



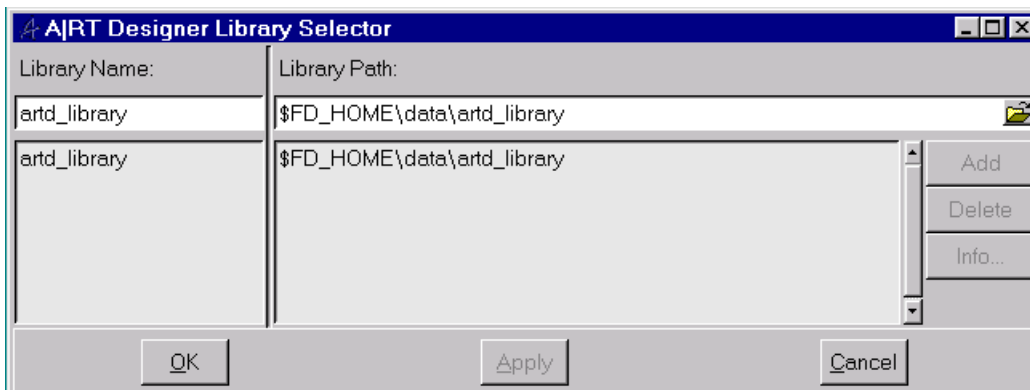
A|RT Designer Workshop FIR Filter

1 Introduction

The major objective of this workshop is to provide the user on the basic operation of A|RT Designer, how to interrogate A|RT Designer through the user interface and the practical application of A|RT Designer using a FIR filter as a design example.

2 A|RT Designer Setup

Apart from software installation and license setup, only one other thing needs to be done: setup of the libraries inside A|RT Designer. Invoke the tool by typing `$FD_HOME/bin/artdesignerpro`. Choose "Options > Library Selector" and modify the paths as shown below.



3 Projects & files

There will be four parts to this workshop that examines the architectural exploration capabilities of A|RT Designer.

3.1 FIR1

Contains the C-code of a FIR band-pass filter with following characteristics:

- The filter is intended to work with a sample frequency of 32 kHz
- The center frequency of the pass-band is 2 kHz
- The pass band has a quality factor of 2.2
- The filter has a set of 71 symmetric filter coefficients

