



Delft University of Technology  
Faculty of Electrical Engineering,  
Mathematics and Computer Science  
Circuits & Systems Group

**Laboratory Exercise for**  
**ET4054**  
**Methods and Algorithms for System Design**

**Instruction Manual,**  
**Academic year 2006 - 2007**

**version *msclab2006***

Dr.ir. T.G.R van Leuken,  
Ing. H.J. Lincklaen Arriëns

## Introduction Laboratory Exercise ET4054

The laboratory exercise will illustrate and teach you to design hardware starting from a ‘high level’ description of a circuit. Details of the Assignments are given later on in this manual.

A short overview is given here:

We will start with translating a number of circuits in Data Flow Graphs, which consists of operations, delays and interconnect. Using MATLAB, you can schedule (using various scheduling methods, resource constraints, resource latencies, etc) the circuit descriptions.

Again, using MATLAB, it is possible to test a circuit whether ‘retiming for minimum clock states’ can result in ‘faster’ circuits.

For each circuit given, Pareto points can be obtained and plotted. You will have to select a ‘best’ circuit that you use for further investigation. The ‘best’ function in this context can be the smallest, fastest, or even the most simple implementation.

Still using MATLAB, you will have to simulate the scheduled circuit and calculate output values depending on user-definable input sequences. These values can be used later on to verify that your hardware operates correctly.

Another MATLAB program can generate corresponding bit-true VHDL code for the data path and the controller part of your circuit, together with a so-called testbench. This testbench and circuit code will have to be simulated with the aid of a VHDL simulator like *ModelSim*. Evidently, the outputs of both MATLAB and the VHDL testbench should be in accordance with each other.

The next (optional) step is to synthesize the design using *Synplify\_Pro*. This program will a.o. generate an EDIF netlist, a schematic diagram of your design and lots of information in its log-files..

## Requirements for the oral exam

For the exam you will have to write a short report, which at least covers the following content:

- prints of the Data Flow Graphs assigned to you, including operations, delays and interconnect (Also print the values of the input and calculated output variables),
- plots of Pareto curves of the Sequencing Graphs, including scheduling parameters,
- prints of feasible retimed circuits, with Pareto points and resulting Sequencing Graphs,
- a short motivation (textual) for the Scheduled Sequencing Graph selected by you for simulation,
- a print-out of the simulation results (wave viewer), and
- (optionally) a print-out of the schematic diagram of your design.

Please note, that if you could not completely finish the ET4054 exercise in the given time, this does **not** exclude you from participating in the oral exam! In case, for whatever reason, you did not complete the exercise, show the plots and printouts of those steps that you did accomplish.

## Location

The Laboratory room is located on the 16th floor, rooms H16.090 and H16.100. A login should be enough to access all functions and tools.

## Getting Started

You have to obtain a user login name and password first: you will be informed how at where during the lectures. After login and starting MATLAB for the first time, you will find a file in your home directory called **Assignment2006\_praks###** that lists your ‘personal assignment’.

You can work at home if you want: in that case you need some MATLAB files, MATLAB itself (Release 14 or higher), a VHDL simulator such as is e.g. available in the Xilinx WEBPACK MXE and optionally a synthesizer. Unfortunately, there exists no license-free or student version of the Synplify\_Pro synthesizer that can however be used in the laboratory room. Instruction can be found in the document *Install\_Guide\_for\_Home\_Users.pdf* on our home page.

Note that you have to login and start MATLAB at least once in the Laboratory room to obtain your ‘personal assignment’.

## Assistance

During lecture hours there will be assistance present in the Laboratory room.

## Some more useful information

The home page for this course is in <http://ens.ewi.tudelft.nl/Education/courses/et4054>

For VHDL: <http://www.vhdl-online.de/~vhdl/tutorial/>

For VHDL testbenches: <http://www.stefanvhdl.com/>

For ModelSim: <http://www.model.com>  
<http://www.fkt.cs.tut.fi/software/modeltech/tutorial/>

For Synplify: <http://www.synplicity.com>  
[http://www.actel.com/documents/synplify\\_ug.pdf](http://www.actel.com/documents/synplify_ug.pdf), Chapters 1 – 5

In general: [www.google.com](http://www.google.com)

We will maintain a FAQ list on the Circuits&Systems webserver:

<http://ens.ewi.tudelft.nl/Education/courses/et4054/FAQ.pdf>

## Contents of this manual

On the next pages, we will give an impression of the ‘design’ of a 5<sup>th</sup> order FIR filter, following more or less the same steps that you are also supposed to follow.

Then we introduce a 5<sup>th</sup> order Cauer (elliptical) filter, from which a set of 16 different Wave Digital structures are derived which will be the subject of your assignment, which are followed by the actual assignment.

In the last part of this manual, more detailed descriptions of the structures are given, viz. for each of them a data-flow block scheme and the accompanying cir-file.

More information about the scheduling programs can be found in the [\*Reference Guide\*](#) for the *Scheduling Toolbox for MATLAB, version msclab2006*, that can be found on our web page.

For detailed information on the subjects covered in this exercise, see

*Synthesis and Optimization of Digital Circuits*,  
Giovanni De Micheli,  
McGraw-Hill, Inc.  
ISBN 0-07-113271-6

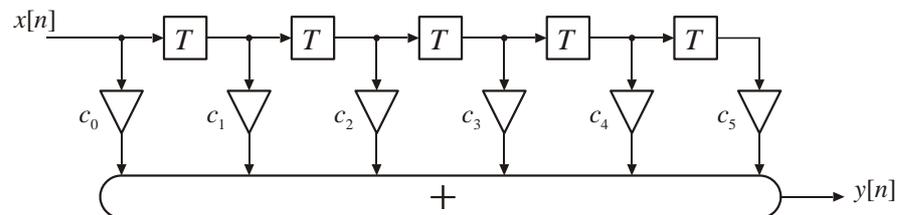
*VLSI Digital Signal Processing Systems*,  
Keshab K. Parhi,  
John Wiley & Sons, Inc.  
ISBN 0-471-24186-5

Delft, November 2006.

Dr.ir. T.G.R van Leuken,  
Ing. H.J. Lincklaen Arriëns

## 1. Detailed example of a 5<sup>th</sup> order FIR filter

In the following, we will give an impression of the design of a simple Finite Impulse Response (FIR) filter. The block diagram of a 5<sup>th</sup> order FIR filter is given in Figure 1.1. Such diagrams can be found in almost any Digital Signal Processing textbooks.



**Figure 1.1** Block diagram of a 5<sup>th</sup> order FIR filter.

The filter is made up of several delay elements that are clocked with the sample frequency. The output of each delay element is individually weighted with a coefficient  $c_i$ , and is combined with the others in a weighted sum. For each input sample  $x[n]$ , we obtain one output sample  $y[n]$ . This can be described mathematically with

$$y[n] = \sum_{i=0}^5 c_i \cdot x[n-i] \quad \dots (1.1)$$

For the coefficients  $c_0$  to  $c_5$  given in Table 1, this filter will show a low-pass transfer function in the frequency domain as shown in Figure 1.2a.

**Table 1.1** coefficients for the 5<sup>th</sup> order FIR filter

$c_0$	=	-0.07556556070608
$c_1$	=	0.09129209297815
$c_2$	=	0.47697917208036
$c_3$	=	0.47697917208036
$c_4$	=	0.09129209297815
$c_5$	=	-0.07556556070608

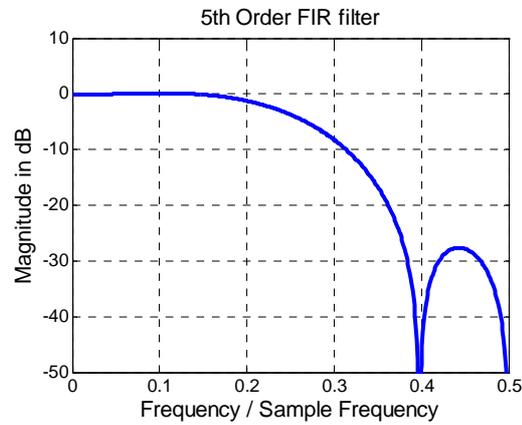
For our laboratory exercise, it is however easier to test the filter in the time domain. It is common practice to use the well-known *unit impulse* or *unit sample* function,  $\delta[n]$ , defined by

$$\delta[n] = \begin{cases} 1, & \text{for } n = 0 \\ 0, & \text{otherwise.} \end{cases} \quad \dots (1.2a)$$

