A_51 = A_51 - L_50 * L_01 = A_51 - L_50 * L_10
          temp[5][2] <= temp[5][2] - i.array_mult_result[11]; // A_52 = A_52 - L_50 * L_02 = A_52 - L_50 * L_20
          temp[5][3] <= temp[5][3] - i.array_mult_result[12]; // A_53 = A_53 - L_50 * L_03 = A_53 - L_50 * L_30
          temp[5][4] <= temp[5][4] - i.array_mult_result[13]; // A_54 = A_54 - L_50 * L_04 = A_54 - L_50 * L_40
          temp[5][5] <= temp[5][5] - i.array_mult_result[14]; // A_55 = A_55 - L_50 * L_05 = A_55 - L_50 * L_50
        end
```

```
        8'd147: begin
          temp[2][2] <= temp[2][2] - i.array_mult_result[0]; // A_22 = A_22 - L_21 * L_12 = A_22 - L_21 * L_21
          temp[3][2] <= temp[3][2] - i.array_mult_result[1]; // A_32 = A_32 - L_31 * L_12 = A_32 - L_31 * L_21
          temp[3][3] <= temp[3][3] - i.array_mult_result[2]; // A_33 = A_33 - L_31 * L_13 = A_33 - L_31 * L_31
          temp[4][2] <= temp[4][2] - i.array_mult_result[3]; // A_42 = A_42 - L_41 * L_12 = A_42 - L_41 * L_21
          temp[4][3] <= temp[4][3] - i.array_mult_result[4]; // A_43 = A_43 - L_41 * L_13 = A_43 - L_41 * L_31
          temp[4][4] <= temp[4][4] - i.array_mult_result[5]; // A_44 = A_44 - L_41 * L_14 = A_44 - L_41 * L_41
          temp[5][2] <= temp[5][2] - i.array_mult_result[6]; // A_52 = A_52 - L_51 * L_12 = A_52 - L_51 * L_21
          temp[5][3] <= temp[5][3] - i.array_mult_result[7]; // A_53 = A_53 - L_51 * L_13 = A_53 - L_51 * L_31
          temp[5][4] <= temp[5][4] - i.array_mult_result[8]; // A_54 = A_54 - L_51 * L_14 = A_54 - L_51 * L_41
          temp[5][5] <= temp[5][5] - i.array_mult_result[9]; // A_55 = A_55 - L_51 * L_15 = A_55 - L_51 * L_51
        end
        8'd165: begin
          temp[3][3] <= temp[3][3] - i.array_mult_result[0]; // A_33 = A_33 - L_32 * L_23 = A_33 - L_32 * L_32
          temp[4][3] <= temp[4][3] - i.array_mult_result[1]; // A_43 = A_43 - L_42 * L_23 = A_43 - L_42 * L_32
          temp[4][4] <= temp[4][4] - i.array_mult_result[2]; // A_44 = A_44 - L_42 * L_24 = A_44 - L_42 * L_42
          temp[5][3] <= temp[5][3] - i.array_mult_result[3]; // A_53 = A_53 - L_52 * L_23 = A_53 - L_52 * L_32
          temp[5][4] <= temp[5][4] - i.array_mult_result[4]; // A_54 = A_54 - L_52 * L_24 = A_54 - L_52 * L_42
          temp[5][5] <= temp[5][5] - i.array_mult_result[5]; // A_55 = A_55 - L_52 * L_25 = A_55 - L_52 * L_52
        end
        8'd183: begin
          temp[4][4] <= temp[4][4] - i.array_mult_result[0]; // A_44 = A_44 - L_43 * L_34 = A_44 - L_43 * L_43
          temp[5][4] <= temp[5][4] - i.array_mult_result[1]; // A_54 = A_54 - L_53 * L_34 = A_54 - L_53 * L_43
          temp[5][5] <= temp[5][5] - i.array_mult_result[2]; // A_55 = A_55 - L_53 * L_35 = A_55 - L_53 * L_53
        end
        8'd201: begin
          temp[5][5] <= temp[5][5] - i.array_mult_result[0]; // A_55 = A_55 - L_54 * L_45 = A_55 - L_54 * L_54
        end
        default: begin
        end
      endcase

endmodule
```

../../rtl/ik_swift_32/inverse/cholesky_block/cholesky_block.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_lt_block (
  input logic clk
);

logic en, rst;
logic [7:0] count;
```

```
logic [5:0] [5:0] [26:0] lt;
logic [5:0] [5:0] [26:0] lt_inverse;

// shared array_mult
logic [14:0] [26:0] array_mult_dataa;
logic [14:0] [26:0] array_mult_datab;
logic [14:0] [26:0] array_mult_result;

// shared array_div
logic [5:0] [26:0] dividends;
logic [26:0] divisor;
logic [5:0] [26:0] quotients;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output count;
//
//   output lt;
//   input lt_inverse;
// endclocking
//
// modport lt_block_tb (clocking cb);

// restrict directions
modport lt_block_dut (

  input clk, en, rst,
  input count,

  input lt,

  input array_mult_result,
  input quotients,
  output array_mult_dataa,
  output array_mult_datab,
  output dividends,
  output divisor,

  output lt_inverse

);

endinterface
```

../../rtl/ik_swift_32/inverse/lt_block/lt_block_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */
module lt_block (
  ifc_lt_block.lt_block_dut i
);

  // LOWER TEMP MATRIX
  // logic [5:0] [5:0] [35:0] temp;

  // LOGIC GOVERNING ARRAY DIV DIVIDENDS
  always_ff @(posedge i.clk)
    if (i.en)
      case (i.count)
        8'd118: begin
          i.dividends[0] <= 27'd65536; // K_00 = 1 / L_00
          i.dividends[1] <= 27'b0;
          i.dividends[2] <= 27'b0;
```

```
        i.dividends[3] <= 27'b0;
        i.dividends[4] <= 27'b0;
        i.dividends[5] <= 27'b0;
      end
      8'd136: begin
        i.dividends[0] <= i.lt_inverse[1][0]; // K_10 = K_10 / L_11
        i.dividends[1] <= 27'd65536; // K_11 = 1 / L_11
        i.dividends[2] <= 27'b0;
        i.dividends[3] <= 27'b0;
        i.dividends[4] <= 27'b0;
        i.dividends[5] <= 27'b0;
      end
      8'd154: begin
        i.dividends[0] <= i.lt_inverse[2][0]; // K_20 = K_20 / L_22
        i.dividends[1] <= i.lt_inverse[2][1]; // K_21 = K_21 / L_22
        i.dividends[2] <= 27'd65536; // K_22 = 1 / L_22
        i.dividends[3] <= 27'b0;
        i.dividends[4] <= 27'b0;
        i.dividends[5] <= 27'b0;
      end
      8'd172: begin
        i.dividends[0] <= i.lt_inverse[3][0]; // K_30 = K_30 / L_33
        i.dividends[1] <= i.lt_inverse[3][1]; // K_31 = K_31 / L_33
        i.dividends[2] <= i.lt_inverse[3][2]; // K_32 = K_32 / L_33
        i.dividends[3] <= 27'd65536; // K_33 = 1 / L_33
        i.dividends[4] <= 27'b0;
        i.dividends[5] <= 27'b0;
      end
      8'd190: begin
        i.dividends[0] <= i.lt_inverse[4][0]; // K_40 = K_40 / L_44
        i.dividends[1] <= i.lt_inverse[4][1]; // K_41 = K_41 / L_44
        i.dividends[2] <= i.lt_inverse[4][2]; // K_42 = K_42 / L_44
        i.dividends[3] <= i.lt_inverse[4][3]; // K_43 = K_43 / L_44
        i.dividends[4] <= 27'd65536; // K_44 = 1 / L_44
        i.dividends[5] <= 27'b0;
      end
      8'd208: begin
        i.dividends[0] <= i.lt_inverse[5][0]; // K_50 = K_50 / L_55
        i.dividends[1] <= i.lt_inverse[5][1]; // K_51 = K_51 / L_55
        i.dividends[2] <= i.lt_inverse[5][2]; // K_52 = K_52 / L_55
        i.dividends[3] <= i.lt_inverse[5][3]; // K_53 = K_53 / L_55
        i.dividends[4] <= i.lt_inverse[5][4]; // K_54 = K_54 / L_55
        i.dividends[5] <= 27'd65536; // K_55 = 1 / L_55
      end
      default: begin
        i.dividends[0] <= 27'b0;
        i.dividends[1] <= 27'b0;
        i.dividends[2] <= 27'b0;
        i.dividends[3] <= 27'b0;
        i.dividends[4] <= 27'b0;
        i.dividends[5] <= 27'b0;
      end
    endcase

