Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Дисциплина: Системы на кристалле

Лабораторная работа №1

Крупно-блочное проектирование микропроцессорной СнК

Группа: P3432

Выполнили:
Глотов Егор Дмитриевич
Чмурова Мария Владиславовна
Ефимов Арслан Альбертович

Преподаватель:
Рыбаков Степан Дмитриевич

г. Санкт-Петербург

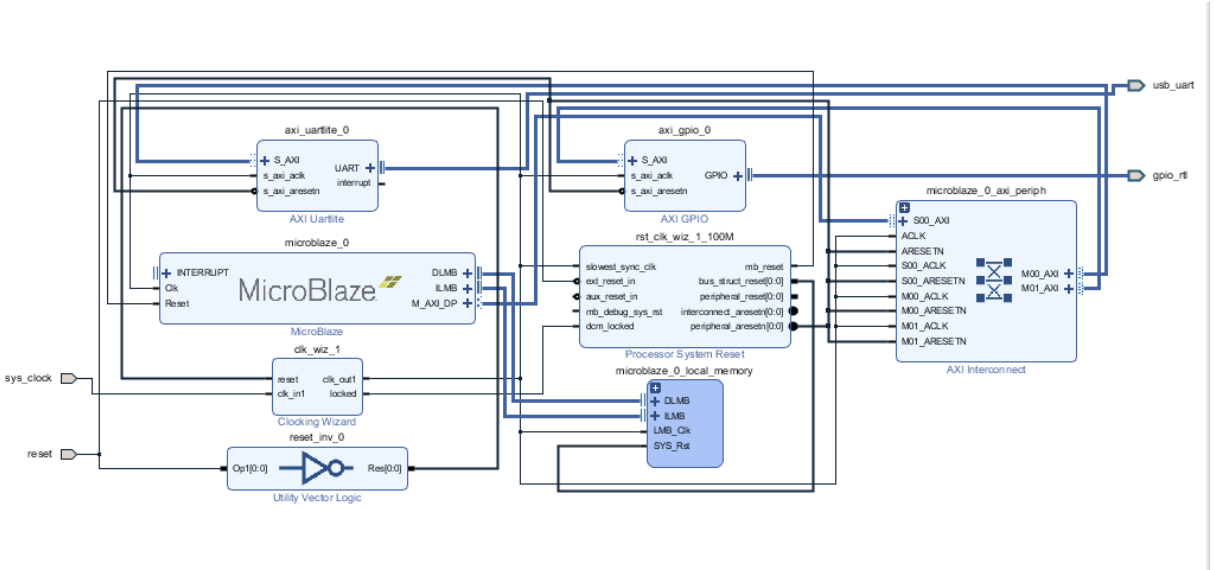
2025 г.

Содержание

Цель

Получить знания и навыки про устройство микропроцессорных систем на кристалле.

Структура разработанной системы и описание ее работы



- Центральный элемент – MicroBlaze

MicroBlaze – soft- процессор, реализуемый в логике ПЛИС. Выполняет программный код и взаимодействует с периферией по стандартной AXI шине. На схеме имеет три интерфейса: DLMB/ILMB – шины для доступа к локальной памяти и M_AXI_DP – master порт шины AXI для доступа к внешним устройствам

- **AXI Interconnect**

Шинная матрица AXI, обеспечивающая связь между master и несколькими slave устройствами. На схеме MicroBlaze подключен к порту S00_AXI. С точки зрения периферии сам AXI Interconnect является master для них. С точки зрения MicroBlaze сам процессор является master для шинной матрицы и инициирует передачу данных.

- **Processor System Reset**

Управляет сбросом всех компонентов системы. Основные входы в этот блок – ext_reset_in (внешний сигнал сброса) и dcm_locked (сигнал от Clock Wizard, показывающий, что сигнал стабильный). На выходе генерируются сигналы сброса процессора, шин и периферийных IP-модулей.

- **Блок управления тактовым сигналом clk_wiz_1**

Блок **Clocking Wizard (clk_wiz_1)** служит для генерации стабильного системного тактового сигнала требуемой частоты. На вход он получает внешний опорный сигнал **clk_in1** и сигнал **reset**, который используется для перезапуска внутреннего генератора при необходимости. На выходе формируется сигнал **clk_out1**, являющийся синхронизированным тактовым сигналом для всех модулей системы. Выход **locked** указывает, что генератор успешно стабилизировал частоту и фазы, и система может начать нормальную работу.

- **Контроллер последовательного канала AXI UARTlite**

Блок axi_uartlite_0 реализует упрощенный контроллер последовательного интерфейса UART для обмена данными между процессором MicroBlaze и внешними устройствами, например компьютером через USB-UART. Он подключён к системе по шине AXI через интерфейс S_AXI, включающий сигналы s_axi_aclk (тактовый сигнал шины) и s_axi_aresetn (активный низкий сброс). Выход interrupt используется для генерации прерываний, уведомляющих процессор о событиях, таких как окончание передачи или получение байта.