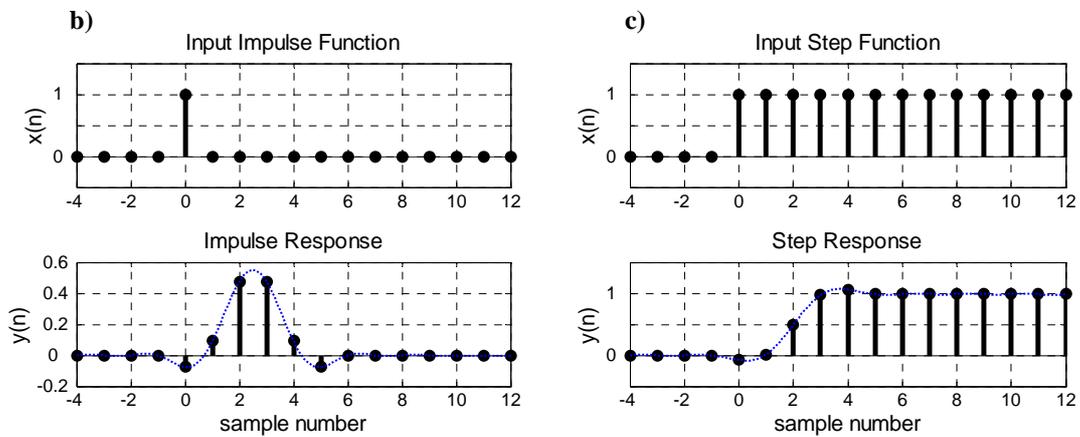
and the *unit step* function,  $u[n]$ , which is defined by

$$u[n] = \begin{cases} 0, & \text{for } n < 0 \\ 1, & \text{for } n \geq 0. \end{cases} \quad \dots (1.2b)$$

for this purpose.



**Figure 1.2a** Magnitude transfer function of the 5<sup>th</sup> order FIR filter.



**Figures 1.2 b and c.** Time response of the FIR filter with the magnitude transfer function given in Figure 2a. The lower plots show the output time samples  $y[n]$  for **b)** a unit impulse function at the input, and **c)** for a unit step function at the input.

When we calculate the output values of our filter for these input functions, e.g. by respectively substituting (1.2a) and (1.2a) in equation (1.1), we obtain Table 1.2.

These values will be used as references when testing our filter implementation. Figures 1.2b and 1.2c also show these signals in a graphical way.

**Table 1.2** *unit impulse response and unit step response for the 5<sup>th</sup> order FIR filter*

First 10 output samples, when at $n = 0$ a <i>unit impulse</i> function is applied to the input:	$n$	First 10 output samples, when at $n = 0$ a <i>unit step</i> function is applied to the input:
-0.07556556070608	0	-0.07556556070608
0.09129209297815	1	0.01572653227208
0.47697917208036	2	0.49270570435244
0.47697917208036	3	0.96968487643279
0.09129209297815	4	1.06097696941095
-0.07556556070608	5	0.98541140870487
0	6	0.98541140870487
0	7	0.98541140870487
0	8	0.98541140870487
0	9	0.98541140870487

Given the previous description of our filter, the following steps will be shown next.

1. we will describe the filter in terms of multiplications and additions/subtraction operations, apart from the delay elements, to obtain a sequencing graph that can be scheduled,
2. using this graph, we will use scheduling software to show the influence of a few different scheduling methods,
3. we will use a retiming algorithm to investigate whether changing the number and locations of the delay elements can result in a smaller number of clock-states, and check the resulting circuit description(s) with the scheduling software,
4. after selecting an appropriate circuit description and scheduling method, we will first use MATLAB to test the implementation which is derived from the scheduled graph (the SSG) in a fixed point environment,
5. we will then describe how to translate the scheduled circuit into a VHDL description,
6. we will simulate this VHDL description using the Modelsim timing simulator, and finally,
7. we will show the synthesis results when the VHDL description has been translated by the Synplify Pro program from Synplicity Inc.

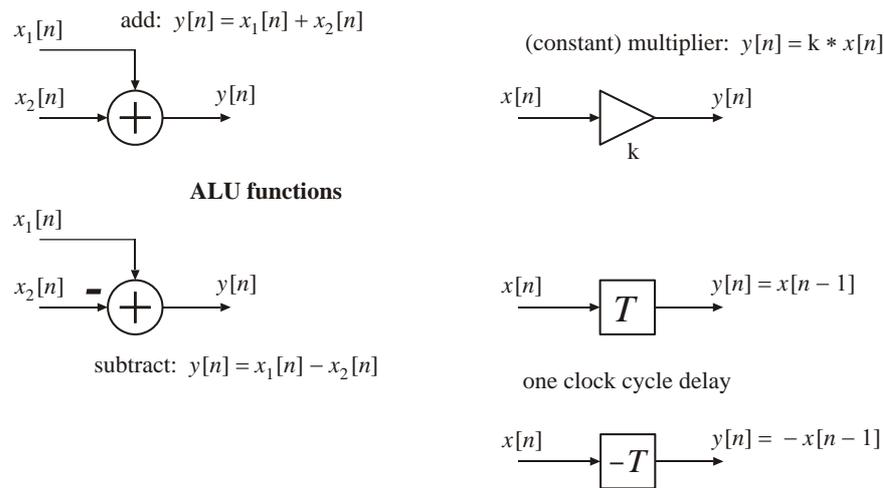
### 1.1 Description of the filter in a .cir file.

We first have to describe our FIR filter in a way that can be understood by the scheduling software. The scheduling software translates the operations and calculations so that they can be implemented on real multipliers and adders/subtractors.

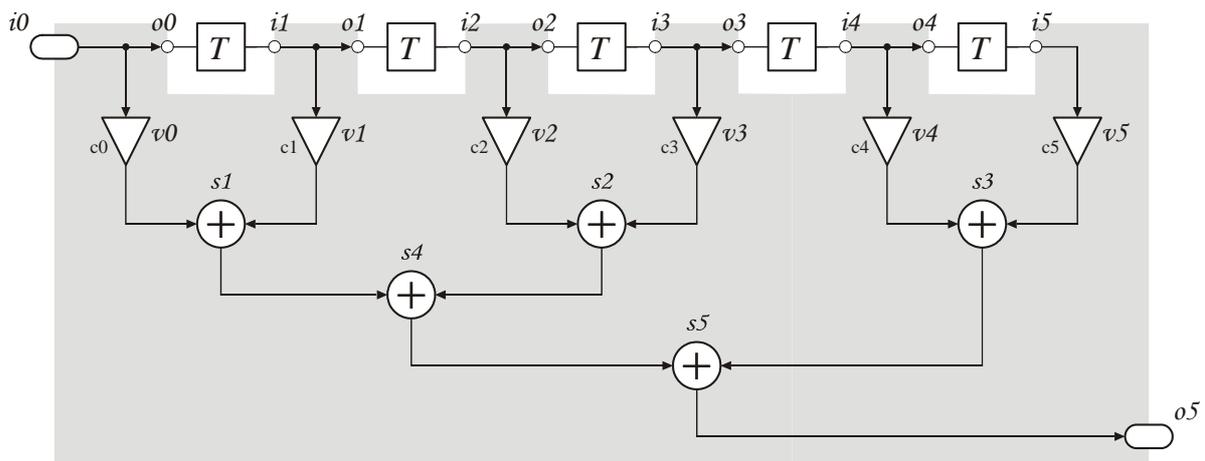
Our scheduling software demands a simple ASCII text file, for which we have defined a number of rules. The name of the text file should have the extension ‘.cir’.

For an extensive description of the syntax, structure and the thoughts behind a .cir-file, see the Reference Guide for the Scheduling Toolbox for Matlab (version msclab2006), to be found on <http://ens.ewi.tudelft.nl/Education/courses/et4054/>.

