

1. Description of design approach

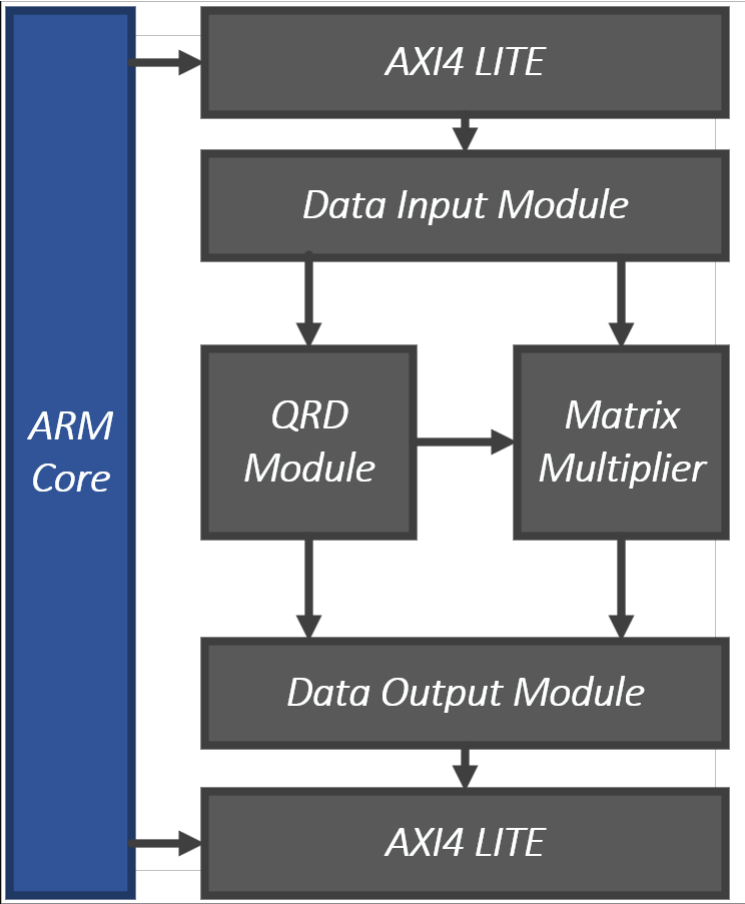
This design is used for the continuous solution of quaternary linear equations. This design uses both ARM and FPGA to calculate linear equations, and AXI4 Lite interface is used between ARM core and FPGA to communicate.

Its workflow is as follows:

- 1. **[ARM]**A and b of $Ax = b$ are obtained in the ARM core, and are transmitted to the FPGA through the AXI Lite interface.
- 2. **[FPGA]**Doing QR Decomposition in FPGA.
- 3. **[FPGA]**Multiply the resulting Q transpose with b.
- 4. **[FPGA]**Pass Q transpose * b and R back to the ARM core through the AXI Lite interface.
- 5. **[ARM]**Do the back-substitution in ARM to get the final result and print(if needed).

• Overall block diagram

The overall design block diagram is shown in the figure below, which is mainly composed of ARM core, AXI Lite interface, QR decomposition module, matrix multiplier and some additional input and output controls.



The design method of each module in the block diagram is as follows.

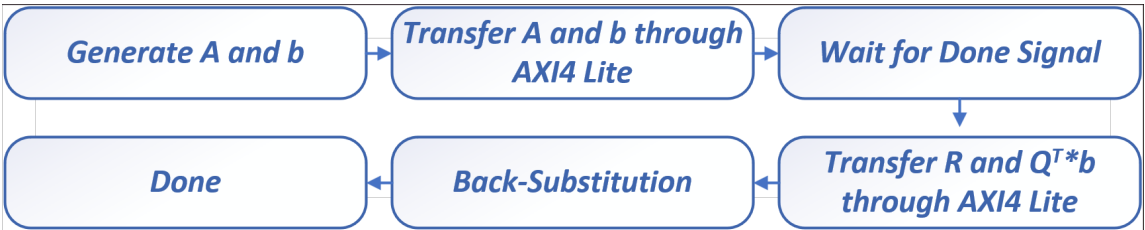
- **Input matrices and vectors using AXI Lite**
Implemented in ARM Vitis IDE
- **Achieve QRD with Cordic Rotation Blocks**
Cordic algorithm designed in Vitis HLS and QRD module built in Model Composer
- **Calculate $Q \times b$ using Matrix Multiplier**
**Matrix Multiplier designed in Vitis HLS and from Project 3 but maintain systolic*
- **Complete equations solution in ARM**
Retrieve data using AXI Lite and Completing back-substitution in ARM

• ARM Core

The ARM core code is divided into two versions, one uses FPGA for hardware acceleration, and the other only uses ARM core for calculation (using the built-in trigonometric functions in the cmath header file).

According to the test results, the calculation speed of FPGA is about **10.942 times** of the calculation speed of ARM core.

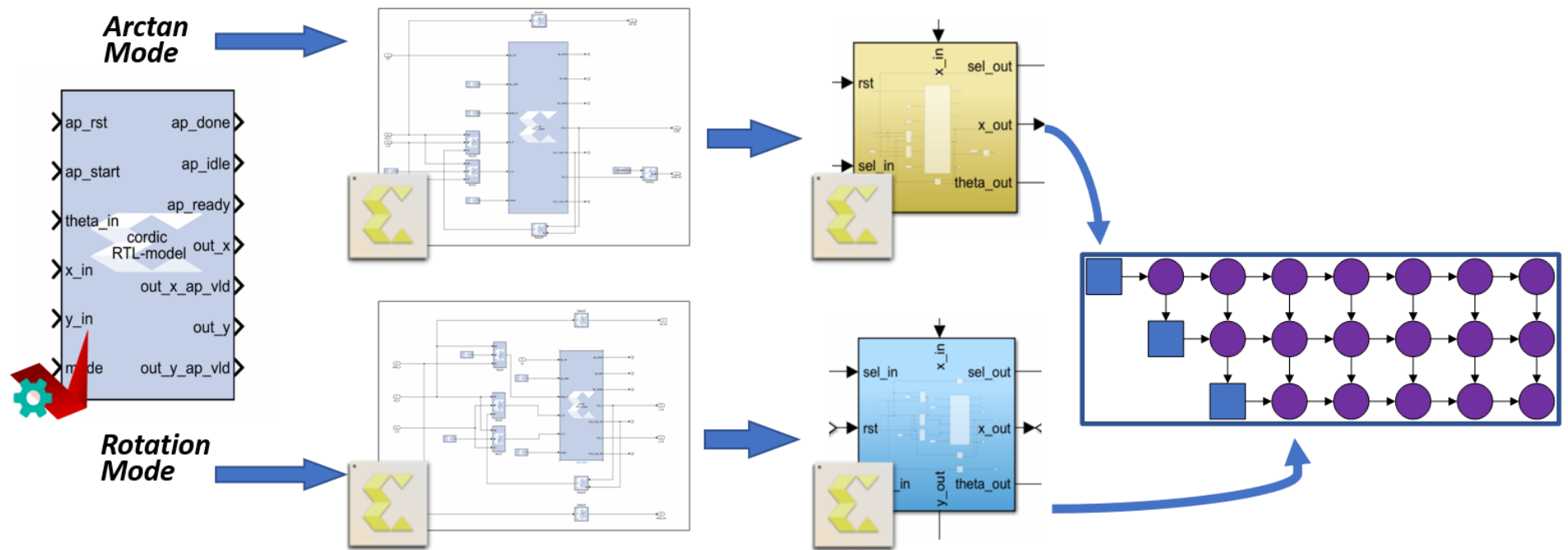
- Workflow of ARM Core program



• Cordic & QRD

The modules of QR decomposition are built using Cordic modules. It uses the second method in Lecture_W10_L1.pdf.

- Build QRD module using Cordic blocks
 - Get vector length and angle using arctan mode
 - Rotate vector by angle with rotate mode



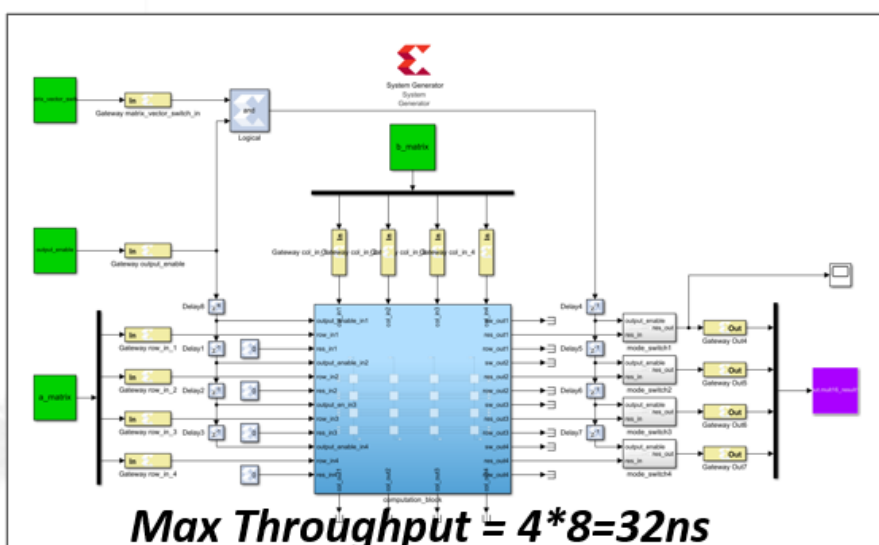
Matrix Multiplier

Both Project2 and Project3 can complete matrix multiplication, but after comparison, it is found that the Project3 solution has more advantages in terms of resource occupation and timing, so this project uses the Project3 solution for construction.

The advantages of the Project3 solution are as follows:

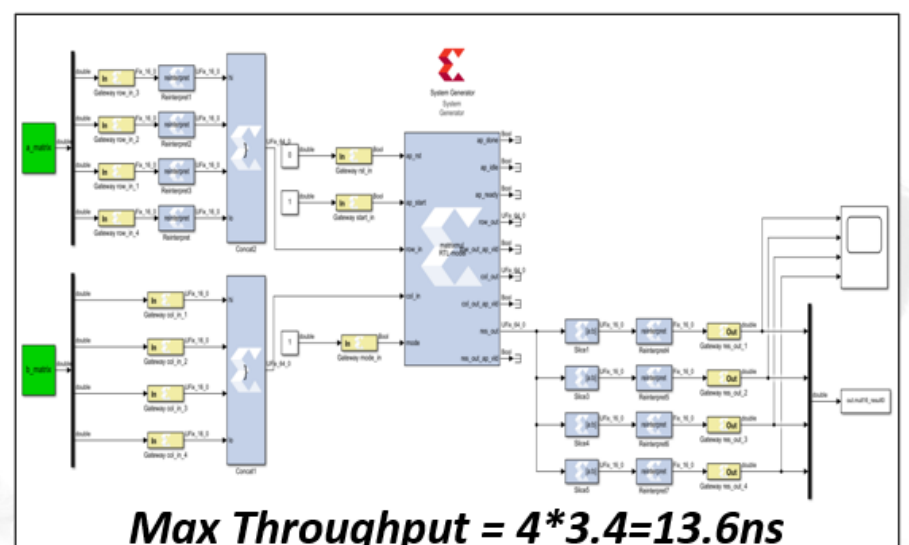
- Saving time/Higher throughput
- Saving resources
- Maintain systolic structure

Project2 Design

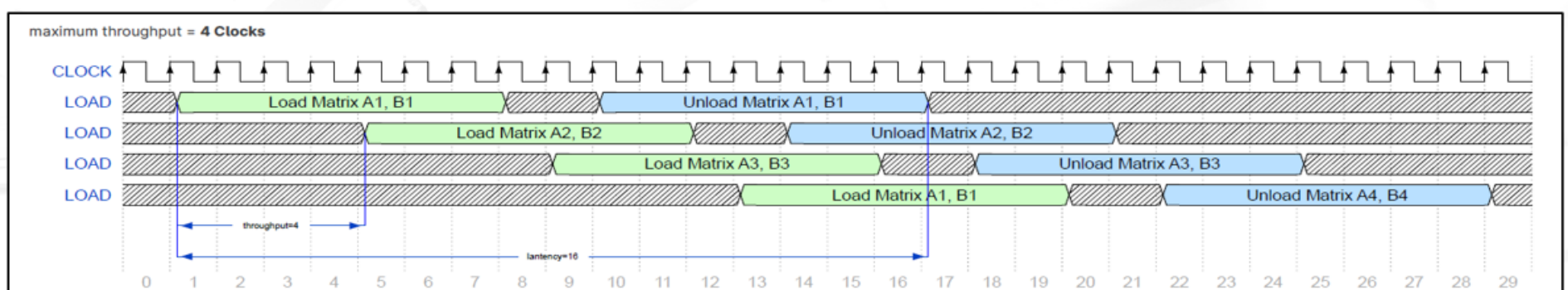


Resource Type	Usage
DSPs	16
LUTs	2413
Registers	3187

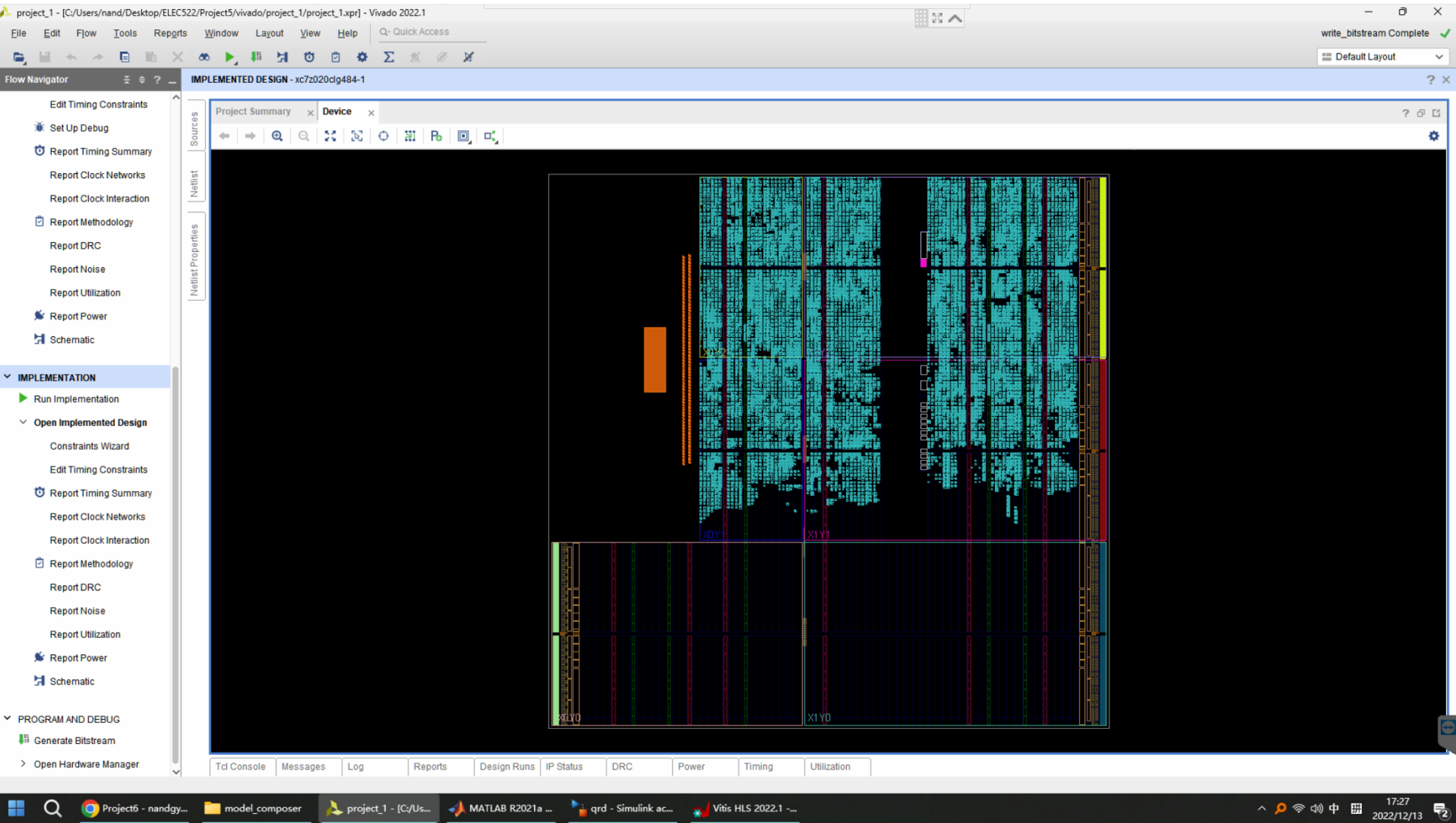
Project3 Design ✓



Resource Type	Usage
DSPs	16
LUTs	919
Registers	2170



Implemented Design

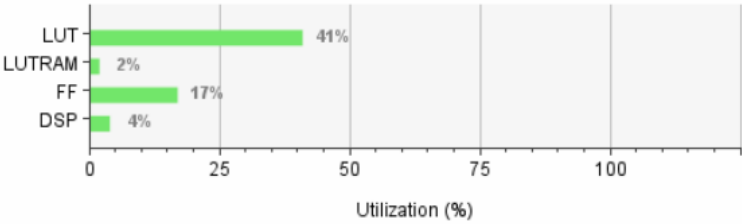


Resources Utilization

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	DSPs (220)	Bonded IOPADs (130)	BUFGCTRL (32)
design_1_wrapper	21851	17718	168	72	6284	21469	382	8	130	1
design_1_i (design_1)	21851	17718	168	72	6284	21469	382	8	0	1
processing_system7_0	0	0	0	0	0	0	0	0	0	1
ps7_0_axi_periph (design_1_ps7_0_axi_periph)	365	426	0	0	152	306	59	0	0	0
qrd_0 (design_1_qrd_0)	21469	17259	168	72	6151	21147	322	8	0	0
rst_ps7_0_100M (design_1_rst_ps7_0_100M)	17	33	0	0	10	16	1	0	0	0

Summary

Resource	Utilization	Available	Utilization %
LUT	21851	53200	41.07
LUTRAM	382	17400	2.20
FF	17718	106400	16.65
DSP	8	220	3.64



