Ryerson University

**Department of Electrical, Computer, & Biomedical Engineering**
Faculty of Engineering & Architectural Science

| | |
|---|---|
| **Course Title:** | System on Chip Design |
| **Course Number:** | COE838 |
| **Semester/Year (e.g.F2016)** | W2024 |

| | |
|---|---|
| **Instructor:** | Dr. Khan |

| | |
|---|---|
| *Assignment/Lab Number:* | Lab 2 |
| *Assignment/Lab Title:* | SoC Accelerator |

| | |
|---|---|
| *Submission Date:* | 2024-01-30 |
| *Due Date:* | 2024-02-01 |

| Student LAST Name | Student FIRST Name | Student Number | Section | Signature* |
|---|---|---|---|---|
| Zelenovic | Danilo | 501032542 | 04 | DZ |
| | | | | |
| | | | | |

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

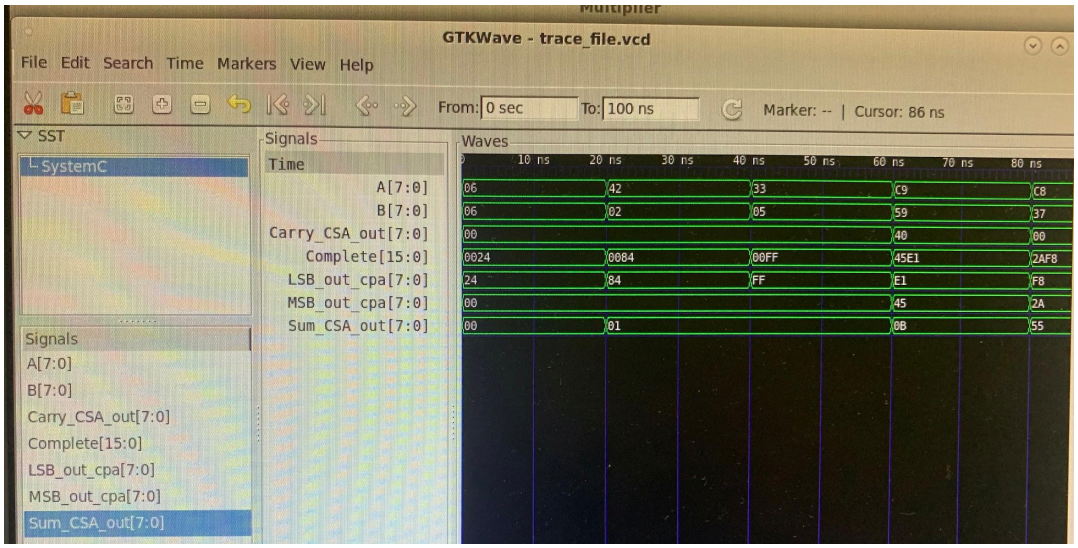
## Introduction

The purpose of this lab is to create a multiplier accelerator using SystemC. An 8x8 array multiplier is to be created. Using the CSA and CPA blocks provided as guidance, as well as the Array Unit Multiplier figures as a guide, the multiplier unit was created.

## Design

To create the array multiplier unit, the block diagram of the 4x4 unit provided was used as guidance. The multiplier is to take 8 “A” inputs, as well as 8 “B” inputs and multiplying them, with the aid of carry save adders as well as carry propagate adders. The CPS block uses a full-adder, taking values A, B and C as inputs, and outputting Carry out and sum as outputs. In comparison, the carry save adder uses an AND gate first with the A and B inputs, producing AB. The AB, Sum, and carry bits are then inputted into the full adder, producing sum and carry out bits. Arrays such as sum_holder and such are declared in order to store intermediate results during the multiplication process. The CSA module performs CSA multiplication by calculating sum and carry values for each bit position using nested loops. The result is simplified, and the LSB is stored in lsb_val, and the carry-out stored in carr_simplier. Next, the CPA module takes the output from the CSA and performs the carry-propagate addition. Using nested loops to add the carry-save partial products, producing the final value. The MSBs are stored in msb_val. The LSBs and MSBs are extracted and written to their respective output ports. The complete_output is formed by combining these bits and are written to the complete output port.