- **Контроллер портов ввода/вывода (AXI GPIO)**

Блок axi_gpio_0 реализует двунаправленные порты общего назначения для управления и считывания логических сигналов. Он также подключён к шине AXI через интерфейс S_AXI с сигналами s_axi_aclk (тактовый сигнал шины) и s_axi_aresetn (активный низкий сброс). Он имеет два порта: GPIO и GPIO2, которые связаны с внешними линиями sw_bi (входы, например, переключатели) и led_bo (выходы, например, светодиоды).

Каждый порт представляет собой набор бит, доступных процессору для чтения и записи через регистры, что позволяет MicroBlaze управлять периферией и считывать состояние кнопок или других устройств ввода.

- **Память данных и команд MicroBlaze**

На схеме microblaze_0_local_memory содержит локальные блоки памяти для инструкций ILMB и для данных DLMB. Сигнал LMB_Clk является общим тактовым сигналом, синхронизирующим работу памяти с остальной системой, а SYS_Rst используется для сброса модуля при инициализации или перезапуске системы.

Листинг алгоритма работы

Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H

#include <stdio.h>
#include <stdlib.h>
#include "xil_printf.h"
#include "xil_io.h"
#include "matrix.h"
#include "sleep.h"

#define matrix_size 3
#define GPIO_BASE_ADDR          0x40000000
#define NO_LD                   0x0000
#define LD0                     0x0001
#define LD1                     0x0002

int execute();
int** multiply_matrices(int **A, int **B);
int** subtract_matrices(int **A, int **B);
void free_matrix(int **matrix);
int** create_matrix();
int** calculate_function(int **A, int **B);


#endif
```

Здесь находится описание основной логики работы функций с матрицами

Matrix.c

```
#include "matrix.h"

int** create_ones_matrix() {
    int** matrix = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        matrix[i] = (int*)malloc(matrix_size * sizeof(int));
        for (int j = 0; j < matrix_size; j++) {
            matrix[i][j] = 1;
        }
    }
    return matrix;
}

int** create_sequence_matrix() {
    int** matrix = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        matrix[i] = (int*)malloc(matrix_size * sizeof(int));
        for (int j = 0; j < matrix_size; j++) {
            matrix[i][j] = j + 1;
        }
    }
```

```
    return matrix;
}

int execute()
{
    int** result;
    int** A = create_ones_matrix();
    int** B = create_sequence_matrix();

    for (int i = 0; i < 3; i++)
    {
        Xil_Out16(GPIO_BASE_ADDR, LD0);
        xil_printf("Iteration %d: started\r\n", i);

        result = calculate_function(A, B);

        Xil_Out16(GPIO_BASE_ADDR, LD1);
        xil_printf("Iteration %d: done\r\n", i);
    }


    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
                uint16_t led_value = (uint16_t)(result[i][j] & 0xFFFF);

                Xil_Out16(GPIO_BASE_ADDR, led_value);

                xil_printf("LEDs show result[%d][%d] = 0x%04X\r\n", i, j,
led_value);

                sleep(2);

                Xil_Out16(GPIO_BASE_ADDR, 0x0000);

                sleep(1);
        }
    }


    free(A);
    free(B);
    free(result);

    return 0;
}

int** calculate_function(int **A, int **B)
{
        int** A_multiply_B = multiply_matrices(A, B);
        int** result = subtract_matrices(A_multiply_B, B);
        free_matrix(A_multiply_B);
        return result;
}

int** create_matrix() {
    int **matrix = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
```

```
        matrix[i] = (int*)malloc(matrix_size * sizeof(int));
    }
    return matrix;
}


int** multiply_matrices(int **A, int **B) {
    int **result = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        result[i] = (int*)malloc(matrix_size * sizeof(int));
    }

    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            result[i][j] = 0;
            for (int k = 0; k < matrix_size; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    return result;
}

int** subtract_matrices(int **A, int **B) {
    int **result = (int**)malloc(matrix_size * sizeof(int*));
    for (int i = 0; i < matrix_size; i++) {
        result[i] = (int*)malloc(matrix_size * sizeof(int));
    }

    for (int i = 0; i < matrix_size; i++) {
        for (int j = 0; j < matrix_size; j++) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }

    return result;
}

void free_matrix(int** matrix) {
    for (int i = 0; i < matrix_size; i++) {
        free(matrix[i]);
    }
    free(matrix);
}
```

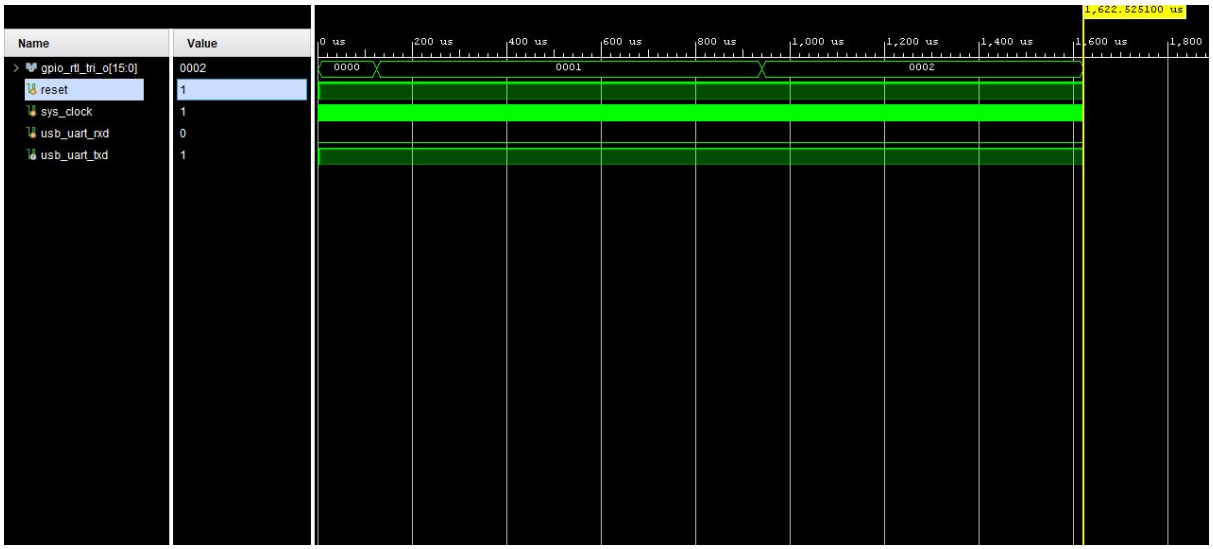
Реализация описанных функций сложения и умножения матриц и запуск самой программы

Helloworld.c

```c
#include <stdio.h>
#include "platform.h"
#include "matrix.h"

int main()
{
    init_platform();

    execute();

    cleanup_platform();
    return 0;
}
```

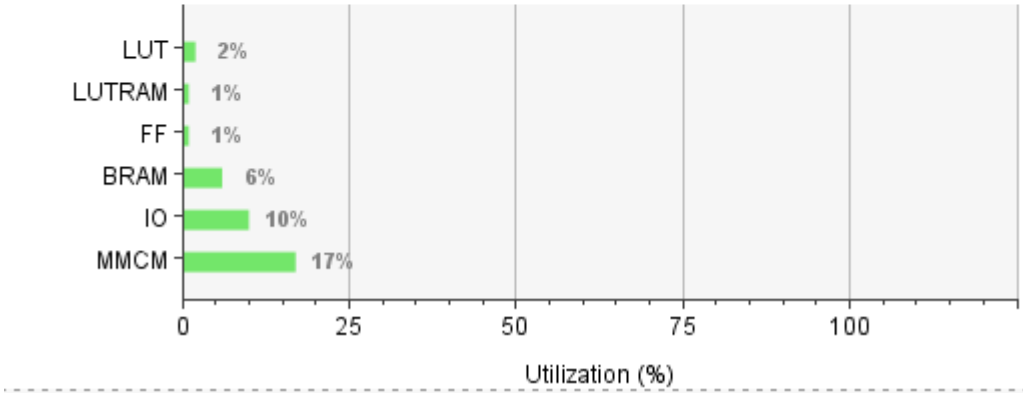
Входная точка в программу.

Работы системы в симуляторе и время работы программы при частоте тактового сигнала 100 МГц



Число занимаемых ресурсов ПЛИС

Resource	Utilization	Available	Utilization %
LUT	1311	63400	2.07
LUTRAM	160	19000	0.84
FF	1639	126800	1.29
BRAM	8	135	5.93
IO	20	210	9.52
MMCM	1	6	16.67



Система получилась очень компактной и использует меньше 20% ресурсов платы, что в целом типично для такого небольшого проекта.

- LUT и LUTRAM - 2.07% и 0.84% соответственно. Используются для реализации логических функций, а также как мини RAM модули
- FF – также около одного процента. Используются для хранения одного бита информации
- BRAM – блочная память, занимаем всего 8 блоков (около 4 Кб на блок)
- MMCM – используем только один генератор тактового сигнала
- IO – примерно 20 линий ввода-вывода занято UART, GPIO и системными сигналами

Вывод

В результате выполнения данной работы мы познакомились с устройством устройство микропроцессорных систем на кристалле посредством реализации простой функции с матрицами.