Timing Report

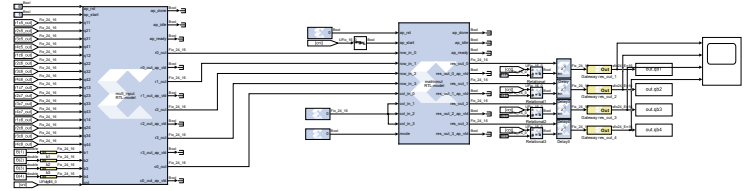
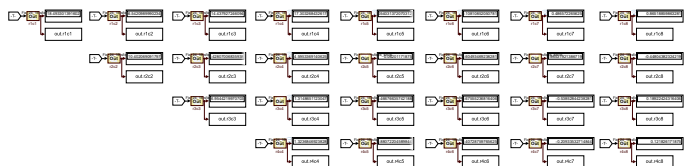
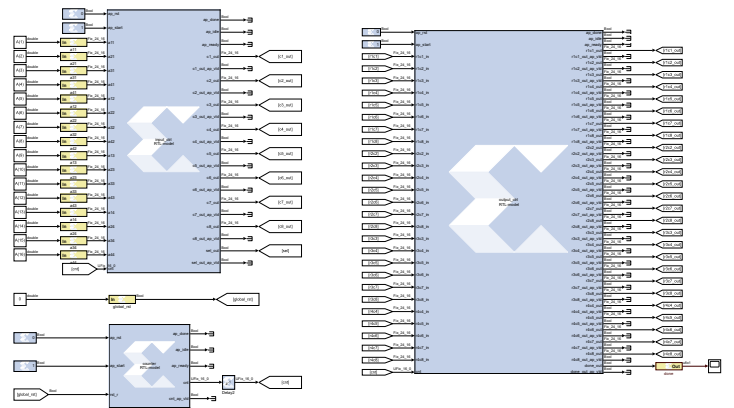
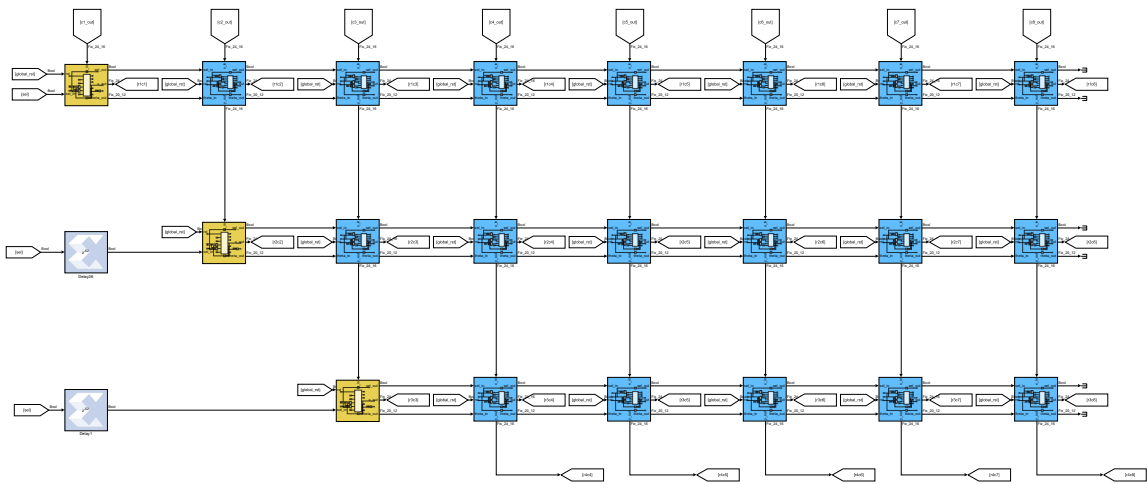
Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.419 ns	Worst Hold Slack (WHS): 0.011 ns	Worst Pulse Width Slack (WPWS): 4.020 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 34424	Total Number of Endpoints: 34424	Total Number of Endpoints: 18113
All user specified timing constraints are met.		

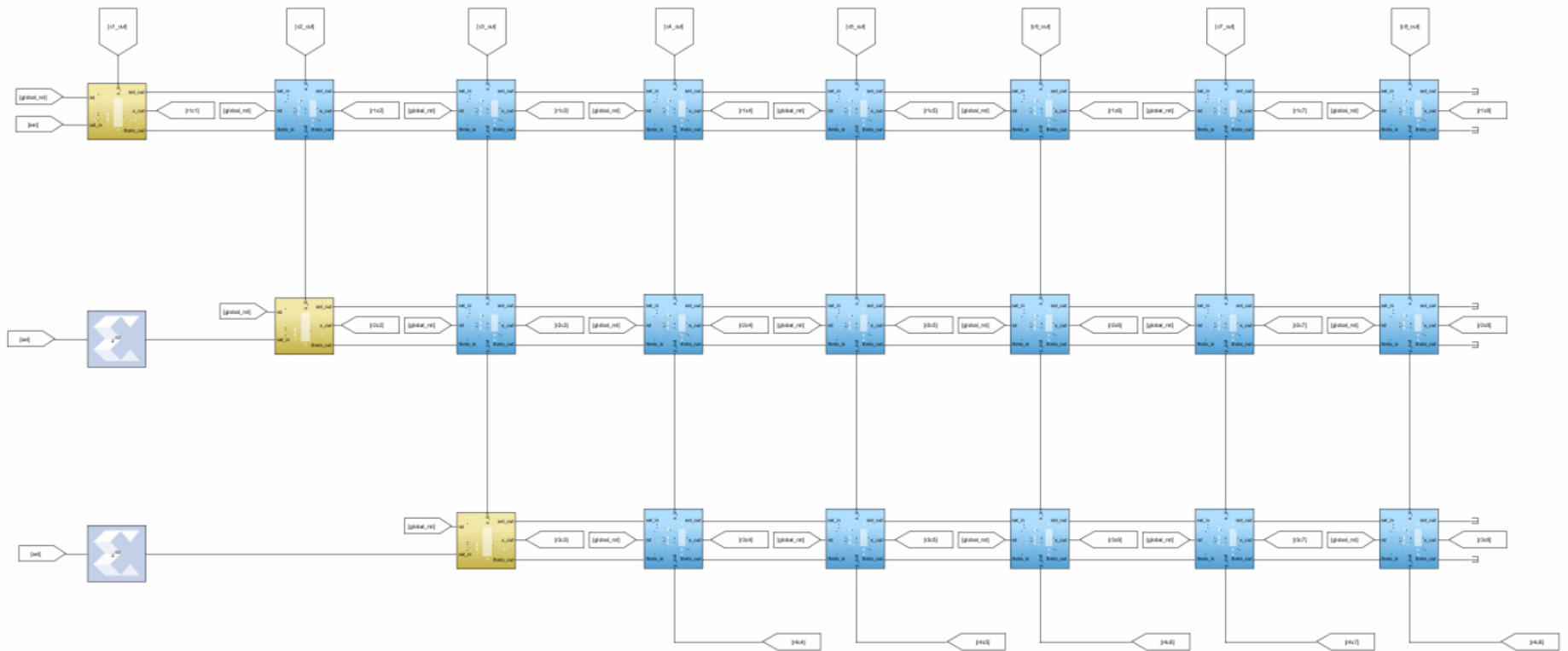
I would have liked to use the AXI FIFO and AXI Stream interface for speed, but I'm having issues with data alignment so haven't gotten around to it yet.