## Results



With some manual calculation examples, we can see that 0x06 times 0x06 certainly would give us 0x0024, and that, as another example, C9 multiplied by 59 would in fact give us 45E1. This can prove that the resulting waveform does output correct values.

## Appendix

**CPA.cpp:**

```
#include "CPA.h"
#include "CSA.h"

#define DATA_SIZE 8

// CSA Definition
sc_uint<DATA_SIZE> sum_csa;
sc_uint<DATA_SIZE> carrout_csa;
sc_uint<DATA_SIZE> ain_csa;
sc_uint<DATA_SIZE> bin_csa;
sc_uint<DATA_SIZE> A_combined_csa;
sc_uint<DATA_SIZE> sin_input_csa;
sc_uint<DATA_SIZE> lsb_val_csa;
sc_uint<DATA_SIZE> carrin_csa;
sc_uint<DATA_SIZE> sum_actual_csa;
sc_uint<DATA_SIZE> carr_actual_csa;
sc_uint<DATA_SIZE> sum_simpler_csa;
sc_uint<DATA_SIZE> carr_simpler_csa;

int row_index_csa = 0;
int col_index_csa = 0;
int iteration_index_csa = 0;
int sum_holder_csa[DATA_SIZE][DATA_SIZE];
int carrout_holder_csa[DATA_SIZE][DATA_SIZE];

// CPA definition
sc_uint<DATA_SIZE> sum_cpa;
sc_uint<DATA_SIZE> carrout_cpa;
sc_uint<DATA_SIZE> sum_holder_cpa;
sc_uint<DATA_SIZE> carrout_holder_cpa;
```

```
sc_uint<DATA_SIZE> sum_csa_cpa;
sc_uint<DATA_SIZE> carrout_csa_cpa;
sc_uint<DATA_SIZE> ain_cpa;
sc_uint<DATA_SIZE> bin_cpa;
sc_uint<DATA_SIZE> carrin_cpa;

sc_uint<DATA_SIZE> least_significant_cpa;
sc_uint<DATA_SIZE * 2> complete_output_cpa;

sc_uint<DATA_SIZE> msb_val_cpa;
int outer_loop_index_cpa, inner_loop_index_cpa = 0;
int loop_index_cpa = 0;

void CPA::CPA_method() {
   cout << endl;

   // CSA Method
   ain_csa = A_input.read();
   bin_csa = B_input.read();
   A_combined_csa = ain_csa & bin_csa;

   for (row_index_csa = 0; row_index_csa < DATA_SIZE; row_index_csa++) {
      for (col_index_csa = 0; col_index_csa < DATA_SIZE; col_index_csa++) {
         if (row_index_csa == 0) {
            sin_input_csa = 0;
            carrin_csa = 0;

            A_combined_csa = ain_csa[col_index_csa] & bin_csa[row_index_csa];
            sum_csa = ((A_combined_csa ^ sin_input_csa) ^ carrin_csa);
            carrout_csa = (A_combined_csa & sin_input_csa) | (sin_input_csa & carrin_csa) |
(A_combined_csa & carrin_csa);

            if (col_index_csa == 0) {
               lsb_val_csa[row_index_csa] = sum_csa[col_index_csa];
            }

            sum_holder_csa[row_index_csa][col_index_csa] = sum_csa;
            carrout_holder_csa[row_index_csa][col_index_csa] = carrout_csa;
         } else if (col_index_csa == (DATA_SIZE - 1) && (row_index_csa != 0)) {
            A_combined_csa = ain_csa[col_index_csa] & bin_csa[row_index_csa];
```

```
            carrin_csa = carrout_holder_csa[row_index_csa - 1][col_index_csa];
            sin_input_csa = 0;

            sum_csa = ((A_combined_csa ^ sin_input_csa) ^ carrin_csa);
            carrout_csa = (A_combined_csa & sin_input_csa) | (sin_input_csa & carrin_csa) |
(A_combined_csa & carrin_csa);

            sum_holder_csa[row_index_csa][col_index_csa] = sum_csa;
            carrout_holder_csa[row_index_csa][col_index_csa] = carrout_csa;
         } else {
            A_combined_csa = ain_csa[col_index_csa] & bin_csa[row_index_csa];
            carrin_csa = carrout_holder_csa[row_index_csa - 1][col_index_csa];
            sin_input_csa = sum_holder_csa[row_index_csa - 1][col_index_csa + 1];

            sum_csa = ((A_combined_csa ^ sin_input_csa) ^ carrin_csa);
            carrout_csa = (A_combined_csa & sin_input_csa) | (sin_input_csa & carrin_csa) |
(A_combined_csa & carrin_csa);

            sum_holder_csa[row_index_csa][col_index_csa] = sum_csa;
            carrout_holder_csa[row_index_csa][col_index_csa] = carrout_csa;

            if (col_index_csa == 0) {
               lsb_val_csa[row_index_csa] = sum_csa[col_index_csa];
            }
