



## **COEN 317: MICROPROCESSOR-BASED SYSTEM**

### **Section UN-X**

#### **LAB REPORT # 4:**

Interrupt with the AXI Timer  
Direct Memory Access

#### **INSTRUCTOR:**

Dr. Fadi El-Hassan

#### **Lab Coordinator:**

Ted Obuchowicz

#### **Submitted by:**

Jaskirat Kaur- 40138320

Bashar -?

**"I certify that this submission is my original work and meets the  
Faculty's Expectations of Originality"**

Date Written: Monday, April 13, 2024

# Introduction

The second part of the lab introduces us to the AXI CDMA (Direct Memory Access). This component facilitates high-speed data transfers from a memory source to a memory destination. Our objective is to configure the CDMA as a master device to oversee two high-performance slave ports, specifically the Master General Purpose ports. These ports will be interconnected using the AXI Interconnect module, thereby ensuring efficient data throughput between the AXI master and the DDR memory system of the processor. Furthermore, one of the lab inquiries involves determining the duration required for data transfer. To accomplish this, we'll employ the AXI Timer, a familiar module introduced in the third lab session, which counts the clock sequence and offers various operational modes.

# Results and Questions:

To set up the hardware for the first part, we follow the steps in the lab manual. We add a CDMA instance and two AXI interconnect instances. We manually connect them instead of using the processor system 7 manual connection. The lab manual explains different ways to make these connections, but we can simply click some boxes in the bus tab to designate ports as master or slave. Once the hardware is set up, we write a program in Xilinx SDK to transfer data between the source and destination. The lab manual gives us a step-by-step guide in pseudocode for this program. Question 1 asks us to use an AXI timer to find out how long it takes to transfer the data. So, we enhance the hardware by adding an AXI Timer/Counter instance. We can connect it manually or let the processor's system 7 handle it. To modify the program in the SDK, we refer to lab 3 where we used the timer to count cycles for a discrete write. Instead of a discrete write, we're now doing a data transfer with the CDMA. We modify the code from the first part by adding a timer variable, initializing it, setting it in capture mode, starting the timer and then stopping it. The code for question 1 is shown in [Appendix A, Code 3](#). This is the output we got:

```
Array Size: 4096 bytes
Time without CDMA: 1602 clock cycles
Time with CDMA: 1063 clock cycles
Speedup: 1.50706

Array Size: 16384 bytes
Time without CDMA: 6285 clock cycles
Time with CDMA: 2603 clock cycles
Speedup: 2.41452

Array Size: 32768 bytes
Time without CDMA: 12533 clock cycles
Time with CDMA: 4648 clock cycles
Speedup: 2.69643

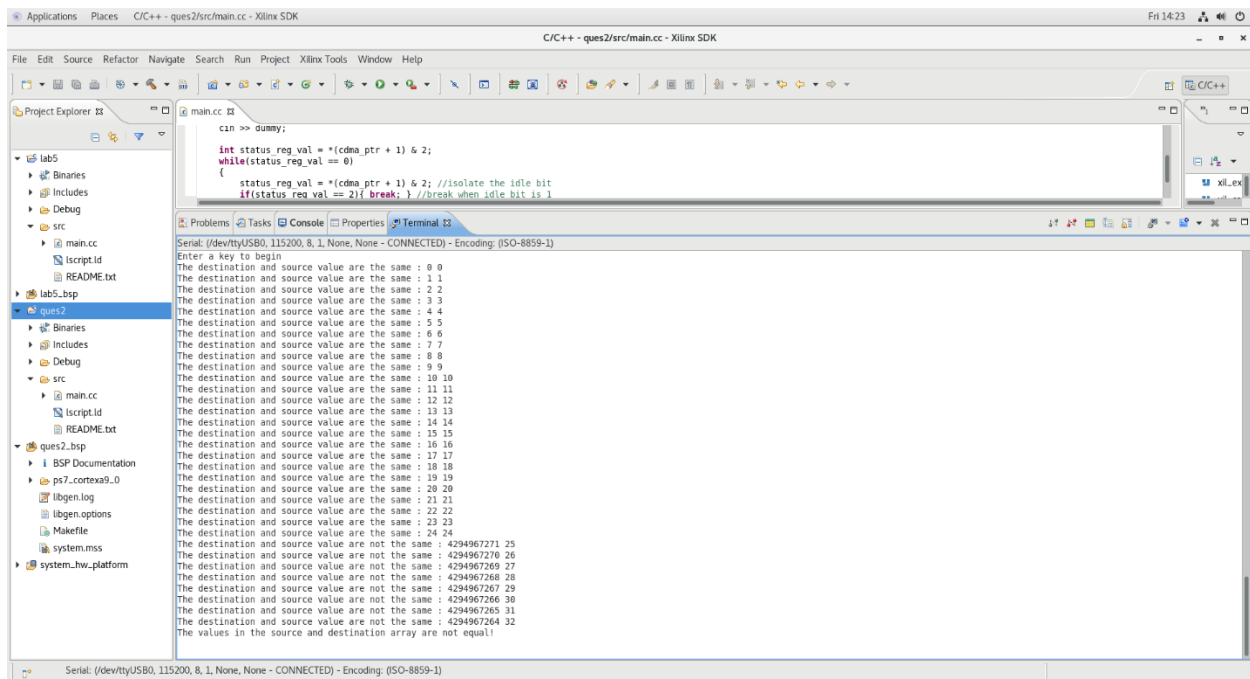
Array Size: 65536 bytes
Time without CDMA: 25033 clock cycles
Time with CDMA: 8749 clock cycles
Speedup: 2.86124

Array Size: 131068 bytes
Time without CDMA: 50007 clock cycles
Time with CDMA: 16938 clock cycles
Speedup: 2.95236

Array Size: 4194304 bytes
Time without CDMA: 1618798 clock cycles
Time with CDMA: 524845 clock cycles
Speedup: 3.08434
```