2. Model Composer integration of QR Decomposition and matrix-vector multiplication

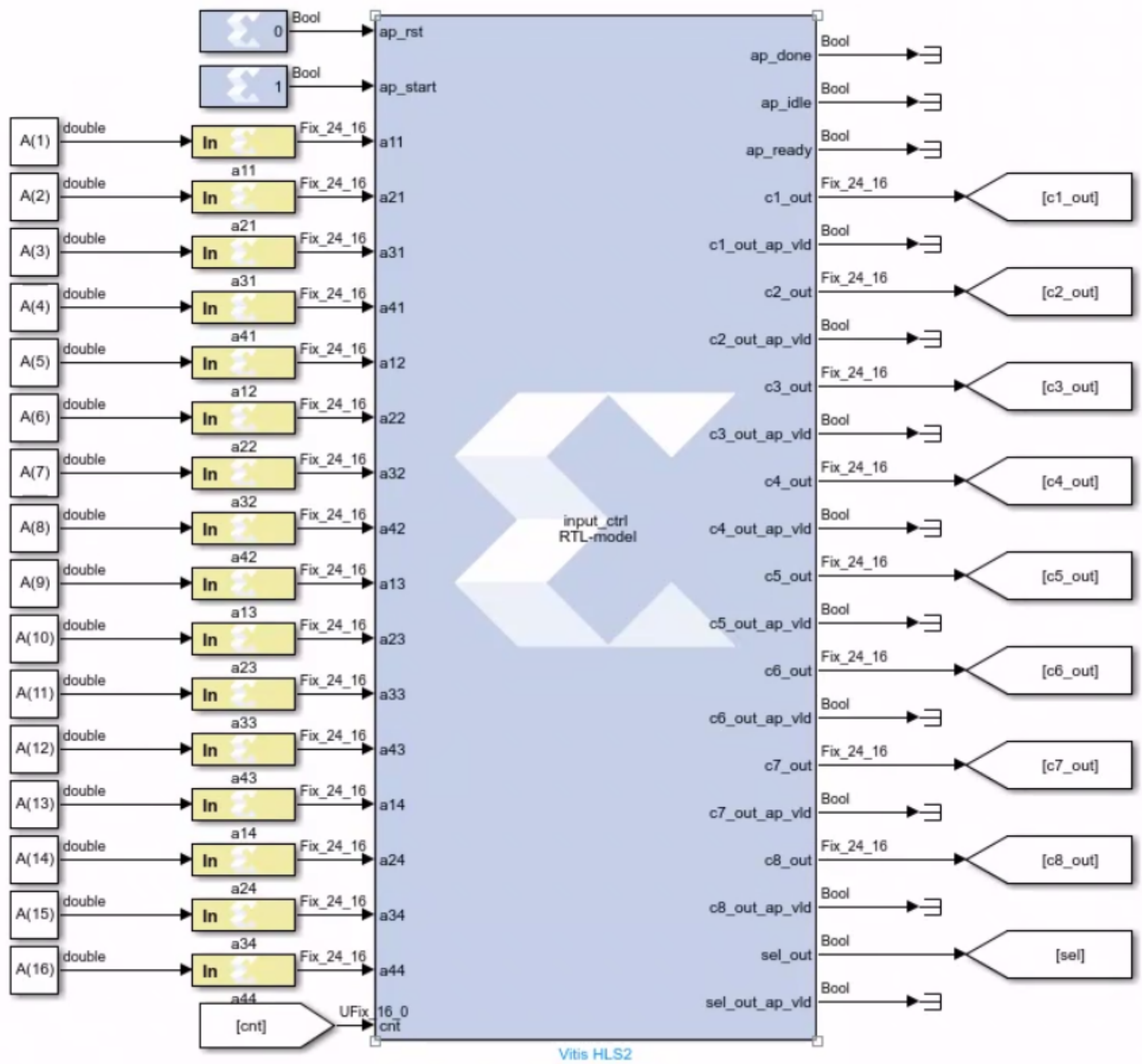
- Block diagrams



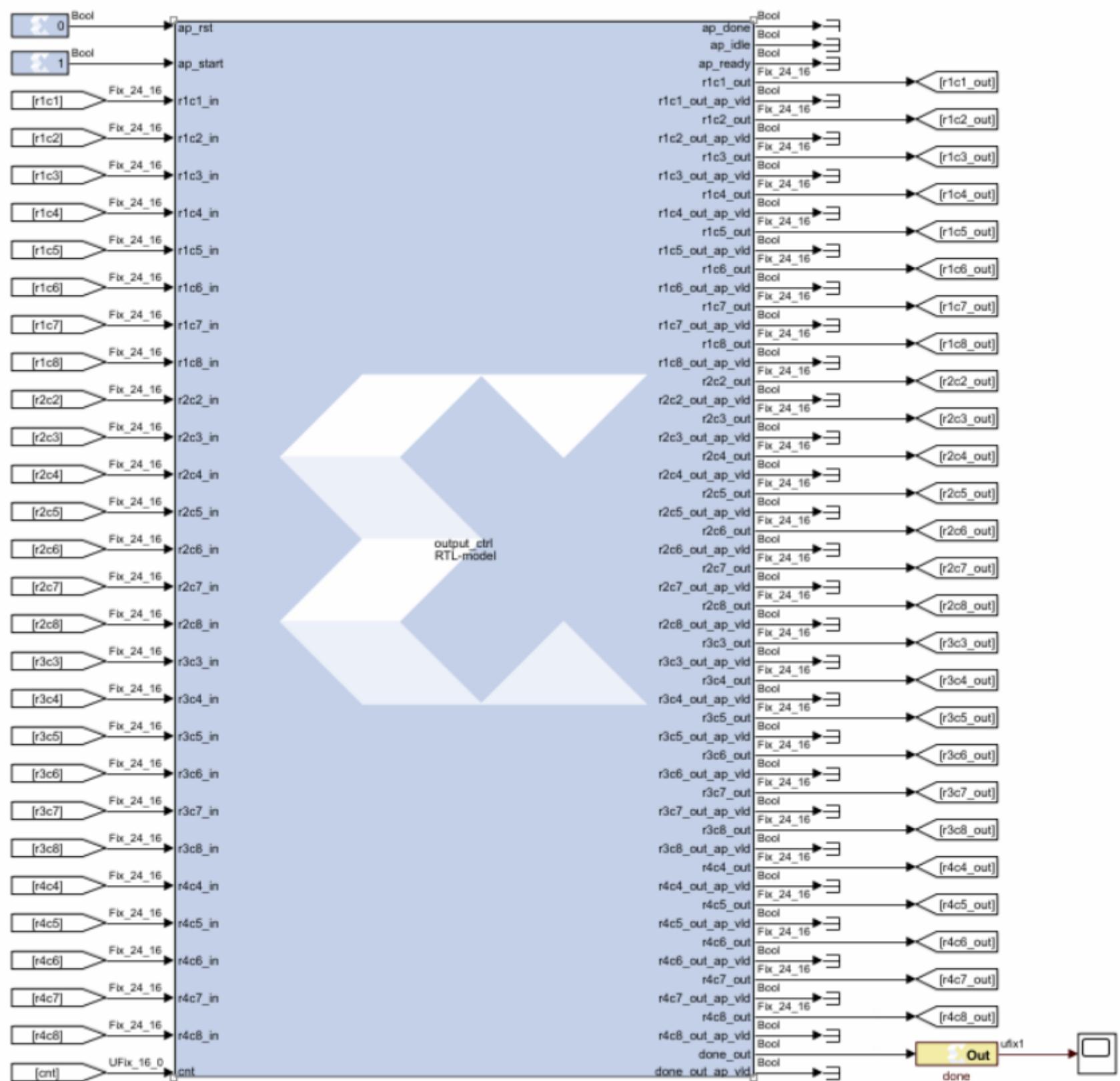
◦ QR Decomposition Module



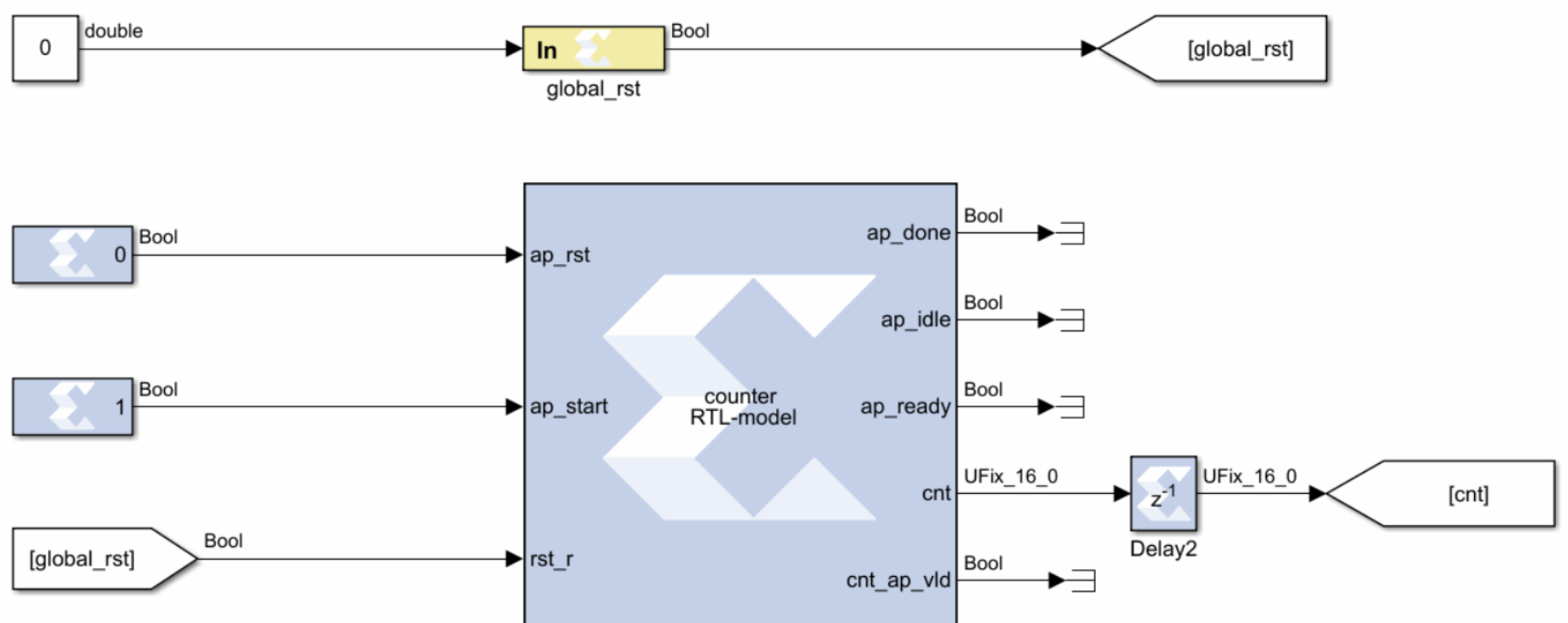
◦ QR Decomposition Input Control Module



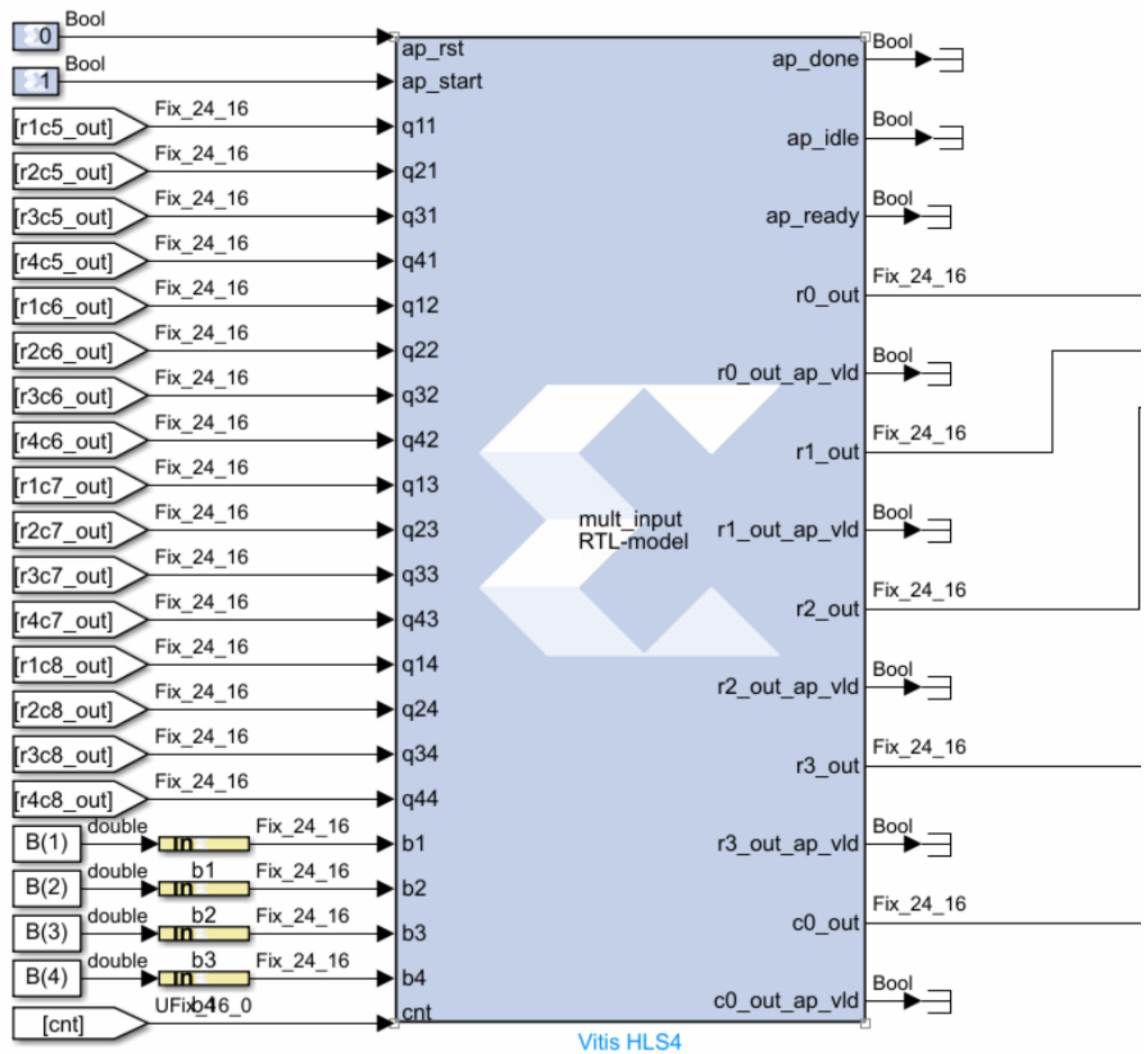
- QR Decomposition Output Control Module



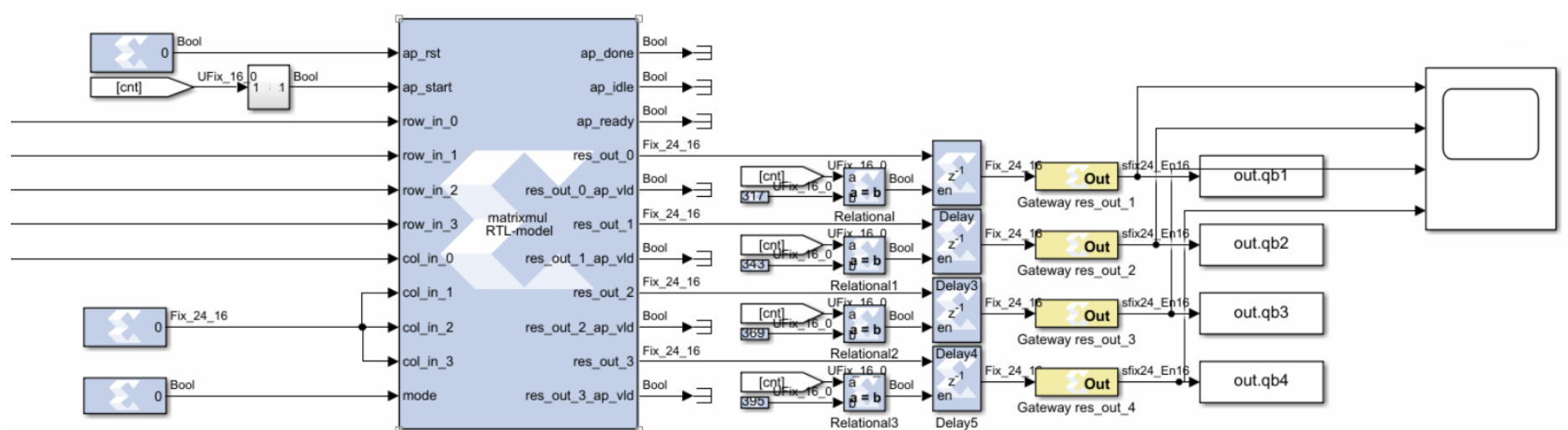
- Reset and Counter Module



- QRD and Matrix Multiplier Connection Module

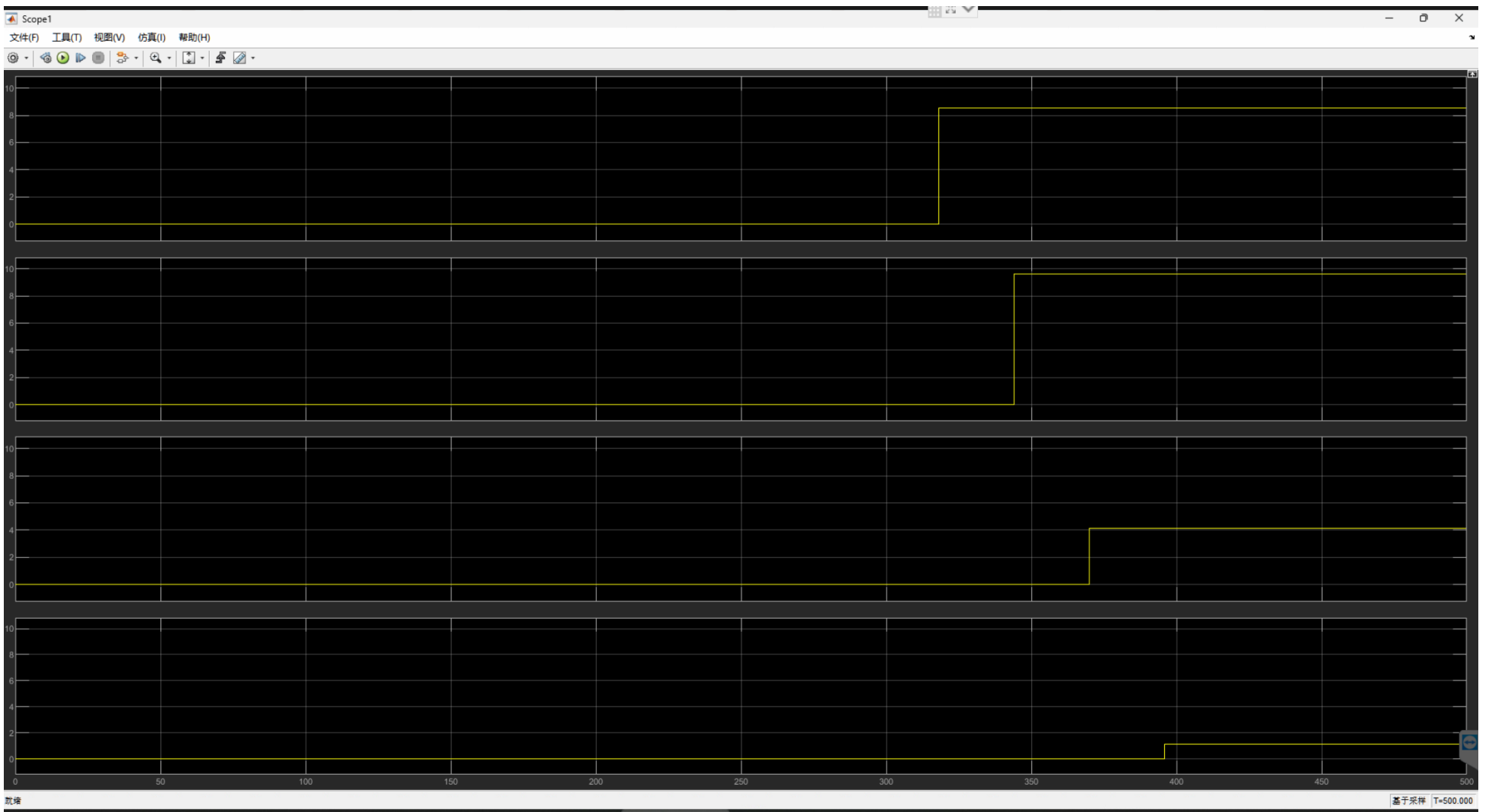


◦ Matrix Multiplier and Latch Module



• Scope/ terminal text outputs

Q transpose * b remains unchanged at the end for the ARM core to read.



- .m code for testing

```
clc;
clear;
% Define data types
full_bits = 24;
frac_bits = 16;
% Test1
disp("Test1:")
A = [1 1 4 3; 2 7 6 5 ; 9 10 3 11; 16 1 14 13];
B = [2; 10; 8; 4];
x = test();
disp("Result x = ");
disp(x);
disp("Result by matlab x = ");
disp(A\B);
disp("Differene between the result from matlab and the module = ")
disp(x - A\B);
% Test2
disp("Test2:")
A = [0.1 0.1 0.4 0.3; 0.2 0.8 0.6 0.5 ; 0.9 0.1 0.3 .2; 0.3 0.1 0.4 0.6];
B = [0.2; 0.1; 0.8; 0.4];
x = test();
disp("Result x = ");
disp(x);
disp("Result by matlab x = ");
disp(A\B);
disp("Differene between the result from matlab and the module = ")
disp(x - A\B);
% Test3
disp("Test3:")
A = [1 0 5 8; 3 2 3 3;-1 -3 5 7;9 -5 3 6];
B = [37; 34; 13; 26];
x = test();
disp("Result x = ");
disp(x);
disp("Result by matlab x = ");
disp(A\B);
disp("Differene between the result from matlab and the module = ")
disp(x - A\B);
% Test4
disp("Test4:")
A = [2 -5 -3 4;5 12 7 1;3 7 6 9;3 3 9 4];
B = [1; 2; 9; 7];
x = test();
disp("Result x = ");
disp(x);
disp("Result by matlab x = ");
disp(A\B);
disp("Differene between the result from matlab and the module = ")
disp(x - A\B);
```


- .m code test results

```
Test1:
Result x =
    0.1956
    1.3314
    0.7555
   -0.8492

Result by matlab x =
    0.1966
    1.3322
    0.7559
   -0.8508

Differene between the result from matlab and the module =
   -0.0010
   -0.0008
  -0.0004
    0.0017

Test2:
Result x =
    0.8034
   -0.3794
    0.2957
    0.1309

Result by matlab x =
    0.8034
   -0.3796
    0.2957
    0.1311

Differene between the result from matlab and the module =
  1.0e-03 *

    0.0195
    0.1602
   -0.0468
   -0.1930

Test3:
Result x =
    2.9997
    5.0000
    2.0008
    2.9994

Result by matlab x =
    3.0000
    5.0000
    2.0000
    3.0000

Differene between the result from matlab and the module =
  1.0e-03 *

   -0.2661
    0.0006
    0.7902
   -0.6310

Test4:
Result x =
   -0.4072
   -0.0686
    0.5794
    0.8028

Result by matlab x =
   -0.4074
   -0.0686
    0.5797
    0.8027

Differene between the result from matlab and the module =
  1.0e-03 *

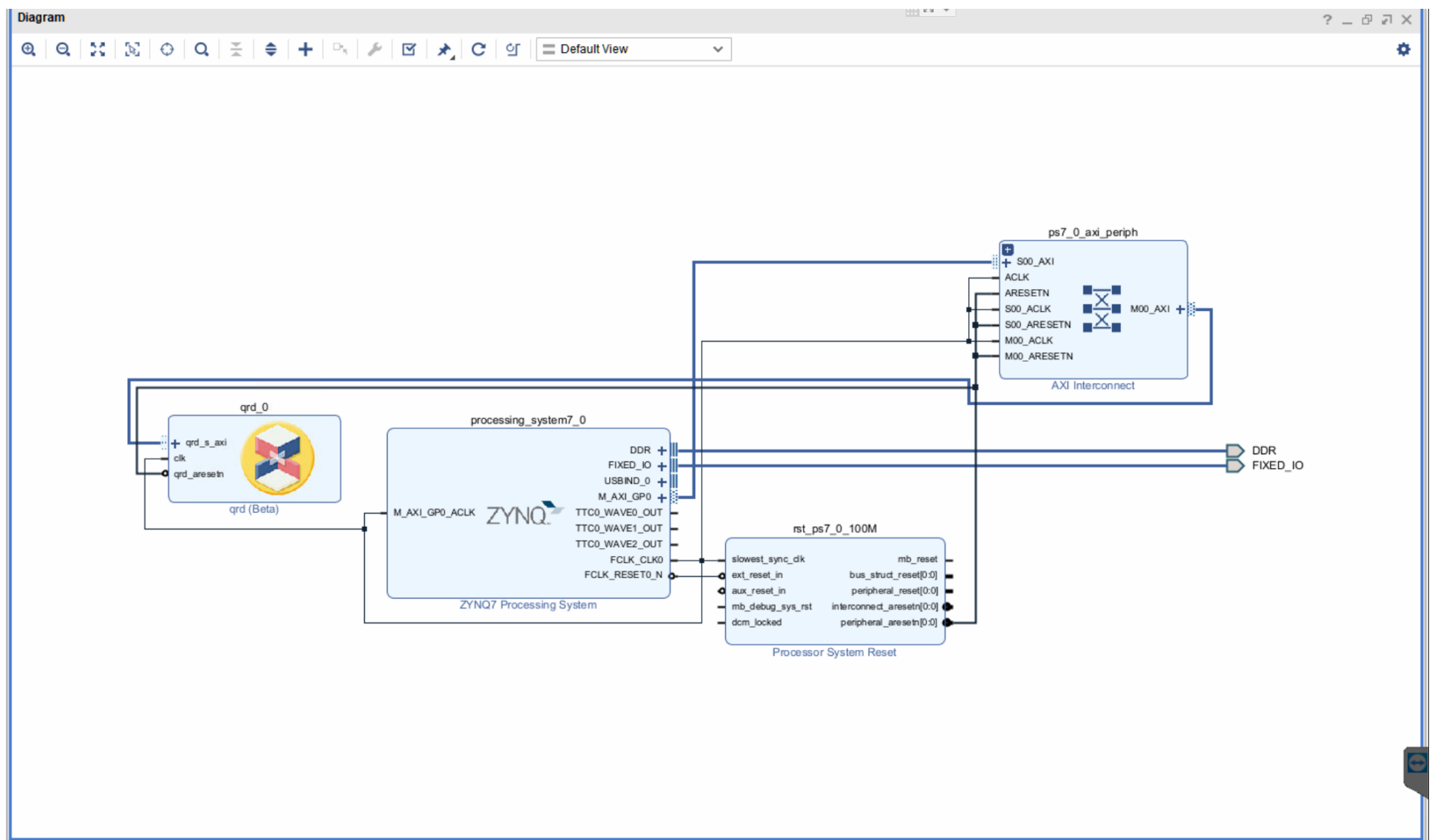
    0.1995
    0.0285
```

-0.2502
0.1078

The test results show that the calculation results using the FPGA accelerator are almost the same as the calculation results of the matlab built-in functions.

