Advanced Topics in Electronic System Level Design

Term Project Report

N26094859 蔡宗穎

Abstract: This term project aims to integrate our DMA module with other components in the TLM 2.0 platform and run applications by using OVP simulator. I will illustrate some examples I thought helpful for Part2 in Part1, and then explain how I implemented the above mentioned platform.

Part 1. Useful Examples

In this part, I will pick 5 examples and illustrate how they can be helpful in Part2.

1.1 usingSystemC

(1) Folder Path: $IMPERAS_HOME/Examples/HelloWorld/usingSystemC

(2) Why choose this example: To learn how a bare metal program looks like and how to change the component to different variants.

(3) How it helps in doing Part2: This example is the platform program base I used to implement this term project. I only need to change the number of the components and define the connection of the processor (riscv_RV32G), ram, and DMA to bus (tlmDecoder).

1.2 SystemC_TLM

(1) Folder Path: $IMPERAS_HOME/Examples/PlatformConstruction/SystemC_TLM

(2) Why choose this example: To learn how a module (UART in this example) is connected to the platform. I also learned how to change tlm_platform.tcl (the hardware definition) while running this example.

(3) How it helps in doing Part2: Although the method of connecting the module in this example is different from that in my implementation, it is also a good example to learn how a module is connected to the platform, and how to write application.c by using the provided API like readReg8, readReg32, writeReg8, and writeReg32.

1.3 creatingDMAC/1.registers, 3. memoryAccess, 4. Interrupts, 5. nativeBehaviors

(1) Folder Path: $IMPERAS_HOME/Examples/Models/Peripherals/creatingDMAC

(2) Why choose this example: To learn what a real DMA looks like, how it can be used, specifically how to write the ISR (Interrupt Handler).

(3) How it helps in doing Part2: The 2 files, dmacRegister.h and riscvInterrupts.h, which the former can be used by changing some definition to let the CPU knows the

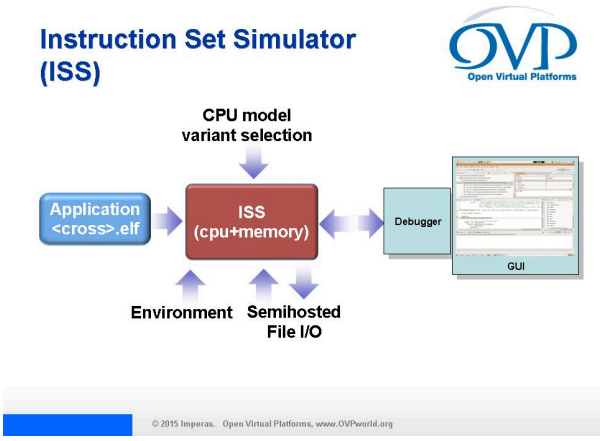
base address of every component, and the later can be used directly for interrupt signal. And the dmaTest.cpp is used as the template to write my application, and helps me a lot to learn how to write the ISR.

1.4 using ISS

(1) Folder Path: $IMPERAS_HOME/Examples/ HelloWorld/usingISS

(2) Why choose this example: To learn how to specify a processor in the program using Instruction Set Simulator

(3) How it helps in doing Part2: Although it is not directly related to this project, it is a good start for me to learn how to use the Instruction Set Simulator.

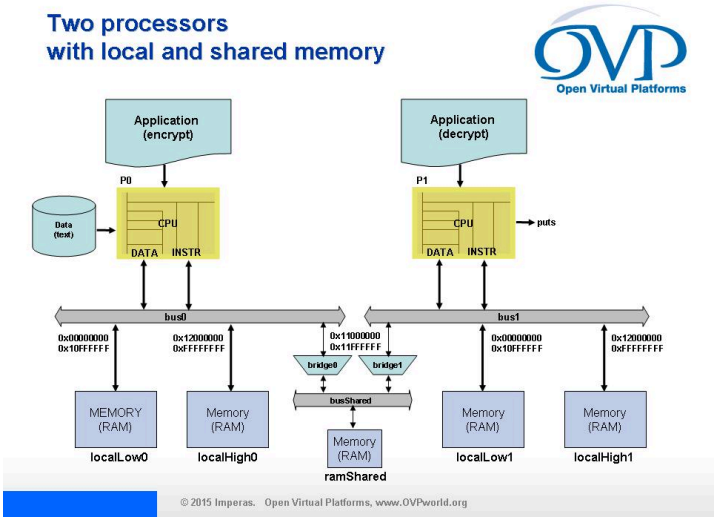


1.5 using ISS

(1) Folder Path: $IMPERAS_HOME/Examples/ HelloWorld/usingISS

(2) Why choose this example: To learn how multicore processors and shared memory are used in OVP.

(3) How it helps in doing Part2: Although it is not directly related to this project, it is a good start for me to learn how the multicore processors and shared memory are used in TLM 2.0 platform, which we should implement in the original project.



Part 2. How I implemented the term project

In this project, I used the folder “SystemC_TLM” as the base folder to implement my project, and I also used some files in the folder “creatingDMAC”, i.e. dmacRegister.h and riscvInterrupts.h. In the following sections, I will explain part of the codes needed to implement in this project.