// LOGIC GOVERNING ARRAY DIV DIVISOR
// remove during integration to enable
// fast forwarding from sqrt
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd118: i.divisor <= i.lt[0][0]; // i.lt[row][row]; // L_00
      8'd136: i.divisor <= i.lt[1][1]; // i.lt[row][row]; // L_11
      8'd154: i.divisor <= i.lt[2][2]; // i.lt[row][row]; // L_22
      8'd172: i.divisor <= i.lt[3][3]; // i.lt[row][row]; // L_33
      8'd190: i.divisor <= i.lt[4][4]; // i.lt[row][row]; // L_44
      8'd208: i.divisor <= i.lt[5][5]; // i.lt[row][row]; // L_55
      default: i.divisor <= 27'b0;
    endcase
```

```
// LOGIC GOVERNING ARRAY MULT DATAA
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd130: begin
        i.array_mult_dataa[0] <= i.lt[1][0]; // K_10 = K_10 - L_10 * K_00
        i.array_mult_dataa[1] <= i.lt[2][0]; // K_20 = K_20 - L_20 * K_00
        i.array_mult_dataa[2] <= i.lt[3][0]; // K_30 = K_30 - L_30 * K_00
        i.array_mult_dataa[3] <= i.lt[4][0]; // K_40 = K_40 - L_40 * K_00
        i.array_mult_dataa[4] <= i.lt[5][0]; // K_50 = K_50 - L_50 * K_00
      end
      8'd148: begin
        i.array_mult_dataa[0] <= i.lt[2][1]; // K_20 = K_20 - L_21 * K_10
        i.array_mult_dataa[1] <= i.lt[3][1]; // K_30 = K_30 - L_31 * K_10
        i.array_mult_dataa[2] <= i.lt[4][1]; // K_40 = K_40 - L_41 * K_10
        i.array_mult_dataa[3] <= i.lt[5][1]; // K_50 = K_50 - L_51 * K_10
        i.array_mult_dataa[4] <= i.lt[2][1]; // K_21 = K_21 - L_21 * K_11
        i.array_mult_dataa[5] <= i.lt[3][1]; // K_31 = K_31 - L_31 * K_11
        i.array_mult_dataa[6] <= i.lt[4][1]; // K_41 = K_41 - L_41 * K_11
        i.array_mult_dataa[7] <= i.lt[5][1]; // K_51 = K_51 - L_51 * K_11
      end
      8'd166: begin
        i.array_mult_dataa[0] <= i.lt[3][2]; // K_30 = K_30 - L_32 * K_20
        i.array_mult_dataa[1] <= i.lt[4][2]; // K_40 = K_40 - L_42 * K_20
        i.array_mult_dataa[2] <= i.lt[5][2]; // K_50 = K_50 - L_52 * K_20
        i.array_mult_dataa[3] <= i.lt[3][2]; // K_31 = K_31 - L_32 * K_21
        i.array_mult_dataa[4] <= i.lt[4][2]; // K_41 = K_41 - L_42 * K_21
        i.array_mult_dataa[5] <= i.lt[5][2]; // K_51 = K_51 - L_52 * K_21
        i.array_mult_dataa[6] <= i.lt[3][2]; // K_32 = K_32 - L_32 * K_22
        i.array_mult_dataa[7] <= i.lt[4][2]; // K_42 = K_42 - L_42 * K_22
        i.array_mult_dataa[8] <= i.lt[5][2]; // K_52 = K_52 - L_52 * K_22
      end
      8'd184: begin
        i.array_mult_dataa[0] <= i.lt[4][3]; // K_40 = K_40 - L_43 * K_30
        i.array_mult_dataa[1] <= i.lt[5][3]; // K_50 = K_50 - L_53 * K_30
        i.array_mult_dataa[2] <= i.lt[4][3]; // K_41 = K_41 - L_43 * K_31
        i.array_mult_dataa[3] <= i.lt[5][3]; // K_51 = K_51 - L_53 * K_31
        i.array_mult_dataa[4] <= i.lt[4][3]; // K_42 = K_42 - L_43 * K_32
        i.array_mult_dataa[5] <= i.lt[5][3]; // K_52 = K_52 - L_53 * K_32
        i.array_mult_dataa[6] <= i.lt[4][3]; // K_43 = K_43 - L_43 * K_33
        i.array_mult_dataa[7] <= i.lt[5][3]; // K_53 = K_53 - L_53 * K_33
      end
      8'd202: begin
        i.array_mult_dataa[0] <= i.lt[5][4]; // K_50 = K_50 - L_54 * K_40
        i.array_mult_dataa[1] <= i.lt[5][4]; // K_51 = K_51 - L_54 * K_41
        i.array_mult_dataa[2] <= i.lt[5][4]; // K_52 = K_52 - L_54 * K_42
        i.array_mult_dataa[3] <= i.lt[5][4]; // K_53 = K_53 - L_54 * K_43
        i.array_mult_dataa[4] <= i.lt[5][4]; // K_54 = K_54 - L_54 * K_44
      end
      default: begin
        i.array_mult_dataa <= {15{27'b0}};
      end
    endcase

// LOGIC GOVERNING ARRAY MULT DATAB
always_ff @(posedge i.clk)
  if (i.en)
    case (i.count)
      8'd130: begin
        i.array_mult_datab[0] <= i.lt_inverse[0][0]; // K_10 = K_10 - L_10 * K_00
        i.array_mult_datab[1] <= i.lt_inverse[0][0]; // K_20 = K_20 - L_20 * K_00
        i.array_mult_datab[2] <= i.lt_inverse[0][0]; // K_30 = K_30 - L_30 * K_00
        i.array_mult_datab[3] <= i.lt_inverse[0][0]; // K_40 = K_40 - L_40 * K_00
        i.array_mult_datab[4] <= i.lt_inverse[0][0]; // K_50 = K_50 - L_50 * K_00
      end
      8'd148: begin
        i.array_mult_datab[0] <= i.lt_inverse[1][0]; // K_20 = K_20 - L_21 * K_10
```