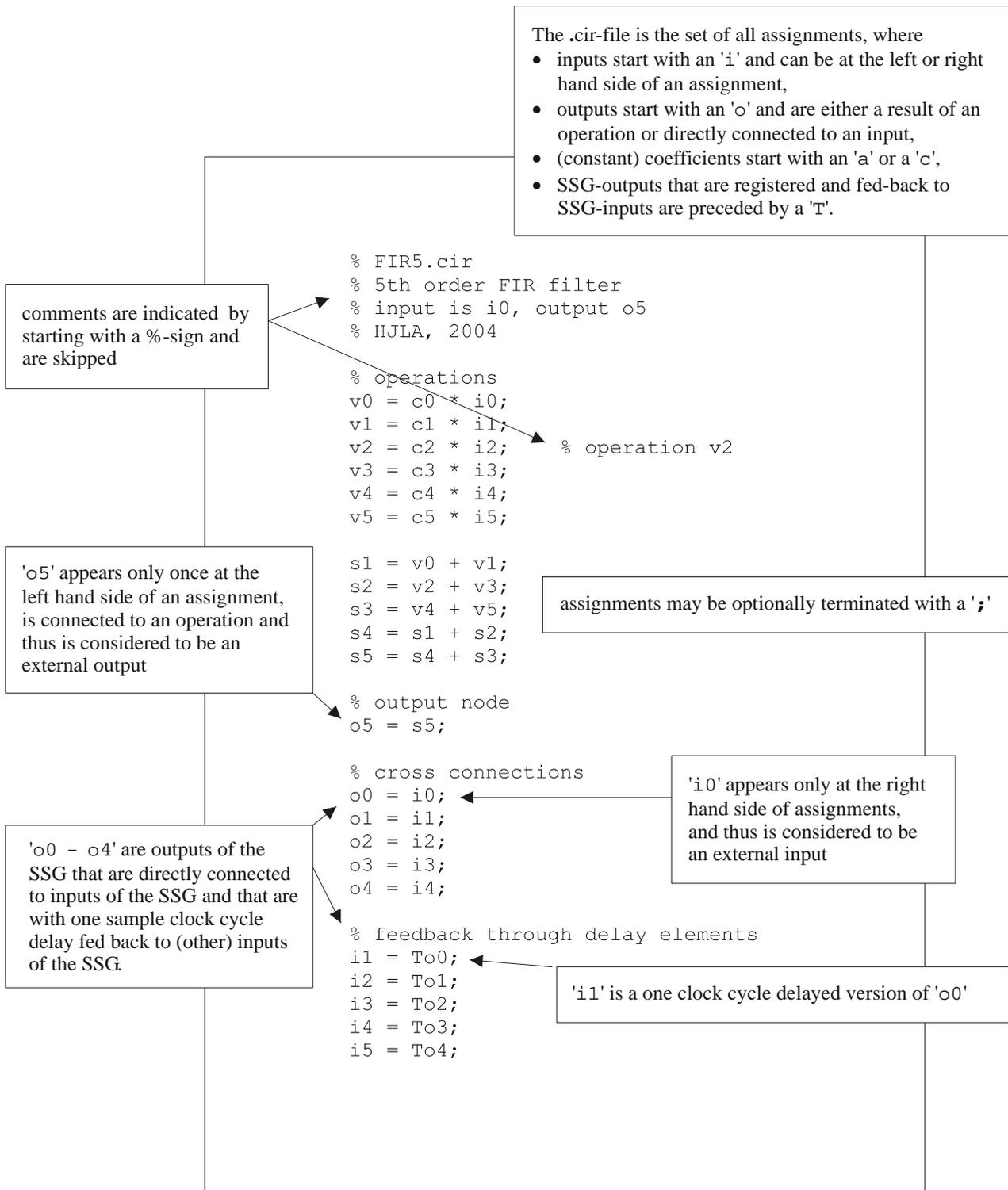
First, we have to label the circuit in a way so that we can write all necessary information in the .cir-file. Therefore, we have to separate the operations that can be scheduled from those that cannot be scheduled. In our case, we can schedule only multiplications, additions and subtractions. These resources have to be combined in one block, while we will use input ports and output ports for connections to delay elements. Figure 1.4 can help in recognizing the parts to be scheduled (the shaded area) and the input and output ports, while Figure 1.3 shows the meaning of the several functional block-elements as they are often used in signal processing literature. Now ‘i0’ is the overall input port of the filter ( $x[n]$ ) and ‘o5’ the output port ( $y[n]$ ). Using the correct syntax and assignment rules, may result in the cir-file shown in Figure 1.5. This .cir-file, let’s call it ‘FIR5.cir’, will be the input for the scheduling software.



**Figure 1.3** Explanation of the available basis elements.



**Figure 1.4** Example of the labeling of the operations, inputs and outputs for the FIR5 filter.



**Figure 1.5** Example of a .cir-file description for the 5<sup>th</sup> order FIR filter.

## 1.2 Description of the scheduling software.

This software consists of a set of MATLAB programs that can be used as separate functions, to be called from the MATLAB command window. All functions are provided with help comments, while a complete description of all function can be found in the Reference Guide ([msclab2006\\_RefGuide.pdf](#)) that can be downloaded from our web page.

There is also a graphical user interface, `schedGUI`, which combines a number of the commonly used functions and eases the comparison of the scheduling methods.

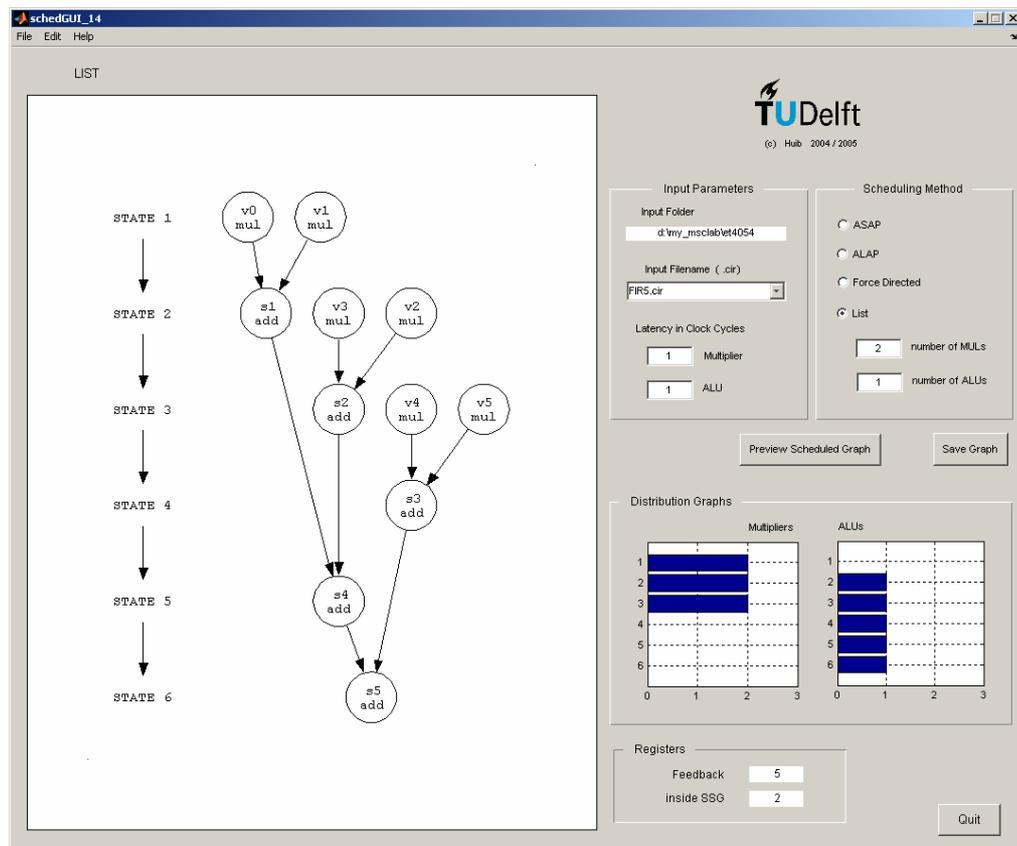
This GUI can be started from the directory in which we have saved our `.cir`-file. Let's denote this directory with  $\langle \$OUR\_CIR\_DIR \rangle$ .

Figure 1.6 shows a screen shot of this `schedGUI`, with the result when our FIR5 filter should be scheduled according to the 'List' method given the resource constraint that only 2 MULTIpliers and only 1 ALU (which can perform both the add and the subtract calculations) are available.

Both MULTIplier and ALU are supposed to have a latency of only one system clock cycle.

It can be seen that the result of the calculations will be valid after 6 system clock cycles, while also the distribution of the resources over the clock cycles is shown.