2.1 Folder and files

```
ESL_Project_DMA
|
|---example.sh
|
|---application
|    |
|    |---dmacRegisters.h
|    |
|    |---riscvInterrupts.h
|    |
|    |---dmaTest.c
|
|---platform
|    |
|    |---dma.h
|    |
|    |---dma.cpp
|    |
|    |---platform.cpp
|    |
|    |---tlmSupport
|    |
|    |---Build
```

1. **example.sh** is the share script to run the overall DMA program.
2. **dmaRegisters.h** defines the addresses needed for the riscv processor to find and send control signals to DMA, and the source/target addresses for DMA to move data.
3. **riscvInterrupts.h** defines the information the riscv processor need to perform interrupt actions.
4. **dmaTest.c** is the application file that controls and displays the overall program.
5. **dma.h** is the head file for DMA to define the input/output and other variables.
6. **dma.cpp** is definition of the process and tlm transport.
7. **platform.cpp** defines all components and instances in the platform, and how the platform is connected.
8. **tlmSupport** is the folder containing all definitions of the tlm-related components.

2.2 Codes of Platform

(1) dma.h & dma.cpp

Since dma.h is almost the same as the assignment 2, except that the interrupt port should be modified to meet the riscv processor way.

```
tlm::tlm_analysis_port<unsigned int> Interrupt;
tlm_utils::simple_target_socket<dma> slave_p;
tlm_utils::simple_initiator_socket<dma> master_p;
```

The following figures shows the difference in dma.cpp. I changed the receiving part for control registers in dm::dma_proc() into dma::b_transport(…) part to ensure that the data is received at the right order and right time.

```cpp
#include "dma.h"

using namespace sc_core;
using namespace sc_dt;
using namespace std;

/*------------------------slave mode-------------------------*/
void dma::b_transport(tlm::tlm_generic_payload& trans, sc_time& delay){
    tlm::tlm_command cmd_s = trans.get_command();
    sc_dt::uint64 addr_s = trans.get_address();
    unsigned char* data_s = trans.get_data_ptr();
    //unsigned int len = trans.get_data_length();

    //cout << "---------------- Bus Transaction (CPU & DMA) -------------"<<endl;
        //cout << sc_time_stamp() << "\n";
    //cout << "Command : "<< (cmd_s ? "write" : "read") <<endl;
        //cout <<  "Address : 0x" << hex << addr_s <<endl;

    if(cmd_s == tlm::TLM_WRITE_COMMAND){
        //addressed write operations
        tmp_addr_mask = addr_s;
        //tmp_addr_mask = addr_s - DMA_base_addr.read();
        tmp_data = *(reinterpret_cast<int*>(data_s));
        slave_r_w = 1;
        //cout << "addr = 0x" << tmp_addr_mask << endl;
            //cout << "data = " << hex << tmp_data << endl;
            //cout << "slave_r_w = 1" << endl;

            switch(tmp_addr_mask){
            case 0x0:
                SOURCE = tmp_data - 0x50000000;
                //dma_state = slave_st;
                //cout << "SOURCE = " << hex << SOURCE << endl;
                //cout << "DMA_slave_st: 0x0" << endl;
            break;
            case 0x4:
                TARGET = tmp_data - 0x50000000;
                //dma_state = slave_st;
                //cout << "TARGET = " << hex << TARGET << endl;
                //cout << "DMA_slave_st: 0x4" << endl;
            break;
            case 0x8:
                SIZE = tmp_data;
                //dma_state = slave_st;
                //cout << "SIZE = " << hex << SIZE << endl;
                //cout << "DMA_slave_st: 0x8" << endl;
            break;
            case 0xc:
                START_CLEAR = tmp_data;
                //cout << "start_clear = " << START_CLEAR << endl;
                //if(tmp_data == 1 && SIZE.read() > 0){
                    //dma_state = dataR_st;
                    //cout << "START = " << hex << tmp_data << endl;
                    //cout << "DMA_slave_st: 0xc_1" << endl;
                    //master_port.r_w.write(0);
                    //master_port.addr = Control_Reg[0x0];
                //}else{
                    //dma_state = slave_st;
                    //slave_r_w = 0;
                    //cout << "DMA_slave_st: 0xc_2" << endl;
                //}
            break;
            default:
                //dma_state = slave_st;
                //cout << "DMA_slave_st: default" << endl;
            break;
        }
    }else{
        //addressed read operations
        slave_r_w = 0;
        //cout << "slave_r_w = 0" << endl;
    }



    wait(delay);  //use the external delay
    trans.set_response_status(tlm::TLM_OK_RESPONSE);
}
```