3. Integration with ARM core on the Zynq ZedBoard

- Block diagram



- Back-substitution and test bench C++ code (Vitis C++ code to start QRD, matrix vector multiplication on PL, then C++ back substitution on PS ARM core.)

```
#include <cmath>
#include <iostream>
#include "ap_fixed.h"
#include "xparameters.h"
#include "xtime_l.h"
#include "xscugic.h"
#include "qr_d.h"

using namespace std;
// #define TIMING_TEST

// Define our fixed point type used here to match the PL accelerator
typedef ap_fixed<24,8, AP_RND, AP_SAT> FIXED_TYPE;

// Create an instance of qr_d
qr_d qr_d_ins;

// The AXI interface sends only "raw bits" so we need to convert standard data types we use to
// match the ap_fixed data type in the PL accelerator.
// Using C++ functions per - RJ Cunningham
int get_int_reinterpret(FIXED_TYPE x) {
    return *(reinterpret_cast<int*>(&x));
}

FIXED_TYPE get_fixed_reinterpret(int x) {
    return *(reinterpret_cast<FIXED_TYPE*>(&x));
}

// Initialize the hardware and perform calculations
void QRD_run(double (*A)[4], double (*Q)[4], double (*R)[4], double *B, double *Y)
{
    // values for simulation and testing using C++ format with ap_fixed
    FIXED_TYPE a11 = A[0][0];
    FIXED_TYPE a21 = A[1][0];
    FIXED_TYPE a31 = A[2][0];
    FIXED_TYPE a41 = A[3][0];
    FIXED_TYPE a12 = A[0][1];
    FIXED_TYPE a22 = A[1][1];
    FIXED_TYPE a32 = A[2][1];
```

```

FIXED_TYPE a42 = A[3][1];
FIXED_TYPE a13 = A[0][2];
FIXED_TYPE a23 = A[1][2];
FIXED_TYPE a33 = A[2][2];
FIXED_TYPE a43 = A[3][2];
FIXED_TYPE a14 = A[0][3];
FIXED_TYPE a24 = A[1][3];
FIXED_TYPE a34 = A[2][3];
FIXED_TYPE a44 = A[3][3];
FIXED_TYPE b1 = B[0];
FIXED_TYPE b2 = B[1];
FIXED_TYPE b3 = B[2];
FIXED_TYPE b4 = B[3];

// Need to reinterpret the 16 bit ap_fixed values into raw unsigned 32 bit value for AXI transfer
unsigned int a11_u32 = get_int_reinterpret(a11);
unsigned int a21_u32 = get_int_reinterpret(a21);
unsigned int a31_u32 = get_int_reinterpret(a31);
unsigned int a41_u32 = get_int_reinterpret(a41);
unsigned int a12_u32 = get_int_reinterpret(a12);
unsigned int a22_u32 = get_int_reinterpret(a22);
unsigned int a32_u32 = get_int_reinterpret(a32);
unsigned int a42_u32 = get_int_reinterpret(a42);
unsigned int a13_u32 = get_int_reinterpret(a13);
unsigned int a23_u32 = get_int_reinterpret(a23);
unsigned int a33_u32 = get_int_reinterpret(a33);
unsigned int a43_u32 = get_int_reinterpret(a43);
unsigned int a14_u32 = get_int_reinterpret(a14);
unsigned int a24_u32 = get_int_reinterpret(a24);
unsigned int a34_u32 = get_int_reinterpret(a34);
unsigned int a44_u32 = get_int_reinterpret(a44);
unsigned int b1_u32 = get_int_reinterpret(b1);
unsigned int b2_u32 = get_int_reinterpret(b2);
unsigned int b3_u32 = get_int_reinterpret(b3);
unsigned int b4_u32 = get_int_reinterpret(b4);

// Define reset signal
bool rst = 1;

// Initialize instance
qrd_initialize(&qrd_ins, XPAR_QRD_0_DEVICE_ID);
// Set reset to valid
qrd_global_rst_write(&qrd_ins, rst);
// Load matrix for decomposition
qrd_a11_write(&qrd_ins, a11_u32);
qrd_a21_write(&qrd_ins, a21_u32);
qrd_a31_write(&qrd_ins, a31_u32);
qrd_a41_write(&qrd_ins, a41_u32);
qrd_a12_write(&qrd_ins, a12_u32);
qrd_a22_write(&qrd_ins, a22_u32);
qrd_a32_write(&qrd_ins, a32_u32);
qrd_a42_write(&qrd_ins, a42_u32);
qrd_a13_write(&qrd_ins, a13_u32);
qrd_a23_write(&qrd_ins, a23_u32);
qrd_a33_write(&qrd_ins, a33_u32);
qrd_a43_write(&qrd_ins, a43_u32);
qrd_a14_write(&qrd_ins, a14_u32);
qrd_a24_write(&qrd_ins, a24_u32);
qrd_a34_write(&qrd_ins, a34_u32);
qrd_a44_write(&qrd_ins, a44_u32);
qrd_b1_write(&qrd_ins, b1_u32);
qrd_b2_write(&qrd_ins, b2_u32);
qrd_b3_write(&qrd_ins, b3_u32);
qrd_b4_write(&qrd_ins, b4_u32);
// Set reset to invalid
rst = 0;
qrd_global_rst_write(&qrd_ins, rst);
// Wait for done
while(qrd_done_read(&qrd_ins) == 0);
#endif
// Unload Matrix R
R[0][0] = double(get_fixed_reinterpret(qrd_r1c1_read(&qrd_ins)));
R[0][1] = double(get_fixed_reinterpret(qrd_r1c2_read(&qrd_ins)));
R[0][2] = double(get_fixed_reinterpret(qrd_r1c3_read(&qrd_ins)));
R[0][3] = double(get_fixed_reinterpret(qrd_r1c4_read(&qrd_ins)));
R[1][1] = double(get_fixed_reinterpret(qrd_r2c2_read(&qrd_ins)));
R[1][2] = double(get_fixed_reinterpret(qrd_r2c3_read(&qrd_ins)));
R[1][3] = double(get_fixed_reinterpret(qrd_r2c4_read(&qrd_ins)));
R[2][2] = double(get_fixed_reinterpret(qrd_r3c3_read(&qrd_ins)));
R[2][3] = double(get_fixed_reinterpret(qrd_r3c4_read(&qrd_ins)));
R[3][3] = double(get_fixed_reinterpret(qrd_r4c4_read(&qrd_ins)));

```

```

// Unload Matrix Q
Q[0][0] = double(get_fixed_reinterpret(qrd_r1c5_read(&qrd_ins)));
Q[1][0] = double(get_fixed_reinterpret(qrd_r1c6_read(&qrd_ins)));
Q[2][0] = double(get_fixed_reinterpret(qrd_r1c7_read(&qrd_ins)));
Q[3][0] = double(get_fixed_reinterpret(qrd_r1c8_read(&qrd_ins)));
Q[0][1] = double(get_fixed_reinterpret(qrd_r2c5_read(&qrd_ins)));
Q[1][1] = double(get_fixed_reinterpret(qrd_r2c6_read(&qrd_ins)));
Q[2][1] = double(get_fixed_reinterpret(qrd_r2c7_read(&qrd_ins)));
Q[3][1] = double(get_fixed_reinterpret(qrd_r2c8_read(&qrd_ins)));
Q[0][2] = double(get_fixed_reinterpret(qrd_r3c5_read(&qrd_ins)));
Q[1][2] = double(get_fixed_reinterpret(qrd_r3c6_read(&qrd_ins)));
Q[2][2] = double(get_fixed_reinterpret(qrd_r3c7_read(&qrd_ins)));
Q[3][2] = double(get_fixed_reinterpret(qrd_r3c8_read(&qrd_ins)));
Q[0][3] = double(get_fixed_reinterpret(qrd_r4c5_read(&qrd_ins)));
Q[1][3] = double(get_fixed_reinterpret(qrd_r4c6_read(&qrd_ins)));
Q[2][3] = double(get_fixed_reinterpret(qrd_r4c7_read(&qrd_ins)));
Q[3][3] = double(get_fixed_reinterpret(qrd_r4c8_read(&qrd_ins)));
#endif

// Unload Vector QT*B
Y[0] = double(get_fixed_reinterpret(qrd_gateway_res_out_1_read(&qrd_ins)));
Y[1] = double(get_fixed_reinterpret(qrd_gateway_res_out_2_read(&qrd_ins)));
Y[2] = double(get_fixed_reinterpret(qrd_gateway_res_out_3_read(&qrd_ins)));
Y[3] = double(get_fixed_reinterpret(qrd_gateway_res_out_4_read(&qrd_ins)));
#endifdef TIMING_TEST

// Print Matrix A
cout << endl << "Matrix A:" << endl;
for(int i = 0; i < 4; i++)
{
    for(int j = 0; j < 4; j++)
    {
        printf("%10.5f\t", A[i][j]);
    }
    cout << endl;
}

// Print Matrix Q
cout << endl << "Corresponding Matrix Q:" << endl;
for(int i = 0; i < 4; i++)
{
    for(int j = 0; j < 4; j++)
    {
        printf("%10.5f\t", Q[i][j]);
    }
    cout << endl;
}

// Print Matrix R
cout << endl << "Corresponding Matrix R:" << endl;
for(int i = 0; i < 4; i++)
{
    for(int j = 0; j < 4; j++)
    {
        printf("%10.5f\t", R[i][j]);
    }
    cout << endl;
}

// Print Vector QT*B
cout << endl << "Corresponding Vector QT*B:" << endl;
for(int i = 0; i < 4; i++)
{
    printf("%10.5f\t", Y[i]);
}

cout << endl << endl;
#endif
}

void calculate(double (*A)[4], double *B, double *X)
{
    double Q[4][4] = {0};
    double R[4][4] = {0};
    double Y[4];

    // Perform QR decomposition
    QRD_run(A,Q,R,B,Y);
    // Do the back substitution
    for (int j = 3; j >= 0; j--)
    {
        if (R[j][j] == 0)
        {
#endifdef TIMING_TEST
            cout << "Matrix is singular!" << endl;
#endif

```



```

    }
    X[j] = Y[j] / R[j][j];
    for (int i = 0; i <= j; i++)
    {
        Y[i] = Y[i] - R[i][j] * X[j];
    }
}
#endif
// Print the corresponding vector X
cout << "Corresponding Vector X:" << endl;
for (int i = 0; i < 4; i++)
{
    printf("%8.4f\t", X[i]);
}
cout << endl;
#endif
}

// Main function
int main()
{
    cout << "--- Start of the Program ---" << endl;
#ifdef TIMING_TEST
    // Variables for timing and counts used for application cycle counts and timing
    unsigned long long tt;
    int tt_print;
    double tt_seconds, ps_time;
    double total_tt = 0;
    XTime start_time_co;
    XTime stop_time_co;
#endif
    // Define equation set 1
    double X[4] = { 0 };
    double A1[4][4] = { {1,1,4,3},
                        {2,7,6,5},
                        {9,10,3,11},
                        {16,1,14,13}
    };
    double B1[4] = {2,10,8,4};
    // Define equation set 2
    double A2[4][4] = { {0.1,0.1,0.4,0.3},
                        {0.2,0.8,0.6,0.5},
                        {0.9,0.1,0.3,0.2},
                        {0.3,0.1,0.4,0.6}
    };
    double B2[4] = {0.2, 0.1, 0.8, 0.4};
    // Define equation set 3
    double A3[4][4] = { {1,0,5,8},
                        {3,2,3,3},
                        {-1,-3,5,7},
                        {9,-5,3,6}
    };
    double B3[4] = {37, 34, 13, 26};
#ifdef TIMING_TEST
    // Get the starting time in cycle counts
    XTime_GetTime(&start_time_co);
#endif
    // Perform First Decomposition
#ifdef TIMING_TEST
    cout << endl << "*****Perform First Decomposition*****" << endl;
#endif
    calculate(A1,B1,X);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #1 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;
#endif
#ifdef TIMING_TEST
    // Get the starting time in cycle counts
    XTime_GetTime(&start_time_co);
#endif
    // Perform Second Decomposition
#ifdef TIMING_TEST

```

```

        cout << endl << "*****Perform Second Decomposition*****" << endl;
    #endif
    calculate(A2,B2,X);
    #ifdef TIMING_TEST
        // Capture the stop time on the processor
        XTime_GetTime(&stop_time_co);
        // Compute timing on PL hardware using the accelerator
        tt = stop_time_co - start_time_co;
        tt_print = (unsigned) tt;
        cout << "Done, Total time steps for solving the #2 system of equations = " << tt_print << endl;
        tt_seconds = COUNTS_PER_SECOND;
        cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
        ps_time = (float) tt_print / tt_seconds;
        cout << "Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
        total_tt += ps_time;
    #endif
    #ifdef TIMING_TEST
        // Get the starting time in cycle counts
        XTime_GetTime(&start_time_co);
    #endif
    // Perform Third Decomposition
    #ifndef TIMING_TEST
        cout << endl << "*****Perform Third Decomposition*****" << endl;
    #endif
    calculate(A3,B3,X);
    #ifdef TIMING_TEST
        // Capture the stop time on the processor
        XTime_GetTime(&stop_time_co);
        // Compute timing on PL hardware using the accelerator
        tt = stop_time_co - start_time_co;
        tt_print = (unsigned) tt;
        cout << "Done, Total time steps for solving the #3 system of equations = " << tt_print << endl;
        tt_seconds = COUNTS_PER_SECOND;
        cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
        ps_time = (float) tt_print / tt_seconds;
        cout << "Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
        total_tt += ps_time;

        cout << "Total time for all three is " << total_tt << endl;
    #endif

    cout << "--- End of the Program ---" << endl;
    return 0;
}

```

- Terminal text outputs

```

--- Start of the Program ---

*****Perform First Decomposition*****

Matrix A:
  1.00000    1.00000    4.00000    3.00000
  2.00000    7.00000    6.00000    5.00000
  9.00000   10.00000    3.00000   11.00000
 16.00000    1.00000   14.00000   13.00000

Corresponding Matrix Q:
  0.05403    0.06201    0.46677   -0.88072
  0.10811    0.60493    0.67555    0.40729
  0.48657    0.65532   -0.53853   -0.20934
  0.86519   -0.44804    0.18922    0.12183

Corresponding Matrix R:
 18.49330    6.54210   14.43793   17.30327
  0.00000   10.40207   -0.42807    4.59534
  0.00000    0.00000    6.95442    1.31487
  0.00000    0.00000    0.00000   -1.32368

Corresponding Vector QT*B:
  8.54248    9.62375    4.13773    1.12405

Corresponding Vector X:
  0.1956    1.3314    0.7555   -0.8492

*****Perform Second Decomposition*****

Matrix A:
  0.10000    0.10000    0.40000    0.30000
  0.20000    0.80000    0.60000    0.50000

```

```
0.90000    0.10000    0.30000    0.20000
0.30000    0.10000    0.40000    0.60000

Corresponding Matrix Q:
0.10265    0.09108    0.76566   -0.62863
0.20514    0.96904   -0.13702    0.00694
0.92334   -0.22914   -0.24733   -0.18362
0.30786    0.01109    0.57785    0.75574

Corresponding Matrix R:
0.97470    0.29747    0.56425    0.50270
0.00000    0.76259    0.55357    0.47264
0.00000    0.00000    0.38097    0.45840
0.00000    0.00000    0.00000    0.23161

Corresponding Vector QT*B:
0.90285   -0.06375    0.17268    0.03035

Corresponding Vector X:
0.8034   -0.3794    0.2956    0.1310

*****Perform Third Decomposition*****

Matrix A:
1.00000    0.00000    5.00000    8.00000
3.00000    2.00000    3.00000    3.00000
-1.00000   -3.00000    5.00000    7.00000
9.00000   -5.00000    3.00000    6.00000

Corresponding Matrix Q:
0.10419    0.07993    0.67598    0.72510
0.31268    0.64906    0.44524   -0.53169
-0.10426   -0.69356    0.56474   -0.43504
0.93831   -0.30223   -0.16078    0.04831

Corresponding Matrix R:
9.59164   -3.75362    3.75319    6.67244
0.00000    4.88986   -2.02742   -4.08144
0.00000    0.00000    7.05721    9.73225
0.00000    0.00000    0.00000    1.45082

Corresponding Vector QT*B:
37.52675    8.15114   43.31061    4.35155

Corresponding Vector X:
2.9997    5.0000    2.0008    2.9994
--- End of the Program ---
```

4. Stand-alone C++ code version of the entire linear equation solver algorithm (with QR Decomposition)

- C++ Code

At first I used the Gaussian elimination method to solve the system of equations, but later switched to the **QR decomposition method**. The following is the test code for QR decomposition to solve linear equations.

```
#include <cmath>
#include <iostream>
#include "ap_fixed.h"
#include "xparameters.h"
#include "xtime_l.h"
#include "xscugic.h"
#include <cstdlib>
#include <ctime>

// #define TIMING_TEST
using namespace std;
// Define our fixed point type used here to match the PL accelerator
typedef ap_fixed<24,8, AP_RND, AP_SAT> FIXED_TYPE;

// Help function to print the 4*4 matrices
void print_matrix(FIXED_TYPE (*a)[4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            printf("%10.5f\t", (double)a[i][j]);
        }
        cout << endl;
    }
}
```

```

    }
}

// Help function to print the vectors
void print_vector(FIXED_TYPE *a)
{
    for (int i = 0; i < 4; i++)
    {
        printf("%10.5f\t", (double)a[i]);
    }
    cout << endl;
}

// Help function to print the 2*2 matrices
void print_rotation_matrix(FIXED_TYPE(*rotation_matrix)[2])
{
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            cout << rotation_matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

// Get the rotation matrix according to the angle
void get_rotation_matrix(FIXED_TYPE theta, FIXED_TYPE(*mat)[2])
{
    mat[0][0] = cos(double(theta));
    mat[0][1] = -sin(double(theta));
    mat[1][0] = sin(double(theta));
    mat[1][1] = cos(double(theta));
}

// Perform the rotation to matrices
void rotate(FIXED_TYPE(*a)[4], FIXED_TYPE(*rotation_matrix)[2],int row1, int row2)
{
    FIXED_TYPE a_res[4][4] = { 0 };
    a_res[row1][0] = rotation_matrix[0][0] * a[row1][0] + rotation_matrix[0][1] * a[row2][0];
    a_res[row1][1] = rotation_matrix[0][0] * a[row1][1] + rotation_matrix[0][1] * a[row2][1];
    a_res[row1][2] = rotation_matrix[0][0] * a[row1][2] + rotation_matrix[0][1] * a[row2][2];
    a_res[row1][3] = rotation_matrix[0][0] * a[row1][3] + rotation_matrix[0][1] * a[row2][3];

    a_res[row2][0] = rotation_matrix[1][0] * a[row1][0] + rotation_matrix[1][1] * a[row2][0];
    a_res[row2][1] = rotation_matrix[1][0] * a[row1][1] + rotation_matrix[1][1] * a[row2][1];
    a_res[row2][2] = rotation_matrix[1][0] * a[row1][2] + rotation_matrix[1][1] * a[row2][2];
    a_res[row2][3] = rotation_matrix[1][0] * a[row1][3] + rotation_matrix[1][1] * a[row2][3];

    for (int i = 0; i < 4; i++)
    {
        a[row1][i] = a_res[row1][i];
        a[row2][i] = a_res[row2][i];
    }
}

// Multiply a matrix with a vector
void matrix_vector_mult(FIXED_TYPE(*matrix)[4], FIXED_TYPE(*vector))
{
    FIXED_TYPE vector_res[4] = { 0 };
    vector_res[0] = matrix[0][0] * vector[0] + matrix[0][1] * vector[1] + matrix[0][2] * vector[2] + matrix[0][3] * vector[3];
    vector_res[1] = matrix[1][0] * vector[0] + matrix[1][1] * vector[1] + matrix[1][2] * vector[2] + matrix[1][3] * vector[3];
    vector_res[2] = matrix[2][0] * vector[0] + matrix[2][1] * vector[1] + matrix[2][2] * vector[2] + matrix[2][3] * vector[3];
    vector_res[3] = matrix[3][0] * vector[0] + matrix[3][1] * vector[1] + matrix[3][2] * vector[2] + matrix[3][3] * vector[3];
    for (int i = 0; i < 4; i++)
    {
        vector[i] = vector_res[i];
    }
}

// Perform the whole calculation to get the final result X
void calculate(FIXED_TYPE(*A)[4],FIXED_TYPE *B)
{
#ifdef TIMING_TEST
    cout << "Matrix A:" << endl;
    print_matrix(A);
#endif
    FIXED_TYPE E[4][4] = { {1,0,0,0},
                            {0,1,0,0},
                            {0,0,1,0},
                            {0,0,0,1} };