```
          i.array_mult_datab[1] <= i.lt_inverse[1][0]; // K_30 = K_30 - L_31 * K_10
          i.array_mult_datab[2] <= i.lt_inverse[1][0]; // K_40 = K_40 - L_41 * K_10
          i.array_mult_datab[3] <= i.lt_inverse[1][0]; // K_50 = K_50 - L_51 * K_10
          i.array_mult_datab[4] <= i.lt_inverse[1][1]; // K_21 = K_21 - L_21 * K_11
          i.array_mult_datab[5] <= i.lt_inverse[1][1]; // K_31 = K_31 - L_31 * K_11
          i.array_mult_datab[6] <= i.lt_inverse[1][1]; // K_41 = K_41 - L_41 * K_11
          i.array_mult_datab[7] <= i.lt_inverse[1][1]; // K_51 = K_51 - L_51 * K_11
        end
        8'd166: begin
          i.array_mult_datab[0] <= i.lt_inverse[2][0]; // K_30 = K_30 - L_32 * K_20
          i.array_mult_datab[1] <= i.lt_inverse[2][0]; // K_40 = K_40 - L_42 * K_20
          i.array_mult_datab[2] <= i.lt_inverse[2][0]; // K_50 = K_50 - L_52 * K_20
          i.array_mult_datab[3] <= i.lt_inverse[2][1]; // K_31 = K_31 - L_32 * K_21
          i.array_mult_datab[4] <= i.lt_inverse[2][1]; // K_41 = K_41 - L_42 * K_21
          i.array_mult_datab[5] <= i.lt_inverse[2][1]; // K_51 = K_51 - L_52 * K_21
          i.array_mult_datab[6] <= i.lt_inverse[2][2]; // K_32 = K_32 - L_32 * K_22
          i.array_mult_datab[7] <= i.lt_inverse[2][2]; // K_42 = K_42 - L_42 * K_22
          i.array_mult_datab[8] <= i.lt_inverse[2][2]; // K_52 = K_52 - L_52 * K_22
        end
        8'd184: begin
          i.array_mult_datab[0] <= i.lt_inverse[3][0]; // K_40 = K_40 - L_43 * K_30
          i.array_mult_datab[1] <= i.lt_inverse[3][0]; // K_50 = K_50 - L_53 * K_30
          i.array_mult_datab[2] <= i.lt_inverse[3][1]; // K_41 = K_41 - L_43 * K_31
          i.array_mult_datab[3] <= i.lt_inverse[3][1]; // K_51 = K_51 - L_53 * K_31
          i.array_mult_datab[4] <= i.lt_inverse[3][2]; // K_42 = K_42 - L_43 * K_32
          i.array_mult_datab[5] <= i.lt_inverse[3][2]; // K_52 = K_52 - L_53 * K_32
          i.array_mult_datab[6] <= i.lt_inverse[3][3]; // K_43 = K_43 - L_43 * K_33
          i.array_mult_datab[7] <= i.lt_inverse[3][3]; // K_53 = K_53 - L_53 * K_33
        end
        8'd202: begin
          i.array_mult_datab[0] <= i.lt_inverse[4][0]; // K_50 = K_50 - L_54 * K_40
          i.array_mult_datab[1] <= i.lt_inverse[4][1]; // K_51 = K_51 - L_54 * K_41
          i.array_mult_datab[2] <= i.lt_inverse[4][2]; // K_52 = K_52 - L_54 * K_42
          i.array_mult_datab[3] <= i.lt_inverse[4][3]; // K_53 = K_53 - L_54 * K_43
          i.array_mult_datab[4] <= i.lt_inverse[4][4]; // K_54 = K_54 - L_54 * K_44
        end
        default: begin
          i.array_mult_datab <= {15{27'b0}};
        end
      endcase

  // LOGIC GOVERNING ARRAY DIV QUOTIENTS
  // LOGIC GOVERNING ARRAY MULT RESULT
  always_ff @(posedge i.clk)
    if (i.en)
      case (i.count)
        8'd111: begin
          // INITIALIZE LOWER TRIANGULAR INVERSE MATRIX
          i.lt_inverse <= {36{27'b0}};
        end
        8'd124: begin
          i.lt_inverse[0][0] <= i.quotients[0][26:0]; // K_00 = 1 / L_00
        end
        8'd135: begin
          i.lt_inverse[1][0] <= i.lt_inverse[1][0] - i.array_mult_result[0]; // K_10 = K_10 - L_10 * K_00
          i.lt_inverse[2][0] <= i.lt_inverse[2][0] - i.array_mult_result[1]; // K_20 = K_20 - L_20 * K_00
          i.lt_inverse[3][0] <= i.lt_inverse[3][0] - i.array_mult_result[2]; // K_30 = K_30 - L_30 * K_00
          i.lt_inverse[4][0] <= i.lt_inverse[4][0] - i.array_mult_result[3]; // K_40 = K_40 - L_40 * K_00
          i.lt_inverse[5][0] <= i.lt_inverse[5][0] - i.array_mult_result[4]; // K_50 = K_50 - L_50 * K_00
        end
        8'd142: begin
          i.lt_inverse[1][0] <= i.quotients[0]; // K_10 = K_10 / L_11
          i.lt_inverse[1][1] <= i.quotients[1]; // K_11 = 1 / L_11
```

```
    end
    8'd153: begin
      i.lt_inverse[2][0] <= i.lt_inverse[2][0] - i.array_mult_result[0]; // K_20 = K_20 - L_21 * K_10
      i.lt_inverse[3][0] <= i.lt_inverse[3][0] - i.array_mult_result[1]; // K_30 = K_30 - L_31 * K_10
      i.lt_inverse[4][0] <= i.lt_inverse[4][0] - i.array_mult_result[2]; // K_40 = K_40 - L_41 * K_10
      i.lt_inverse[5][0] <= i.lt_inverse[5][0] - i.array_mult_result[3]; // K_50 = K_50 - L_51 * K_10
      i.lt_inverse[2][1] <= i.lt_inverse[2][1] - i.array_mult_result[4]; // K_21 = K_21 - L_21 * K_11
      i.lt_inverse[3][1] <= i.lt_inverse[3][1] - i.array_mult_result[5]; // K_31 = K_31 - L_31 * K_11
      i.lt_inverse[4][1] <= i.lt_inverse[4][1] - i.array_mult_result[6]; // K_41 = K_41 - L_41 * K_11
      i.lt_inverse[5][1] <= i.lt_inverse[5][1] - i.array_mult_result[7]; // K_51 = K_51 - L_51 * K_11
    end
    8'd160: begin
      i.lt_inverse[2][0] <= i.quotients[0]; // K_20 = K_20 / L_22
      i.lt_inverse[2][1] <= i.quotients[1]; // K_21 = K_21 / L_22
      i.lt_inverse[2][2] <= i.quotients[2]; // K_22 = 1 / L_22
    end
    8'd171: begin
      i.lt_inverse[3][0] <= i.lt_inverse[3][0] - i.array_mult_result[0]; // K_30 = K_30 - L_32 * K_20
      i.lt_inverse[4][0] <= i.lt_inverse[4][0] - i.array_mult_result[1]; // K_40 = K_40 - L_42 * K_20
      i.lt_inverse[5][0] <= i.lt_inverse[5][0] - i.array_mult_result[2]; // K_50 = K_50 - L_52 * K_20
      i.lt_inverse[3][1] <= i.lt_inverse[3][1] - i.array_mult_result[3]; // K_31 = K_31 - L_32 * K_21
      i.lt_inverse[4][1] <= i.lt_inverse[4][1] - i.array_mult_result[4]; // K_41 = K_41 - L_42 * K_21
      i.lt_inverse[5][1] <= i.lt_inverse[5][1] - i.array_mult_result[5]; // K_51 = K_51 - L_52 * K_21
      i.lt_inverse[3][2] <= i.lt_inverse[3][2] - i.array_mult_result[6]; // K_32 = K_32 - L_32 * K_22
      i.lt_inverse[4][2] <= i.lt_inverse[4][2] - i.array_mult_result[7]; // K_42 = K_42 - L_42 * K_22
      i.lt_inverse[5][2] <= i.lt_inverse[5][2] - i.array_mult_result[8]; // K_52 = K_52 - L_52 * K_22
    end
    8'd178: begin
      i.lt_inverse[3][0] <= i.quotients[0]; // K_30 = K_30 / L_33
      i.lt_inverse[3][1] <= i.quotients[1]; // K_31 = K_31 / L_33
      i.lt_inverse[3][2] <= i.quotients[2]; // K_32 = K_32 / L_33
      i.lt_inverse[3][3] <= i.quotients[3]; // K_33 = 1 / L_33
    end
    8'd189: begin
      i.lt_inverse[4][0] <= i.lt_inverse[4][0] - i.array_mult_result[0]; // K_40 = K_40 - L_43 * K_30
      i.lt_inverse[5][0] <= i.lt_inverse[5][0] - i.array_mult_result[1]; // K_50 = K_50 - L_53 * K_30
      i.lt_inverse[4][1] <= i.lt_inverse[4][1] - i.array_mult_result[2]; // K_41 = K_41 - L_43 * K_31
      i.lt_inverse[5][1] <= i.lt_inverse[5][1] - i.array_mult_result[3]; // K_51 = K_51 - L_53 * K_31
      i.lt_inverse[4][2] <= i.lt_inverse[4][2] - i.array_mult_result[4]; // K_42 = K_42 - L_43 * K_32
      i.lt_inverse[5][2] <= i.lt_inverse[5][2] - i.array_mult_result[5]; // K_52 = K_52 - L_53 * K_32
      i.lt_inverse[4][3] <= i.lt_inverse[4][3] - i.array_mult_result[6]; // K_43 = K_43 - L_43 * K_33
      i.lt_inverse[5][3] <= i.lt_inverse[5][3] - i.array_mult_result[7]; // K_53 = K_53 - L_53 * K_33
    end
```