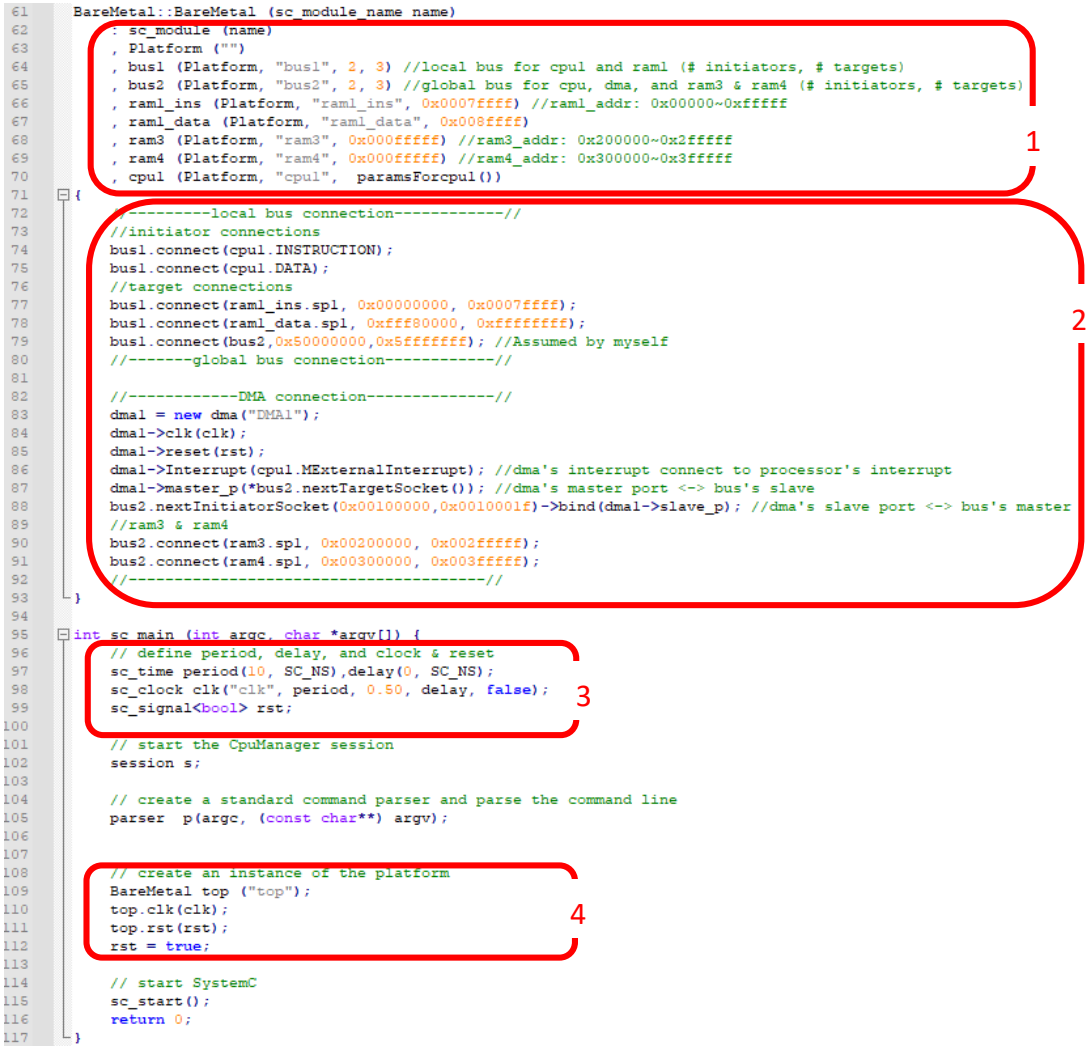
```
81
82  /*------------------------state proc-------------------------*/
83  void dma::dma_proc(){
84      //reset behavior
85      SOURCE = 0;
86      TARGET = 0;
87      SIZE = 0;
88      START_CLEAR = 0;
89      Interrupt.write(0);
90      slave_r_w = 0;
91      data_reg = 0;
92
93      while(1){
94          wait();
95          tlm::tlm_generic_payload* trans_m = new tlm::tlm_generic_payload;
96          tlm::tlm_command cmd_m;
97          sc_time delay = sc_time(10,SC_NS);
98          sc_uint<32> addr_s, addr_t;
99          uint32_t data_get;
100         uint32_t data_out;
101
102         switch(dma_state){
103             case slave_st: //State 0:Read from slave port
104
105                 if((slave_r_w == 1) && (START_CLEAR == 0)){
106                     //cout << "tmp_addr_mask: " << tmp_addr_mask << endl;
107                     dma_state = slave_st;
108                     //cout << "0-1" << endl;
109                 }
110                 else if (START_CLEAR == 1){//cout
111                     dma_state = dataR_st;
112                     //cout << "0-2" << endl;
113                 }
114             break;
115             case dataR_st: //State 1:Get source addr. from SOURCE register
116                 //slave_port.r_w.write(0);
117                 if((START_CLEAR == 1)&& (SIZE > 0)){
118                     //Data_Reg = master_port.data_in.read();
119                     //addr_s = SOURCE.read();
120                     //addr_t = TARGET.read();
121
122
123                     //addr_s = SOURCE.read();
124                     cmd_m = tlm::TLM_READ_COMMAND;
125                     trans_m->set_command(cmd_m);
126                     trans_m->set_address(SOURCE);
127                     trans_m->set_data_ptr(reinterpret_cast<unsigned char*>(&data_get));
128                     trans_m->set_data_length(4);
```