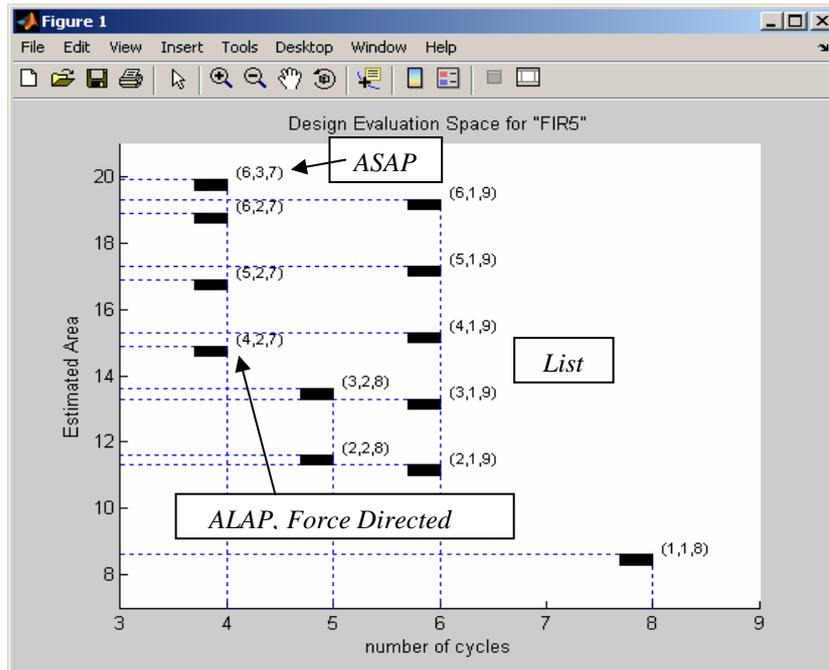
Also can be seen that the GUI expects that 2 additional REGisters will be needed to store intermediate results. It is important to note that `schedGUI` expects that the VHDL descriptions of our MULTIplier and ALU components already provide for a register immediately following the MULTIplier and the ALU, so that we use registered MULs and ALUs that are clocked at the end of the clock cycle in which the actual computation is done, and that can store this result until it needs to be clocked again for storing another result.



**Figure 1.6** Screenshot of `schedGUI` showing the Scheduled Sequencing Graph (SSG) for our FIR5 filter when a resource constrained List scheduling method has been applied.

### 1.3 Design exploration.

With the `xplore`-function, it is possible to get information about the design evaluation space for a specific `.cir` description. In a plot, area versus minimal delay trade-off points will be shown with the area of the different resource types (i.e. MULs, ALUs and REGs) as parameter. Combining the Pareto points for different realizations can lead to an optimized design from a given perspective.



**Figure 1.7** Design space exploration for the FIR5 filter. It appears that –for this particular (, small) circuit– the schedules resulting from the ALAP and the Force Directed methods are exactly the same.

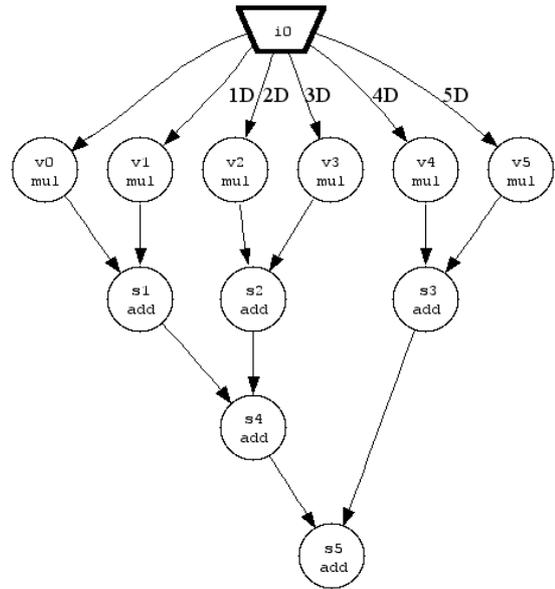
Figure 1.7 shows the results of executing `xplore('FIR5.cir',[ 2; 1; 0.7 ])`. For the correct syntax, again, see the Reference Guide or use the `help xplore` –command. In this example, the parameter `[ 2; 1; 0.7 ]` is used to define the relative sizes of a Multiplier, an ALU and a simple Register. The values chosen here are strictly arbitrary with the only intention to obtain a nice picture. In reality, a synthesized multiplier will take much more area than an ALU. On the other hand, if on e.g. an FPGA dedicated multipliers are available that are not needed by other parts of a design, the cost factor of a multiplier can be regarded to be zero.

The triplets  $(x,y,z)$  next to the black rectangles in Figure 1.7 represent the number of resources for that particular scheduling choice, viz. respectively the number of Multipliers, the number of ALUs and the number of Registers (this value includes the feedback registers, the registers needed inside the SSG for lifetime extension of intermediate computational results, and registers after each external input and before each external output) needed.

Note: the List scheduling method is sensitive for the sequence in which the assignments appear in the `.cir` file.

### 1.4 Retiming.

**Figure 1.8** Data-flow graph obtained with `view_DFG('FIR5.cir')`



Given the data-flow graph of our FIR5 filter in Figure 1.8, is it possible –without affecting the input/output characteristics of the circuit– to reduce the minimum number of cycles needed to obtain output results?

To answer this question, the MATLAB function `retime_minCycles(filename)` is available.

If a retiming for cycle minimization turns out to be feasible, actions to be taken are written to the command window. Next to that a new .cir-file will be written in the working directory that reflects the retimed circuit. This .cir-file, the name of which is mentioned on screen, can be the source for further explorations.

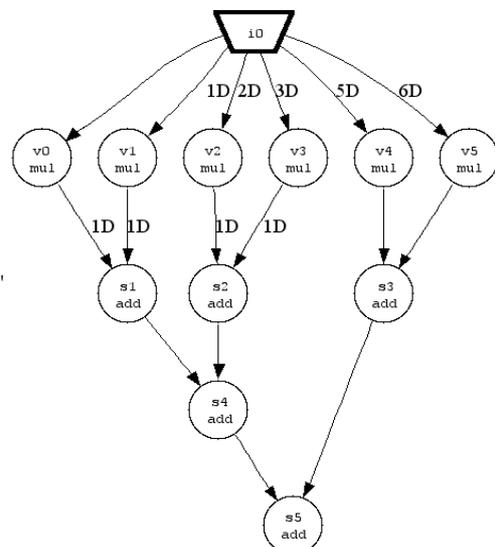
The process stops when no further reductions can be realized.

Shown below are the first results when `retime_minCycles('FIR5.cir')` will be executed.

```
Retiming possible to decrease longest path to 3 state(s)
NOTE: Output will be delayed 1 additional 'sample'cycle(s)
Actions needed:
  Change 1T delay from i3 to v4 into a 2T delay.
  Change connection from v0 to s1 into a 1T delay.
  Change connection from v1 to s1 into a 1T delay.
  Change connection from v2 to s2 into a 1T delay.
  Change connection from v3 to s2 into a 1T delay.
```

Created file 'FIR5\_3cyc.cir' for this retimed circuit ...

**Figure 1.9a** Data-flow graph of 'FIR5\_3cyc.cir'

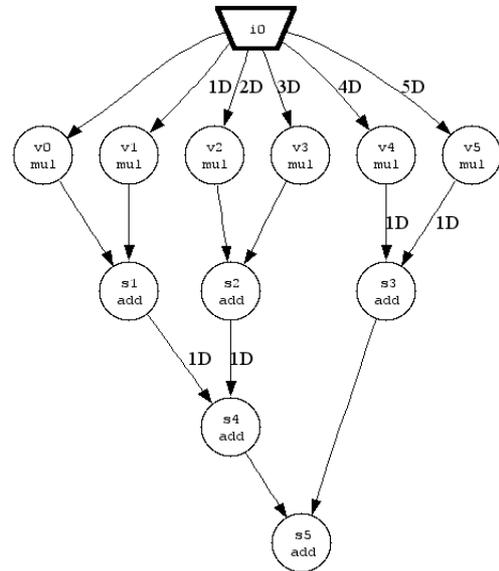


Retiming possible to decrease longest path to 2 state(s)

Actions needed:

- Change connection from v4 to s3 into a 1T delay.
- Change connection from v5 to s3 into a 1T delay.
- Change connection from s1 to s4 into a 1T delay.
- Change connection from s2 to s4 into a 1T delay.