```
        8'd196: begin
          i.lt_inverse[4][0] <= i.quotients[0]; // K_40 = K_40 / L_44
          i.lt_inverse[4][1] <= i.quotients[1]; // K_41 = K_41 / L_44
          i.lt_inverse[4][2] <= i.quotients[2]; // K_42 = K_42 / L_44
          i.lt_inverse[4][3] <= i.quotients[3]; // K_43 = K_43 / L_44
          i.lt_inverse[4][4] <= i.quotients[4]; // K_44 = 1 / L_44
        end
        8'd207: begin
          i.lt_inverse[5][0] <= i.lt_inverse[5][0] - i.array_mult_result[0]; // K_50 = K_50
    - L_54 * K_40
          i.lt_inverse[5][1] <= i.lt_inverse[5][1] - i.array_mult_result[1]; // K_51 = K_51
    - L_54 * K_41
          i.lt_inverse[5][2] <= i.lt_inverse[5][2] - i.array_mult_result[2]; // K_52 = K_52
    - L_54 * K_42
          i.lt_inverse[5][3] <= i.lt_inverse[5][3] - i.array_mult_result[3]; // K_53 = K_53
    - L_54 * K_43
          i.lt_inverse[5][4] <= i.lt_inverse[5][4] - i.array_mult_result[4]; // K_54 = K_54
    - L_54 * K_44
        end
        8'd214: begin
          i.lt_inverse[5][0] <= i.quotients[0]; // K_50 = K_50 / L_55
          i.lt_inverse[5][1] <= i.quotients[1]; // K_51 = K_51 / L_55
          i.lt_inverse[5][2] <= i.quotients[2]; // K_52 = K_52 / L_55
          i.lt_inverse[5][3] <= i.quotients[3]; // K_53 = K_53 / L_55
          i.lt_inverse[5][4] <= i.quotients[4]; // K_54 = K_54 / L_55
          i.lt_inverse[5][5] <= i.quotients[5]; // K_55 = 1 / L_55
        end
        default: begin
        end
      endcase

endmodule
```

../../rtl/ik_swift_32/inverse/lt_block/lt_block.sv

6.2.6 Sine Cosine Functional Units

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_sincos (
  input logic clk
);

logic en, rst;
logic [20:0] angle;
logic [20:0] sin;
logic [20:0] cos;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output angle;
//   input sin;
//   input cos;
// endclocking
//
// modport sincos_tb (clocking cb);

// restrict directions
modport sincos (
  input clk,
  input en,
```

```
  input rst,
  input angle,
  output sin,
  output cos
);

endinterface
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/sincos_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module sincos (
  ifc_sincos.sincos i
);

  // pipeline delay registers for sin
  logic [20:0] angle_delay_1;
  logic [20:0] angle_delay_2;

  always_ff @(posedge i.clk)
    if (i.en) begin
      angle_delay_1 <= i.angle;
      angle_delay_2 <= angle_delay_1;
    end

  sin sin_block (
    .clk ( i.clk ),
    .en ( i.en ),
    .rst ( i.rst ),
    .angle ( angle_delay_2),
    .sin ( i.sin )
  );

  cos cos_block (
    .clk ( i.clk ),
    .en ( i.en ),
    .rst ( i.rst ),
    .angle ( i.angle ),
    .cos ( i.cos )
  );

endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/sincos.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module sin (
  input logic clk, en, rst,
  input logic [20:0] angle,
  output logic [20:0] sin
);

  // x * x
  logic [41:0] angle_2_result;
  logic [20:0] angle_2_round;
  mult_21 angle_2 (
    .clken ( en ),
    .clock ( clk ),
```

```
    .dataa ( angle[20:0] ),
    .datab ( angle[20:0] ),
    .result ( angle_2_result )
  );
  always_ff @(posedge clk)
    if (en)
      angle_2_round <= angle_2_result[15] ? angle_2_result[36:16] + 1'b1 : angle_2_result
    [36:16];

  // 0.405284735 * x * x
  logic [41:0] angle_2_405_result;
  logic [20:0] angle_2_405_round;
  mult_21_coeff_26561 angle_2_405 (
    .clken ( en ),
    .clock ( clk ),
    .dataa ( angle_2_round[20:0] ),
    .result ( angle_2_405_result )
  );
  always_ff @(posedge clk)
    if (en)
      angle_2_405_round <= angle_2_405_result[15] ? angle_2_405_result[36:16] + 1'b1 :
    angle_2_405_result[36:16];

  // 1.27323954 * x
  logic [41:0] angle_1_273_result;
  logic [20:0] angle_1_273_round;
  mult_21_coeff_83443 angle_1_273 (
    .clken ( en ),
    .clock ( clk ),
    .dataa ( angle[20:0] ),
    .result ( angle_1_273_result )
  );
  always_ff @(posedge clk)
    if (en)
      angle_1_273_round <= angle_1_273_result[15] ? angle_1_273_result[36:16] + 1'b1 :
    angle_1_273_result[36:16];

  // pipeline delay registers for est
  logic [20:0] angle_delay_1;
  logic [20:0] angle_delay_2;
  logic [20:0] angle_delay_3;
  logic [20:0] angle_delay_4;
  logic [20:0] angle_delay_5;
  logic [20:0] angle_delay_6;
  logic [20:0] angle_delay_7;
  logic [20:0] angle_delay_8;
  logic [20:0] angle_1_273_round_delay_1;
  logic [20:0] angle_1_273_round_delay_2;
  logic [20:0] angle_1_273_round_delay_3;
  logic [20:0] angle_1_273_round_delay_4;

  // if (x < 0)
  //  est = 1.27323954 * x + 0.405284735 * x * x;
  // else
  //  est = 1.27323954 * x - 0.405284735 * x * x;
  logic [20:0] est;
  always_ff @(posedge clk)
    if (en) begin
      angle_delay_1 <= angle;
      angle_delay_2 <= angle_delay_1;
      angle_delay_3 <= angle_delay_2;
      angle_delay_4 <= angle_delay_3;
      angle_delay_5 <= angle_delay_4;
      angle_delay_6 <= angle_delay_5;
      angle_delay_7 <= angle_delay_6;
      angle_delay_8 <= angle_delay_7;
      angle_1_273_round_delay_1 <= angle_1_273_round;
      angle_1_273_round_delay_2 <= angle_1_273_round_delay_1;
```

```
      angle_1_273_round_delay_3 <= angle_1_273_round_delay_2;
      angle_1_273_round_delay_4 <= angle_1_273_round_delay_3;
      est <= angle_delay_8[20]==1'b1 ? angle_1_273_round_delay_4+angle_2_405_round :
    angle_1_273_round_delay_4-angle_2_405_round; // ask if negative number
    end

  // if (est < 0)
  // est_norm = sin*-sin
  // else
  // est_norm = sin*sin
  logic [41:0] est_2_result;
  logic [20:0] est_2_round;
  logic [20:0] est_2_norm;
  mult_21 est_2 (
    .clken ( en ),
    .clock ( clk ),
    .dataa ( est[20:0] ),
    .datab ( est[20:0] ),
    .result ( est_2_result )
  );

  // pipeline delay registers for est_2_norm
  // pipeline delay registers for est_2_norm_minus_est
  // pipeline delay registers for sin
  logic [10:0] [20:0] est_delay;
  assign est_delay[0] = est;
  always_ff @(posedge clk)
    if (en)
      est_delay[10:1] <= est_delay[9:0];

  always_ff @(posedge clk)
    if (en) begin
      est_2_round <= est_2_result[15] ? est_2_result[36:16] + 1'b1 : est_2_result[36:16];
      est_2_norm <= est_delay[4][20]==1'b1 ? -est_2_round : est_2_round; // ask if negative
    number
    end

  // sin = .225 * (est_2_norm - est) + est;
  logic [20:0] est_2_norm_minus_est;
  always_ff @(posedge clk)
    if (en)
      est_2_norm_minus_est <= est_2_norm - est_delay[5];

  logic [41:0] est_225_result;
  logic [20:0] est_225_round;
  mult_21_coeff_14746 est_225 (
    .clken ( en ),
    .clock ( clk ),
    .dataa ( est_2_norm_minus_est[20:0] ),
    .result ( est_225_result )
  );
  always_ff @(posedge clk)
    if (en)
      est_225_round <= est_225_result[15] ? est_225_result[36:16] + 1'b1 : est_225_result
    [36:16];

  always_ff @(posedge clk)
    if (en)
      sin <= est_225_round + est_delay[10];

endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/sin.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
```

```
 */

module cos (
  input logic clk, en, rst,
  input logic [20:0] angle,
  output logic [20:0] cos
);

  //compute cosine: cos(angle) = sin(angle + PI/2)

  logic [20:0] plus_half_pi; // angle + PI/2
  always_ff @(posedge clk)
    if (en)
      plus_half_pi <= angle + 21'd102944; // angle += 1.57079632;

  logic [20:0] new_angle;
  // if (angle>3.14159265) angle-=6.28318531;
  always_ff @(posedge clk)
    if (en)
      new_angle <= plus_half_pi>21'd205887 && plus_half_pi[20]==1'b0 ? plus_half_pi-21'
    d411775 : plus_half_pi; // also check for positivity

  sin sin (
    .clk ( clk ),
    .en ( en ),
    .rst ( rst ),
    .angle ( new_angle ),
    .sin ( cos )
  );

endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/cos.sv