(2) platform.cpp

```
19  #include "tlm/tlmModule.hpp"
20  #include "tlm/tlmDecoder.hpp"
21  #include "tlm/tlmMemory.hpp"
22  #include "dma.h"
23  // Processor configuration
24  #include "riscv.ovpworld.org/processor/riscv/1.0/tlm/riscv_RV32G.igen.hpp"
25
26
27
28  using namespace sc_core;
29
30  ////////////////////////////////////////////////////////////////////////////////
31  //                      BareMetal Class                                       //
32  ////////////////////////////////////////////////////////////////////////////////
33
34  class BareMetal : public sc_module {
35
36    public:
37      BareMetal (sc_module_name name);
38      sc_in <bool>          clk;
39      sc_in <bool>          rst;
40      tlmModule             Platform;
41      tlmDecoder            bus1; //local bus
42      tlmDecoder            bus2; //global bus
43      tlmRam                ram1_ins;
44      tlmRam                ram1_data;
45      tlmRam                ram3;
46      tlmRam                ram4;
47      dma*                  dma1;
48      riscv_RV32G           cpu1;
49
50
51    private:
52
53      params paramsForcpu1() {
54          params p;
55          p.set("defaultsemihost", true);
56          return p;
57      }
58
59  }; /* BareMetal */
```

This file is revised from the example.

In this part, it declares the components in the class “BareMetal.” The platform, bus, and ram are declared using classes in tlmSupport. The dma module we declared must using pointers.



This part is illustrated in 4 sections.

1 shows the declaration of # of initiators and # of targets for the bus, and the size in the memory (ram1_ins, ram1_data, ram3, and ram4). Note that the ram connected to the processor must be separated into 2 parts, i.e. instruction memory (ram1_ins) and data memory (ram1_data), in order to cover the last address 0xffffffff.

2 shows the connection between all components. The approach for connecting the bus and pre-defined components is using “bus.connect(target, address_low, address_high)”. The approach for connecting the bus and our own module is slightly different, it used the pre-defined function (nextInitiatorSocket, nextTargetSocket) declared in tlmDecoder.hpp. Also, the interrupt signal is loaded by connecting the Interrupt port in DMA and MExternalInterrupt port in the processor riscv_RV32G.

3 shows the declaration of period, delay in sc_time class, clk in sc_clock class, and rst in sc_signal class.

4 shows the creation of the instance of the platform, and the corresponding port with clk and rst.

2.3 Codes of Application

(1) dmaRegisters.h & riscvInterrupts.h

In dmaRegisters.h, it defines the addressed used for riscv_RV32G processor, and some of them are used in the dmaTest.c (application file). This file is revised from the example file.

```
////////////////////// Registers for DMA Controller based on riscv_RV32G ////////////////////////////

#ifndef DMAC_REGISTERS_H
#define DMAC_REGISTERS_H

#define bus2                    0x50000000

#define RAM1_data_BASE_ADDR     0x00000000
#define RAM1_ins_BASE_ADDR      0x00000000
#define RAM3_BASE_ADDR          0x50200000
#define RAM4_BASE_ADDR          0x50300000

#define RAM3_MOVE_BASE_ADDR     0x50200400
#define RAM4_MOVE_BASE_ADDR     0x50300400
#define MODE_ADDR               0x50200800  //Switch for moving mod. If 0: RAM3 => RAM4, else : RAM4 => RAM3.

#define DMA_BASE_ADDR           0x50100000


#define DMA_SOURCE              0x50100000
#define DMA_TARGET              0x50100004
#define DMA_SIZE                0x50100008
#define DMA_START_CLEAR         0x5010000C
```

Since riscvInterrupts.h is not revised or changed, so I will skip it here. It provides some methods for interrupt signal to the processor.

(2) dmaTest.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "dmacRegisters.h"

typedef unsigned int  Uns32;
typedef unsigned char Uns8;

#include "riscvInterrupts.h"

#define LOG(_FMT, ...)  printf( "[DMA Program]: " _FMT,  ## __VA_ARGS__)
int send_time = 0;
int count = 0;
int global_count = 0;

void int_init(void (*handler)()) {

    // Set MTVEC register to point to handler function in direct mode
    int handler_int = (int) handler & ~0x1;
    write_csr(mtvec, handler_int);

    // Enable Machine mode external interrupts
    set_csr(mie, MIE_MEIE);
}

void int_enable() {
    set_csr(mstatus, MSTATUS_MIE);
}

static inline void writeReg32(Uns32 address, Uns32 offset, Uns32 value)
{
    *(volatile Uns32*) (address + offset) = value;
}

static inline Uns32 readReg32(Uns32 address, Uns32 offset)
{
    return *(volatile Uns32*) (address + offset);
}

static inline void writeReg8(Uns32 address, Uns32 offset, Uns8 value)
{
    *(volatile Uns8*) (address + offset) = value;
}

static inline Uns8 readReg8(Uns32 address, Uns32 offset)
{
    return *(volatile Uns8*) (address + offset);
}

 volatile static Uns32 interruptCount = 0;
```

```c
void interruptHandler(void)
{
    if(send_time == 0){
        LOG("Interrupt received.\n");
        writeReg32(DMA_BASE_ADDR, (uint)(0xC), (uint)(0));
        LOG("CPU send Clear signal (START_CLEAR = 0).\n");
        send_time++;
        global_count++;
    }