```



```

    FIXED_TYPE rotation_matrix[2][2] = { 0 };
    // rotation #1
    FIXED_TYPE theta = -atan((double)(A[1][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 0, 1);
    rotate(E, rotation_matrix, 0, 1);
    // rotation #2
    theta = -atan((double)(A[2][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 0, 2);
    rotate(E, rotation_matrix, 0, 2);
    // rotation #3
    theta = -atan((double)(A[3][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 0, 3);
    rotate(E, rotation_matrix, 0, 3);
    // rotation #4
    theta = -atan((double)(A[2][1] / A[1][1]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 1, 2);
    rotate(E, rotation_matrix, 1, 2);
    // rotation #5
    theta = -atan((double)(A[3][1] / A[1][1]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 1, 3);
    rotate(E, rotation_matrix, 1, 3);
    // rotation #6
    theta = -atan((double)(A[3][2] / A[2][2]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 2, 3);
    rotate(E, rotation_matrix, 2, 3);
    // Print the result
    FIXED_TYPE(*R)[4] = A;
    FIXED_TYPE(*QT)[4] = E;
#ifdef TIMING_TEST
    cout << "Corresponding Matrix R:" << endl;
    print_matrix(R);
    cout << "Corresponding Matrix Q:" << endl;
    FIXED_TYPE Q[4][4];
    for(int i = 0; i<4; i++)
    {
        for(int j = 0; j<4; j++)
        {
            Q[i][j] = QT[j][i];
        }
    }
    print_matrix(Q);
#endif
    // Calculate QT*B
    matrix_vector_mult(QT, B);
#ifdef TIMING_TEST
    cout << "Corresponding Vector QT*B:" << endl;
    print_vector(B);
#endif
    // Check if the matrix is singular
    FIXED_TYPE X[4] = { 0 };
    for (int j = 3; j >= 0; j--)
    {
        if (R[j][j] == 0)
        {
#ifdef TIMING_TEST
            cout << "Matrix is singular!" << endl;
#endif
        }
        X[j] = B[j] / R[j][j];
        for (int i = 0; i <= j; i++)
        {
            B[i] = B[i] - R[i][j] * X[j];
        }
    }
#ifdef TIMING_TEST
    cout << "Corresponding Vector X:" << endl;
    print_vector(X);
#endif
}

int main()
{
    cout << "-----Start of the Program-----" << endl;
#ifdef TIMING_TEST
    // variables for timing and counts used for application cycle counts and timing

```

```

    unsigned long long tt;
    int tt_print;
    double total_tt = 0;
    double tt_seconds, ps_time;
    XTime start_time_co;
    XTime stop_time_co;
    // Get the starting time in cycle counts
#endif
    // Define equation set 1
    FIXED_TYPE A1[4][4] = { {1,1,4,3},
                             {2,7,6,5},
                             {9,10,3,11},
                             {16,1,14,13}

    };
    FIXED_TYPE B1[4] = {2,10,8,4};
    // Define equation set 2
    FIXED_TYPE A2[4][4] = { {0.1,0.1,0.4,0.3},
                             {0.2,0.8,0.6,0.5},
                             {0.9,0.1,0.3,0.2},
                             {0.3,0.1,0.4,0.6}

    };
    FIXED_TYPE B2[4] = {0.2, 0.1, 0.8, 0.4};
    // Define equation set 3
    FIXED_TYPE A3[4][4] = { {1,0,5,8},
                             {3,2,3,3},
                             {-1,-3,5,7},
                             {9,-5,3,6}

    };
    FIXED_TYPE B3[4] = {37, 34, 13, 26};
#ifdef TIMING_TEST
    // Capture the start time on the processor
    XTime_GetTime(&start_time_co);
#endif
#ifdef TIMING_TEST
    cout << endl << "*****Perform First Decomposition*****" << endl;
#endif
    calculate(A1,B1);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #1 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;
#endif
#ifdef TIMING_TEST
    // Capture the start time on the processor
    XTime_GetTime(&start_time_co);
#endif;
#ifdef TIMING_TEST
    cout << endl << "*****Perform Second Decomposition*****" << endl;
#endif
    calculate(A2,B2);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #2 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;
#endif
#ifdef TIMING_TEST
    // Capture the start time on the processor
    XTime_GetTime(&start_time_co);
#endif
#ifdef TIMING_TEST
    cout << endl << "*****Perform Third Decomposition*****" << endl;
#endif
    calculate(A3,B3);
#ifdef TIMING_TEST
    // Capture the stop time on the processor

```

```

XTime_GetTime(&stop_time_co);
// Compute timing on PL hardware using the accelerator
tt = stop_time_co - start_time_co;
tt_print = (unsigned) tt;
cout << "Done, Total time steps for solving the #3 system of equations = " << tt_print << endl;
tt_seconds = COUNTS_PER_SECOND;
cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
ps_time = (float) tt_print / tt_seconds;
cout << "Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
total_tt += ps_time;

cout << "Total time for all three is " << total_tt << endl;
#endif
cout << "-----End of the Program-----" << endl;
return 0;
}

```

- Terminal text outputs

```
-----Start of the Program-----
```

```
*****Perform First Decomposition*****
```

```
Matrix A:
```

```

1.00000    1.00000    4.00000    3.00000
2.00000    7.00000    6.00000    5.00000
9.00000   10.00000    3.00000   11.00000
16.00000    1.00000   14.00000   13.00000

```

```
Corresponding Matrix R:
```

```

18.49324    6.54279   14.43774   17.30356
 0.00017   10.40157   -0.42914    4.59412
 0.00008    0.00005   -6.95465   -1.31520
-0.00003   -0.00003   -0.00002   -1.32310

```

```
Corresponding Matrix Q:
```

```

 0.05408    0.06212   -0.46672   -0.88055
 0.10814    0.60497   -0.67554    0.40738
 0.48665    0.65529    0.53850   -0.20930
 0.86519   -0.44809   -0.18929    0.12184

```

```
Corresponding Vector QT*B:
```

```

8.54349    9.62384   -4.13797    1.12561

```

```
Corresponding Vector X:
```

```

0.19656    1.33215    0.75586   -0.85072

```

```
*****Perform Second Decomposition*****
```

```
Matrix A:
```

```

0.10001    0.10001    0.39999    0.30000
0.20000    0.80000    0.60001    0.50000
0.89999    0.10001    0.30000    0.20000
0.30000    0.10001    0.39999    0.60001

```

```
Corresponding Matrix R:
```

```

0.97467    0.29753    0.56429    0.50273
0.00000    0.76253    0.55354    0.47264
0.00000    0.00000   -0.38098   -0.45847
0.00000    0.00000    0.00000    0.23167

```

```
Corresponding Matrix Q:
```

```

0.10260    0.09109   -0.76553   -0.62857
0.20518    0.96904    0.13702    0.00708
0.92337   -0.22914    0.24727   -0.18364
0.30780    0.01103   -0.57796    0.75571

```

```
Corresponding Vector QT*B:
```

```

0.90286   -0.06377   -0.17276    0.03036

```

```
Corresponding Vector X:
```

```

0.80336   -0.37953    0.29573    0.13106

```

```
*****Perform Third Decomposition*****
```

```
Matrix A:
```

```

1.00000    0.00000    5.00000    8.00000
3.00000    2.00000    3.00000    3.00000
-1.00000   -3.00000    5.00000    7.00000
9.00000   -5.00000    3.00000    6.00000

```

```
Corresponding Matrix R:
```

```

9.59160   -3.75323    3.75325    6.67244
0.00002    4.89009   -2.02725   -4.08113
0.00002   -0.00005   -7.05725   -9.73257
-0.00003   -0.00003   -0.00005    1.45009

```

```
Corresponding Matrix Q:
```

```

0.10426    0.08002   -0.67604    0.72507
0.31277    0.64905   -0.44519   -0.53172
-0.10426   -0.69351   -0.56474   -0.43503
0.93831   -0.30229    0.16077    0.04834

```

```
Corresponding Vector QT*B:
```

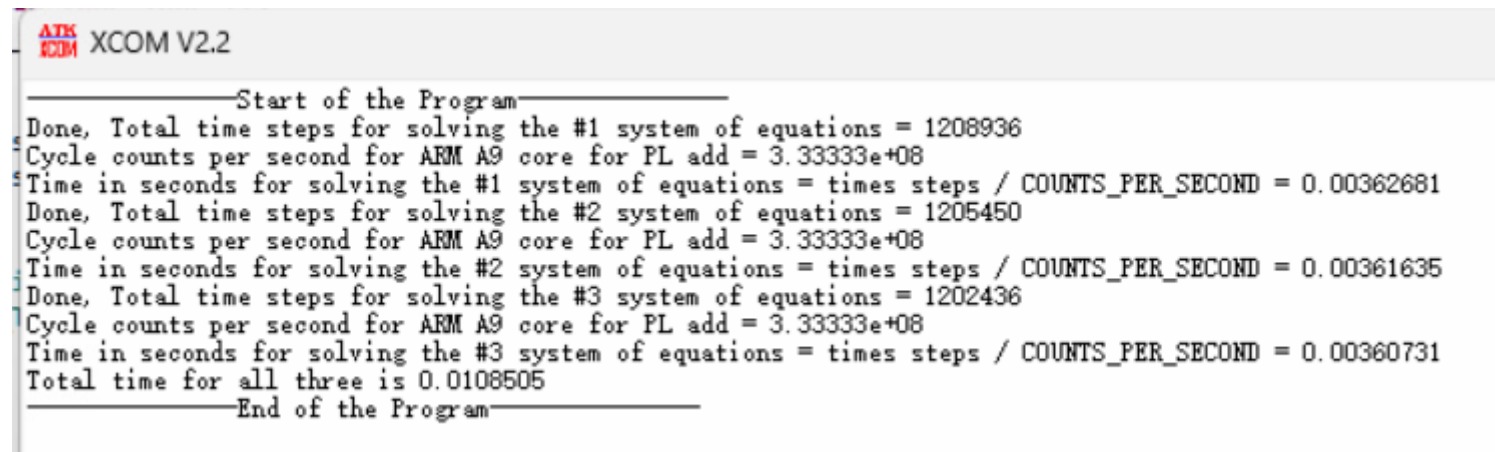
```

37.53268      8.15302    -43.31171      4.35037
Corresponding Vector X:
3.00006      5.00006      1.99983      3.00006
-----End of the Program-----

```

- Timing results

It took 0.01085 seconds to complete three calculations.



```

ATK
COM XCOM V2.2
-----Start of the Program-----
Done, Total time steps for solving the #1 system of equations = 1208936
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = 0.00362681
Done, Total time steps for solving the #2 system of equations = 1205450
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = 0.00361635
Done, Total time steps for solving the #3 system of equations = 1202436
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = 0.00360731
Total time for all three is 0.0108505
-----End of the Program-----

```

5. Timing analysis of the design tradeoffs between the totally stand-alone C++ version and the FPGA SoC PL accelerated version.

- C++ testbench of the FPGA SoC PL accelerated version

```

#include <cmath>
#include <iostream>
#include "ap_fixed.h"
#include "xparameters.h"
#include "xtime_l.h"
#include "xscugic.h"
#include "qrd.h"

using namespace std;
#define TIMING_TEST

// Define our fixed point type used here to match the PL accelerator
typedef ap_fixed<24,8, AP_RND, AP_SAT> FIXED_TYPE;

// Create an instance of qrd
qrd qrd_ins;

// The AXI interface sends only "raw bits" so we need to convert standard data types we use to
// match the ap_fixed data type in the PL accelerator.
// Using C++ functions per - RJ Cunningham
int get_int_reinterpret(FIXED_TYPE x) {
    return *(reinterpret_cast<int *>(&x));
}

FIXED_TYPE get_fixed_reinterpret(int x) {
    return *(reinterpret_cast<FIXED_TYPE *>(&x));
}

// Initialize the hardware and perform calculations
void QRD_run(double (*A)[4], double (*Q)[4], double (*R)[4], double *B, double *Y)
{
    // values for simulation and testing using C++ format with ap_fixed
    FIXED_TYPE a11 = A[0][0];
    FIXED_TYPE a21 = A[1][0];
    FIXED_TYPE a31 = A[2][0];
    FIXED_TYPE a41 = A[3][0];
    FIXED_TYPE a12 = A[0][1];
    FIXED_TYPE a22 = A[1][1];
    FIXED_TYPE a32 = A[2][1];
    FIXED_TYPE a42 = A[3][1];
    FIXED_TYPE a13 = A[0][2];
    FIXED_TYPE a23 = A[1][2];
    FIXED_TYPE a33 = A[2][2];
    FIXED_TYPE a43 = A[3][2];
    FIXED_TYPE a14 = A[0][3];
    FIXED_TYPE a24 = A[1][3];
    FIXED_TYPE a34 = A[2][3];
    FIXED_TYPE a44 = A[3][3];
    FIXED_TYPE b1 = B[0];
    FIXED_TYPE b2 = B[1];
    FIXED_TYPE b3 = B[2];
    FIXED_TYPE b4 = B[3];

    // Need to reinterpret the 16 bit ap_fixed values into raw unsigned 32 bit value for AXI transfer

```



```

    unsigned int a11_u32 = get_int_reinterpret(a11);
    unsigned int a21_u32 = get_int_reinterpret(a21);
    unsigned int a31_u32 = get_int_reinterpret(a31);
    unsigned int a41_u32 = get_int_reinterpret(a41);
    unsigned int a12_u32 = get_int_reinterpret(a12);
    unsigned int a22_u32 = get_int_reinterpret(a22);
    unsigned int a32_u32 = get_int_reinterpret(a32);
    unsigned int a42_u32 = get_int_reinterpret(a42);
    unsigned int a13_u32 = get_int_reinterpret(a13);
    unsigned int a23_u32 = get_int_reinterpret(a23);
    unsigned int a33_u32 = get_int_reinterpret(a33);
    unsigned int a43_u32 = get_int_reinterpret(a43);
    unsigned int a14_u32 = get_int_reinterpret(a14);
    unsigned int a24_u32 = get_int_reinterpret(a24);
    unsigned int a34_u32 = get_int_reinterpret(a34);
    unsigned int a44_u32 = get_int_reinterpret(a44);
    unsigned int b1_u32 = get_int_reinterpret(b1);
    unsigned int b2_u32 = get_int_reinterpret(b2);
    unsigned int b3_u32 = get_int_reinterpret(b3);
    unsigned int b4_u32 = get_int_reinterpret(b4);

    // Define reset signal
    bool rst = 1;

    // Initialize instance
    qrd_initialize(&qrd_ins, XPAR_QRD_0_DEVICE_ID);
    // Set reset to valid
    qrd_global_rst_write(&qrd_ins, rst);
    // Load matrix for decomposition
    qrd_a11_write(&qrd_ins, a11_u32);
    qrd_a21_write(&qrd_ins, a21_u32);
    qrd_a31_write(&qrd_ins, a31_u32);
    qrd_a41_write(&qrd_ins, a41_u32);
    qrd_a12_write(&qrd_ins, a12_u32);
    qrd_a22_write(&qrd_ins, a22_u32);
    qrd_a32_write(&qrd_ins, a32_u32);
    qrd_a42_write(&qrd_ins, a42_u32);
    qrd_a13_write(&qrd_ins, a13_u32);
    qrd_a23_write(&qrd_ins, a23_u32);
    qrd_a33_write(&qrd_ins, a33_u32);
    qrd_a43_write(&qrd_ins, a43_u32);
    qrd_a14_write(&qrd_ins, a14_u32);
    qrd_a24_write(&qrd_ins, a24_u32);
    qrd_a34_write(&qrd_ins, a34_u32);
    qrd_a44_write(&qrd_ins, a44_u32);
    qrd_b1_write(&qrd_ins, b1_u32);
    qrd_b2_write(&qrd_ins, b2_u32);
    qrd_b3_write(&qrd_ins, b3_u32);
    qrd_b4_write(&qrd_ins, b4_u32);
    // Set reset to invalid
    rst = 0;
    qrd_global_rst_write(&qrd_ins, rst);
    // wait for done
    while(qrd_done_read(&qrd_ins) == 0);
#ifdef TIMING_TEST
    // Unload Matrix R
    R[0][0] = double(get_fixed_reinterpret(qrd_r1c1_read(&qrd_ins)));
    R[0][1] = double(get_fixed_reinterpret(qrd_r1c2_read(&qrd_ins)));
    R[0][2] = double(get_fixed_reinterpret(qrd_r1c3_read(&qrd_ins)));
    R[0][3] = double(get_fixed_reinterpret(qrd_r1c4_read(&qrd_ins)));
    R[1][1] = double(get_fixed_reinterpret(qrd_r2c2_read(&qrd_ins)));
    R[1][2] = double(get_fixed_reinterpret(qrd_r2c3_read(&qrd_ins)));
    R[1][3] = double(get_fixed_reinterpret(qrd_r2c4_read(&qrd_ins)));
    R[2][2] = double(get_fixed_reinterpret(qrd_r3c3_read(&qrd_ins)));
    R[2][3] = double(get_fixed_reinterpret(qrd_r3c4_read(&qrd_ins)));
    R[3][3] = double(get_fixed_reinterpret(qrd_r4c4_read(&qrd_ins)));
    // Unload Matrix Q
    Q[0][0] = double(get_fixed_reinterpret(qrd_r1c5_read(&qrd_ins)));
    Q[1][0] = double(get_fixed_reinterpret(qrd_r1c6_read(&qrd_ins)));
    Q[2][0] = double(get_fixed_reinterpret(qrd_r1c7_read(&qrd_ins)));
    Q[3][0] = double(get_fixed_reinterpret(qrd_r1c8_read(&qrd_ins)));
    Q[0][1] = double(get_fixed_reinterpret(qrd_r2c5_read(&qrd_ins)));
    Q[1][1] = double(get_fixed_reinterpret(qrd_r2c6_read(&qrd_ins)));
    Q[2][1] = double(get_fixed_reinterpret(qrd_r2c7_read(&qrd_ins)));
    Q[3][1] = double(get_fixed_reinterpret(qrd_r2c8_read(&qrd_ins)));
    Q[0][2] = double(get_fixed_reinterpret(qrd_r3c5_read(&qrd_ins)));
    Q[1][2] = double(get_fixed_reinterpret(qrd_r3c6_read(&qrd_ins)));
    Q[2][2] = double(get_fixed_reinterpret(qrd_r3c7_read(&qrd_ins)));
    Q[3][2] = double(get_fixed_reinterpret(qrd_r3c8_read(&qrd_ins)));
    Q[0][3] = double(get_fixed_reinterpret(qrd_r4c5_read(&qrd_ins)));
    Q[1][3] = double(get_fixed_reinterpret(qrd_r4c6_read(&qrd_ins)));

