



**A|RT Designer
Homework Assignment
EE214A**

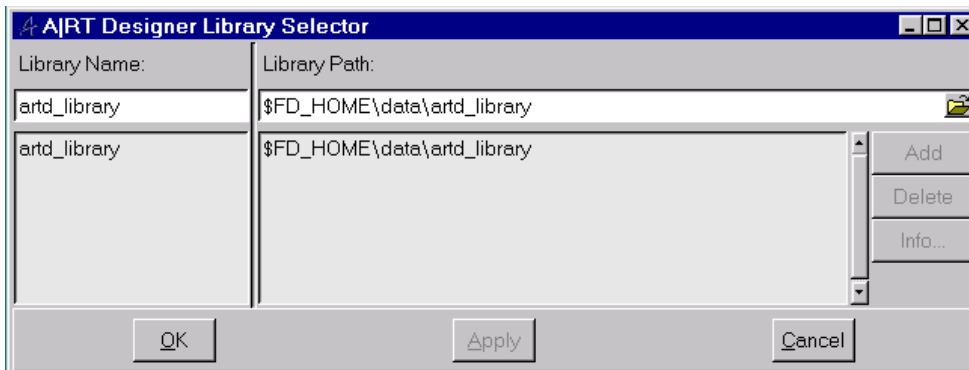
NAME : _____

1 Introduction

The major objective of the assignment is to instruct how A|RT Designer works and how ART Designer is driven using a very basic starting algorithm.

2 Setup

Apart from software installation and license setup, only one other thing needs to be done: setup of the libraries inside A|RT Designer. Choose "Options > Library Selector," modify the paths to reflect your installation.



3 Projects & files

This section gives detailed information about the designs used in this assignment.

3.1 Problem_1

The algorithm used for this introduction demo of A|RT Designer contains two arrays that are multiplied pair-wise using a loop. However, the second array will be multiplied pair-wise in reverse order. On the product, some conditional operations are performed. The first problem consists of a default run of A|RT Designer, once without the scheduler option "loop-folding" and once with loop folding enabled.

src.cxx - This is the source code for the first problem and is contained in the `Problem_1` directory. You will need to create a new project and then import this code into that project.

```
#include <fxp.h>

void demo (
    const Fix<8,7> A[8],
    const Fix<8,7> B[8],
    Fix<8,7>& overall_result,
    Uint<4>& position)
{
    #pragma OUT overall_result position

    static Fix<8,7> result = 0;
    Fix<8,7> current_result = 0;
    Uint<4> max = "0b1111";

    for (Uint<4> i = 0; i < 8; i++)
    {
        Fix<8,7> product = A[i] * B[7-i];
        if (product > current_result)
        {
            current_result = -product;
            position = max+1;
        } // end if
        else
        {
            current_result = product<<2;
            position = max-1;
        } // end else
    } // end for

    result = result + current_result;
    overall_result = result;
}
```