```
// megafunction wizard: %LPM_MULT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: lpm_mult

// ============================================================
// File Name: mult_21_coeff_26561.v
// Megafunction Name(s):
//      lpm_mult
//
// Simulation Library Files(s):
//      lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.3 Build 178 02/12/2014 SJ Web Edition
// ************************************************************


//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.
```

```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module mult_21_coeff_26561 (
  clken,
  clock,
  dataa,
  result);

  input   clken;
  input   clock;
  input [20:0]  dataa;
  output  [41:0]  result;

  wire [41:0] sub_wire0;
  wire [20:0] sub_wire1 = 21'd26561;
  wire [41:0] result = sub_wire0[41:0];

  lpm_mult  lpm_mult_component (
        .clock (clock),
        .datab (sub_wire1),
        .clken (clken),
        .dataa (dataa),
        .result (sub_wire0),
        .aclr (1'b0),
        .sum (1'b0));
  defparam
    lpm_mult_component.lpm_hint = "DEDICATED_MULTIPLIER_CIRCUITRY=NO,INPUT_B_IS_CONSTANT=YES
    ,MAXIMIZE_SPEED=1",
    lpm_mult_component.lpm_pipeline = 3,
    lpm_mult_component.lpm_representation = "SIGNED",
    lpm_mult_component.lpm_type = "LPM_MULT",
    lpm_mult_component.lpm_widtha = 21,
    lpm_mult_component.lpm_widthb = 21,
    lpm_mult_component.lpm_widthp = 42;


endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21_coeff_26561/mult_21_coeff_26561.v

```
// megafunction wizard: %LPM_MULT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: lpm_mult

// ============================================================
// File Name: mult_21_coeff_83443.v
// Megafunction Name(s):
//      lpm_mult
//
// Simulation Library Files(s):
//      lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.3 Build 178 02/12/2014 SJ Web Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module mult_21_coeff_83443 (
  clken,
  clock,
  dataa,
  result);

  input   clken;
  input   clock;
  input [20:0]  dataa;
  output  [41:0]  result;

  wire [41:0] sub_wire0;
  wire [20:0] sub_wire1 = 21'd83443;
  wire [41:0] result = sub_wire0[41:0];

  lpm_mult  lpm_mult_component (
        .clock (clock),
        .datab (sub_wire1),
        .clken (clken),
        .dataa (dataa),
        .result (sub_wire0),
        .aclr (1'b0),
        .sum (1'b0));
  defparam
    lpm_mult_component.lpm_hint = "DEDICATED_MULTIPLIER_CIRCUITRY=NO,INPUT_B_IS_CONSTANT=YES,MAXIMIZE_SPEED=1",
    lpm_mult_component.lpm_pipeline = 3,
    lpm_mult_component.lpm_representation = "SIGNED",
    lpm_mult_component.lpm_type = "LPM_MULT",
    lpm_mult_component.lpm_widtha = 21,
    lpm_mult_component.lpm_widthb = 21,
    lpm_mult_component.lpm_widthp = 42;


endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21_coeff_83443/mult_21_coeff_83443.v

```
// megafunction wizard: %LPM_MULT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: lpm_mult

// ============================================================
// File Name: mult_21_coeff_14746.v
// Megafunction Name(s):
//      lpm_mult
//
// Simulation Library Files(s):
//      lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
```

```
//
// 13.1.3 Build 178 02/12/2014 SJ Web Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module mult_21_coeff_14746 (
  clken,
  clock,
  dataa,
  result);

  input   clken;
  input   clock;
  input [20:0]  dataa;
  output  [41:0]  result;

  wire [41:0] sub_wire0;
  wire [20:0] sub_wire1 = 21'd14746;
  wire [41:0] result = sub_wire0[41:0];

  lpm_mult  lpm_mult_component (
        .clock (clock),
        .datab (sub_wire1),
        .clken (clken),
        .dataa (dataa),
        .result (sub_wire0),
        .aclr (1'b0),
        .sum (1'b0));
  defparam
    lpm_mult_component.lpm_hint = "DEDICATED_MULTIPLIER_CIRCUITRY=NO,INPUT_B_IS_CONSTANT=YES
    ,MAXIMIZE_SPEED=1",
    lpm_mult_component.lpm_pipeline = 3,
    lpm_mult_component.lpm_representation = "SIGNED",
    lpm_mult_component.lpm_type = "LPM_MULT",
    lpm_mult_component.lpm_widtha = 21,
    lpm_mult_component.lpm_widthb = 21,
    lpm_mult_component.lpm_widthp = 42;


endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21_coeff_14746/mult_21_coeff_14746.v

```
// megafunction wizard: %LPM_MULT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: lpm_mult

// ============================================================
```

```
// File Name: mult_21.v
// Megafunction Name(s):
//      lpm_mult
//
// Simulation Library Files(s):
//      lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.3 Build 178 02/12/2014 SJ Web Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module mult_21 (
  clken,
  clock,
  dataa,
  datab,
  result);

  input   clken;
  input   clock;
  input [20:0]  dataa;
  input [20:0]  datab;
  output  [41:0]  result;

  wire [41:0] sub_wire0;
  wire [41:0] result = sub_wire0[41:0];

  lpm_mult  lpm_mult_component (
        .clock (clock),
        .datab (datab),
        .clken (clken),
        .dataa (dataa),
        .result (sub_wire0),
        .aclr (1'b0),
        .sum (1'b0));
  defparam
    lpm_mult_component.lpm_hint = "DEDICATED_MULTIPLIER_CIRCUITRY=YES,MAXIMIZE_SPEED=1",
    lpm_mult_component.lpm_pipeline = 3,
    lpm_mult_component.lpm_representation = "SIGNED",
    lpm_mult_component.lpm_type = "LPM_MULT",
    lpm_mult_component.lpm_widtha = 21,
    lpm_mult_component.lpm_widthb = 21,
    lpm_mult_component.lpm_widthp = 42;


endmodule
```