For question 2, we investigate what happens if we try to write data beyond the maximum allowed bits in the BBT register. We can modify the code to write data beyond the limit and compare the source with the destination to see what happens. The code for question 2 is shown in [Appendix A, Code 3](#).

When we run the code, we see that the destination data doesn't change when we write beyond the maximum allowed bits. This is because bits 25 to 32 are reserved and shouldn't be modified. Therefore, it showcases garbage values in those destinations:



The screenshot displays the Xilinx IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Run, Project, Xilinx Tools, Window, and Help. The Project Explorer on the left shows a project structure with folders for lab5, ques2, and system. The main editor window shows the source code for main.c, which includes a loop that writes data to a destination array. The terminal window at the bottom shows the output of the program, which includes a series of messages comparing source and destination values. The output shows that for the first 24 iterations, the source and destination values are the same. However, for the remaining iterations (25 to 32), the destination values are garbage, indicating that the write operation is not modifying the reserved bits of the BBT register.

```
Serial: (/dev/ttyUSB0, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
Enter a key to begin
The destination and source value are the same : 0 0
The destination and source value are the same : 1 1
The destination and source value are the same : 2 2
The destination and source value are the same : 3 3
The destination and source value are the same : 4 4
The destination and source value are the same : 5 5
The destination and source value are the same : 6 6
The destination and source value are the same : 7 7
The destination and source value are the same : 8 8
The destination and source value are the same : 9 9
The destination and source value are the same : 10 10
The destination and source value are the same : 11 11
The destination and source value are the same : 12 12
The destination and source value are the same : 13 13
The destination and source value are the same : 14 14
The destination and source value are the same : 15 15
The destination and source value are the same : 16 16
The destination and source value are the same : 17 17
The destination and source value are the same : 18 18
The destination and source value are the same : 19 19
The destination and source value are the same : 20 20
The destination and source value are the same : 21 21
The destination and source value are the same : 22 22
The destination and source value are the same : 23 23
The destination and source value are the same : 24 24
The destination and source value are not the same : 4294967271 25
The destination and source value are not the same : 4294967270 26
The destination and source value are not the same : 4294967269 27
The destination and source value are not the same : 4294967268 28
The destination and source value are not the same : 4294967267 29
The destination and source value are not the same : 4294967266 30
The destination and source value are not the same : 4294967265 31
The destination and source value are not the same : 4294967264 32
The values in the source and destination array are not equal!
```

# Conclusion

In conclusion, through part 1 and 2 of lab 4, we successfully configured hardware components such as the CDMA and AXI interconnects to facilitate data transfer operations. By integrating these hardware elements into our programming environment using Xilinx SDK, we were able to develop a program that effectively transferred data between a source and destination. Leveraging an AXI timer allowed us to accurately measure the time required for data transfer, while further experimentation revealed the consequences of attempting to write data beyond the permissible limits in the BBT register. These hands-on activities not only enhanced our understanding of hardware configuration and programming techniques but also provided valuable insights into the robustness and limitations of the implemented hardware systems.