```

```

    Q[2][3] = double(get_fixed_reinterpret(qrd_r4c7_read(&qrd_ins)));
    Q[3][3] = double(get_fixed_reinterpret(qrd_r4c8_read(&qrd_ins)));
#endif
    // Unload Vector QT*B
    Y[0] = double(get_fixed_reinterpret(qrd_gateway_res_out_1_read(&qrd_ins)));
    Y[1] = double(get_fixed_reinterpret(qrd_gateway_res_out_2_read(&qrd_ins)));
    Y[2] = double(get_fixed_reinterpret(qrd_gateway_res_out_3_read(&qrd_ins)));
    Y[3] = double(get_fixed_reinterpret(qrd_gateway_res_out_4_read(&qrd_ins)));
#ifdef TIMING_TEST
    // Print Matrix A
    cout << endl << "Matrix A:" << endl;
    for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%10.5f\t", A[i][j]);
        }
        cout << endl;
    }
    // Print Matrix Q
    cout << endl << "Corresponding Matrix Q:" << endl;
    for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%10.5f\t", Q[i][j]);
        }
        cout << endl;
    }
    // Print Matrix R
    cout << endl << "Corresponding Matrix R:" << endl;
    for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%10.5f\t", R[i][j]);
        }
        cout << endl;
    }
    // Print Vector QT*B
    cout << endl << "Corresponding Vector QT*B:" << endl;
    for(int i = 0; i < 4; i++)
    {
        printf("%10.5f\t", Y[i]);
    }
    cout << endl << endl;
#endif
}

void calculate(double (*A)[4], double *B, double *X)
{
    double Q[4][4] = {0};
    double R[4][4] = {0};
    double Y[4];

    // Perform QR decomposition
    QRD_run(A,Q,R,B,Y);
    // Do the back substitution
    for (int j = 3; j >= 0; j--)
    {
        if (R[j][j] == 0)
        {
#ifdef TIMING_TEST
            cout << "Matrix is singular!" << endl;
#endif
        }
        X[j] = Y[j] / R[j][j];
        for (int i = 0; i <= j; i++)
        {
            Y[i] = Y[i] - R[i][j] * X[j];
        }
    }
#ifdef TIMING_TEST
    // Print the corresponding vector X
    cout << "Corresponding Vector X:" << endl;
    for (int i = 0; i < 4; i++)
    {
        printf("%8.4f\t", X[i]);
    }
    cout << endl;

```

```

#endif
}

// Main function
int main()
{
    cout << "--- Start of the Program ---" << endl;
#ifdef TIMING_TEST
    // Variables for timing and counts used for application cycle counts and timing
    unsigned long long tt;
    int tt_print;
    double tt_seconds, ps_time;
    double total_tt = 0;
    XTime start_time_co;
    XTime stop_time_co;
#endif
    // Define equation set 1
    double X[4] = { 0 };
    double A1[4][4] = { {1,1,4,3},
                        {2,7,6,5},
                        {9,10,3,11},
                        {16,1,14,13}
    };
    double B1[4] = {2,10,8,4};
    // Define equation set 2
    double A2[4][4] = { {0.1,0.1,0.4,0.3},
                        {0.2,0.8,0.6,0.5},
                        {0.9,0.1,0.3,0.2},
                        {0.3,0.1,0.4,0.6}
    };
    double B2[4] = {0.2, 0.1, 0.8, 0.4};
    // Define equation set 3
    double A3[4][4] = { {1,0,5,8},
                        {3,2,3,3},
                        {-1,-3,5,7},
                        {9,-5,3,6}
    };
    double B3[4] = {37, 34, 13, 26};
#ifdef TIMING_TEST
    // Get the starting time in cycle counts
    XTime_GetTime(&start_time_co);
#endif
    // Perform First Decomposition
#ifdef TIMING_TEST
    cout << endl << "*****Perform First Decomposition*****" << endl;
#endif
    calculate(A1,B1,X);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #1 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;
#endif
#ifdef TIMING_TEST
    // Get the starting time in cycle counts
    XTime_GetTime(&start_time_co);
#endif
    // Perform Second Decomposition
#ifdef TIMING_TEST
    cout << endl << "*****Perform Second Decomposition*****" << endl;
#endif
    calculate(A2,B2,X);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #2 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;

```

```

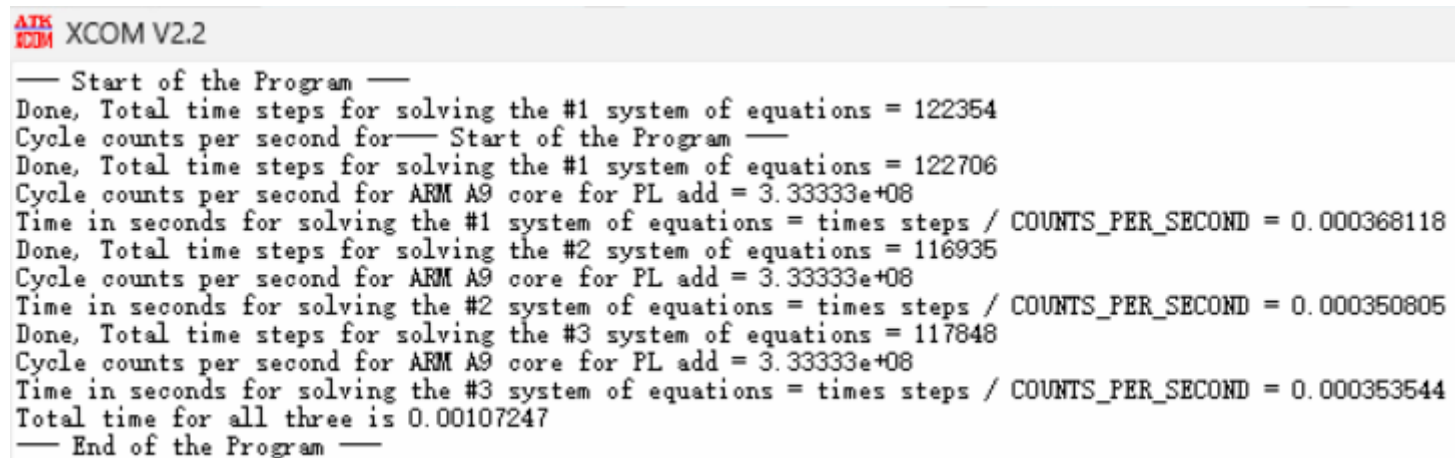
#endif
#ifdef TIMING_TEST
    // Get the starting time in cycle counts
    XTime_GetTime(&start_time_co);
#endif
    // Perform Third Decomposition
#endifdef TIMING_TEST
    cout << endl << "*****Perform Third Decomposition*****" << endl;
#endif
    calculate(A3,B3,X);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #3 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;

    cout << "Total time for all three is " << total_tt << endl;
#endif

    cout << "--- End of the Program ---" << endl;
    return 0;
}

```

- o Screenshot of the timing result



```

XCOM V2.2
--- Start of the Program ---
Done, Total time steps for solving the #1 system of equations = 122354
Cycle counts per second for--- Start of the Program ---
Done, Total time steps for solving the #1 system of equations = 122706
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = 0.000368118
Done, Total time steps for solving the #2 system of equations = 116935
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = 0.000350805
Done, Total time steps for solving the #3 system of equations = 117848
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = 0.000353544
Total time for all three is 0.00107247
--- End of the Program ---

```

- C++ testbench of the totally stand-alone C++ version

```

#include <cmath>
#include <iostream>
#include "ap_fixed.h"
#include "xparameters.h"
#include "xtime_l.h"
#include "xscugic.h"
#include <cstdlib>
#include <ctime>

#define TIMING_TEST
using namespace std;
// Define our fixed point type used here to match the PL accelerator
typedef ap_fixed<24,8, AP_RND, AP_SAT> FIXED_TYPE;

// Help function to print the 4*4 matrices
void print_matrix(FIXED_TYPE (*a)[4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            printf("%10.5f\t", (double)a[i][j]);
        }
        cout << endl;
    }
}

// Help function to print the vectors
void print_vector(FIXED_TYPE *a)
{
    for (int i = 0; i < 4; i++)
    {
        printf("%10.5f\t", (double)a[i]);
    }
}

```



```

    }
    cout << endl;
}

// Help function to print the 2*2 matrices
void print_rotation_matrix(FIXED_TYPE(*rotation_matrix)[2])
{
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            cout << rotation_matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

// Get the rotation matrix according to the angle
void get_rotation_matrix(FIXED_TYPE theta, FIXED_TYPE(*mat)[2])
{
    mat[0][0] = cos(double(theta));
    mat[0][1] = -sin(double(theta));
    mat[1][0] = sin(double(theta));
    mat[1][1] = cos(double(theta));
}

// Perform the rotation to matrices
void rotate(FIXED_TYPE(*a)[4], FIXED_TYPE(*rotation_matrix)[2],int row1, int row2)
{
    FIXED_TYPE a_res[4][4] = { 0 };
    a_res[row1][0] = rotation_matrix[0][0] * a[row1][0] + rotation_matrix[0][1] * a[row2][0];
    a_res[row1][1] = rotation_matrix[0][0] * a[row1][1] + rotation_matrix[0][1] * a[row2][1];
    a_res[row1][2] = rotation_matrix[0][0] * a[row1][2] + rotation_matrix[0][1] * a[row2][2];
    a_res[row1][3] = rotation_matrix[0][0] * a[row1][3] + rotation_matrix[0][1] * a[row2][3];

    a_res[row2][0] = rotation_matrix[1][0] * a[row1][0] + rotation_matrix[1][1] * a[row2][0];
    a_res[row2][1] = rotation_matrix[1][0] * a[row1][1] + rotation_matrix[1][1] * a[row2][1];
    a_res[row2][2] = rotation_matrix[1][0] * a[row1][2] + rotation_matrix[1][1] * a[row2][2];
    a_res[row2][3] = rotation_matrix[1][0] * a[row1][3] + rotation_matrix[1][1] * a[row2][3];

    for (int i = 0; i < 4; i++)
    {
        a[row1][i] = a_res[row1][i];
        a[row2][i] = a_res[row2][i];
    }
}

// Multiply a matrix with a vector
void matrix_vector_mult(FIXED_TYPE(*matrix)[4], FIXED_TYPE(*vector))
{
    FIXED_TYPE vector_res[4] = { 0 };
    vector_res[0] = matrix[0][0] * vector[0] + matrix[0][1] * vector[1] + matrix[0][2] * vector[2] + matrix[0][3] * vector[3];
    vector_res[1] = matrix[1][0] * vector[0] + matrix[1][1] * vector[1] + matrix[1][2] * vector[2] + matrix[1][3] * vector[3];
    vector_res[2] = matrix[2][0] * vector[0] + matrix[2][1] * vector[1] + matrix[2][2] * vector[2] + matrix[2][3] * vector[3];
    vector_res[3] = matrix[3][0] * vector[0] + matrix[3][1] * vector[1] + matrix[3][2] * vector[2] + matrix[3][3] * vector[3];
    for (int i = 0; i < 4; i++)
    {
        vector[i] = vector_res[i];
    }
}

// Perform the whole calculation to get the final result X
void calculate(FIXED_TYPE(*A)[4],FIXED_TYPE *B)
{
#ifdef TIMING_TEST
    cout << "Matrix A:" << endl;
    print_matrix(A);
#endif
    FIXED_TYPE E[4][4] = { {1,0,0,0},
                            {0,1,0,0},
                            {0,0,1,0},
                            {0,0,0,1} };
    FIXED_TYPE rotation_matrix[2][2] = { 0 };
    // rotation #1
    FIXED_TYPE theta = -atan((double)(A[1][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 0, 1);
    rotate(E, rotation_matrix, 0, 1);
    // rotation #2
    theta = -atan((double)(A[2][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);

```

```

rotate(A, rotation_matrix, 0, 2);
rotate(E, rotation_matrix, 0, 2);
// rotation #3
theta = -atan((double)(A[3][0] / A[0][0]));
get_rotation_matrix(theta, rotation_matrix);
rotate(A, rotation_matrix, 0, 3);
rotate(E, rotation_matrix, 0, 3);
// rotation #4
theta = -atan((double)(A[2][1] / A[1][1]));
get_rotation_matrix(theta, rotation_matrix);
rotate(A, rotation_matrix, 1, 2);
rotate(E, rotation_matrix, 1, 2);
// rotation #5
theta = -atan((double)(A[3][1] / A[1][1]));
get_rotation_matrix(theta, rotation_matrix);
rotate(A, rotation_matrix, 1, 3);
rotate(E, rotation_matrix, 1, 3);
// rotation #6
theta = -atan((double)(A[3][2] / A[2][2]));
get_rotation_matrix(theta, rotation_matrix);
rotate(A, rotation_matrix, 2, 3);
rotate(E, rotation_matrix, 2, 3);
// Print the result
FIXED_TYPE(*R)[4] = A;
FIXED_TYPE(*QT)[4] = E;
#ifdef TIMING_TEST
    cout << "Corresponding Matrix R:" << endl;
    print_matrix(R);
    cout << "Corresponding Matrix Q:" << endl;
    FIXED_TYPE Q[4][4];
    for(int i = 0; i<4; i++)
    {
        for(int j = 0; j<4; j++)
        {
            Q[i][j] = QT[j][i];
        }
    }
    print_matrix(Q);
#endif
// Calculate QT*B
matrix_vector_mult(QT, B);
#ifdef TIMING_TEST
    cout << "Corresponding Vector QT*B:" << endl;
    print_vector(B);
#endif
// Check if the matrix is singular
FIXED_TYPE X[4] = { 0 };
for (int j = 3; j >= 0; j--)
{
    if (R[j][j] == 0)
    {
#ifdef TIMING_TEST
        cout << "Matrix is singular!" << endl;
#endif
    }
    X[j] = B[j] / R[j][j];
    for (int i = 0; i <= j; i++)
    {
        B[i] = B[i] - R[i][j] * X[j];
    }
}
#ifdef TIMING_TEST
    cout << "Corresponding Vector X:" << endl;
    print_vector(X);
#endif
}

int main()
{
    cout << "-----Start of the Program-----" << endl;
#ifdef TIMING_TEST
    // Variables for timing and counts used for application cycle counts and timing
    unsigned long long tt;
    int tt_print;
    double total_tt = 0;
    double tt_seconds, ps_time;
    XTime start_time_co;
    XTime stop_time_co;
    // Get the starting time in cycle counts
#endif
    // Define equation set 1

```

```

    FIXED_TYPE A1[4][4] = { {1,1,4,3},
                           {2,7,6,5},
                           {9,10,3,11},
                           {16,1,14,13}
    };
    FIXED_TYPE B1[4] = {2,10,8,4};
    // Define equation set 2
    FIXED_TYPE A2[4][4] = { {0.1,0.1,0.4,0.3},
                           {0.2,0.8,0.6,0.5},
                           {0.9,0.1,0.3,0.2},
                           {0.3,0.1,0.4,0.6}
    };
    FIXED_TYPE B2[4] = {0.2, 0.1, 0.8, 0.4};
    // Define equation set 3
    FIXED_TYPE A3[4][4] = { {1,0,5,8},
                           {3,2,3,3},
                           {-1,-3,5,7},
                           {9,-5,3,6}
    };
    FIXED_TYPE B3[4] = {37, 34, 13, 26};
#ifdef TIMING_TEST
    // Capture the start time on the processor
    XTime_GetTime(&start_time_co);
#endif
#ifdef TIMING_TEST
    cout << endl << "*****Perform First Decomposition*****" << endl;
#endif
    calculate(A1,B1);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #1 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;
#endif
#ifdef TIMING_TEST
    // Capture the start time on the processor
    XTime_GetTime(&start_time_co);
#endif
#ifdef TIMING_TEST
    cout << endl << "*****Perform Second Decomposition*****" << endl;
#endif
    calculate(A2,B2);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #2 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
    total_tt += ps_time;
#endif
#ifdef TIMING_TEST
    // Capture the start time on the processor
    XTime_GetTime(&start_time_co);
#endif
#ifdef TIMING_TEST
    cout << endl << "*****Perform Third Decomposition*****" << endl;
#endif
    calculate(A3,B3);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
    tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
    cout << "Done, Total time steps for solving the #3 system of equations = " << tt_print << endl;
    tt_seconds = COUNTS_PER_SECOND;
    cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
    ps_time = (float) tt_print / tt_seconds;
    cout << "Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;

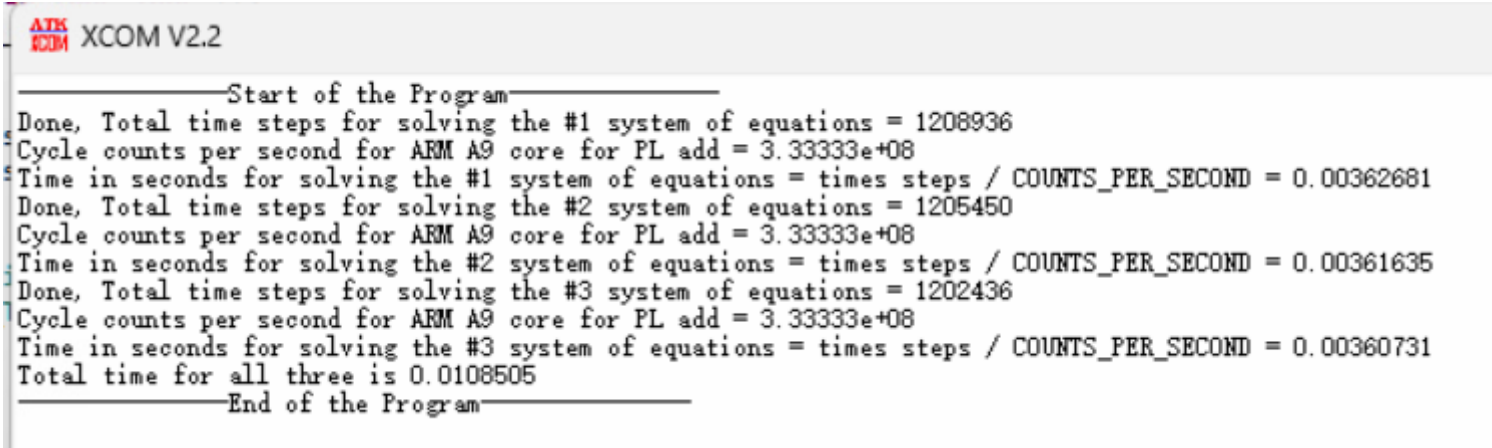
```

```
total_tt += ps_time;

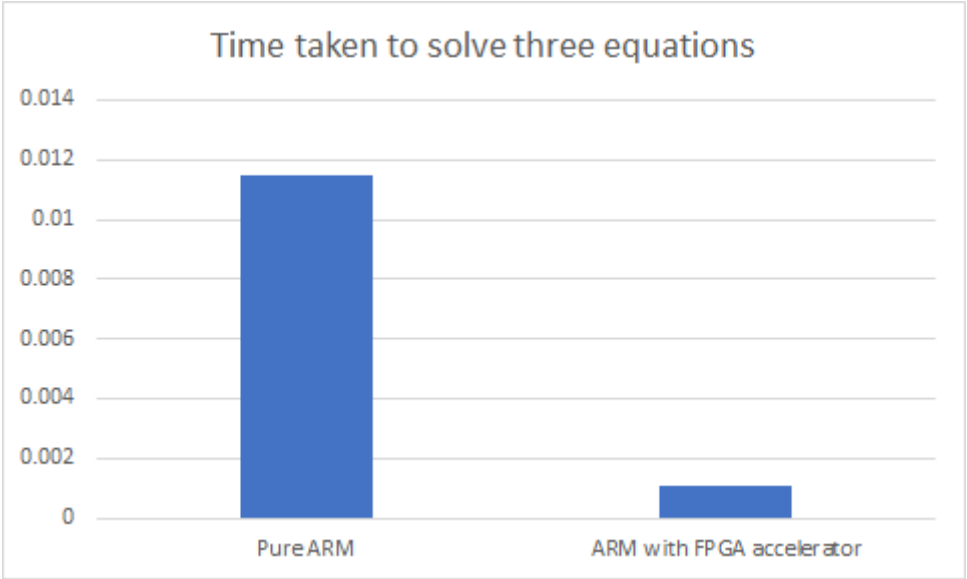
cout << "Total time for all three is " << total_tt << endl;
#endif
cout << "-----End of the Program-----" << endl;
return 0;

}
```

- Screenshot of the timing result



- Tradeoff
 - If only the ARM core is used for calculation, it will consume more than 10 times the time, but it will not occupy any FPGA resources. If you use the pipeline and AXI Stream interface, the speed should be further improved.
 - If the project uses the FPGA-accelerated version, it will occupy 41% of LUT, 2% of LUTRAM, 17% of FF and 4% of DSP. This will not only take up more resources, but also increase the power consumption of the chip.
 - The following is a comparison of the time used to solve the three equations.
 - Which method to use depends on the sensitivity of the specific application to time and resources.