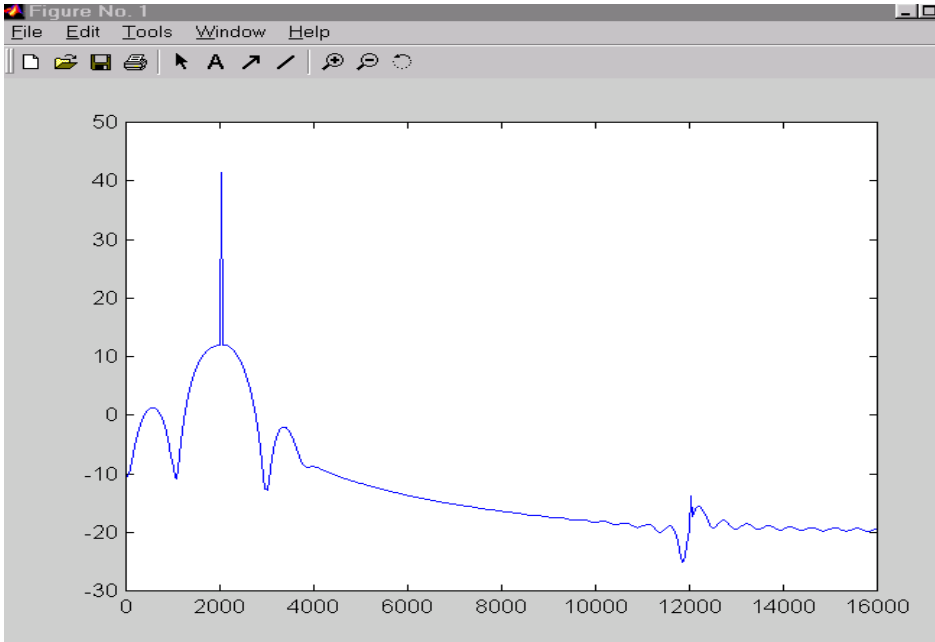


Figure 1: the frequency response of the filter in MATLAB

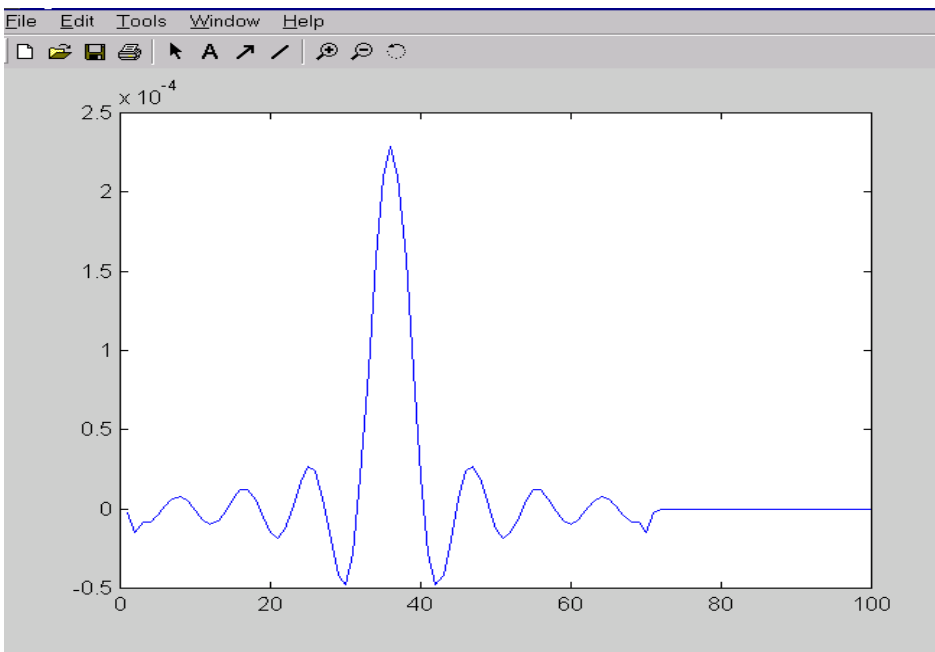


Figure 2: the impulse response of the filter in MATLAB

src.cxx

```
#include <fxp.h>
#include <artd_acu.h>

#define T_IW  Fix<16,15>
#define T_OW  Fix<16,15>
#define T_AW  Fix<32,31>
#define T_CW  Fix<15,16>

/*-----*/
/* Definition of the coefficient array */
#define NR_TAPS 71

const T_CW COEF[NR_TAPS] = {
    -0.0027 , -0.0149 , -0.0084 , -0.0083 , -0.0036 ,  0.0020 ,  0.0064 ,
    0.0076 ,  0.0047 , -0.0010 , -0.0069 , -0.0099 , -0.0079 , -0.0014 ,
    0.0067 ,  0.0125 ,  0.0124 ,  0.0056 , -0.0053 , -0.0151 , -0.0185 ,
    -0.0124 ,  0.0016 ,  0.0175 ,  0.0271 ,  0.0239 ,  0.0065 , -0.0193 ,
    -0.0420 , -0.0481 , -0.0277 ,  0.0205 ,  0.0879 ,  0.1575 ,  0.2097 ,
    0.2291 ,  0.2097 ,  0.1575 ,  0.0879 ,  0.0205 , -0.0277 , -0.0481 ,
    -0.0420 , -0.0193 ,  0.0065 ,  0.0239 ,  0.0271 ,  0.0175 ,  0.0016 ,
    -0.0124 , -0.0185 , -0.0151 , -0.0053 ,  0.0056 ,  0.0124 ,  0.0125 ,
    0.0067 , -0.0014 , -0.0079 , -0.0099 , -0.0069 , -0.0010 ,  0.0047 ,
    0.0076 ,  0.0064 ,  0.0020 , -0.0036 , -0.0083 , -0.0084 , -0.0149 ,
    -0.0027 };
// number of bits needed to address all taps, need one more for modulo
#define ADDRESS 8
#define T_AD  Int<ADDRESS>

int array_init(T_IW ARRAY[NR_TAPS-1])
{
    T_AD index=0;
    initloop: for (T_AD i=0; i<NR_TAPS-2; ++i)
    {
        ARRAY[index]=0;
        acuIncx: index = Artd_acuIncx(index);
    }
    return 0;
}

/*-----*/
/* Initialisation of the taps via a dummy function */

T_IW DELAY[NR_TAPS-1];
int dummy=array_init(DELAY);

void FIR( const T_IW IN, T_OW& OUT)
{
    #pragma OUT OUT

    /*-----*/
    /* Definition of the summation of the tap multipliers */
    static T_AD baseptr = 0;
    T_AD mod=NR_TAPS-1;
    T_AD index = Artd_acuPassx(baseptr);
    T_AW ACC = IN * COEF[0];

    loopi: for (T_AD i=1; i<NR_TAPS; ++i)
    {
        ACC += T_AW(DELAY[index] * COEF[i]);
        index = Artd_acuIncxMod(index,mod);
    }

    /*-----*/
    /* Definition of the output */
}
```

```

baseptr = Artd_acuDecxMod(baseptr,mod);
DELAY[baseptr] = IN;

OUT = ACC
}