## **APPENDIX A**

### **Code 1**

### **Code 2**

### **Code 3**

```
#include "xil_exception.h"
#include "xil_cache.h"
#include "xparameters.h"
#include <iostream>
#include "xtmrctr.h"

using namespace std;

// Function to transfer data without CDMA
unsigned int transferWithoutCDMA(u32* source_ptr, u32* destination_ptr, int num_integers,
XTmrCtr& TimerInstancePtr) {
    XTmrCtr_Start(&TimerInstancePtr, 0);
    for(int i = 0; i < num_integers; i++) {
        *(destination_ptr + i) = *(source_ptr + i);
    }
    XTmrCtr_Stop(&TimerInstancePtr, 0);
    return XTmrCtr_GetValue(&TimerInstancePtr, 0);
}

// Function to transfer data with CDMA
unsigned int transferWithCDMA(u32* cdma_ptr, u32* source_ptr, u32* destination_ptr, int
num_bytes, XTmrCtr& TimerInstancePtr) {
    // Reset CDMA
    *(cdma_ptr) = 0x00000004;
    // Configure CDMA
    *(cdma_ptr) = 0x00000020;
    // Load source and destination addresses
    *(cdma_ptr + 6) = (u32)source_ptr;
    *(cdma_ptr + 8) = (u32)destination_ptr;
    // Flush cache
    Xil_DCacheFlush();
    // Set number of bytes to transfer
    *(cdma_ptr + 10) = num_bytes;

    XTmrCtr_Start(&TimerInstancePtr, 0);
    // Wait for transfer to complete
    while(!(*(cdma_ptr + 1) & 2)) {}
    XTmrCtr_Stop(&TimerInstancePtr, 0);
    return XTmrCtr_GetValue(&TimerInstancePtr, 0);
}

int main() {
    // Declare variables
```

```

u32* source_ptr = (u32*)XPAR_PS7_DDR_0_S_AXI_HP0_BASEADDR;
u32* destination_ptr = (u32*)XPAR_PS7_DDR_0_S_AXI_HP2_BASEADDR;
u32* cdma_ptr = (u32*)XPAR_AXI_CDMA_0_BASEADDR;
XTmrCtr TimerInstancePtr;
int xStatus;

// Initialize AXI Timer
xStatus = XTmrCtr_Initialize(&TimerInstancePtr, XPAR_AXI_TIMER_0_DEVICE_ID);
if(xStatus != XST_SUCCESS) {
    cout << "TIMER INIT FAILED" << endl;
    return 0;
}
XTmrCtr_SetResetValue(&TimerInstancePtr, 0, 0);
XTmrCtr_SetOptions(&TimerInstancePtr, XPAR_AXI_TIMER_0_DEVICE_ID,
XTC_CAPTURE_MODE_OPTION);

// Array sizes to test
int array_sizes[] = {1024, 4096, 8192, 16384, 32767, 1048576};
int num_sizes = sizeof(array_sizes) / sizeof(array_sizes[0]);

// Measure transfer time for each array size
for(int i = 0; i < num_sizes; i++) {
    int num_integers = array_sizes[i];
    int num_bytes = num_integers * sizeof(u32);

    // Without CDMA
    unsigned int cycles_without_cdma = transferWithoutCDMA(source_ptr, destination_ptr,
num_integers, TimerInstancePtr);

    // With CDMA
    unsigned int cycles_with_cdma = transferWithCDMA(cdma_ptr, source_ptr, destination_ptr,
num_bytes, TimerInstancePtr);

    // Output results
    cout << "Array Size: " << num_bytes << " bytes" << endl;
    cout << "Time without CDMA: " << cycles_without_cdma << " clock cycles" << endl;
    cout << "Time with CDMA: " << cycles_with_cdma << " clock cycles" << endl;

    // Compute speedup
    float speedup = static_cast<float>(cycles_without_cdma) / cycles_with_cdma;
    cout << "Speedup: " << speedup << endl << endl;
}

return 0;
}

```

## **Code 4**

```

#include "xil_exception.h"
#include "xil_cache.h"
#include "xparameters.h"

```

```

#include <iostream>

using namespace std;

int main()
{
    //initialize the three pointers
    u32* cdma_ptr = (u32*) XPAR_AXI_CDMA_0_BASEADDR;
    u32* source_ptr = (u32*) XPAR_PS7_DDR_0_S_AXI_HP0_BASEADDR;
    u32* destination_ptr = (u32*) XPAR_PS7_DDR_0_S_AXI_HP2_BASEADDR;

    //initializing the contents of the source array within the maximum size
    for(int i=0; i <= 32; i++)
    {
        *(source_ptr + i) = i;
    }

    //initializing the contents of the destination array
    for(int i=0; i <= 32; i++)
    {
        *(destination_ptr + i) = -i;
    }

    //reset the cdma
    *(cdma_ptr) = 0x00000004;

    //configure the cdma
    *(cdma_ptr) = 0x00000020;

    //load address of source array
    *(cdma_ptr + 6) = 0x20000000;

    //load address of destination array
    *(cdma_ptr + 8) = 0x30000000;

    //flush the cash
    Xil_DCacheFlush();

    //number of bytes to transfer
    *(cdma_ptr + 10) = 0x64;

    //take in a value to start the process
    char dummy;
    cout << "Enter a key to begin" << endl;
    cin >> dummy;

    int status_reg_val = *(cdma_ptr + 1) & 2;
    while(status_reg_val == 0)
    {
        status_reg_val = *(cdma_ptr + 1) & 2; //isolate the idle bit
    }
}

```



```

    if(status_reg_val == 2){ break; } //break when idle bit is 1
}

bool are_equal = true;
//compare the contents after the transfer
for(int i=0 ; i <= 32; i++)
{
    if(*(destination_ptr + i) != *(source_ptr + i))
    {
        are_equal = false;
        cout <<"The destination and source value are not the same : " << *(destination_ptr + i)
<< " " << *(source_ptr + i) << endl;
    }
    else {
        cout <<"The destination and source value are the same : " << *(destination_ptr + i) << " "
<< *(source_ptr + i) << endl;
    }
}

//output the result
if (are_equal){ cout << "The values in the source and destination array are equal!" << endl;}
else { cout << "The values in the source and destination array are not equal!" << endl;}

return 0;
}

```