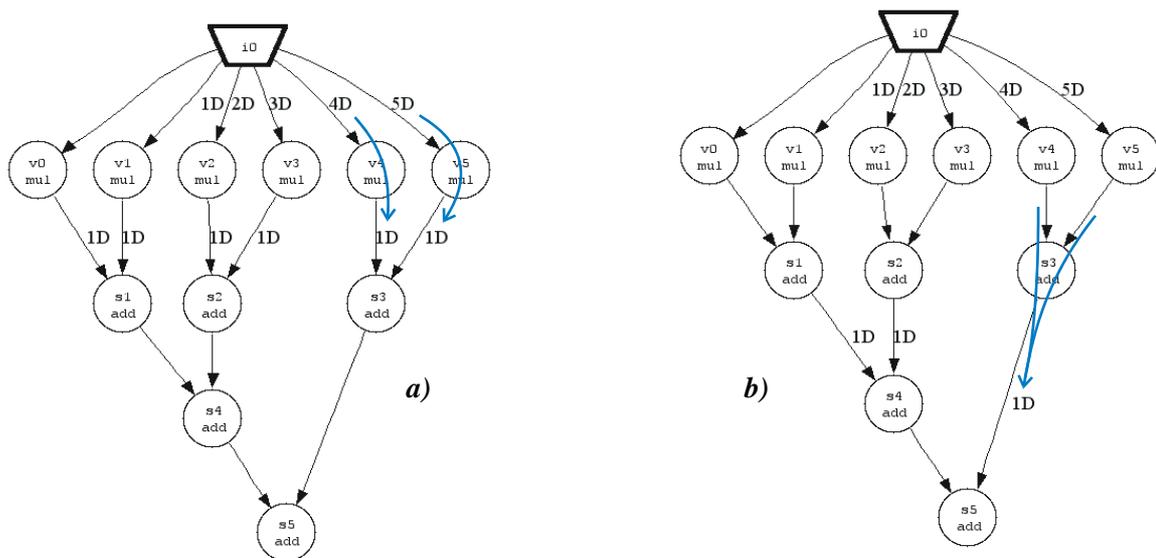
Created file 'FIR5\_2cyc.cir' for this retimed circuit ...



**Figure 1.9b** Data-flow graph of 'FIR5\_2cyc.cir'

Retiming possible to decrease longest pat  
 --- aborted here ---

Note that the underlying retiming algorithm stops when it finds a solution that satisfies its goal, viz. a reduction in cycles. This doesn't necessarily mean that the registers are inserted in the most efficient way. For the FIR5 circuit here (adders and multiplications having the same latency), it turns out to be possible to obtain solutions that 'looks nicer' (Figure 1.10a) or that use less registers (Figure 1.10b).



**Figure 1.10** a) Functionally equivalent realization of Figure 1.9a after relocating 2 delay elements, and b) equivalent realization of Figure 1.9b after combining 2 delay elements.

## 1.5 A MATLAB testbench.

The purpose of this testbench is to provide reference data that can be used to verify the VHDL output with. Testing and working with MATLAB is much easier and faster, and far less error prone, than starting with VHDL immediately, unless –maybe– you are a very experienced VHDL programmer.

A very important issue to note is that the VHDL definitions of the MUL, ALU and REG entities are working with fixed-point data and fixed-point arithmetic, in contrary with the floating-point computations that we have used so far.

It is a bit too complex and area consuming to work with floating-point arithmetic in this application, and also –given the limited dynamic range– not a requirement. In practice, most of the commercial DSP applications work with some kind of fixed-point format anyhow.

The MATLAB testbench adheres to this fixed-point format.

Suppose that we are satisfied with the result of a particular scheduling method, and that we want to implement that circuit. Since we will be working with fixed-point hardware for our computations, we will have to establish first with how many bit lines we will have to represent our coefficients and input and output values in order to meet our accuracy constraints. For this purpose the MATLAB function `gen_mTB` can be used, which results in a MATLAB testbench that simulates the final VHDL implementation using the same fixed-point arithmetic.

Here, no accuracy constraints are given, and we opt for a representation with 15 fractional bits. See the Reference Guide for a short description concerning signed fixed-point arithmetic and our notation of bus width, number of fraction bits, etc.

Suppose we want to see the filter’s behavior when a step function is applied at the input.

From Figure 1.2c and Table 1.2, it can be seen that the step response at the output exceeds the value of +1.0 (for sample number 4). This implicates that we need at least 2 bits to represent the signed integer part of this value, so we will use a [17 15] signed fixed-point notation here for our coefficients and in/outputs.

The MATLAB testbench (also the VHDL testbench that will be used later on) expects to read its input values from a .INP-file, a description of which is given in Appendix C. In this .INP-file, the constant coefficients should be defined first, followed by the input samples, line by line.

The utility `gen_INP` is intended to automate the creation of such an .INP-file.

Start however with running from `<$OUR_CIR_DIR>`

```
>> gen_mTB( 'FIR5.cir', 1, 'list', 1,1, 2,1 );
```

since this will notify you in which order the coefficients should be listed in the .INP-file.

`gen_mTB` creates a sub-directory `FIR5/matlab` under `<$OUR_CIR_DIR>` in which it will write its testbench files. These testbench files consist of (see the Reference Guide)

```
testbench_FIR5_auto.m, FIR5_SSG_auto.m, MUL.m and ALU.m.
```

These should be completed with the .INP-file mentioned before, e.g. by issuing

```
>> gen_INP( 'FIR5/matlab/FIR5.INP', [17 15], coeffs, ones(1,10) );
```

where `coeffs` is a cell-array containing the names and the values of the coefficients  $c_0 \dots c_5$  as were given in Table 1.1.

For this example, the coefficient data has been stored in a file `coeffs_FIR5.mat`. Use

```
>> load coeffs_FIR5.mat
```

first, to retrieve the `coeffs` cell-array into MATLAB’s workspace.

Running, from the sub-directory `<$OUR_CIR_DIR>/FIR5/matlab` (alternatively, add it to MATLAB's path definition)

```
>> testbench_FIR5_auto( [17 15 0], 1, 0 );
```

will perform the simulation and write the results to the screen as well as to a file `FIR5.OUT`.

If the last '0' parameter is changed in a '1', full debug information will be presented.

In case of doubt which syntax to use, see `testbench_FIR5_auto.m`'s own help function.

`FIR5.OUT` is a file in a specific hex-format, which can be translated into decimal format using the function `read_OUT.m`.

Note that the fixed-point parameter format that `testbench_FIR5_auto` works with, needs a third parameter which defines a number of additional bits with which the integer part of the fixed-point computations inside the SSG-part need to be expanded. It will be surely possible that intermediate results that occur inside the computational process will be greater than, or more negative than, the values that can be represented with the number of bits needed for input and output values.

In this example, no additional bits are needed, so the third value can be a zero.

It is obvious that we can only proceed when this testbench performs without any errors, and results in the expected output values (regarding the effects of the quantization errors caused by the limited number of bits).

Table 1.3 lists the values obtained with `read_OUT`, for the input sequence `[ 0 0 1 1 1 ... 1 1 1 ]`.

**Table 1.3** result of `read_OUT` for  $u[n-2]$

1:	x"00000" =	0.000000000000000
2:	x"00000" =	0.000000000000000
3:	x"00000" =	0.000000000000000
4:	x"1F654" =	-0.07556152343750
5:	x"00203" =	0.01571655273438
6:	x"03F11" =	0.49270629882813
7:	x"07C1F" =	0.96969604492188
8:	x"087CE" =	1.06097412109375
9:	x"07E22" =	0.98541259765625
10:	x"07E22" =	0.98541259765625
11:	x"07E22" =	0.98541259765625
12:	x"07E22" =	0.98541259765625
13:	x"07E22" =	0.98541259765625

It can be checked that the maximum absolute difference between these output values and the reference values is about  $1.12 \cdot 10^{-5}$ .