```

architecture.pra (PRAGMA FILE)

```

instantiate("artd_library","inport","inport_1");
instantiate("artd_library","outport","outport_1");

instantiate("artd_library","alu","alu_1");
instantiate("artd_library","multp","multp_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","mbc_23","ctrl");

```

- Click on "Open" radio button. Traverse to the FIR1 folder and select this project. This is purely behavioral C-code and utilizes the A|RT Library fixed-point data-types. Compile the source by clicking on the Compile button. A C testbench is automatically generated.
- View the architecture definition pragma file as shown above by clicking on the Pragma button next to "Create Architecture." The resources are instantiated at a high level (MULT, RAM, ALU, etc.). The exact details (instructions and word-width) are derived later on. Build the architecture by clicking on the ">" button. Pop up the Architecture Report by clicking the "Arch" button.
- In order to map the algorithm on the above resources, click on the ">" button in Architecture Mapping.
- Now schedule the algorithm by clicking ">" next to Schedule Operations. It takes 215 cycles to complete the FIR filter algorithm. Click on "Sched" to view the instructions executed by the processor on each cycle. Pop up EXU Load view by clicking on Load and have a look at the activity on each resource on each cycle of the processor clocking. You can see from the report that 215 cycles are required for a 71-tap filter. This is excessive and FIR2 is a solution where 3 ACU's (Address Calculation Units) are instantiated in the design, all with their own separate ROMCTRL.

3.2 FIR2

FIR1 requires a 215 cycles, which is excessive for a 71-tap FIR filter. To improve the performance, the loop will be reduced in this step from 3 cycles to a single-cycle loop. This will be realized by instantiating 3 ACU's, which all have a separate ROMCTRL. To reduce the number of initialization cycles, the delay-line is no longer initialized to zero.

src.cxx

Labels are added and no initialization of the delay-line is done

```
#include <fxp.h>
#include <artd_acu.h>
#include <artd_alu.h>

#define T_IW Fix<16,15>
#define T_OW Fix<16,15>
#define T_AW Fix<32,31>
#define T_CW Fix<15,16>

/*-----*/
/* Definition of the coefficient array */
#define NR_TAPS 71

const T_CW COEF[NR_TAPS] ={
    -0.0027 , -0.0149 , -0.0084 , -0.0083 , -0.0036 , 0.0020 , 0.0064 ,
    0.0076 , 0.0047 , -0.0010 , -0.0069 , -0.0099 , -0.0079 , -0.0014 ,
    0.0067 , 0.0125 , 0.0124 , 0.0056 , -0.0053 , -0.0151 , -0.0185 ,
    -0.0124 , 0.0016 , 0.0175 , 0.0271 , 0.0239 , 0.0065 , -0.0193 ,
    -0.0420 , -0.0481 , -0.0277 , 0.0205 , 0.0879 , 0.1575 , 0.2097 ,
    0.2291 , 0.2097 , 0.1575 , 0.0879 , 0.0205 , -0.0277 , -0.0481 ,
    -0.0420 , -0.0193 , 0.0065 , 0.0239 , 0.0271 , 0.0175 , 0.0016 ,
    -0.0124 , -0.0185 , -0.0151 , -0.0053 , 0.0056 , 0.0124 , 0.0125 ,
    0.0067 , -0.0014 , -0.0079 , -0.0099 , -0.0069 , -0.0010 , 0.0047 ,
    0.0076 , 0.0064 , 0.0020 , -0.0036 , -0.0083 , -0.0084 , -0.0149 ,
    -0.0027 };

// number of bits needed to address all taps, need one more for modulo
#define ADDRESS 8
#define T_AD Int<ADDRESS>

int array_init(T_IW ARRAY[NR_TAPS-1])
{
    T_AD index=0;
    initloop: for (T_AD i=0; i<NR_TAPS-2; ++i)
    {
        ARRAY[index]=0;
        acuIncx: index = Artd_acuIncx(index);
    }
    return 0;
}

/*-----*/
/* Initialisation of the taps via a dummy function */
T_IW DELAY[NR_TAPS-1];
// int dummy=array_init(DELAY);

void FIR( const T_IW IN, T_OW& OUT)
/* Direct form FIR */
{
    #pragma OUT OUT

    /*-----*/
    /* Definition of the summation of the tap multipliers */
    static T_AD baseptr = 0;
    T_AD mod=NR_TAPS-1;
```

```

pass1: T_AD index = Artd_acuPassx(baseptr);
first_product: T_AW ACC = IN * COEF[0];

loopi: for (T_AD i=1; i<NR_TAPS; ++i)
{
    read1: T_IW M1 = DELAY[index];
    T_AW M = M1 * COEF[i];
    ACC += M;
    acuIncx1: index = Artd_acuIncxMod(index,mod);
}

/*-----*/
/* Definition of the output */
acuDecx1: baseptr = Artd_acuDecxMod(baseptr,mod);
DELAY[baseptr] = IN;

OUT = ACC;
}