}

int main(int argc, char **argv)
{
    //Step1. Initialization of RAM3 & RAM4.
    LOG("Step1. Initialization of RAM3 & RAM4.\n");
    LOG("Step1_1_1. Initialization of RAM3.\n");
    for(int i = 0; i< 2048;i++){
        writeReg8(RAM3_BASE_ADDR, i, (Uns8)0);
    }
    LOG("Step1_1_2. Initialization of RAM4.\n");
    for(int i = 0; i< 2048;i++){
        writeReg8(RAM4_BASE_ADDR, i, (Uns8)0);
    }
    LOG("Step1_2_1. Checking first 64 grouped-data in RAM3.\n");
     for(int i = 0; i< 64;i++){
       printf("Data in RAM3: ADDR[%x] to [%x]\n",(RAM3_BASE_ADDR + i*64), (RAM3_BASE_ADDR + (i+1)*64 - 1));
       for(int j = 0;j < 16;j++){
        printf("%d_",readReg8((RAM3_BASE_ADDR + i*64), j ));
       }
       printf("\n");

    }
    printf("------------------------------------------------\n");

    LOG("Step1_2_1. Checking first 64 grouped-data in RAM4.\n");
     for(int i = 0; i< 64;i++){
       printf("Data in RAM4: ADDR[%x] to [%x]\n",(RAM4_BASE_ADDR + i*64), (RAM4_BASE_ADDR + (i+1)*64 - 1));
       for(int j = 0;j < 16;j++){
        printf("%d_",readReg8((RAM4_BASE_ADDR + i*64), j ));
       }
       printf("\n");

    }
    printf("------------------------------------------------\n");
    LOG("---------- Step1 Done. ----------\n");

    //Step2. Writing 1KB data to RAM3 & RAM4.
    LOG("Step2. Writing 1KB data to RAM3 & RAM4.\n");
    LOG("Step2_1. Writing 1KB data for the repeatition of 0123456789ABCDEF to RAM3.\n");
    Uns8 CharData = '0';
    for(int i = 0; i < 1024;i++){
        writeReg8(RAM3_MOVE_BASE_ADDR, i, CharData);

        if((CharData >= '0' && CharData < '9') || (CharData >= 'A' && CharData < 'F')){
            CharData = CharData + 1;
        }else if(CharData == '9'){
            CharData = 'A';
        }else if(CharData == 'F'){
            CharData = '0';
        }
    }
    LOG("Step2_2. Writing 1KB data for the repeatition of FEDCBA9876543210 to RAM4.\n");
    CharData = 'F';
    for(int i = 0; i < 1024;i++){
        writeReg8(RAM4_MOVE_BASE_ADDR, i, CharData);

        if((CharData > '0' && CharData <= '9') || (CharData > 'A' && CharData <= 'F')){
            CharData = CharData - 1;
        }else if(CharData == 'A'){
            CharData = '9';
        }else if(CharData == '0'){
            CharData = 'F';
        }
    }
    LOG("Step2_3. Checking Data in RAM3 & RAM4.\n");
    LOG("Step2_3_1. Checking first 64 grouped-data in RAM3.\n");
    for(int i = 0; i< 16;i++){
       printf("Data in RAM3: ADDR[%x] to [%x]\n",(RAM3_BASE_ADDR + i*64), (RAM3_BASE_ADDR + (i+1)*64 - 1));
       for(int j = 0;j < 16;j++){
        printf("%d_",readReg8((RAM3_BASE_ADDR + i*64), j ));
       }
       printf("\n");
    }
    for(int i = 16; i< 32;i++){
       printf("Data in RAM3: ADDR[%x] to [%x]\n",(RAM3_BASE_ADDR + i*64), (RAM3_BASE_ADDR + (i+1)*64 - 1));
       for(int j = 0;j < 16;j++){
        printf("%c_",readReg8((RAM3_BASE_ADDR + i*64), j ));
       }
       printf("\n");
    }
    for(int i = 32; i< 64;i++){
       printf("Data in RAM3: ADDR[%x] to [%x]\n",(RAM3_BASE_ADDR + i*64), (RAM3_BASE_ADDR + (i+1)*64 - 1));
       for(int j = 0;j < 16;j++){
        printf("%d_",readReg8((RAM3_BASE_ADDR + i*64), j ));
       }

       printf("\n");

    }
```

1 is to flush all cells in ram3 and ram4 using the pre-defined functions writeReg8 in the file. Note that because core dumped will occur if I set the size to 1MB, so I set to 2kB for demonstration.

2 is to print data in ram3 and ram4 using the pre-defined functions readReg8, LOG and printf in the file.

3 is to set 0 to F and F to 0 in ram3 and ram4, respectively. I used if-else statements to automatic set another group after a group of 0 to F or F to 0 is finished.

4 is the interrupt handler (ISR). After the processor received the interrupt signal from DMA, it will send clear signal to DMA, which is writing the control register at 0xC in DMA here. Since the frequency of the processor is much faster than the DMA, so it will send clear signal to DMA for many times, and writing the control register at send_time = 0 could let the DMA only receive the clear signal once.

