**A GENERAL PURPOSE REAL-TIME SoC BASED MATRIX MANIPULATOR**

**Project**

**ECE 5730 –Embedded System Design Using FPGAs**

**Submitted by**

**Cory Ness**

**Kokila Subramanian**

**Shivani Devatha**

**Submitted to**

**Professor Subramaniam Ganesan**



**Fall 2021**

**Abstract**

Matrix manipulation includes operations such as addition, subtraction, multiplication, inverse, and transpose etc., on the array of data stored as rows and columns. It is an essential process used in diverse fields of science and commerce including but not limited to Computer technology, Optics, Geology, Cryptography, Network Theory, Robotics and Animations, and Finance. Real-time matrix operations include a large number of computations, required for process control, and data & signal processing, which directly impacts the system performance. Real-time matrix calculation becomes a bottleneck for performance of fast system applications as it requires large computation power, memory and time.

In this project, we present a design which can perform matrix manipulations such as addition, subtraction, scalar and matrix array multiplication, determinant of matrix, co-factor and other vital matrix operations required for real-time signal processing. The matrix manipulator is developed using Intel DE1-SoC development board with Cyclone V processor interfaced with TRDB\_LTM 4.3 Inch Digital Touch Panel development kit (LCD module) to interpret the results. It is achieved by using NIOS II soft-core processor implemented on DE1-SoC board, the inputs are fed from a host computer onto FPGA board via UART serial port, a USB to TTL convertor module will be used to establish serial communication. The final computational results are displayed on the LCD.

**Objective**

Real-time signal processing system involves large computations and complex process control operations on massive array of data which is highly time-consuming. Currently, most of matrix operations are performed using software. The expansion of matrix dimension and reduction in processing speed has become bottleneck for the most of the existing real-time fast system applications. The objective of this project is to develop a Real-time Matrix-Manipulator using UART to TTL convertor for serial communication, Intel DE1-SoC FPGA Development Kit, and TRDB- TLM Touch screen LCD Display. The input matrix values are served into the FPGA board using Serial communication from a remote system. The matrix operations are executed in NIOS-II processor available in DE1-SoC board. The manipulated resultant matrix is displayed in the LCD display. The NIOS –II processor is used to control system operation and perform basic matrix operations such as addition, subtraction, Multiplication, finding determinant, co-factor, and other essential matrix operations required for real-time signal processing.

**System-level Block Diagram**

The system-level Block diagram for the Real-time Matrix Manipulator is shown in figure 1. The major functional blocks include the Remote PC, UART TTL convertor, the DE1-SoC board and the TRDB-TLM LCD display.



**TRDB-TLM LCD DISPLAY**

LCD

Controller

SDRAM

UART RX

NIOS II

Matrix Operation Algorithm

SSRAM

UART TX

UART

PIO

**USB to UART**

Serial communication

**DE1-SoC**

**PC**

**Figure 1: System-level Block Diagram**

**ADD some supporting points for Block-diagram**

**Matrix Manipulation Theory**

Matrices are representation of array of data commonly used in extensive real-world data processing, research and scientific studies. Matrix Multiplication provides good approximation for complicated calculation of real-time sensitive engineering applications signal processing, image processing, network theory and many more. Matrix manipulation is engaged with every part of day-to-day life including data base management, encryption that helps security, 3D gaming, robotics and animation, create models for economic and business, construction, physics and geology. The common matric calculations involved are addition, subtraction, multiplication, transpose, finding the determinant value and inverse of the matrix.

***Pseudo code for Matrix Addition:***

Matrix addition can be performed only when both the matrices under operation are of same size. It is commonly employed in data, image and signal processing. The void matrix\_addition() performs the following activity:

for (i=0;i<r1;i++)

{

for (j=0;j<c1;j++)

{

res[i][j]= a[i][j]+b[i][j];

}

}

***Pseudo code for Matrix Subtraction***

Matrix subtraction is a linear algebraic operation which can be performed only when both the matrices under operation are of same size. The void matrix\_subraction() performs the following operation:

for (i=0;i<r1;i++)

{

for (j=0;j<c1;j++)

{

res[i][j]= a[i][j]-b[i][j];

}

}

***Pseudo code for Multiplication:***

Matrix Multiplication can be either scalar or matrix multiplication, and is the most challenging algebraic operation. Matrix multiplication can be either scalar, vector or matrix multiplication. Scalar multiplication is a element-wise multiplication of matrix elements using a real or complex value. In the below defined code, *void scalar\_matrix\_multiplication()* executes matrix multiplication using constant value k.

for (i=0;i<r1;i++)

{

for (j=0;j<c1;j++)

{

res[i][j] = k\* a[i][j];