```

architecture.pra

```

instantiate("artd_library","inport_noaddr","inport");
instantiate("artd_library","outport_noaddr","outport");

instantiate("artd_library","alu","alu_1");
instantiate("artd_library","multp","multp_1");

instantiate("artd_library","romctrl","romctrl_loop");
instantiate("artd_library","romctrl","romctrl_rom");
instantiate("artd_library","romctrl","romctrl_ram");

instantiate("artd_library","acu","acu_loop");
instantiate("artd_library","acu","acu_rom");
instantiate("artd_library","acu","acu_ram");

instantiate("artd_library","rom","rom_coeff");
instantiate("artd_library","ram","ram_delay");

instantiate("artd_library","mbc_23","ctrl");

```

Algmapping.pra

```

dedicate("acu_rom","rom_coeff");
dedicate("romctrl_rom","acu_rom");

assign_operation("/FIR/loopi/read1","acu_ram");
assign_operation("/FIR/acuDecx1","acu_ram");
assign_operation("/FIR/loopi/acuIncx1","acu_ram");
assign_operation("/FIR/pass1","acu_ram");

dedicate("romctrl_ram","acu_ram");

```

- Labels have been added to the source-code and no initialization of the delay-line has been done. Have a look at the C-code to see the addition of the labels and the change in the initiation of the loop.
- The architecture definition pragma file that the new units: 2 ACU's and 2 ROMCTRL's are added.

- The algorithm mapping pragmas are used to map operations on the extra ACU's that are used to dedicate separate ROMCTRL's to the different ACU's.
- The result is a reduced cycle count to 76. Better, not perfect.

3.3 FIR3

In this version a trade-off is done between the number of multiplexers and other design parameters (cycle-count and the controller size). By inspecting the "Mux" report of FIR2, we can see that one port at the dx input of the ALU can be saved. We have to modify the source in order to force the result of the first multiplication on the Y-register of the ALU, rather than on the X-register.

src.cxx

```

#include <fxp.h>
#include <artd_acu.h>
#include <artd_alu.h>

#define T_IW  Fix<16,15>
#define T_OW  Fix<16,15>
#define T_AW  Fix<32,31>
#define T_CW  Fix<15,16>

/*-----*/
/* Definition of the coefficient array          */
/*-----*/

#define NR_TAPS 71

const T_CW COEF[NR_TAPS] = {
    -0.0027 , -0.0149 , -0.0084 , -0.0083 , -0.0036 ,  0.0020 ,  0.0064 ,
    0.0076 ,  0.0047 , -0.0010 , -0.0069 , -0.0099 , -0.0079 , -0.0014 ,
    0.0067 ,  0.0125 ,  0.0124 ,  0.0056 , -0.0053 , -0.0151 , -0.0185 ,
    -0.0124 ,  0.0016 ,  0.0175 ,  0.0271 ,  0.0239 ,  0.0065 , -0.0193 ,
    -0.0420 , -0.0481 , -0.0277 ,  0.0205 ,  0.0879 ,  0.1575 ,  0.2097 ,
    0.2291 ,  0.2097 ,  0.1575 ,  0.0879 ,  0.0205 , -0.0277 , -0.0481 ,
    -0.0420 , -0.0193 ,  0.0065 ,  0.0239 ,  0.0271 ,  0.0175 ,  0.0016 ,
    -0.0124 , -0.0185 , -0.0151 , -0.0053 ,  0.0056 ,  0.0124 ,  0.0125 ,
    0.0067 , -0.0014 , -0.0079 , -0.0099 , -0.0069 , -0.0010 ,  0.0047 ,
    0.0076 ,  0.0064 ,  0.0020 , -0.0036 , -0.0083 , -0.0084 , -0.0149 ,
    -0.0027 };

// number of bits needed to address all taps, need one more for modulo
#define ADDRESS 8
#define T_AD  Int<ADDRESS>

int array_init(T_IW ARRAY[NR_TAPS])
{
    T_AD index=0;
    initloop: for (T_AD i=0; i<NR_TAPS-1; ++i)
    {
        ARRAY[index]=0;
        acuIncx: index = Artd_acuIncx(index);
    }
    return 0;
}

/*-----*/
/* Initialisation of the taps via a dummy function          */
/*-----*/

T_IW DELAY[NR_TAPS];
// int dummy=array_init(DELAY);

void FIR( const T_IW IN, T_OW& OUT)
/* Direct form FIR */
{
    #pragma OUT OUT