```
221     printf("-------------------------------------------------\n");
222     LOG("Step2_3_2. Checking first 64 grouped-data in RAM4.\n");
223     for(int i = 0; i< 16;i++){
224        printf("Data in RAM4: ADDR[%x] to [%x]\n",(RAM4_BASE_ADDR + i*64), (RAM4_BASE_ADDR + (i+1)*64 - 1));
225        for(int j = 0;j < 16;j++){
226         printf("%d_",readReg8((RAM4_BASE_ADDR + i*64), j ));
227        }
228        printf("\n");
229     }
230     for(int i = 16; i< 32;i++){
231        printf("Data in RAM4: ADDR[%x] to [%x]\n",(RAM4_BASE_ADDR + i*64), (RAM4_BASE_ADDR + (i+1)*64 - 1));
232        for(int j = 0;j < 16;j++){
233         printf("%c_",readReg8((RAM4_BASE_ADDR + i*64), j ));
234        }
235        printf("\n");
236     }
237     for(int i = 32; i< 64;i++){
238        printf("Data in RAM4: ADDR[%x] to [%x]\n",(RAM4_BASE_ADDR + i*64), (RAM4_BASE_ADDR + (i+1)*64 - 1));
239        for(int j = 0;j < 16;j++){
240         printf("%d_",readReg8((RAM4_BASE_ADDR + i*64), j ));
241        }
242        printf("\n");
243     }
244     printf("-------------------------------------------------\n");
245     LOG("---------Step2 Done.---------\n");
246
247
248     //API to launch ISR
249     int_init(trap_entry);
250     int_enable();
251
252     LOG("Step3. Moving 1KB data from RAM3 to RAM4 and from RAM4 to RAM3 for 3 times.\n");
253     while(count < 3){
254        if(readReg8(MODE_ADDR, 0) == 0 ){
255              send_time = 0;
256              LOG("Step3_1. CPU calls DMA for %d times to move data from RAM3 to RAM4.\n", count + 1);
257              writeReg32(DMA_BASE_ADDR, (uint)(0x0), (uint)(RAM3_MOVE_BASE_ADDR));
258              writeReg32(DMA_BASE_ADDR, (uint)(0x4), (uint)(RAM4_BASE_ADDR));
259              writeReg32(DMA_BASE_ADDR, (uint)(0x8), (uint)(256));
260              writeReg32(DMA_BASE_ADDR, (uint)(0xC), (uint)(1));
261              LOG("CPU finished calling DMA for %d times to move data from RAM3 to RAM4.\n", count + 1);
262              LOG("CPU waiting for interrupt signal from the DMA.\n", count + 1);
263              wfi();
264              LOG("Step3_2. Checking first 64 grouped-data in RAM4.\n");
265
266           for(int i = 0; i< 32;i++){
267              printf("Data in RAM4: ADDR[%x] to [%x]\n",(RAM4_BASE_ADDR + i*64), (RAM4_BASE_ADDR + (i+1)*64 - 1));
268              for(int j = 0;j < 16;j++){
269               printf("%c_",readReg8((RAM4_BASE_ADDR + i*64), j ));
270              }
271              printf("\n");
272           }
273           for(int i = 32; i< 64;i++){
274              printf("Data in RAM4: ADDR[%x] to [%x]\n",(RAM4_BASE_ADDR + i*64), (RAM4_BASE_ADDR + (i+1)*64 - 1));
275              for(int j = 0;j < 16;j++){
276               printf("%d_",readReg8((RAM4_BASE_ADDR + i*64), j ));
277              }
278              printf("\n");
279           }
280           printf("-------------------------------------------------\n");

281           LOG("Step3_3. Changing Mode to another (RAM4 => RAM3) \n");
282           writeReg8(MODE_ADDR, 0, 1);
283           LOG("CPU finished changing mode. Now the mode is %x. \n", readReg8(MODE_ADDR, 0));
284
```

5

7