         }
      }
   }

   for (iteration_index_csa = 0; iteration_index_csa < DATA_SIZE; iteration_index_csa++) {
      sum_simpler_csa[iteration_index_csa] = sum_holder_csa[DATA_SIZE -
1][iteration_index_csa];
      carr_simpler_csa[iteration_index_csa] = carrout_holder_csa[DATA_SIZE -
1][iteration_index_csa];
   }

   lsb_out.write(lsb_val_csa);
   carry_out_csa.write(carr_simpler_csa);
   sum_out_csa.write(sum_simpler_csa);

   // CSA Method
```

```
    sum_csa_cpa = sum_simpler_csa;
    carrout_csa_cpa = carr_simpler_csa;
    least_significant_cpa = lsb_val_csa;

    for (outer_loop_index_cpa = 0; outer_loop_index_cpa < DATA_SIZE;
outer_loop_index_cpa++) {
        if (outer_loop_index_cpa == 0) {
            carrin_cpa = 0;
            ain_cpa = sum_csa_cpa[outer_loop_index_cpa + 1];
            bin_cpa = carrout_csa_cpa[outer_loop_index_cpa];

            sum_cpa = ((ain_cpa ^ bin_cpa) ^ carrin_cpa);
            carrout_cpa = (ain_cpa & bin_cpa) | (carrin_cpa & bin_cpa) | (ain_cpa & carrin_cpa);

            sum_holder_cpa[outer_loop_index_cpa] = sum_cpa;
            msb_val_cpa[outer_loop_index_cpa] = sum_cpa;
            carrout_holder_cpa[outer_loop_index_cpa] = carrout_cpa;
        } else if (outer_loop_index_cpa == (DATA_SIZE - 1)) {
            carrin_cpa = 0;
            bin_cpa = carrout_holder_cpa[outer_loop_index_cpa - 1];
            ain_cpa = carrout_csa_cpa[outer_loop_index
            carrout_csa_cpa = carrout_csa;

    for (outer_loop_index_cpa = 0; outer_loop_index_cpa < DATA_SIZE;
outer_loop_index_cpa++) {
        if (outer_loop_index_cpa == 0) {
            carrin_cpa = 0;
            ain_cpa = sum_csa_cpa[outer_loop_index_cpa + 1];
            bin_cpa = carrout_csa_cpa[outer_loop_index_cpa];

            sum_cpa = ((ain_cpa ^ bin_cpa) ^ carrin_cpa);
            carrout_cpa = (ain_cpa & bin_cpa) | (carrin_cpa & bin_cpa) | (ain_cpa & carrin_cpa);

            sum_holder_cpa[outer_loop_index_cpa] = sum_cpa;
            msb_val_cpa[outer_loop_index_cpa] = sum_cpa;
            carrout_holder_cpa[outer_loop_index_cpa] = carrout_cpa;
        } else if (outer_loop_index_cpa == (DATA_SIZE - 1)) {
            carrin_cpa = 0;
            bin_cpa = carrout_holder_cpa[outer_loop_index_cpa - 1];
```

```
            ain_cpa = carrout_csa_cpa[outer_loop_index_cpa];

            sum_cpa = ((ain_cpa ^ bin_cpa) ^ carrin_cpa);
            carrout_cpa = (ain_cpa & bin_cpa) | (carrin_cpa & bin_cpa) | (ain_cpa & carrin_cpa);

            sum_holder_cpa[outer_loop_index_cpa] = sum_cpa;
            msb_val_cpa[outer_loop_index_cpa] = sum_cpa;
            carrout_holder_cpa[outer_loop_index_cpa] = carrout_cpa;
        } else {
            carrin_cpa = carrout_holder_cpa[outer_loop_index_cpa - 1];
            bin_cpa = carrout_csa_cpa[outer_loop_index_cpa];
            ain_cpa = sum_csa_cpa[outer_loop_index_cpa + 1];

            sum_cpa = ((ain_cpa ^ bin_cpa) ^ carrin_cpa);
            carrout_cpa = (ain_cpa & bin_cpa) | (carrin_cpa & bin_cpa) | (ain_cpa & carrin_cpa);

            sum_holder_cpa[outer_loop_index_cpa] = sum_cpa;
            msb_val_cpa[outer_loop_index_cpa] = sum_cpa;
            carrout_holder_cpa[outer_loop_index_cpa] = carrout_cpa;
        }

        cout << sum_holder_cpa[outer_loop_index_cpa];
    }

    msb_out.write(msb_val_cpa);

    for (loop_index_cpa = 0; loop_index_cpa < (DATA_SIZE * 2); loop_index_cpa++) {
        if (loop_index_cpa < DATA_SIZE) {
            complete_output_cpa[loop_index_cpa] = least_significant_cpa[loop_index_cpa];
        } else {
            complete_output_cpa[loop_index_cpa] = msb_val_cpa[loop_index_cpa - DATA_SIZE];
        }
    }

    complete.write(complete_output_cpa);
}
```