```

```

/*-----*/
/* Definition of the summation of the tap multipliers */

static T_AD baseptr = 0;
T_AD mod=NR_TAPS;
acuIncx1: T_AD index = Artd_acuIncxMod(baseptr,mod);
DELAY[baseptr] = IN;
T_AW M = DELAY[baseptr] * COEF[0];
T_AW ACC = Artd_aluPassy(M);
acuDecx1: baseptr = Artd_acuDecxMod(baseptr,mod);

loopi: for (T_AD i=1; i<NR_TAPS; ++i)
{
    read1: T_IW M1 = DELAY[index];
    T_AW M = M1 * COEF[i];
    ACC += M;
    acuIncx2: index = Artd_acuIncxMod(index,mod);
}

/*-----*/
/* Definition of the output */

OUT = ACC;
}

```

architecture.pra

Identical to FIR2

Algmapping.pra

Identical to FIR2

By eliminating unnecessary muxes, area can be optimized. In this version of the FIR filter, a trade-off has been done between the number of muxes needed and the cycle-count.

- Nothing has changed in the architecture definition and algorithm pragma files, meaning that the same architecture is used.
- Run all steps.

The number of muxes has been reduced from 79 to 38! The number of cycles and the number of micro-ROM rows are increased by 1, whereas the width of the micro-ROM is decreased thanks to the reduced number of muxes.

3.4 FIR4

To further speed up the design, a second multiplier is introduced to the datapath. The source-code of design2 is rewritten with 2 delay-lines to keep the 2 multipliers busy. As a consequence the ROM and RAM are split into two equal parts.