../../rtl/ik_swift_32/full_jacobian/full_mat/t_block/sincos/mult_21/mult_21.v

6.2.7 Arithmetic Functional Units

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_mat_mult (
  input logic clk
);

parameter n = 6;

logic en, rst, mat_mode;
logic [5:0] [5:0] [26:0] dataa;
logic [5:0] [5:0] [26:0] datab;
logic [5:0] [5:0] [26:0] result;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output mat_mode;
//   output dataa;
//   output datab;
//
//   input result;
// endclocking
//
// modport mat_mult_tb (clocking cb);

// restrict directions
modport mat_mult (
  input clk,
  input en,
  input rst,
  input mat_mode,

  input dataa,
  input datab,

  output result
);

endinterface
```

../../rtl/ik_swift_32/mat_mult/mat_mult_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module mat_mult (
  ifc_mat_mult.mat_mult i
);

  parameter n = 6;

  logic [n-1:0] [n-1:0] [26:0] mult_array_dataa;
  logic [n-1:0] [n-1:0] [26:0] mult_array_datab;
  logic [n-1:0] [n-1:0] [26:0] mult_array_result;
```

```
mult_array #(n) mult_array (
  .clk(i.clk),
  .en(i.en),
  .dataa(mult_array_dataa),
  .datab(mult_array_datab),
  .result(mult_array_result)
);

// LOGIC GOVERNING COUNT
logic [3:0] count;
always_ff @(posedge i.clk) begin
  if ( i.rst || !i.mat_mode ) begin // if parallel multiplier mode, clear counter
    count <= 0;
  end else if ( i.en && i.mat_mode ) begin
    if ( count==n-1'b1 ) begin
      count <= 0;
    end else begin
      count <= count + 1'b1;
    end
  end
end

// LOGIC GOVERNING MULT_ARRAY_DATAA/B
always_ff @(posedge i.clk)
  if (i.en)
    case(i.mat_mode)
      1'b0: begin // parallel multiplier mode
        mult_array_dataa <= i.dataa;
        mult_array_datab <= i.datab;
      end
      1'b1: begin // matrix multiplier mode
        case(count)
          4'd0: begin
            mult_array_dataa <= {{6{i.dataa[5][5]}}, {6{i.dataa[4][5]}}, {6{i.dataa
  [3][5]}}, {6{i.dataa[2][5]}}, {6{i.dataa[1][5]}}, {6{i.dataa[0][5]}}};
            mult_array_datab <= {6{i.datab[5][5], i.datab[5][4], i.datab[5][3], i.datab
  [5][2], i.datab[5][1], i.datab[5][0]}};
          end
          4'd1: begin
            mult_array_dataa <= {{6{i.dataa[5][4]}}, {6{i.dataa[4][4]}}, {6{i.dataa
  [3][4]}}, {6{i.dataa[2][4]}}, {6{i.dataa[1][4]}}, {6{i.dataa[0][4]}}};
            mult_array_datab <= {6{i.datab[4][5], i.datab[4][4], i.datab[4][3], i.datab
  [4][2], i.datab[4][1], i.datab[4][0]}};
          end
          4'd2: begin
            mult_array_dataa <= {{6{i.dataa[5][3]}}, {6{i.dataa[4][3]}}, {6{i.dataa
  [3][3]}}, {6{i.dataa[2][3]}}, {6{i.dataa[1][3]}}, {6{i.dataa[0][3]}}};
            mult_array_datab <= {6{i.datab[3][5], i.datab[3][4], i.datab[3][3], i.datab
  [3][2], i.datab[3][1], i.datab[3][0]}};
          end
          4'd3: begin
            mult_array_dataa <= {{6{i.dataa[5][2]}}, {6{i.dataa[4][2]}}, {6{i.dataa
  [3][2]}}, {6{i.dataa[2][2]}}, {6{i.dataa[1][2]}}, {6{i.dataa[0][2]}}};
            mult_array_datab <= {6{i.datab[2][5], i.datab[2][4], i.datab[2][3], i.datab
  [2][2], i.datab[2][1], i.datab[2][0]}};
          end
          4'd4: begin
            mult_array_dataa <= {{6{i.dataa[5][1]}}, {6{i.dataa[4][1]}}, {6{i.dataa
  [3][1]}}, {6{i.dataa[2][1]}}, {6{i.dataa[1][1]}}, {6{i.dataa[0][1]}}};
            mult_array_datab <= {6{i.datab[1][5], i.datab[1][4], i.datab[1][3], i.datab
  [1][2], i.datab[1][1], i.datab[1][0]}};
          end
          4'd5: begin
            mult_array_dataa <= {{6{i.dataa[5][0]}}, {6{i.dataa[4][0]}}, {6{i.dataa
  [3][0]}}, {6{i.dataa[2][0]}}, {6{i.dataa[1][0]}}, {6{i.dataa[0][0]}}};
            mult_array_datab <= {6{i.datab[0][5], i.datab[0][4], i.datab[0][3], i.datab
  [0][2], i.datab[0][1], i.datab[0][0]}};
```

```
            end
            default: begin
              mult_array_dataa <= {36{27'b0}};
              mult_array_datab <= {36{27'b0}};
            end
          endcase
        end
        default: begin
          mult_array_dataa <= {36{27'b0}};
          mult_array_datab <= {36{27'b0}};
        end
      endcase

  // LOGIC GOVERNING RESULT
  genvar index, jndex;
  generate
    for ( index=n-1 ; index>=0 ; index-- ) begin: adder_row
      for ( jndex=n-1 ; jndex>=0 ; jndex-- ) begin: adder_col
        always_ff @(posedge i.clk) begin
          if ( i.rst ) begin
            i.result[index][jndex] <= 27'b0;
          end else if (i.en) begin
            if ( i.mat_mode ) begin // matrix multiplier mode
              if ( i.en && (count==4'd5) ) begin
                i.result[index][jndex] <= mult_array_result[index][jndex];
              end
              if ( i.en && (count==4'd0 || count==4'd1 || count==4'd2 || count==4'd3 ||
    count==4'd4) ) begin
                i.result[index][jndex] <= i.result[index][jndex] + mult_array_result[index][
    jndex]; // accumulate
              end
            end else begin // parallel multiplier mode
              i.result[index][jndex] <= mult_array_result[index][jndex];
            end // end parallel multiplier mode
          end // end not reset
        end // end always_ff
      end // end col loop
    end // end row loop
  endgenerate

endmodule
```