architecture.pra

The default architecture is used.

```
// Datapath
//-----
instantiate("artd_library","inport","inport_1");
instantiate("artd_library","outport","outport_1");

instantiate("artd_library","alu","alu_1");

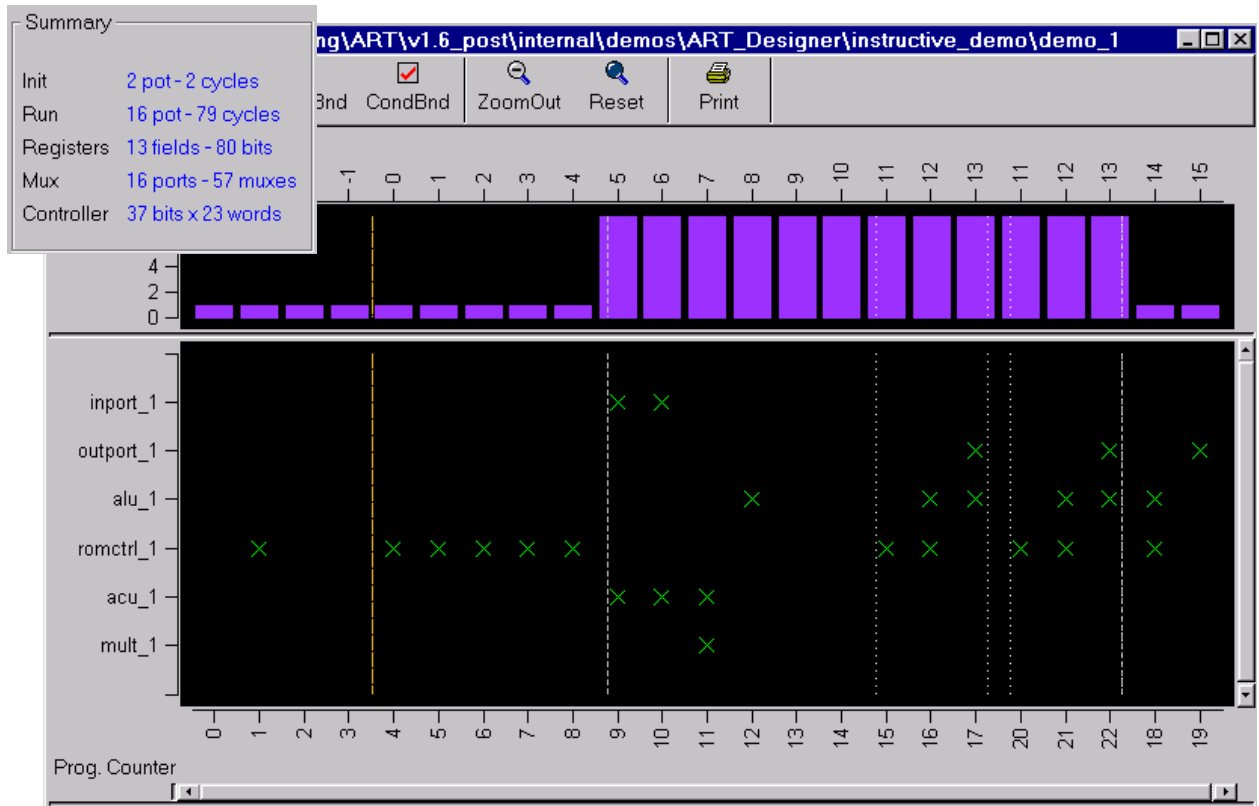
instantiate("artd_library","romctrl","romctrl_1");
instantiate("artd_library","acu","acu_1");
instantiate("artd_library","rom","rom_1");
instantiate("artd_library","ram","ram_1");

instantiate("artd_library","mult","mult_1");

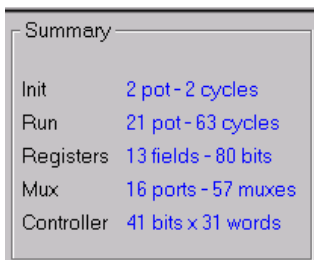
// Multibranch Controller (ctrldelay = 3, jumpdelay = 2)
//-----
instantiate("artd_library","mbc_23","ctrl");
```

NOTE: If you enable unconstrained loop folding the cycle-count is less. Loop folding reduces the total machine cycle-count of a for loop by increasing the available parallelism within every iteration. Higher parallelism can be obtained by moving individual RT's (Register Transfers) between iterations.

- Compile the source code to check for errors.
 - View the architecture definition pragma file. For the first problem, the default architecture is used and the resources are instantiated at a high level (MULT, RAM, ALU, etc.). Build the architecture. Pop up the Architecture Report and review the resources allocated for this architecture.
 - Run the scheduler with loop folding disabled. You can do this by going into the "Options," "Schedule Operations" menu and unchecking "Unconstrained Folding."
- Pop up Load view to view the EXU (Execution Unit) activity on a cycle basis. Also, you should clearly see the loop-bounds and the bounds of the if and else branches. The architecture should require 79 cycles to execute as reported in the "Summary".



Several optimization techniques can be used for improving the basic schedule. You can activate/deactivate them either in the "Options" menu, or by means of the pragmas. In this case, we will enable loop folding using the "Options" menu. The loop folding optimization will try to reduce the total machine cycle count for loop by increasing the available parallelism within every iteration. Moving individual register transfers between iterations can obtain higher parallelism. Enable loop folding in the "Options" menu and re-run scheduling. Pop up the Load View and show the differences. The architecture should require 63 cycles as shown in the "Summary" report in the user interface.



3.2 Problem_2

In this problem, an architecture other than the default is used. Inspecting the load view of Problem_1 in combination with the schedule report reveals that in the loop, the INPORT is accessed two times, once for every input variable, that the ACU is very busy in the loop and that the ROMCTRL and ALU are heavily loaded in the conditional branches. As a consequence, to improve the design, you will add a second INPORT is instantiated with a separate ACU and ROMCTRL. At the same time, we will use the second ACU to decrease the load on the ALU inside the conditional branches.