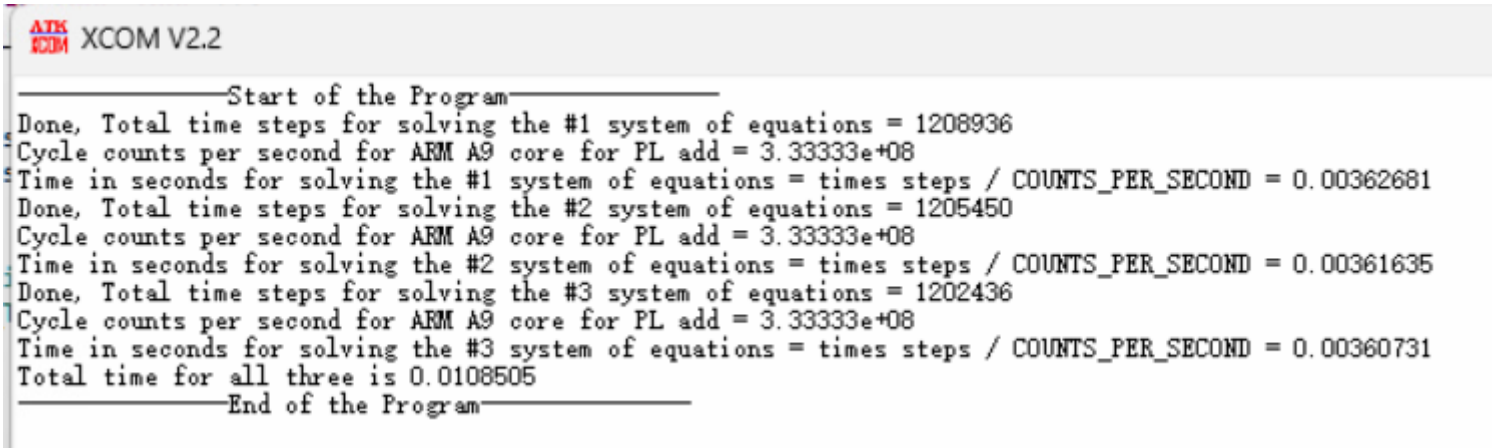


6. Demonstration of the array being able to solve three different instances of systems of equations for both the stand-alone C++ versions and also the FPGA SoC PL accelerated version.

- Documentation of the timings for each system of equations, and the total time for all three for both the stand-alone C++ versions and also the FPGA SoC PL accelerated version

It turns out that solving each equation took about the same amount of time. But using the FPGA-accelerated version will be about 10 times faster than the only ARM core version.

- Timing test result of the stand-alone C++ version



- Timing test result of the FPGA SoC PL accelerated version

```

— Start of the Program —
Done, Total time steps for solving the #1 system of equations = 122354
Cycle counts per second for— Start of the Program —
Done, Total time steps for solving the #1 system of equations = 122706
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = 0.000368118
Done, Total time steps for solving the #2 system of equations = 116935
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = 0.000350805
Done, Total time steps for solving the #3 system of equations = 117848
Cycle counts per second for ARM A9 core for PL add = 3.33333e+08
Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = 0.000353544
Total time for all three is 0.00107247
— End of the Program —

```

- The stand-alone C++ version
 - C++ code

```

#include <cmath>
#include <iostream>
#include "ap_fixed.h"
#include "xparameters.h"
#include "xtime_l.h"
#include "xscugic.h"
#include <cstdlib>
#include <ctime>

// #define TIMING_TEST
using namespace std;
// Define our fixed point type used here to match the PL accelerator
typedef ap_fixed<24,8, AP_RND, AP_SAT> FIXED_TYPE;

// Help function to print the 4*4 matrices
void print_matrix(FIXED_TYPE (*a)[4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            printf("%10.5f\t", (double)a[i][j]);
        }
        cout << endl;
    }
}

// Help function to print the vectors
void print_vector(FIXED_TYPE *a)
{
    for (int i = 0; i < 4; i++)
    {
        printf("%10.5f\t", (double)a[i]);
    }
    cout << endl;
}

// Help function to print the 2*2 matrices
void print_rotation_matrix(FIXED_TYPE(*rotation_matrix)[2])
{
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            cout << rotation_matrix[i][j] << "\t";
        }
        cout << endl;
    }
}

// Get the rotation matrix according to the angle
void get_rotation_matrix(FIXED_TYPE theta, FIXED_TYPE(*mat)[2])
{
    mat[0][0] = cos(double(theta));
    mat[0][1] = -sin(double(theta));
    mat[1][0] = sin(double(theta));
    mat[1][1] = cos(double(theta));
}

// Perform the rotation to matrices
void rotate(FIXED_TYPE(*a)[4], FIXED_TYPE(*rotation_matrix)[2], int row1, int row2)
{
    FIXED_TYPE a_res[4][4] = { 0 };
    a_res[row1][0] = rotation_matrix[0][0] * a[row1][0] + rotation_matrix[0][1] * a[row2][0];

```



```

a_res[row1][1] = rotation_matrix[0][0] * a[row1][1] + rotation_matrix[0][1] * a[row2][1];
a_res[row1][2] = rotation_matrix[0][0] * a[row1][2] + rotation_matrix[0][1] * a[row2][2];
a_res[row1][3] = rotation_matrix[0][0] * a[row1][3] + rotation_matrix[0][1] * a[row2][3];

a_res[row2][0] = rotation_matrix[1][0] * a[row1][0] + rotation_matrix[1][1] * a[row2][0];
a_res[row2][1] = rotation_matrix[1][0] * a[row1][1] + rotation_matrix[1][1] * a[row2][1];
a_res[row2][2] = rotation_matrix[1][0] * a[row1][2] + rotation_matrix[1][1] * a[row2][2];
a_res[row2][3] = rotation_matrix[1][0] * a[row1][3] + rotation_matrix[1][1] * a[row2][3];

for (int i = 0; i < 4; i++)
{
    a[row1][i] = a_res[row1][i];
    a[row2][i] = a_res[row2][i];
}
}

// Multiply a matrix with a vector
void matrix_vector_mult(FIXED_TYPE(*matrix)[4], FIXED_TYPE(*vector))
{
    FIXED_TYPE vector_res[4] = { 0 };
    vector_res[0] = matrix[0][0] * vector[0] + matrix[0][1] * vector[1] + matrix[0][2] * vector[2] + matrix[0][3] * vector[3];
    vector_res[1] = matrix[1][0] * vector[0] + matrix[1][1] * vector[1] + matrix[1][2] * vector[2] + matrix[1][3] * vector[3];
    vector_res[2] = matrix[2][0] * vector[0] + matrix[2][1] * vector[1] + matrix[2][2] * vector[2] + matrix[2][3] * vector[3];
    vector_res[3] = matrix[3][0] * vector[0] + matrix[3][1] * vector[1] + matrix[3][2] * vector[2] + matrix[3][3] * vector[3];
    for (int i = 0; i < 4; i++)
    {
        vector[i] = vector_res[i];
    }
}

// Perform the whole calculation to get the final result x
void calculate(FIXED_TYPE(*A)[4], FIXED_TYPE *B)
{
#ifdef TIMING_TEST
    cout << "Matrix A:" << endl;
    print_matrix(A);
#endif
    FIXED_TYPE E[4][4] = { {1,0,0,0},
                           {0,1,0,0},
                           {0,0,1,0},
                           {0,0,0,1} };
    FIXED_TYPE rotation_matrix[2][2] = { 0 };
    // rotation #1
    FIXED_TYPE theta = -atan((double)(A[1][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 0, 1);
    rotate(E, rotation_matrix, 0, 1);
    // rotation #2
    theta = -atan((double)(A[2][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 0, 2);
    rotate(E, rotation_matrix, 0, 2);
    // rotation #3
    theta = -atan((double)(A[3][0] / A[0][0]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 0, 3);
    rotate(E, rotation_matrix, 0, 3);
    // rotation #4
    theta = -atan((double)(A[2][1] / A[1][1]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 1, 2);
    rotate(E, rotation_matrix, 1, 2);
    // rotation #5
    theta = -atan((double)(A[3][1] / A[1][1]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 1, 3);
    rotate(E, rotation_matrix, 1, 3);
    // rotation #6
    theta = -atan((double)(A[3][2] / A[2][2]));
    get_rotation_matrix(theta, rotation_matrix);
    rotate(A, rotation_matrix, 2, 3);
    rotate(E, rotation_matrix, 2, 3);
    // Print the result
    FIXED_TYPE(*R)[4] = A;
    FIXED_TYPE(*QT)[4] = E;
#ifdef TIMING_TEST
    cout << "Corresponding Matrix R:" << endl;
    print_matrix(R);
    cout << "Corresponding Matrix Q:" << endl;
    FIXED_TYPE Q[4][4];
    for(int i = 0; i<4; i++)

```

```

    {
        for(int j = 0; j<4; j++)
        {
            Q[i][j] = QT[j][i];
        }
    }
    print_matrix(Q);
#endif
    // Calculate QT*B
    matrix_vector_mult(QT, B);
#ifdef TIMING_TEST
    cout << "Corresponding Vector QT*B:" << endl;
    print_vector(B);
#endif
    // Check if the matrix is singular
    FIXED_TYPE X[4] = { 0 };
    for (int j = 3; j >= 0; j--)
    {
        if (R[j][j] == 0)
        {
#ifdef TIMING_TEST
            cout << "Matrix is singular!" << endl;
#endif
        }
        X[j] = B[j] / R[j][j];
        for (int i = 0; i <= j; i++)
        {
            B[i] = B[i] - R[i][j] * X[j];
        }
    }
#ifdef TIMING_TEST
    cout << "Corresponding Vector X:" << endl;
    print_vector(X);
#endif
}

int main()
{
    cout << "-----Start of the Program-----" << endl;
#ifdef TIMING_TEST
    // Variables for timing and counts used for application cycle counts and timing
    unsigned long long tt;
    int tt_print;
    double total_tt = 0;
    double tt_seconds, ps_time;
    XTime start_time_co;
    XTime stop_time_co;
    // Get the starting time in cycle counts
#endif
    // Define equation set 1
    FIXED_TYPE A1[4][4] = { {1,1,4,3},
                            {2,7,6,5},
                            {9,10,3,11},
                            {16,1,14,13}
    };
    FIXED_TYPE B1[4] = {2,10,8,4};
    // Define equation set 2
    FIXED_TYPE A2[4][4] = { {0.1,0.1,0.4,0.3},
                            {0.2,0.8,0.6,0.5},
                            {0.9,0.1,0.3,0.2},
                            {0.3,0.1,0.4,0.6}
    };
    FIXED_TYPE B2[4] = {0.2, 0.1, 0.8, 0.4};
    // Define equation set 3
    FIXED_TYPE A3[4][4] = { {1,0,5,8},
                            {3,2,3,3},
                            {-1,-3,5,7},
                            {9,-5,3,6}
    };
    FIXED_TYPE B3[4] = {37, 34, 13, 26};
#ifdef TIMING_TEST
    // Capture the start time on the processor
    XTime_GetTime(&start_time_co);
#endif
#ifdef TIMING_TEST
    cout << endl << "*****Perform First Decomposition*****" << endl;
#endif
    calculate(A1,B1);
#ifdef TIMING_TEST
    // Capture the stop time on the processor
    XTime_GetTime(&stop_time_co);

```

```

// Compute timing on PL hardware using the accelerator
tt = stop_time_co - start_time_co;
tt_print = (unsigned) tt;
cout << "Done, Total time steps for solving the #1 system of equations = " << tt_print << endl;
tt_seconds = COUNTS_PER_SECOND;
cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
ps_time = (float) tt_print / tt_seconds;
cout << "Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
total_tt += ps_time;
#endif
#ifdef TIMING_TEST
// Capture the start time on the processor
XTime_GetTime(&start_time_co);
#endif;
#ifndef TIMING_TEST
cout << endl << "*****perform Second Decomposition*****" << endl;
#endif
calculate(A2,B2);
#ifdef TIMING_TEST
// Capture the stop time on the processor
XTime_GetTime(&stop_time_co);
// Compute timing on PL hardware using the accelerator
tt = stop_time_co - start_time_co;
tt_print = (unsigned) tt;
cout << "Done, Total time steps for solving the #2 system of equations = " << tt_print << endl;
tt_seconds = COUNTS_PER_SECOND;
cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
ps_time = (float) tt_print / tt_seconds;
cout << "Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
total_tt += ps_time;
#endif
#ifdef TIMING_TEST
// Capture the start time on the processor
XTime_GetTime(&start_time_co);
#endif
#ifndef TIMING_TEST
cout << endl << "*****perform Third Decomposition*****" << endl;
#endif
calculate(A3,B3);
#ifdef TIMING_TEST
// Capture the stop time on the processor
XTime_GetTime(&stop_time_co);
// Compute timing on PL hardware using the accelerator
tt = stop_time_co - start_time_co;
tt_print = (unsigned) tt;
cout << "Done, Total time steps for solving the #3 system of equations = " << tt_print << endl;
tt_seconds = COUNTS_PER_SECOND;
cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
ps_time = (float) tt_print / tt_seconds;
cout << "Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
total_tt += ps_time;

cout << "Total time for all three is " << total_tt << endl;
#endif
cout << "-----End of the Program-----" << endl;
return 0;
}

```

o Result

```

-----Start of the Program-----

*****Perform First Decomposition*****
Matrix A:
  1.00000    1.00000    4.00000    3.00000
  2.00000    7.00000    6.00000    5.00000
  9.00000   10.00000    3.00000   11.00000
 16.00000    1.00000   14.00000   13.00000
Corresponding Matrix R:
 18.49324    6.54279   14.43774   17.30356
  0.00017   10.40157   -0.42914    4.59412
  0.00008    0.00005   -6.95465   -1.31520
 -0.00003   -0.00003   -0.00002   -1.32310
Corresponding Matrix Q:
  0.05408    0.06212   -0.46672   -0.88055
  0.10814    0.60497   -0.67554    0.40738
  0.48665    0.65529    0.53850   -0.20930
  0.86519   -0.44809   -0.18929    0.12184
Corresponding Vector QT*B:
  8.54349    9.62384   -4.13797    1.12561
Corresponding Vector X:

```

```

0.19656    1.33215    0.75586    -0.85072

*****Perform Second Decomposition*****
Matrix A:
0.10001    0.10001    0.39999    0.30000
0.20000    0.80000    0.60001    0.50000
0.89999    0.10001    0.30000    0.20000
0.30000    0.10001    0.39999    0.60001
Corresponding Matrix R:
0.97467    0.29753    0.56429    0.50273
0.00000    0.76253    0.55354    0.47264
0.00000    0.00000    -0.38098    -0.45847
0.00000    0.00000    0.00000    0.23167
Corresponding Matrix Q:
0.10260    0.09109    -0.76553    -0.62857
0.20518    0.96904    0.13702    0.00708
0.92337    -0.22914    0.24727    -0.18364
0.30780    0.01103    -0.57796    0.75571
Corresponding Vector QT*B:
0.90286    -0.06377    -0.17276    0.03036
Corresponding Vector X:
0.80336    -0.37953    0.29573    0.13106

*****Perform Third Decomposition*****
Matrix A:
1.00000    0.00000    5.00000    8.00000
3.00000    2.00000    3.00000    3.00000
-1.00000   -3.00000    5.00000    7.00000
9.00000   -5.00000    3.00000    6.00000
Corresponding Matrix R:
9.59160   -3.75323    3.75325    6.67244
0.00002    4.89009   -2.02725   -4.08113
0.00002   -0.00005   -7.05725   -9.73257
-0.00003   -0.00003   -0.00005    1.45009
Corresponding Matrix Q:
0.10426    0.08002   -0.67604    0.72507
0.31277    0.64905   -0.44519   -0.53172
-0.10426   -0.69351   -0.56474   -0.43503
0.93831   -0.30229    0.16077    0.04834
Corresponding Vector QT*B:
37.53268    8.15302   -43.31171    4.35037
Corresponding Vector X:
3.00006    5.00006    1.99983    3.00006
-----End of the Program-----