**CPA.h:**

```
#ifndef CPA_H
#define CPA_H

#include <systemc.h>
#include "CSA.h"

#define DATA_SIZE 8

SC_MODULE(CPA) {
   sc_in<sc_uint<DATA_SIZE>> B_input;
   sc_in<sc_uint<DATA_SIZE>> A_input;
   sc_in<sc_uint<DATA_SIZE>> carry_in;
   sc_in<sc_uint<DATA_SIZE>> least_sig;
   sc_out<sc_uint<DATA_SIZE>> carry_out;
   sc_out<sc_uint<DATA_SIZE>> sum_out;

   sc_out<sc_uint<DATA_SIZE>> lsb_out;
   sc_out<sc_uint<DATA_SIZE>> sum_out_csa;
   sc_out<sc_uint<DATA_SIZE>> carry_out_csa;

   sc_out<sc_uint<DATA_SIZE>> msb_out;
   sc_out<sc_uint<DATA_SIZE * 2>> complete;

   void CPA_method();

   SC_CTOR(CPA) {
      SC_METHOD(CPA_method);
      dont_initialize();
      sensitive << A_input << B_input << carry_in;
   }
};

#endif
```

**CSA.h:**

```
#ifndef CSA_H
```

```
#define CSA_H

#include <systemc.h>

#define DATA_SIZE 8

SC_MODULE(CSA) {
   sc_in<sc_uint<DATA_SIZE>> B_input;
   sc_in<sc_uint<DATA_SIZE>> A_input;
   sc_in<sc_uint<DATA_SIZE>> S_input;
   sc_in<sc_uint<DATA_SIZE>> carry_in;
   sc_out<sc_uint<DATA_SIZE>> carry_out;
   sc_out<sc_uint<DATA_SIZE>> sum_out;
   sc_out<sc_uint<DATA_SIZE>> lsb_out;

   void CSA_method();

   SC_CTOR(CSA) {
      SC_METHOD(CSA_method);
      dont_initialize();
      sensitive << A_input << B_input << S_input << carry_in;
   }
};

#endif
```