src.cxx

In Problem_2, you have several tasks. First, you will need to modify the Architecture Pragma file (architecture.pra) to include a second instantiation of the INPORT. Second, you will need to attach a label to the loop to identify the loop to the scheduler. Third, attach labels to the increment and decrement code fragments in the loop to allow A|RT Designer to map the specific increment and decrement operations to separate ACUs. The original source code is shown below. You will need to modify this code with the labels and then rerun in A|RT Designer. You can either modify the existing code in the Problem_1 project or create a new project called Problem_2 and import the Problem_1 source code in. Hints are shown below:

```
#include <fxp.h>

void demo (
    const Fix<8,7> A[8],
    const Fix<8,7> B[8],
    Fix<8,7>& overall_result,
    Uint<4>& position)
{
    #pragma OUT overall_result position

    static Fix<8,7> result = 0;
    Fix<8,7> current_result = 0;
    Uint<4> max = "0b1111";

    for (Uint<4> i = 0; i < 8; i++) <- LABEL
    {
        Fix<8,7> product = A[i] * B[7-i];
        ifproduct: if (product > current_result)
        {
            current_result = -product;
            position = max+1; <- LABEL
        } // end if
        else
        {
            current_result = product << 2;
            position = max-1; <- LABEL
        } // end else
    } // end for

    result = result + current_result;
    overall_result = result;
}
```

architecture.pra

```
instantiate("artd_library","inport","inport_1");
<USE THE ABOVE LINE AS A TEMPLATE TO CREATE A SECOND "inport" INSTANTIATION>
instantiate("artd_library","outport","outport_1");

instantiate("artd_library","alu","alu_1");

instantiate("artd_library","romctrl","romctrl_1");
<USE THE ABOVE LINE AS A TEMPLATE TO CREATE A SECOND "romctrl" INSTANTIATION>
instantiate("artd_library","acu","acu_1");
<USE THE ABOVE LINE AS A TEMPLATE TO CREATE A SECOND "acu" INSTANTIATION>
instantiate("artd_library","rom","rom_1");
instantiate("artd_library","ram","ram_1");

instantiate("artd_library","mult","mult_1");
instantiate("artd_library","mbc_23","ctrl");
```

algmapping.pra

```
// allocate a second input port and corresponding resources
assign_variable("/demo/B","inport_2");
dedicate("acu_2","inport_2");
dedicate("romctrl_2","acu_2");

// use second acu to increase parallelism with alu

assign_operation("/demo/loopi/ifproduct/incr","acu_2");
<USE THE ABOVE assign_operation STATEMENT TO ASSIGN THE LABELED INCREMENT AND DECREMENT
CODE FRAGMENTS TO AN ACU>
```

If you correctly label the source code and correctly add the additional ACUs to the design in the architecture definition pragma file, you can reduce the cycle count. What is the number returned in the "Summary" (Run) report? _____

The execution of the conditional operations in the loop takes two cycles. Using cross-highlighting reveals that the second cycle is needed to write the results to the OUTPORT. Reducing the number of cycles to a single-cycle can be realized by rewriting the source in a way that only at the end of the function the result is written to the OUTPORT.

3.3 Problem_3

In this third version the source-code is going to be rewritten in such a way that the condition operations in the loop only take one cycle instead of two (see load View of Problem_2 in combination with schedule report by using cross-highlighting). To achieve this, a dummy variable is introduced which is only going to be written to the output at the end of the function instead of every time that the conditional operations are called. As a consequence, the number of cycles needed for the loop is being reduced with one. Rewrite the code from Problem_2 to incorporate a dummy variable in the loop. Hints are contained below:

src.cxx

```
#include <fxp.h>

void demo (
    const Fix<8,7> A[8],
    const Fix<8,7> B[8],
    Fix<8,7>& overall_result,
    Uint<4>& position)
{
    #pragma OUT overall_result position

    static Fix<8,7> result = 0;
    Fix<8,7> current_result = 0;
    Uint<4> max = "0bull111";
    Uint<4> tmp_pos; <- DUMMY VARIABLE NAME

    loopi: for (Uint<4> i = 0; i < 8; i++)
    {
        Fix<8,7> product = A[i] * B[7-i];

        ifproduct: if (product > current_result)
        {
            current_result = -product;
            incr: position = max+1; <- USE DUMMY VARIABLE TO HOLD VALUE
        } // end if
        else
        {
            current_result = product << 2;
            decr: position = max-1; <- USE DUMMY VARIABLE TO HOLD VALUE
        } // end else

    } // end for
    result = result + current_result;
    overall_result = result; <- YOU WILL NEED TO ADD A CODE FRAGMENT THAT WRITES THE VALUE
    OF position HERE ASSIGNING THE VALUE OF THE DUMMY VARIABLE ABOVE
}
}
```

architecture.pra

No changes need to be made.

algmapping.pra

No changes need to be made.

A dummy variable is introduced in the code to avoid the extra cycle needed for the conditional operations in the loop. Nothing will change in the architecture definition pragma file, meaning that the same architecture is used. The algorithm pragma file also remains the same. Run all steps. The cycle-count as reported in the "Summary" is decreased from to _____ cycles.