}

}

Multiplication of matrices can be executed under the condition that the number of rows of first matrix should be equal to number of columns in the second matrix (i.e. [A]i x j\* [B]m x n = [C]i x n.

if (r1 != c2)

{

printf("Matrix multiplication cannot be performed\n");

}

else

for (i=0;i<r1;i++)

{

for (j=0;j<c2;j++)

{

res[i][j]=0;

for (k=0; k<c1;k++)

res[i][j] += a[i][k]\*b[k][j];

}

}

***Pseudo code for Transpose of Matrix:***

Transpose is simply interchanging the rows and columns. It is generally employed when multiple matrix exists and the dimensions without transposing is not amendable for matrix operations. In real-time one usage of transpose is data manipulation, where the application requires the use of orthonormal matrix. In image processing switching rows and columns of pixels can result in exchanging the domain and space of the image transformation. In neural networks, frequent process weight and inputs of different sizes where the dimensions do not meet the requirement of algorithm can be accomplished using simple transpose of matrix.

for (i=0;i<c1;i++)

{

for (j=0;j<r1;j++)

{

res[i][j] = a[j][i];

}

}

***Pseudo code for Determinant of Matrix:***

***Add Pseudo code and explanation***

***VHDL code for UART Decoder:***

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

USE ieee.numeric\_std.all;

ENTITY Uart\_Decoder IS

GENERIC(

BaudRate : IN INTEGER := 9600;

ClockRate : IN INTEGER := 50\_000\_000

);

PORT(

Clk, Rx : IN STD\_LOGIC;

Complete : OUT STD\_LOGIC;

Data : OUT STD\_LOGIC\_VECTOR(7 DOWNTO 0)

);

END Uart\_Decoder;

ARCHITECTURE Behavior OF Uart\_Decoder IS

SIGNAL started : STD\_LOGIC; -- Whether we've started or not

SIGNAL bit\_location : Integer RANGE 0 TO 9 := 0; -- Where to save the bit

SIGNAL data\_reg : STD\_LOGIC\_VECTOR(9 DOWNTO 0); -- Local copy of the data, including start and stop bit

BEGIN

PROCESS (Rx, bit\_location)

BEGIN

IF bit\_location = 9 THEN

started <= '0';

ELSIF Rx = '0' AND Rx'event THEN

started <= '1';

END IF;

END PROCESS;

PROCESS (Clk, Rx, started)

VARIABLE counter : Integer RANGE 0 TO (ClockRate / BaudRate) := 0;

BEGIN

IF Clk = '1' AND Clk'event THEN

IF started = '1' THEN

counter := counter + 1;

IF counter = (ClockRate / BaudRate) THEN

counter := 0;

ELSIF counter = (ClockRate / (BaudRate \* 2)) THEN

data\_reg(bit\_location) <= Rx;

bit\_location <= bit\_location + 1;

END IF;

ELSE

bit\_location <= 0;

counter := 0;

END IF;

END IF;

END PROCESS;

Complete <= NOT started;

Data <= data\_reg(8 DOWNTO 1);

END Behavior;

***VHDL code for UART Decoder:***

The VHDL code for the UART decoder works as follows:

1. Input the clock rate and baud rate using the generic map, as this will not change throughout operation.
2. Wait for the start-bit from UART Rx (High->Low Transition).
3. Begin counting the clock cycles until we reach the necessary baud rate.
4. Halfway-through the count, sample the Rx line and save it in data\_register, this ensures we sample when the data line is stable.
5. Once the count reaches the baud value, reset the count
6. Repeat until all 10 bits have been sampled (start bit, data bits, stop bit).
7. Once all bits have been sampled, set the Complete signal to indicate Data is stable
8. Wait for next start-bit.

This has proven to be very reliable in testing with the UART-TTL converter.

**Hardware Components**

The developed Real-Time Matrix Manipulator consists of the following hardware components:

1. UART-TTL Convertor
2. DE1-SoC Development Kit
3. TRDB\_ LTM LCD Display

***UART-TTL Converter:***

The USB UART-TTL converter used is a FT232Rl FTDI chip on a breakout board, making use of the Uart Rx and Tx lines going to pins 1 and 3 on the 20 pin expansion header. It allows any computer to open a serial port and communicate to the DE1-SoC over a COM port, inputting the equations needed for the Matrix equations.

***DE1-SOC FPGA DEVELOPMENT KIT:***

The DE1-Soc Development kit provides a hardware platform that combines the System-on Chip (SoC) FPGA and dual-core Cyclone- A9 embedded core which can be used for wide industrial application improvement. It provides users the power of flexibility in design, paired with high-processor, low-performance