## 1.6 The VHDL testbench.

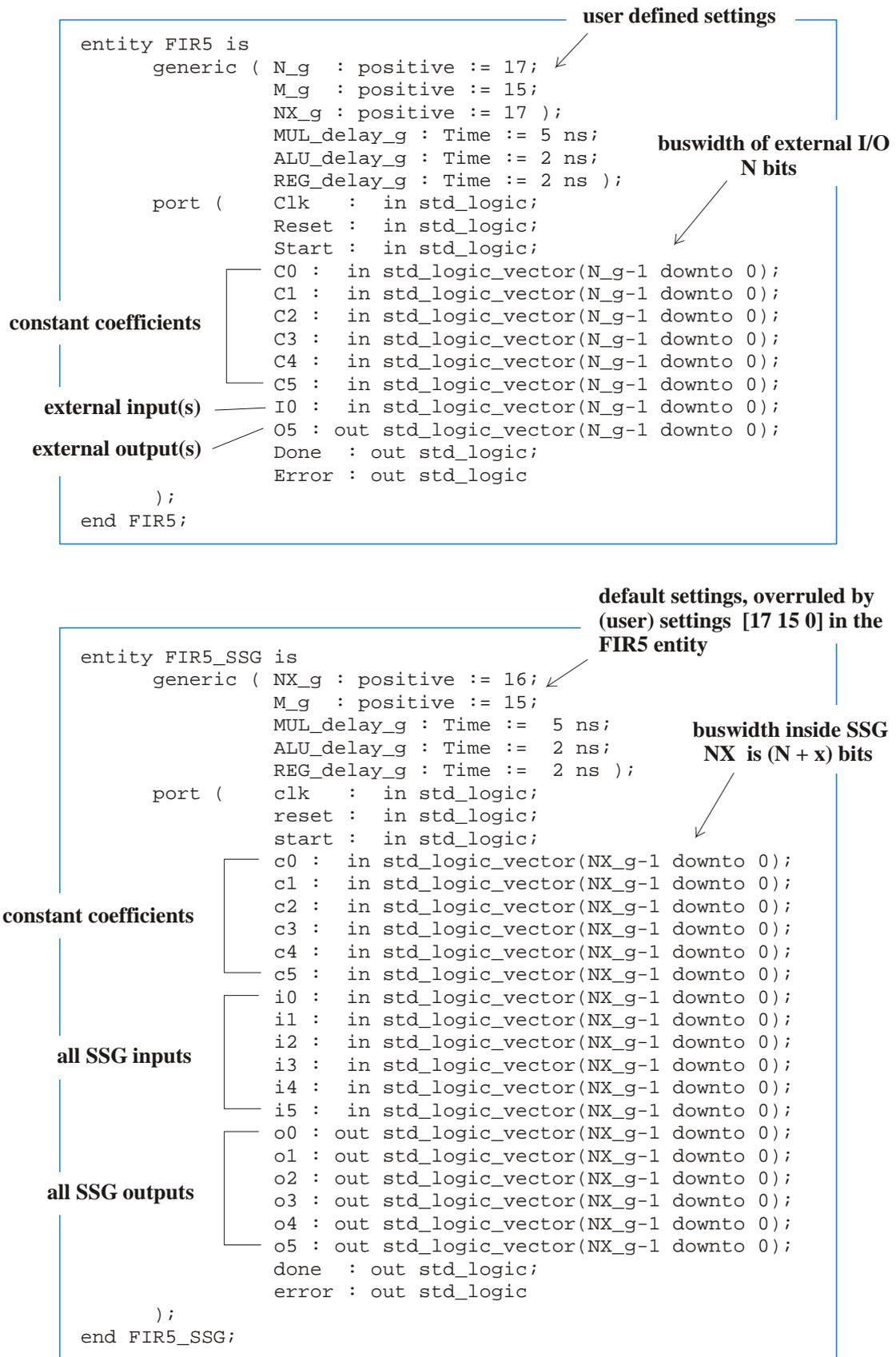
Now, we can use (from the `<$OUR_CIR_DIR>`-directory again),

```
>> gen_VHD( 'FIR5.cir', [17 15 0], 1, 'list', 1,1, 2,1 );
```

to generate the vhd design files and the vhd testbench files, respectively named

`resources_reg.vhd`, `FIR5_SSG_auto.vhd`, `FIR5_auto.vhd` and  
`testbench_FIR5_auto.vhd` and `txt_util2.vhd`.

These files will have been written in the sub-directory `<$OUR_CIR_DIR>/FIR5/vhdl/`.



**Figure 1.11** Listing of the entities FIR\_5 and FIR5\_SSG.

In here `resources_reg.vhd` contains the definitions of

- the registered multiplier with synchronous reset and clock enable (`MUL_R`),
- the registered ALU with synchronous reset and clock enable (`ALU_R`) that use the signal `OPCODE` to discriminate whether the input values have to be added or subtracted (`OPCODE = OP_ADD | OP_SUB`). There is also overflow, an additional output signal that goes high in case of an overflow error. And,
- the entity `REG_R`, which is a register with clock-enable and synchronous reset.

For simulation purposes, these entities show generic delays that are overruled when latencies other than 1 are assigned when calling `gen_VHD`.

The file `txt_util2.vhd` contains a.o. the code needed to perform file access and to read and write hexadecimal values.

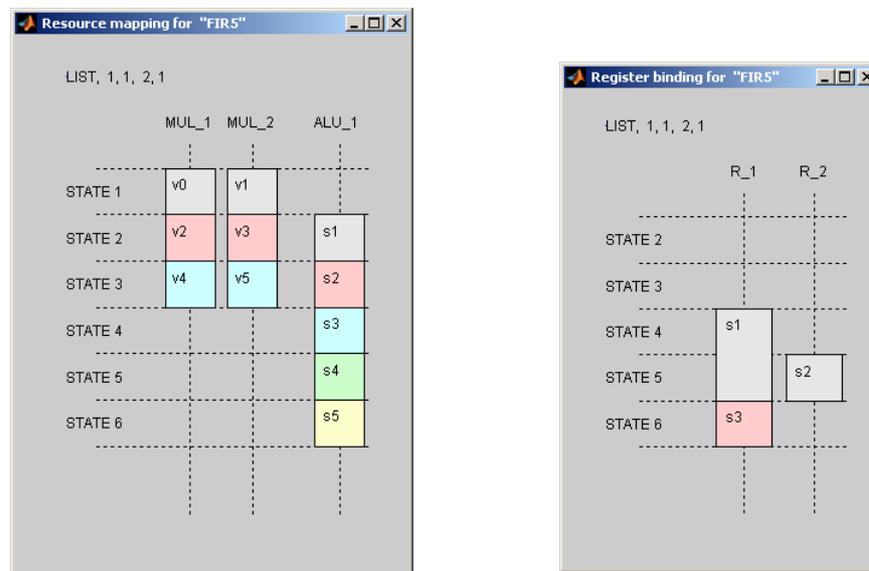
Finally, `FIR5_SSG_auto.vhd`, `FIR5_auto.vhd` and `testbench_FIR5_auto.vhd` are the descriptions of the circuit and its test environment in increasing hierarchy level.

Figure 1.11 lists the entity definitions of `FIR5_SSG` and `FIR5`. `FIR5_SSG` is instantiated as a component inside `FIR5`'s architecture.

Notice that the feedback registers in `FIR5_auto.vhd` are inferred automatically (instead of by inserting `REG` components) by the way in which the VHDL code is written.

All files mentioned, together with the `.INP`-file, are needed for the simulation.

During its run, `gen_VHD(_template)` generates a couple of plots, in which it shows the mapping of the operations on the two `MULs` and single `ALU`, and where exactly it needs the two additional registers (see Figure 1.12).