../../rtl/ik_swift_32/mat_mult/mat_mult.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module mult_array (
  clk, en,
  dataa,
  datab,
  result
);

  parameter n = 6;

  input clk, en;
  input [n-1:0] [n-1:0] [26:0] dataa;
  input [n-1:0] [n-1:0] [26:0] datab;
  output [n-1:0] [n-1:0] [26:0] result;

  logic [n-1:0] [n-1:0] [53:0] mult_result;
  logic [n-1:0] [n-1:0] [26:0] mult_round;

  genvar i, j;
```

```
  generate
    for ( i=n-1 ; i>=0 ; i-- ) begin: mult_27_row
      for ( j=n-1 ; j>=0 ; j-- ) begin: mult_27_col
        mult_27 mult_27_inst (
          .clken(en),
          .clock(clk),
          .dataa(dataa[i][j]),
          .datab(datab[i][j]),
          .result(mult_result[i][j])
        );
        always_ff @(posedge clk)
          if (en)
            mult_round[i][j] <= mult_result[i][j][15] ? mult_result[i][j][42:16] + 1'b1 :
    mult_result[i][j][42:16];
      end
    end
  endgenerate

  assign result = mult_round;

endmodule
```

../../rtl/ik_swift_32/mat_mult/mult_array.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_array_mult (
  input logic clk
);

parameter n = 15;

logic en, rst;
logic [n-1:0] [26:0] dataa;
logic [n-1:0] [26:0] datab;
logic [n-1:0] [26:0] result;

// clocking cb @(posedge clk);
//  output en;
//  output rst;
//  output dataa;
//  output datab;
//
//  input result;
// endclocking
//
// modport array_mult_tb (clocking cb);

// restrict directions
modport array_mult (
  input clk,
  input en,
  input rst,

  input dataa,
  input datab,

  output result
);

endinterface
```

../../rtl/ik_swift_32/array_mult/array_mult_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module array_mult (
  ifc_array_mult.array_mult i
);

  parameter n = 15;

  logic [n-1:0] [53:0] mult_result;
  logic [n-1:0] [26:0] mult_round;

  genvar index;
  generate
    for ( index=n-1 ; index>=0 ; index-- ) begin: mult_27_row
      mult_27 mult_27_inst (
        .clken(i.en),
        .clock(i.clk),
        .dataa(i.dataa[index]),
        .datab(i.datab[index]),
        .result(mult_result[index])
      );
      always_ff @(posedge i.clk)
        if (i.en)
          mult_round[index] <= mult_result[index][15] ? mult_result[index][42:16] + 1'b1 :
    mult_result[index][42:16];
    end
  endgenerate

  assign i.result = mult_round;

endmodule
```

../../rtl/ik_swift_32/array_mult/array_mult.sv

```
// megafunction wizard: %LPM_MULT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: lpm_mult

// ============================================================
// File Name: mult_27.v
// Megafunction Name(s):
//      lpm_mult
//
// Simulation Library Files(s):
//      lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.3 Build 178 02/12/2014 SJ Web Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module mult_27 (
  clken,
  clock,
  dataa,
  datab,
  result);

  input   clken;
  input   clock;
  input  [26:0]  dataa;
  input  [26:0]  datab;
  output  [53:0]  result;

  wire [53:0] sub_wire0;
  wire [53:0] result = sub_wire0[53:0];

  lpm_mult  lpm_mult_component (
        .clock (clock),
        .datab (datab),
        .clken (clken),
        .dataa (dataa),
        .result (sub_wire0),
        .aclr (1'b0),
        .sum (1'b0));
  defparam
    lpm_mult_component.lpm_hint = "DEDICATED_MULTIPLIER_CIRCUITRY=YES,MAXIMIZE_SPEED=1",
    lpm_mult_component.lpm_pipeline = 3,
    lpm_mult_component.lpm_representation = "SIGNED",
    lpm_mult_component.lpm_type = "LPM_MULT",
    lpm_mult_component.lpm_widtha = 27,
    lpm_mult_component.lpm_widthb = 27,
    lpm_mult_component.lpm_widthp = 54;


endmodule
```

../../rtl/ik_swift_32/mult_27/mult_27.v

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_sqrt_43 (
  input logic clk
);

logic en, rst;
logic [26:0] radical;
logic [26:0] q;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//
//   output radical;
//   input q;
// endclocking
//
```

```
// modport sqrt_43_tb (clocking cb);

// restrict directions
modport sqrt_43 (
  input clk,
  input en,
  input rst,

  input radical,
  output q
);

endinterface
```

../../rtl/ik_swift_32/inverse/cholesky_block/sqrt_43/sqrt_43_interface.sv

```
// megafunction wizard: %ALTSQRT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: ALTSQRT

// ============================================================
// File Name: sqrt_43.v
// Megafunction Name(s):
//      ALTSQRT
//
// Simulation Library Files(s):
//      altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.4 Build 182 03/12/2014 SJ Web Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module sqrt_43 (
  clk,
  ena,
  radical,
  q,
  remainder);

  input   clk;
  input   ena;
  input [42:0]  radical;
  output   [21:0]  q;
  output   [22:0]  remainder;

```

```
  wire [21:0] sub_wire0;
  wire [22:0] sub_wire1;
  wire [21:0] q = sub_wire0[21:0];
  wire [22:0] remainder = sub_wire1[22:0];

  altsqrt ALTSQRT_component (
        .clk (clk),
        .ena (ena),
        .radical (radical),
        .q (sub_wire0),
        .remainder (sub_wire1)
        // synopsys translate_off
        ,
        .aclr ()
        // synopsys translate_on
        );
  defparam
    ALTSQRT_component.pipeline = 5,
    ALTSQRT_component.q_port_width = 22,
    ALTSQRT_component.r_port_width = 23,
    ALTSQRT_component.width = 43;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: PIPELINE NUMERIC "5"
// Retrieval info: CONSTANT: Q_PORT_WIDTH NUMERIC "22"
// Retrieval info: CONSTANT: R_PORT_WIDTH NUMERIC "23"
// Retrieval info: CONSTANT: WIDTH NUMERIC "43"
// Retrieval info: USED_PORT: clk 0 0 0 0 INPUT NODEFVAL "clk"
// Retrieval info: USED_PORT: ena 0 0 0 0 INPUT NODEFVAL "ena"
// Retrieval info: USED_PORT: q 0 0 22 0 OUTPUT NODEFVAL "q[21..0]"
// Retrieval info: USED_PORT: radical 0 0 43 0 INPUT NODEFVAL "radical[42..0]"
// Retrieval info: USED_PORT: remainder 0 0 23 0 OUTPUT NODEFVAL "remainder[22..0]"
// Retrieval info: CONNECT: @clk 0 0 0 0 clk 0 0 0 0
// Retrieval info: CONNECT: @ena 0 0 0 0 ena 0 0 0 0
// Retrieval info: CONNECT: @radical 0 0 43 0 radical 0 0 43 0
// Retrieval info: CONNECT: q 0 0 22 0 @q 0 0 22 0
// Retrieval info: CONNECT: remainder 0 0 23 0 @remainder 0 0 23 0
// Retrieval info: GEN_FILE: TYPE_NORMAL sqrt_43.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL sqrt_43.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL sqrt_43.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL sqrt_43.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL sqrt_43_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL sqrt_43_bb.v FALSE
// Retrieval info: LIB_FILE: altera_mf
```

../../rtl/ik_swift_32/inverse/cholesky_block/sqrt_43/sqrt_43.v

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

interface ifc_array_div (
  input logic clk
);

parameter n = 6;

logic en, rst;
```

```
logic [n-1:0] [26:0] dividends;
logic [26:0] divisor;
logic [n-1:0] [26:0] quotients;

// clocking cb @(posedge clk);
//   output en;
//   output rst;
//   output dividends;
//   output divisor;
//
//   input quotients;
// endclocking
//
// modport array_div_tb (clocking cb);

// restrict directions
modport array_div (
  input clk,
  input en,
  input rst,

  input dividends,
  input divisor,

  output quotients
);

endinterface
```

../../rtl/ik_swift_32/inverse/array_div/array_div_interface.sv

```
/*
 * Yipeng Huang, Richard Townsend, Lianne Lairmore
 * Columbia University
 */

module array_div (
  ifc_array_div.array_div i
);

  parameter n = 6;

  logic [n-1:0] [42:0] quotient;
  logic [n-1:0] [26:0] remain;

  genvar index;
  generate
    for ( index=n-1 ; index>=0 ; index-- ) begin: div_43_row
      div_43  div_43_inst (
        .clken( i.en ),
        .clock( i.clk ),
        .denom ( i.divisor ),
        .numer ( {i.dividends[index],16'b0} ),
        .quotient ( quotient[index] ),
        .remain ( remain[index] )
        );
      assign i.quotients[index] = quotient[index][26:0];
    end
  endgenerate

endmodule
```

../../rtl/ik_swift_32/inverse/array_div/array_div.sv

```
// megafunction wizard: %LPM_DIVIDE%
// GENERATION: STANDARD
```

```
// VERSION: WM1.0
// MODULE: LPM_DIVIDE

// ============================================================
// File Name: div_43.v
// Megafunction Name(s):
//      LPM_DIVIDE
//
// Simulation Library Files(s):
//      lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.3 Build 178 02/12/2014 SJ Web Edition
// ************************************************************