```
285         }else if(readReg8(MODE_ADDR, 0) == 1 ){
286                 send_time = 0;
287                 LOG("Step3_1. CPU calls DMA for %d times to move data from RAM4 to RAM3.\n", count + 1);
288                 writeReg32(DMA_BASE_ADDR, (uint)(0x0), (uint)(RAM4_MOVE_BASE_ADDR));
289                 writeReg32(DMA_BASE_ADDR, (uint)(0x4), (uint)(RAM3_BASE_ADDR));
290                 writeReg32(DMA_BASE_ADDR, (uint)(0x8), (uint)(256)); //
291                 writeReg32(DMA_BASE_ADDR, (uint)(0xC), (uint)(1));
292                 LOG("CPU finished calling DMA for %d times to move data from RAM4 to RAM3.\n", count + 1);
293                 LOG("CPU waiting for interrupt signal from the DMA.\n", count + 1);
294                 wfi();
295                 LOG("Step3_2. Checking first 64 grouped-data in RAM3.\n");
296
297             for(int i = 0; i< 32;i++){
298                 printf("Data in RAM3: ADDR[%x] to [%x]\n",(RAM3_BASE_ADDR + i*64), (RAM3_BASE_ADDR + (i+1)*64 - 1));
299                 for(int j = 0;j < 16;j++){
300                  printf("%c_",readReg8((RAM3_BASE_ADDR + i*64), j ));
301                 }
302                 printf("\n");
303             }
304             for(int i = 32; i< 64;i++){
305                 printf("Data in RAM3: ADDR[%x] to [%x]\n",(RAM3_BASE_ADDR + i*64), (RAM3_BASE_ADDR + (i+1)*64 - 1));
306                 for(int j = 0;j < 16;j++){
307                  printf("%d_",readReg8((RAM3_BASE_ADDR + i*64), j ));
308                 }
309                 printf("\n");
310             }
311             printf("-------------------------------------------------\n");
312             LOG("Step3_3. Changing Mode to another (RAM3 => RAM4).\n");
313             writeReg8(MODE_ADDR, 0, 0);
314             LOG("CPU finished changing mode. Now the mode is %x. \n", readReg8(MODE_ADDR, 0));
315             count++;
316         }else{
317                 LOG("Error: The value in MODE_ADDR must be wrong.\n");
318         }
319
320     }
321     LOG("DMA finished moving data from RAM3 to RAM4 and from RAM4 to RAM3.\n");
322     LOG("Overall data moved for %d times.\n", count);
323     LOG("Overall interrupt signal is pulled up for %d times.\n", global_count);
324     LOG("---------Step3 Done.---------\n");
325     LOG("---------End of DMA Program.---------\n");
326
327     return 1;
328 }
```

6

8

In 5 and 6, there are two modes for moving data. When the value in MODE_ADDR is 0, the DMA moves data from ram3 to ram4 and from ram4 to ram3 otherwise. Also, I use the pre-defined function writeReg32 to change the source address, target address, size of the data, and start_clear in the control registers of the DMA. And I also reset send_time to 0 in the beginning of the moving to indicate that this is a new move (send_time is used in the Interrupt Handler, or so called ISR). Note that since the data length set in TLM 2.0 in dma.cpp is 4, so the size here is 256, not 1024(1kB).

7 and 8 shows the change in mode using the pre-defined function writeReg8.

2.4 Examples of the display

Since the result is very long, I will only screenshot some of them. The whole result is recorded in the file result.txt.

(1) Initialization of ram3 & ram4

```
Data in RAM3: ADDR[502003c0] to [502003ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200400] to [5020043f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200440] to [5020047f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200480] to [502004bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[502004c0] to [502004ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200500] to [5020053f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200540] to [5020057f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200580] to [502005bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[502005c0] to [502005ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200600] to [5020063f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200640] to [5020067f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200680] to [502006bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[502006c0] to [502006ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200700] to [5020073f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200740] to [5020077f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200780] to [502007bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[502007c0] to [502007ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200800] to [5020083f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200840] to [5020087f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
```

```
Data in RAM4: ADDR[503003c0] to [503003ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300400] to [5030043f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300440] to [5030047f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300480] to [503004bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[503004c0] to [503004ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300500] to [5030053f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300540] to [5030057f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300580] to [503005bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[503005c0] to [503005ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300600] to [5030063f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300640] to [5030067f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300680] to [503006bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[503006c0] to [503006ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300700] to [5030073f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300740] to [5030077f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300780] to [503007bf]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[503007c0] to [503007ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300800] to [5030083f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300840] to [5030087f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
```

(2) Set 0 – F and F – 0 in ram3 & ram4, respectively.

```
Data in RAM3: ADDR[502003c0] to [502003ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200400] to [5020043f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200440] to [5020047f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200480] to [502004bf]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[502004c0] to [502004ff]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200500] to [5020053f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200540] to [5020057f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200580] to [502005bf]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[502005c0] to [502005ff]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200600] to [5020063f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200640] to [5020067f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200680] to [502006bf]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[502006c0] to [502006ff]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200700] to [5020073f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200740] to [5020077f]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200780] to [502007bf]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[502007c0] to [502007ff]
0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
Data in RAM3: ADDR[50200800] to [5020083f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM3: ADDR[50200840] to [5020087f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
```

```
Data in RAM4: ADDR[503003c0] to [503003ff]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300400] to [5030043f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300440] to [5030047f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300480] to [503004bf]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[503004c0] to [503004ff]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300500] to [5030053f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300540] to [5030057f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300580] to [503005bf]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[503005c0] to [503005ff]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300600] to [5030063f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300640] to [5030067f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300680] to [503006bf]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[503006c0] to [503006ff]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300700] to [5030073f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300740] to [5030077f]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300780] to [503007bf]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[503007c0] to [503007ff]
F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
Data in RAM4: ADDR[50300800] to [5030083f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
Data in RAM4: ADDR[50300840] to [5030087f]
0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
```

(3) Demonstration of moving data, starting and exiting of ISR. (First move here.)