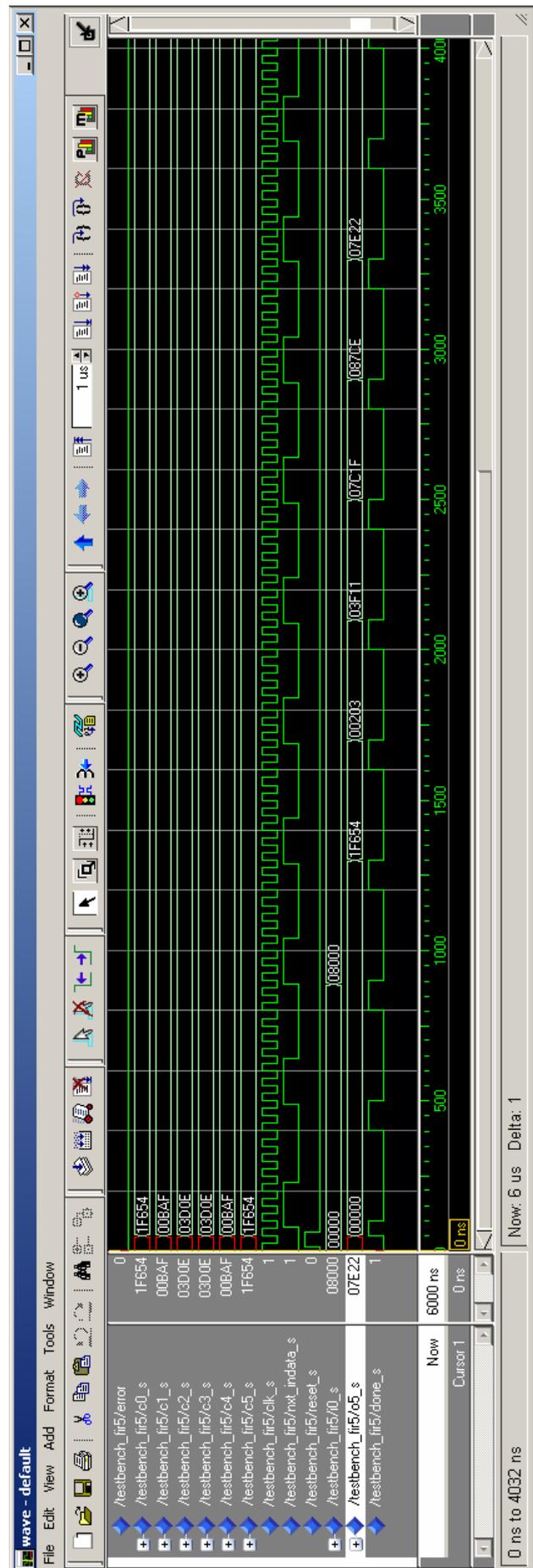
**Figure 1.12** The Resource mapping and Registry binding as reported by `gen_VHD`.

## 1.7 Simulation results.

Figure 1.13 shows the Wave window that can be obtained by running the VHDL files through the ModelSim simulator. All in- and output signals of the FIR5 block are shown here.

See the [Reference Guide](#) for an explanation of the stimuli signals and time steps that are used for the simulation.

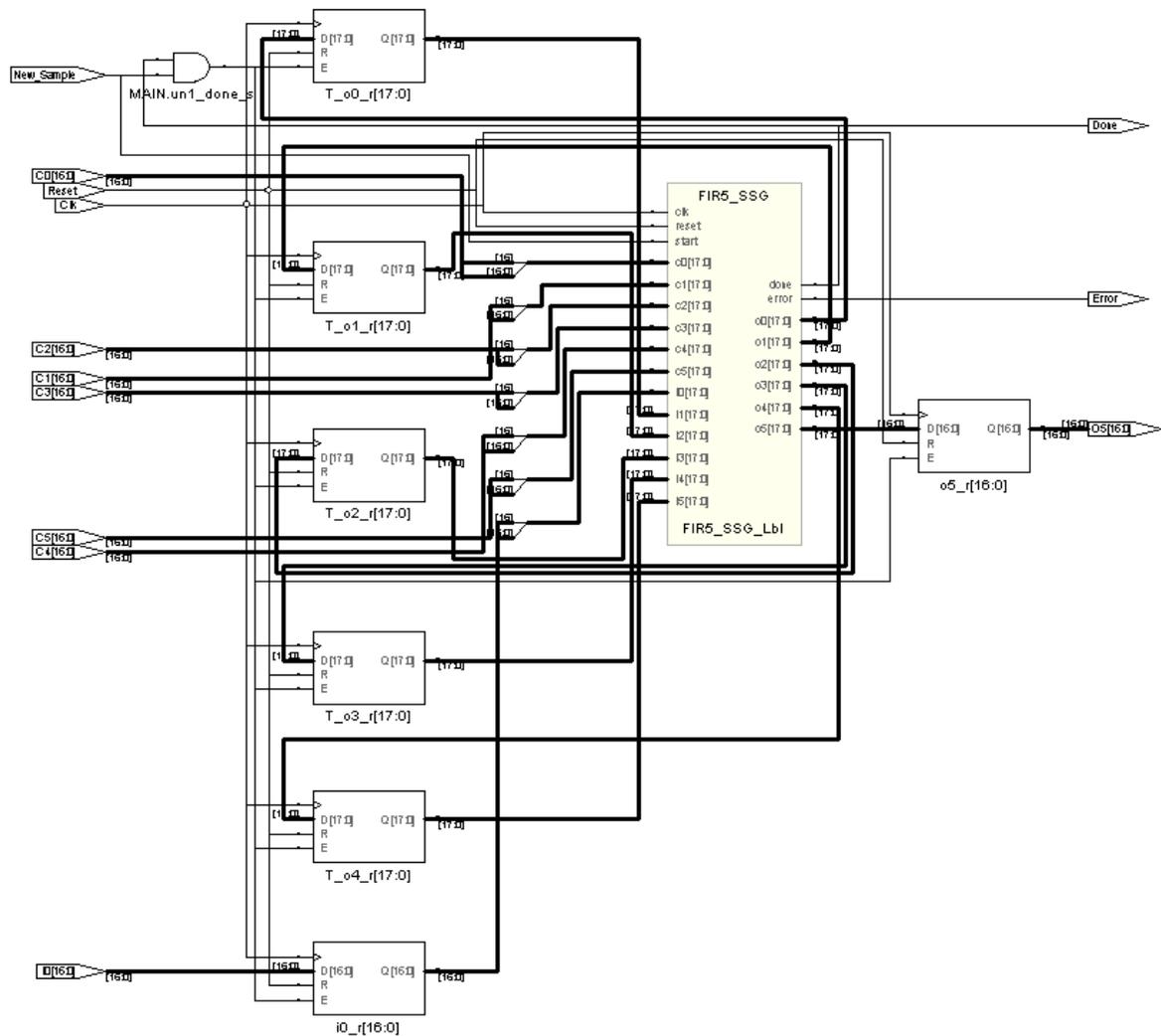
Comparison of the FIR5.OUT-files written by the MATLAB and the VHDL test-benches reveals that they are completely identical (except for the comment lines), from which can be concluded that the MATLAB and VHDL simulations show a bit true (and at the STATES level also a cycle true) correspondence.



**Figure 1.13** Waveform output of the FIR5 vhdl design after simulation with ModelSim.

## 1.8 Synthesis results.

In Figure 1.14, the result of a synthesis run for subsequently lower architectural levels is given. As target device for the implementation was chosen a Xilinx FPGA from the Spartan family, the XC3S2000. The devices from the Spartan family are equipped with dedicated, ready-to-use 18 by 18 bit multipliers; the XC3S2000 even offers 40 of them. In fact, these multipliers can also be configured as a multiplier with a clocked, clock enabled, resettable registered output, exactly as had been described before. The VHDL definition of MUL\_R in `resources_reg.vhd` is such, that these registered versions will be selected by the synthesizer.