**Main:**

```
#include <systemc.h>
#include "CPA.h"
#include "CSA.h"

#define DATA_SIZE 8

int sc_main(int argc, char* argv[]) {
   int truth_iterations = 8;
   int i = 0;
```

```
int time = 9;

sc_trace_file *tf; // Create VCD file for tracing

sc_signal<sc_uint<DATA_SIZE>> sum_cpa;
sc_signal<sc_uint<DATA_SIZE>> carrout_cpa;
sc_signal<sc_uint<DATA_SIZE>> ain_cpa;
sc_signal<sc_uint<DATA_SIZE>> bin_cpa;
sc_signal<sc_uint<DATA_SIZE>> carrin_cpa;

sc_signal<sc_uint<DATA_SIZE>> lsb_out_cpa;
sc_signal<sc_uint<DATA_SIZE>> sum_out_csa;
sc_signal<sc_uint<DATA_SIZE>> carry_out_csa;

sc_signal<sc_uint<DATA_SIZE>> msb_out_cpa;
sc_signal<sc_uint<DATA_SIZE>> lsb_in_cpa;
sc_signal<sc_uint<DATA_SIZE * 2>> complete_byte;

sc_signal<sc_uint<DATA_SIZE>> sum_csa;
sc_signal<sc_uint<DATA_SIZE>> carrout_csa;
sc_signal<sc_uint<DATA_SIZE>> ain_csa;
sc_signal<sc_uint<DATA_SIZE>> bin_csa;
sc_signal<sc_uint<DATA_SIZE>> sin_csa;
sc_signal<sc_uint<DATA_SIZE>> carrin_csa;
sc_signal<sc_uint<DATA_SIZE>> lsb_out_csa;

sc_clock clk("clk", 10, SC_NS, 0.5); // Create a clock signal

CPA testing_CPA("CPA"); // Create Device Under Test (DUT)

testing_CPA.B_input(bin_cpa);
testing_CPA.A_input(ain_cpa);
testing_CPA.carry_in(carrin_cpa);
testing_CPA.carry_out(carrout_cpa);
testing_CPA.sum_out(sum_cpa);

testing_CPA.lsb_out(lsb_out_cpa);
testing_CPA.sum_out_csa(sum_out_csa);
testing_CPA.carry_out_csa(carry_out_csa);
```

```
testing_CPA.msb_out(msb_out_cpa);
testing_CPA.complete(complete_byte);
testing_CPA.least_sig(lsb_in_cpa);

// Create wave file and trace the signals executing
tf = sc_create_vcd_trace_file("trace_file");
tf->set_time_unit(1, SC_NS);

sc_trace(tf, msb_out_cpa, "MSB_out_cpa");
sc_trace(tf, complete_byte, "Complete");

sc_trace(tf, sum_out_csa, "Sum_CSA_out");
sc_trace(tf, carry_out_csa, "Carry_CSA_out");
sc_trace(tf, bin_cpa, "B");
sc_trace(tf, ain_cpa, "A");
sc_trace(tf, lsb_out_cpa, "LSB_out_cpa");

// cout << "\nExecuting CPA example... check .vcd produced" << endl;
// start the testbench

ain_cpa.write(4);
bin_cpa.write(8);
sc_start(20, SC_NS);

ain_cpa.write(255);
bin_cpa.write(255);
sc_start(20, SC_NS);

ain_cpa.write(55);
bin_cpa.write(5);
sc_start(20, SC_NS);

ain_cpa.write(200);
bin_cpa.write(600);
sc_start(20, SC_NS);

ain_cpa.write(20);
bin_cpa.write(55);
sc_start(20, SC_NS);
```

```
    ain_cpa.write(90);
    bin_cpa.write(20);
    sc_start(20, SC_NS);

    sc_close_vcd_trace_file(tf);

    return 0;
}
```