```
551 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
552 -------------------------------------------------
553 [DMA Program]: ---------Step2 Done.---------
554 [DMA Program]: Step3. Moving 1KB data from RAM3 to RAM4 and from RAM4 to RAM3 for 3 times.
555 [DMA Program]: Step3_1. CPU calls DMA for 1 times to move data from RAM3 to RAM4.
556 [DMA Program]: CPU finished calling DMA for 1 times to move data from RAM3 to RAM4.
557 [DMA Program]: CPU waiting for interrupt signal from the DMA.
558 [DMA Program]: Interrupt received.
559 [DMA Program]: CPU send Clear signal (START_CLEAR = 0).
560 [DMA Program]: Step3_2. Checking first 64 grouped-data in RAM4.
```

(4) ram3 and ram4 after moving for 3 times.

```
1138 Data in RAM4: ADDR[503003c0] to [503003ff]
1139 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1140 Data in RAM4: ADDR[50300400] to [5030043f]
1141 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1142 Data in RAM4: ADDR[50300440] to [5030047f]
1143 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1144 Data in RAM4: ADDR[50300480] to [503004bf]
1145 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1146 Data in RAM4: ADDR[503004c0] to [503004ff]
1147 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1148 Data in RAM4: ADDR[50300500] to [5030053f]
1149 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1150 Data in RAM4: ADDR[50300540] to [5030057f]
1151 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1152 Data in RAM4: ADDR[50300580] to [503005bf]
1153 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1154 Data in RAM4: ADDR[503005c0] to [503005ff]
1155 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1156 Data in RAM4: ADDR[50300600] to [5030063f]
```

```
1157 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1158 Data in RAM4: ADDR[50300640] to [5030067f]
1159 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1160 Data in RAM4: ADDR[50300680] to [503006bf]
1161 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1162 Data in RAM4: ADDR[503006c0] to [503006ff]
1163 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1164 Data in RAM4: ADDR[50300700] to [5030073f]
1165 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1166 Data in RAM4: ADDR[50300740] to [5030077f]
1167 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1168 Data in RAM4: ADDR[50300780] to [503007bf]
1169 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1170 Data in RAM4: ADDR[503007c0] to [503007ff]
1171 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1172 Data in RAM4: ADDR[50300800] to [5030083f]
1173 0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
1174 Data in RAM4: ADDR[50300840] to [5030087f]
1175 0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
```

```
1275 Data in RAM3: ADDR[502003c0] to [502003ff]
1276 F_E_D_C_B_A_9_8_7_6_5_4_3_2_1_0_
1277 Data in RAM3: ADDR[50200400] to [5020043f]
1278 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1279 Data in RAM3: ADDR[50200440] to [5020047f]
1280 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1281 Data in RAM3: ADDR[50200480] to [502004bf]
1282 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1283 Data in RAM3: ADDR[502004c0] to [502004ff]
1284 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1285 Data in RAM3: ADDR[50200500] to [5020053f]
1286 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1287 Data in RAM3: ADDR[50200540] to [5020057f]
1288 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1289 Data in RAM3: ADDR[50200580] to [502005bf]
1290 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1291 Data in RAM3: ADDR[502005c0] to [502005ff]
1292 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
```

```
1293 Data in RAM3: ADDR[50200600] to [5020063f]
1294 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1295 Data in RAM3: ADDR[50200640] to [5020067f]
1296 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1297 Data in RAM3: ADDR[50200680] to [502006bf]
1298 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1299 Data in RAM3: ADDR[502006c0] to [502006ff]
1300 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1301 Data in RAM3: ADDR[50200700] to [5020073f]
1302 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1303 Data in RAM3: ADDR[50200740] to [5020077f]
1304 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1305 Data in RAM3: ADDR[50200780] to [502007bf]
1306 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1307 Data in RAM3: ADDR[502007c0] to [502007ff]
1308 0_1_2_3_4_5_6_7_8_9_A_B_C_D_E_F_
1309 Data in RAM3: ADDR[50200800] to [5020083f]
1310 1_0_0_0_0_0_0_0_0_0_0_0_0_0_0_0_
```

(5) Record for # of moves and # of interrupt signals in the end of the DMA program.

```
1374 [DMA Program]: Step3_3. Changing Mode to another (RAM3 => RAM4).
1375 [DMA Program]: CPU finished changing mode. Now the mode is 0.
1376 [DMA Program]: DMA finished moving data from RAM3 to RAM4 and from RAM4 to RAM3.
1377 [DMA Program]: Overall data moved for 3 times.
1378 [DMA Program]: Overall interrupt signal is pulled up for 6 times.
1379 [DMA Program]: ---------Step3 Done.---------
1380 [DMA Program]: ---------End of DMA Program.---------
1381
1382 Info: /OSCI/SystemC: Simulation stopped by user.
1383
1384 OVPsim finished: Thu Jul  1 14:21:17 2021
```