//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.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module div_43 (
  clken,
  clock,
  denom,
  numer,
  quotient,
  remain);

  input   clken;
  input   clock;
  input [26:0]  denom;
  input [42:0]  numer;
  output  [42:0]  quotient;
  output  [26:0]  remain;

  wire [26:0] sub_wire0;
  wire [42:0] sub_wire1;
  wire [26:0] remain = sub_wire0[26:0];
  wire [42:0] quotient = sub_wire1[42:0];

  lpm_divide  LPM_DIVIDE_component (
        .clock (clock),
        .clken (clken),
        .denom (denom),
        .numer (numer),
        .remain (sub_wire0),
        .quotient (sub_wire1),
        .aclr (1'b0));
  defparam
    LPM_DIVIDE_component.lpm_drepresentation = "SIGNED",
    LPM_DIVIDE_component.lpm_hint = "MAXIMIZE_SPEED=6,LPM_REMAINDERPOSITIVE=FALSE",
    LPM_DIVIDE_component.lpm_nrepresentation = "SIGNED",
```

```
    LPM_DIVIDE_component.lpm_pipeline = 5,
    LPM_DIVIDE_component.lpm_type = "LPM_DIVIDE",
    LPM_DIVIDE_component.lpm_widthd = 27,
    LPM_DIVIDE_component.lpm_widthn = 43;


endmodule
```

../../rtl/ik_swift_32/inverse/array_div/div_43/div_43.v

7 Appendix: FPGA Utilization Estimate

In order to estimate the area and timing costs of the accelerator, we created a table enumerating the costs of each of the custom functional units and submodules we proposed, in terms of the pipeline depth of the block, and the DSP, lookup table, and register costs of the block. These figures are based on estimates given by the Altera MegaFunction User Manual and by generating the IP designs in Quartus MegaWizard. These estimates are shown in Figure 10.

References

[1] S. Caselli, E. Faldella, and F. Zanichelli, "Performance evaluation of processor architectures for robotics," in *CompEuro '91. Advanced Computer Technology, Reliable Systems and Applications. 5th Annual European Computer Conference. Proceedings.*, pp. 667–671, May 1991.

[2] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Commun. ACM*, vol. 52, pp. 56–67, Oct. 2009.

[3] kirillv@github.com, "cpp-inverse-kinematics-library,"

[4] "Introduction to homogeneous transformations & robot kinematics,"

[5] H. H. Asada, "Introduction to robotics chapter 5 differential motion,"

[6] M. Baczynski, "Fast and accurate sine/cosine approximation,"

	count	precision	delay	DSP	% DSP Use	ALM	% ALM Use	Regs	18 x 18 Mult
Functional Units									
LPM_ADD_SUB adder subtractor	1	20-bit	2	0	0%	11	0%	0	0
LPM_MULT square multiplier	1	16-bit	3	0	0%	146	0%	151	0
LPM_MULT variable multiplier	1	16-bit	3	2	2%	294	1%	274	2
LPM_DIVIDE divider	1	30-bit	5	0	0%	642	2%	0	0
ALTSQRT square root	1	20-bit	5	0	0%	94	0%	0	0
sincos									
LPM_MULT square multiplier	3	16-bit		0	0%	438	1%	453	0
LPM_MULT variable multiplier	7	16-bit		14	13%	2058	5%	1918	14
coefficient multiplier	10	16-bit		20	18%	2940	7%	2740	20
subtotal	1	16-bit	20	34	30%	5436	13%	5111	34
4x4 matrix multiplier									
LPM_MULT variable multiplier	16	16-bit		32	29%	4704	11%	4384	32
LPM_ADD_SUB adder subtractor	16	20-bit		0	0%	176	0%	0	0
subtotal	1	16-bit	14	32	29%	4880	12%	4384	32
3x1 vector vector cross product									
LPM_MULT variable multiplier	6	16-bit		12	11%	1764	4%	1644	12
LPM_ADD_SUB adder subtractor	3	20-bit		0	0%	33	0%	0	0
subtotal	1	16-bit	5	12	11%	1797	4%	1644	12
6x6 6x1 matrix vector multiplier									
LPM_MULT variable multiplier	6	16-bit		12	11%	1764	4%	1644	12
LPM_ADD_SUB adder subtractor	6	20-bit		0	0%	66	0%	0	0
subtotal	1	16-bit	20	12	11%	1830	4%	1644	12
6x6 matrix multiplier									
LPM_MULT variable multiplier	36	16-bit		72	64%	10584	25%	9864	72
LPM_ADD_SUB adder subtractor	36	20-bit		0	0%	396	1%	0	0
subtotal	1	16-bit	20	72	64%	10980	26%	9864	72
6x6 cholesky decomposition									
ALTSQRT square root	1	20-bit		0	0%	94	0%	0	0
LPM_DIVIDE divider	5	30-bit		0	0%	3210	8%	0	0
LPM_MULT variable multiplier	25	16-bit		50	45%	7350	18%	6850	50
LPM_ADD_SUB adder subtractor	25	20-bit		0	0%	275	1%	0	0
subtotal	1	16-bit	80	50	45%	10929	26%	6850	50
6x6 lower triangular matrix inversion									
LPM_DIVIDE divider	6	30-bit		0	0%	3852	9%	0	0
LPM_MULT variable multiplier	9	16-bit		18	16%	2646	6%	2466	18
LPM_ADD_SUB adder subtractor	6	20-bit		0	0%	66	0%	0	0
subtotal	1	16-bit	50	18	16%	6564	16%	2466	18
D-H Transformation Block									
sincos	2	16-bit		68	61%	10872	26%	10222	68
LPM_MULT variable multiplier	6	16-bit		12	11%	1764	4%	1644	12
subtotal			23	80	71%	12636	30%	11866	80
6 Degree of Freedom Full Matrix Block									
D-H Transformation Block	1	16-bit		68	61%	12636	30%	11866	80
4x4 matrix multiplier	1	16-bit		32	29%	4880	12%	4384	32
subtotal			152	32	29%	17516	42%	16250	112
Jacobian Block									
LPM_ADD_SUB adder subtractor	3	20-bit		0	0%	33	0%	0	0
3x1 vector vector cross product	1	16-bit		12	11%	1797	4%	1644	12
subtotal	1	16-bit	7	12	11%	1830	4%	1644	12
Damped Least Squares Block									
6x6 matrix multiplier	1	16-bit		72	64%	10980	26%	9864	72
LPM_ADD_SUB adder subtractor	6	20-bit		0	0%	66	0%	0	0
6x6 cholesky decomposition	1	16-bit		50	45%	10929	26%	6850	50
6x6 lower triangular matrix inversion	1	16-bit		18	16%	6564	16%	2466	18
6x6 6x1 matrix vector multiplier	1	16-bit		12	11%	1830	4%	1644	12
subtotal	1	16-bit	212	152	136%	30369	73%	20824	152
GRAND TOTAL									
				196	175%	49715	120%	38718	276
FGPA RESOURCES									
Cyclone V SX C6 (5CSXFC6D6F31)				112	100%	41509	100%	166036	224

Figure 10: An estimate of area and timing costs of the accelerator submodules and custom functional units.