src.cxx

```
#include <fxp.h>
#include <artd_acu.h>
#include <artd_alu.h>
#define T_IW Fix<16,15>
#define T_OW Fix<16,15>
#define T_AW Fix<32,31>
#define T_CW Fix<15,16>

/*-----*/
/* Definition of the coefficient array */

const T_CW COEF0 = -0.0027 ;

const T_CW COEF1[35] = { -0.0149 , -0.0084 , -0.0083 , -0.0036 , 0.0020 , 0.0064 ,
0.0076 , 0.0047 , -0.0010 , -0.0069 , -0.0099 , -0.0079 , -0.0014 ,
0.0067 , 0.0125 , 0.0124 , 0.0056 , -0.0053 , -0.0151 , -0.0185 ,
-0.0124 , 0.0016 , 0.0175 , 0.0271 , 0.0239 , 0.0065 , -0.0193 ,
-0.0420 , -0.0481 , -0.0277 , 0.0205 , 0.0879 , 0.1575 , 0.2097 , 0.2291 };

const T_CW COEF2[35]= {0.2097 , 0.1575 , 0.0879 , 0.0205 , -0.0277 , -0.0481 ,
-0.0420 , -0.0193 , 0.0065 , 0.0239 , 0.0271 , 0.0175 , 0.0016 ,
-0.0124 , -0.0185 , -0.0151 , -0.0053 , 0.0056 , 0.0124 , 0.0125 ,
0.0067 , -0.0014 , -0.0079 , -0.0099 , -0.0069 , -0.0010 , 0.0047 ,
0.0076 , 0.0064 , 0.0020 , -0.0036 , -0.0083 , -0.0084 , -0.0149 , -0.0027 };

#define NR_TAPS 70
// number of bits needed to address all memories, need one more for modulo
#define ADDRESS 7
#define T_AD Int<ADDRESS>

int array_init(T_IW ARRAY[NR_TAPS/2])
{
    T_AD index=0;
    initloop: for (T_AD i=0; i<NR_TAPS/2-1; ++i)
    {
        ARRAY[index]=0;
        acuIncx: index = Artd_acuIncx(index);
    }
    return 0;
}

/*-----*/
/* Initialisation of the taps via a dummy function */

T_IW DELAY1[NR_TAPS/2], DELAY2[NR_TAPS/2];
// int dummy1 = array_init(DELAY1);
// int dummy2 = array_init(DELAY2);

void FIR( const T_IW IN, T_OW& OUT)
/* Direct form FIR */
{
    #pragma OUT OUT

    /*-----*/
    /* Definition of the summation of the tap multipliers */

    static T_AD baseptr = 0;
```

```

T_AD mod = NR_TAPS/2;
pass1: T_AD index = Artd_acuPassx(baseptr);
T_AW ACC1 = IN * COEF0;
T_AW ACC2 = 0;

loopi: for (T_AD i=0; i<NR_TAPS/2; ++i)
{
  read1: T_IW M1 = DELAY1[index];
  T_AW M1P = M1 * COEF1[i];
  alu1: ACC1 += M1P;
  read2: T_IW M2 = DELAY2[index];
  T_CW M2TMP = COEF2[i];
  mult2: T_AW M2P = M2 * M2TMP;
  alu2: ACC2 += M2P;
  acuIncx1: index = Artd_acuIncxMod(index,mod);
}

/*-----*/
/* Definition of the output */
*/

pass2: T_AW ACC2F = Artd_aluPassx(ACC2);
OUT = ACC1 + ACC2F;
acuDecx1: baseptr = Artd_acuDecxMod(baseptr,mod);
DELAY2[baseptr] = DELAY1[baseptr];
DELAY1[baseptr] = IN;
}

```

architecture.pra

```

instantiate("artd_library","inport_noaddr","inport");
instantiate("artd_library","outport_noaddr","outport");

instantiate("artd_library","alu","alu_1");
instantiate("artd_library","alu","alu_2");
instantiate("artd_library","mult","mult_1");
instantiate("artd_library","mult","mult_2");

instantiate("artd_library","romctrl","romctrl_loop");
instantiate("artd_library","romctrl","romctrl_rom");
instantiate("artd_library","romctrl","romctrl_ram");

instantiate("artd_library","acu","acu_loop");
instantiate("artd_library","acu","acu_rom");
instantiate("artd_library","acu","acu_ram");

instantiate("artd_library","rom","rom_coeff");
instantiate("artd_library","rom","rom2_coeff");
instantiate("artd_library","ram","ram_delay");
instantiate("artd_library","ram","ram2_delay");

instantiate("artd_library","mbc_23","ctrl");

```

algmapping.pra

```

//splitting coefficients and delay line
assign_variable("/COEF2","rom2_coeff");
assign_variable("/DELAY2","ram2_delay");

//using mult 2 and acu 2 in parallel with mult_1 and acu_1
assign_operation("/FIR/loopi/mult2","mult_2");
assign_operation("/FIR/loopi/alu2","alu_2");

//assigning RAM addressing
assign_operation("/FIR/loopi/read1","acu_ram");

```

```
assign_operation("/FIR/loopi/read2","acu_ram");
assign_operation("/FIR/acuDecx1","acu_ram");
assign_operation("/FIR/loopi/acuIncx1","acu_ram");
assign_operation("/FIR/pass1","acu_ram");
dedicate("romctrl_ram","acu_ram");
dedicate("acu_ram","ram_delay");
dedicate("acu_ram","ram2_delay");

//assigning ROM addressing
dedicate("acu_rom","rom_coeff");
dedicate("acu_rom","rom2_coeff");
dedicate("romctrl_rom","acu_rom");

assign_operation("/FIR/pass2","alu_2");
```

- Open the architecture creation pragma file and observe the changes: 2 multipliers and 2 memory (RAM, ROM) instances.
- Pop up the algorithm-mapping pragma file to highlight additional pragmas.
- Run all steps.

We now have an acceptable solution that only takes 42 cycles for a 71-tap filter, a 50 percent reduction in cycle count. The cost on the other hand is an increase of the total area: two multipliers, more registers, more multiplexers and a larger controller width.