```

- The FPGA SoC PL accelerated version
 - C++ code

```

#include <cmath>
#include <iostream>
#include "ap_fixed.h"
#include "xparameters.h"
#include "xtime_l.h"
#include "xscugic.h"
#include "qrd.h"

using namespace std;
#define TIMING_TEST

// Define our fixed point type used here to match the PL accelerator
typedef ap_fixed<24,8, AP_RND, AP_SAT> FIXED_TYPE;

// Create an instance of qrd
qrd qrd_ins;

// The AXI interface sends only "raw bits" so we need to convert standard data types we use to
// match the ap_fixed data type in the PL accelerator.
// Using C++ functions per - RJ Cunningham
int get_int_reinterpret(FIXED_TYPE x) {
    return *(reinterpret_cast<int *>(&x));
}

FIXED_TYPE get_fixed_reinterpret(int x) {
    return *(reinterpret_cast<FIXED_TYPE *>(&x));
}

// Initialize the hardware and perform calculations
void QRD_run(double (*A)[4], double (*Q)[4], double (*R)[4], double *B, double *Y)
{

```

```

// values for simulation and testing using C++ format with ap_fixed
FIXED_TYPE a11 = A[0][0];
FIXED_TYPE a21 = A[1][0];
FIXED_TYPE a31 = A[2][0];
FIXED_TYPE a41 = A[3][0];
FIXED_TYPE a12 = A[0][1];
FIXED_TYPE a22 = A[1][1];
FIXED_TYPE a32 = A[2][1];
FIXED_TYPE a42 = A[3][1];
FIXED_TYPE a13 = A[0][2];
FIXED_TYPE a23 = A[1][2];
FIXED_TYPE a33 = A[2][2];
FIXED_TYPE a43 = A[3][2];
FIXED_TYPE a14 = A[0][3];
FIXED_TYPE a24 = A[1][3];
FIXED_TYPE a34 = A[2][3];
FIXED_TYPE a44 = A[3][3];
FIXED_TYPE b1 = B[0];
FIXED_TYPE b2 = B[1];
FIXED_TYPE b3 = B[2];
FIXED_TYPE b4 = B[3];

// Need to reinterpret the 16 bit ap_fixed values into raw unsigned 32 bit value for AXI transfer
unsigned int a11_u32 = get_int_reinterpret(a11);
unsigned int a21_u32 = get_int_reinterpret(a21);
unsigned int a31_u32 = get_int_reinterpret(a31);
unsigned int a41_u32 = get_int_reinterpret(a41);
unsigned int a12_u32 = get_int_reinterpret(a12);
unsigned int a22_u32 = get_int_reinterpret(a22);
unsigned int a32_u32 = get_int_reinterpret(a32);
unsigned int a42_u32 = get_int_reinterpret(a42);
unsigned int a13_u32 = get_int_reinterpret(a13);
unsigned int a23_u32 = get_int_reinterpret(a23);
unsigned int a33_u32 = get_int_reinterpret(a33);
unsigned int a43_u32 = get_int_reinterpret(a43);
unsigned int a14_u32 = get_int_reinterpret(a14);
unsigned int a24_u32 = get_int_reinterpret(a24);
unsigned int a34_u32 = get_int_reinterpret(a34);
unsigned int a44_u32 = get_int_reinterpret(a44);
unsigned int b1_u32 = get_int_reinterpret(b1);
unsigned int b2_u32 = get_int_reinterpret(b2);
unsigned int b3_u32 = get_int_reinterpret(b3);
unsigned int b4_u32 = get_int_reinterpret(b4);

// Define reset signal
bool rst = 1;

// Initialize instance
qrd_initialize(&qrd_ins, XPAR_QRD_0_DEVICE_ID);
// Set reset to valid
qrd_global_rst_write(&qrd_ins, rst);
// Load matrix for decomposition
qrd_a11_write(&qrd_ins, a11_u32);
qrd_a21_write(&qrd_ins, a21_u32);
qrd_a31_write(&qrd_ins, a31_u32);
qrd_a41_write(&qrd_ins, a41_u32);
qrd_a12_write(&qrd_ins, a12_u32);
qrd_a22_write(&qrd_ins, a22_u32);
qrd_a32_write(&qrd_ins, a32_u32);
qrd_a42_write(&qrd_ins, a42_u32);
qrd_a13_write(&qrd_ins, a13_u32);
qrd_a23_write(&qrd_ins, a23_u32);
qrd_a33_write(&qrd_ins, a33_u32);
qrd_a43_write(&qrd_ins, a43_u32);
qrd_a14_write(&qrd_ins, a14_u32);
qrd_a24_write(&qrd_ins, a24_u32);
qrd_a34_write(&qrd_ins, a34_u32);
qrd_a44_write(&qrd_ins, a44_u32);
qrd_b1_write(&qrd_ins, b1_u32);
qrd_b2_write(&qrd_ins, b2_u32);
qrd_b3_write(&qrd_ins, b3_u32);
qrd_b4_write(&qrd_ins, b4_u32);
// Set reset to invalid
rst = 0;
qrd_global_rst_write(&qrd_ins, rst);
// wait for done
while(qrd_done_read(&qrd_ins) == 0);
#endif TIMING_TEST
// Unload Matrix R
R[0][0] = double(get_fixed_reinterpret(qrd_r1c1_read(&qrd_ins)));
R[0][1] = double(get_fixed_reinterpret(qrd_r1c2_read(&qrd_ins)));

```

```

R[0][2] = double(get_fixed_reinterpret(qrd_r1c3_read(&qrd_ins)));
R[0][3] = double(get_fixed_reinterpret(qrd_r1c4_read(&qrd_ins)));
R[1][1] = double(get_fixed_reinterpret(qrd_r2c2_read(&qrd_ins)));
R[1][2] = double(get_fixed_reinterpret(qrd_r2c3_read(&qrd_ins)));
R[1][3] = double(get_fixed_reinterpret(qrd_r2c4_read(&qrd_ins)));
R[2][2] = double(get_fixed_reinterpret(qrd_r3c3_read(&qrd_ins)));
R[2][3] = double(get_fixed_reinterpret(qrd_r3c4_read(&qrd_ins)));
R[3][3] = double(get_fixed_reinterpret(qrd_r4c4_read(&qrd_ins)));
// Unload Matrix Q
Q[0][0] = double(get_fixed_reinterpret(qrd_r1c5_read(&qrd_ins)));
Q[1][0] = double(get_fixed_reinterpret(qrd_r1c6_read(&qrd_ins)));
Q[2][0] = double(get_fixed_reinterpret(qrd_r1c7_read(&qrd_ins)));
Q[3][0] = double(get_fixed_reinterpret(qrd_r1c8_read(&qrd_ins)));
Q[0][1] = double(get_fixed_reinterpret(qrd_r2c5_read(&qrd_ins)));
Q[1][1] = double(get_fixed_reinterpret(qrd_r2c6_read(&qrd_ins)));
Q[2][1] = double(get_fixed_reinterpret(qrd_r2c7_read(&qrd_ins)));
Q[3][1] = double(get_fixed_reinterpret(qrd_r2c8_read(&qrd_ins)));
Q[0][2] = double(get_fixed_reinterpret(qrd_r3c5_read(&qrd_ins)));
Q[1][2] = double(get_fixed_reinterpret(qrd_r3c6_read(&qrd_ins)));
Q[2][2] = double(get_fixed_reinterpret(qrd_r3c7_read(&qrd_ins)));
Q[3][2] = double(get_fixed_reinterpret(qrd_r3c8_read(&qrd_ins)));
Q[0][3] = double(get_fixed_reinterpret(qrd_r4c5_read(&qrd_ins)));
Q[1][3] = double(get_fixed_reinterpret(qrd_r4c6_read(&qrd_ins)));
Q[2][3] = double(get_fixed_reinterpret(qrd_r4c7_read(&qrd_ins)));
Q[3][3] = double(get_fixed_reinterpret(qrd_r4c8_read(&qrd_ins)));
#endif

// Unload Vector QT*B
Y[0] = double(get_fixed_reinterpret(qrd_gateway_res_out_1_read(&qrd_ins)));
Y[1] = double(get_fixed_reinterpret(qrd_gateway_res_out_2_read(&qrd_ins)));
Y[2] = double(get_fixed_reinterpret(qrd_gateway_res_out_3_read(&qrd_ins)));
Y[3] = double(get_fixed_reinterpret(qrd_gateway_res_out_4_read(&qrd_ins)));
#endifdef TIMING_TEST

// Print Matrix A
cout << endl << "Matrix A:" << endl;
for(int i = 0; i < 4; i++)
{
    for(int j = 0; j < 4; j++)
    {
        printf("%10.5f\t", A[i][j]);
    }
    cout << endl;
}

// Print Matrix Q
cout << endl << "Corresponding Matrix Q:" << endl;
for(int i = 0; i < 4; i++)
{
    for(int j = 0; j < 4; j++)
    {
        printf("%10.5f\t", Q[i][j]);
    }
    cout << endl;
}

// Print Matrix R
cout << endl << "Corresponding Matrix R:" << endl;
for(int i = 0; i < 4; i++)
{
    for(int j = 0; j < 4; j++)
    {
        printf("%10.5f\t", R[i][j]);
    }
    cout << endl;
}

// Print Vector QT*B
cout << endl << "Corresponding Vector QT*B:" << endl;
for(int i = 0; i < 4; i++)
{
    printf("%10.5f\t", Y[i]);
}
cout << endl << endl;
#endif
}

void calculate(double (*A)[4], double *B, double *X)
{
    double Q[4][4] = {0};
    double R[4][4] = {0};
    double Y[4];

    // Perform QR decomposition
    QRD_run(A,Q,R,B,Y);

```



```

// Do the back substitution
for (int j = 3; j >= 0; j--)
{
    if (R[j][j] == 0)
    {
#ifdef TIMING_TEST
        cout << "Matrix is singular!" << endl;
#endif
    }
    X[j] = Y[j] / R[j][j];
    for (int i = 0; i <= j; i++)
    {
        Y[i] = Y[i] - R[i][j] * X[j];
    }
}

#ifdef TIMING_TEST
// Print the corresponding vector X
cout << "Corresponding Vector X:" << endl;
for (int i = 0; i < 4; i++)
{
    printf("%8.4f\t", X[i]);
}
cout << endl;
#endif
}

// Main function
int main()
{
    cout << "--- Start of the Program ---" << endl;
#ifdef TIMING_TEST
// Variables for timing and counts used for application cycle counts and timing
unsigned long long tt;
int tt_print;
double tt_seconds, ps_time;
double total_tt = 0;
XTime start_time_co;
XTime stop_time_co;
#endif
// Define equation set 1
double X[4] = { 0 };
double A1[4][4] = { {1,1,4,3},
                    {2,7,6,5},
                    {9,10,3,11},
                    {16,1,14,13}
};
double B1[4] = {2,10,8,4};
// Define equation set 2
double A2[4][4] = { {0.1,0.1,0.4,0.3},
                    {0.2,0.8,0.6,0.5},
                    {0.9,0.1,0.3,0.2},
                    {0.3,0.1,0.4,0.6}
};
double B2[4] = {0.2, 0.1, 0.8, 0.4};
// Define equation set 3
double A3[4][4] = { {1,0,5,8},
                    {3,2,3,3},
                    {-1,-3,5,7},
                    {9,-5,3,6}
};
double B3[4] = {37, 34, 13, 26};
#ifdef TIMING_TEST
// Get the starting time in cycle counts
XTime_GetTime(&start_time_co);
#endif
// Perform First Decomposition
#ifdef TIMING_TEST
cout << endl << "*****Perform First Decomposition*****" << endl;
#endif
calculate(A1,B1,X);
#ifdef TIMING_TEST
// Capture the stop time on the processor
XTime_GetTime(&stop_time_co);
// Compute timing on PL hardware using the accelerator
tt = stop_time_co - start_time_co;
tt_print = (unsigned) tt;
cout << "Done, Total time steps for solving the #1 system of equations = " << tt_print << endl;
tt_seconds = COUNTS_PER_SECOND;
cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
ps_time = (float) tt_print / tt_seconds;
cout << "Time in seconds for solving the #1 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;

```

```

        total_tt += ps_time;
    #endif
    #ifdef TIMING_TEST
        // Get the starting time in cycle counts
        XTime_GetTime(&start_time_co);
    #endif
        // Perform Second Decomposition
    #ifndef TIMING_TEST
        cout << endl << "*****Perform Second Decomposition*****" << endl;
    #endif
        calculate(A2,B2,X);
    #ifdef TIMING_TEST
        // Capture the stop time on the processor
        XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
        tt = stop_time_co - start_time_co;
    tt_print = (unsigned) tt;
        cout << "Done, Total time steps for solving the #2 system of equations = " << tt_print << endl;
        tt_seconds = COUNTS_PER_SECOND;
        cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
        ps_time = (float) tt_print / tt_seconds;
        cout << "Time in seconds for solving the #2 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
        total_tt += ps_time;
    #endif
    #ifdef TIMING_TEST
        // Get the starting time in cycle counts
        XTime_GetTime(&start_time_co);
    #endif
        // Perform Third Decomposition
    #ifndef TIMING_TEST
        cout << endl << "*****Perform Third Decomposition*****" << endl;
    #endif
        calculate(A3,B3,X);
    #ifdef TIMING_TEST
        // Capture the stop time on the processor
        XTime_GetTime(&stop_time_co);
    // Compute timing on PL hardware using the accelerator
        tt = stop_time_co - start_time_co;
        tt_print = (unsigned) tt;
        cout << "Done, Total time steps for solving the #3 system of equations = " << tt_print << endl;
        tt_seconds = COUNTS_PER_SECOND;
        cout << "Cycle counts per second for ARM A9 core for PL add = " << tt_seconds << endl;
        ps_time = (float) tt_print / tt_seconds;
        cout << "Time in seconds for solving the #3 system of equations = times steps / COUNTS_PER_SECOND = " << ps_time << endl;
        total_tt += ps_time;

        cout << "Total time for all three is " << total_tt << endl;
    #endif

    cout << "--- End of the Program ---" << endl;
    return 0;
}

```

o Result

```

--- Start of the Program ---

*****Perform First Decomposition*****

Matrix A:
    1.00000    1.00000    4.00000    3.00000
    2.00000    7.00000    6.00000    5.00000
    9.00000   10.00000    3.00000   11.00000
   16.00000    1.00000   14.00000   13.00000

Corresponding Matrix Q:
    0.05403    0.06201    0.46677   -0.88072
    0.10811    0.60493    0.67555    0.40729
    0.48657    0.65532   -0.53853   -0.20934
    0.86519   -0.44804    0.18922    0.12183

Corresponding Matrix R:
   18.49330    6.54210   14.43793   17.30327
    0.00000   10.40207   -0.42807    4.59534
    0.00000    0.00000    6.95442    1.31487
    0.00000    0.00000    0.00000   -1.32368

Corresponding Vector QT*B:
    8.54248    9.62375    4.13773    1.12405

```

```

Corresponding Vector X:
0.1956      1.3314      0.7555      -0.8492

*****Perform Second Decomposition*****

Matrix A:
0.10000      0.10000      0.40000      0.30000
0.20000      0.80000      0.60000      0.50000
0.90000      0.10000      0.30000      0.20000
0.30000      0.10000      0.40000      0.60000

Corresponding Matrix Q:
0.10265      0.09108      0.76566      -0.62863
0.20514      0.96904      -0.13702      0.00694
0.92334      -0.22914      -0.24733      -0.18362
0.30786      0.01109      0.57785      0.75574

Corresponding Matrix R:
0.97470      0.29747      0.56425      0.50270
0.00000      0.76259      0.55357      0.47264
0.00000      0.00000      0.38097      0.45840
0.00000      0.00000      0.00000      0.23161

Corresponding Vector QT*B:
0.90285      -0.06375      0.17268      0.03035

Corresponding Vector X:
0.8034      -0.3794      0.2956      0.1310

*****Perform Third Decomposition*****

Matrix A:
1.00000      0.00000      5.00000      8.00000
3.00000      2.00000      3.00000      3.00000
-1.00000     -3.00000      5.00000      7.00000
9.00000     -5.00000      3.00000      6.00000

Corresponding Matrix Q:
0.10419      0.07993      0.67598      0.72510
0.31268      0.64906      0.44524      -0.53169
-0.10426     -0.69356      0.56474      -0.43504
0.93831      -0.30223      -0.16078      0.04831

Corresponding Matrix R:
9.59164     -3.75362      3.75319      6.67244
0.00000      4.88986     -2.02742     -4.08144
0.00000      0.00000      7.05721      9.73225
0.00000      0.00000      0.00000      1.45082

Corresponding Vector QT*B:
37.52675      8.15114      43.31061      4.35155

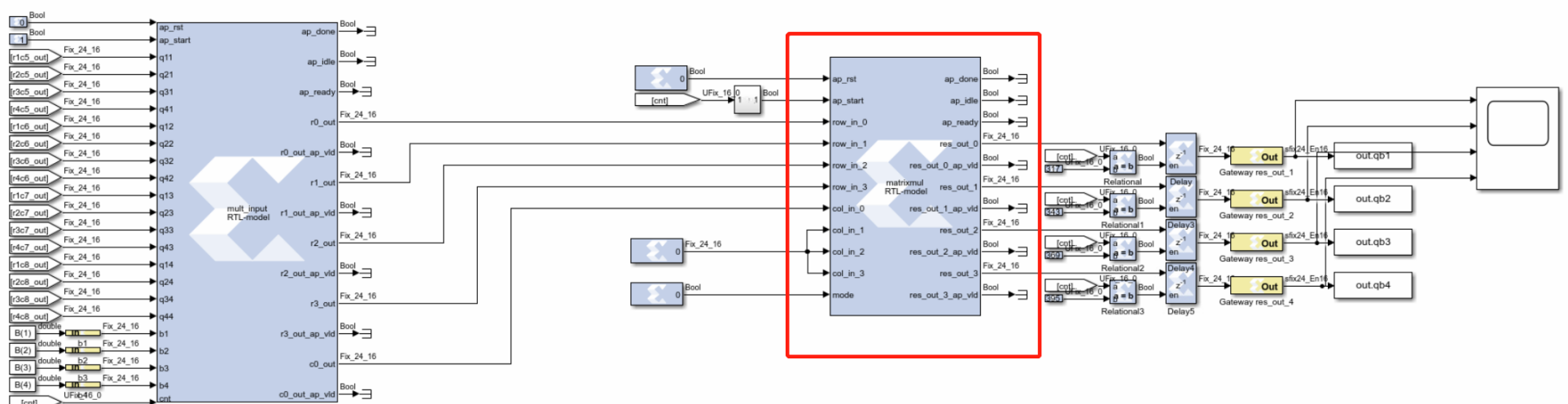
Corresponding Vector X:
2.9997      5.0000      2.0008      2.9994
--- End of the Program ---

```

6. Extra credit options

- Matrix-Vector on FPGA

The matrix-vector multiplication in the project is done in the FPGA with the Matrix Multiplier version of Project3.



We can read $Q^T \cdot b$ at the gateway out on the right.