**Figure 1.14a** *RTL view of the synthesized FIR5 top level as drawn by Synplify\_Pro.*

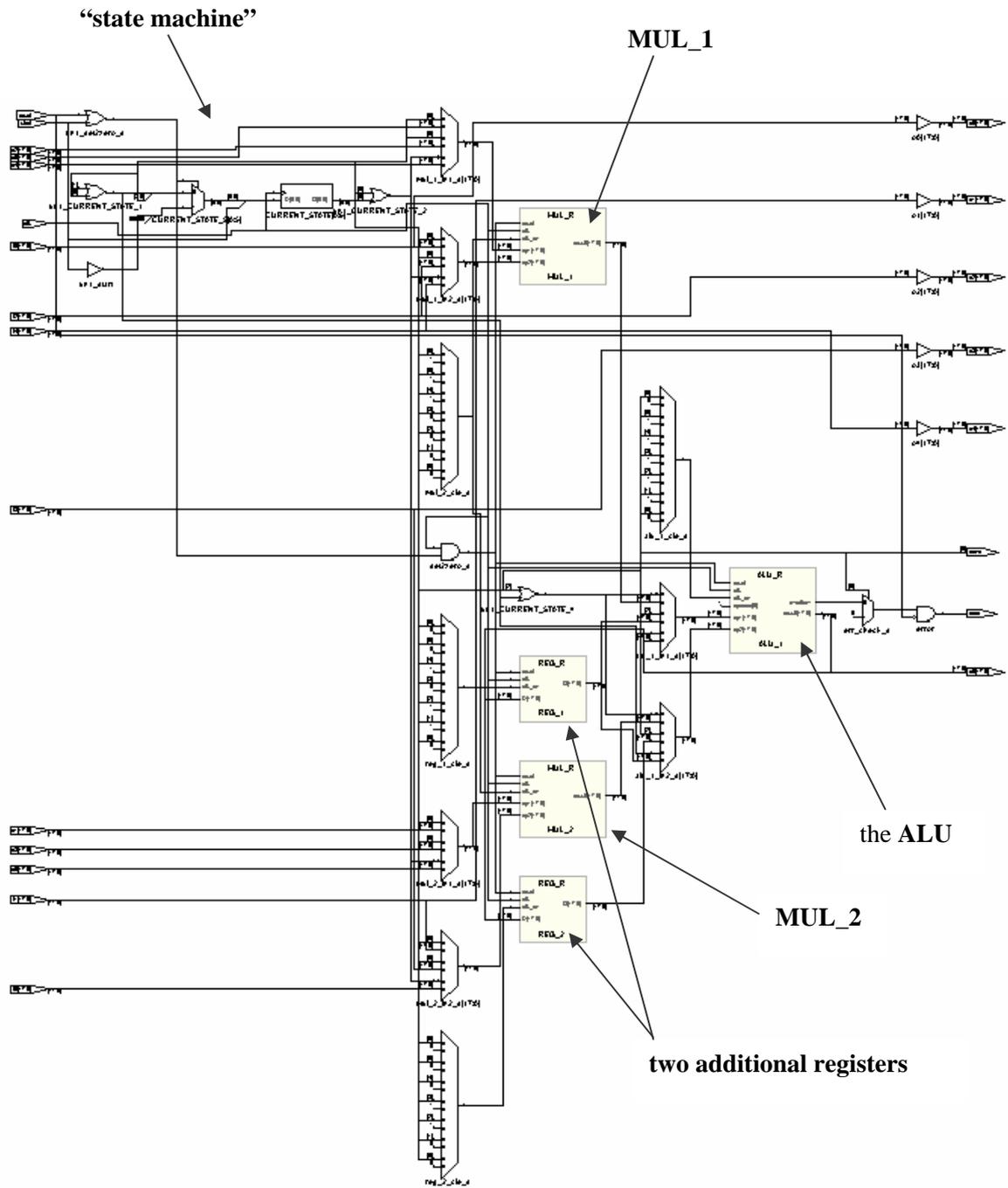


Figure 1.14b RTL view inside the FIR5\_SSG-block.

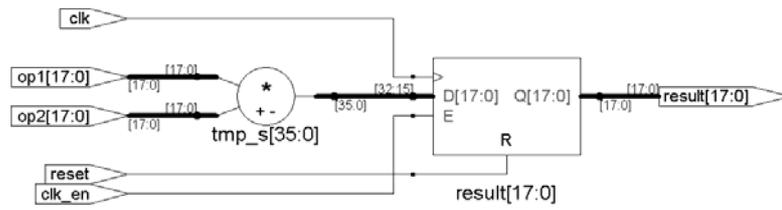


Figure 1.14c RTL view of MUL\_R.

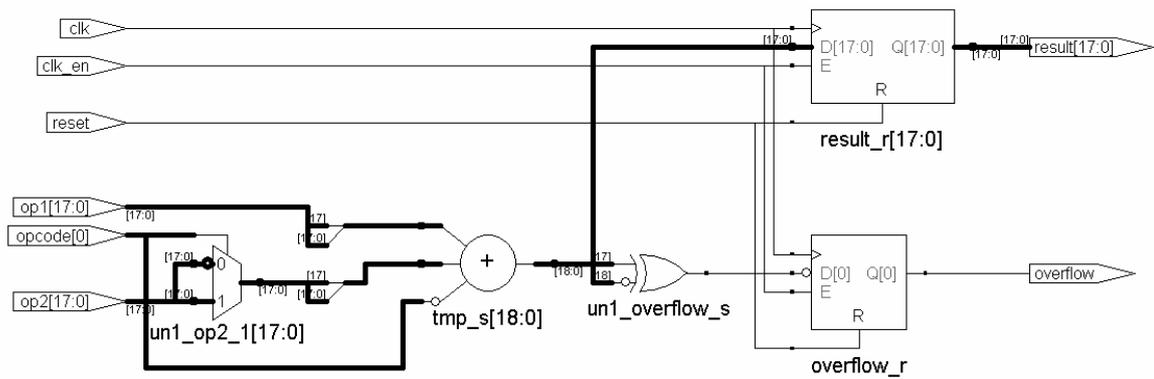


Figure 1.14d RTL view of ALU\_R.

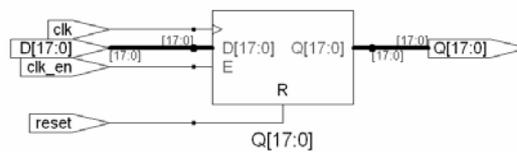
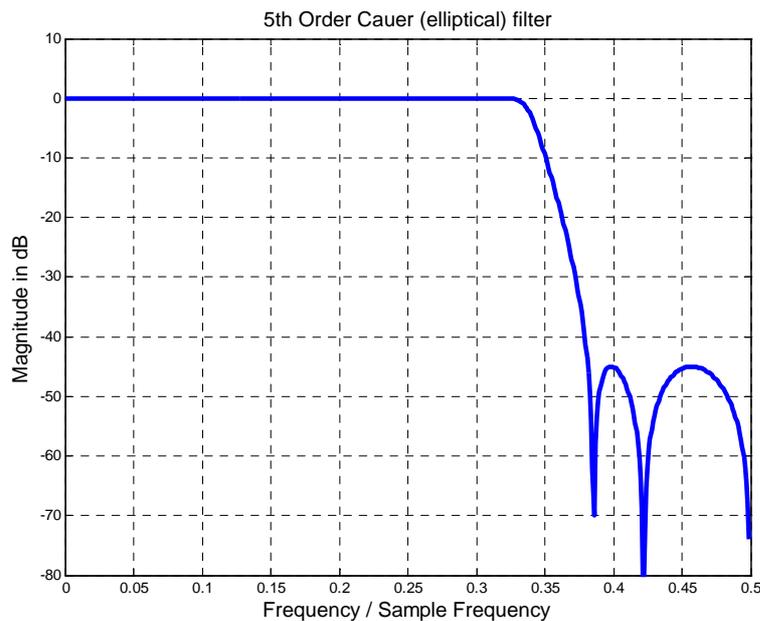


Figure 1.14e RTL view of REG\_R.

## 2. A 5<sup>th</sup> order Cauer low-pass filter.

For the purpose of this laboratory exercise, we have chosen an implementation of a more complex 5<sup>th</sup> order filter, the magnitude response function of which is given in Figure 2.1.

For those who are interested: it is a 5<sup>th</sup> Cauer filter (in the American literature commonly referred to as an elliptical filter), with an equiripple passband showing a ripple of maximally 0.1 dB and a stopband (also equiripple) always below or equal to –45 dB. The –3dB cutoff frequency is positioned at 0.34 of the sample frequency.



**Figure 2.1a** Magnitude transfer function of the 5<sup>th</sup> order Cauer filter.

The filter can be described with its transfer function

$$H(z) = \frac{\sum_{i=0}^5 b_i z^{-i}}{\sum_{j=0}^5 a_j z^{-j}} \quad \dots (2.1)$$

or alternatively with its outputs sequence

$$y[n] = \sum_{i=0}^5 b_i \cdot x[n-i] - \sum_{j=1}^5 a_j \cdot y[n-j] \quad \dots (2.2)$$

so again sums of products, but this time with feedback of the output samples. Table 2.1 shows the coefficient values belonging to the characteristic of Figure 2.1a.

It is surely possible to create a structure that shows a resemblance with the FIR filter shown earlier. In fact, many different architectures have been studied in the literature to implement this kind of transfer functions. We will choose here for an implementation in the form of a Wave

Digital Filter (WDF). This type of filter implementations show outstanding stability and accuracy properties, although they may look a little bit complicated. For our scheduling purposes, however, this is a nice side effect.

**Table 2.1** coefficient for the 5<sup>th</sup> order Caer filter of Figure 2.1

$b_0$	0.20391160908339	$a_0$	1.00000000000000
$b_1$	0.87047736117673	$a_1$	1.47740114108676
$b_2$	1.61565809470470	$a_2$	1.72643982821984
$b_3$	1.61565809470470	$a_3$	0.82548383012058
$b_4$	0.87047736117673	$a_4$	0.34088842960612
$b_5$	0.20391160908339	$a_5$	0.00988090089635

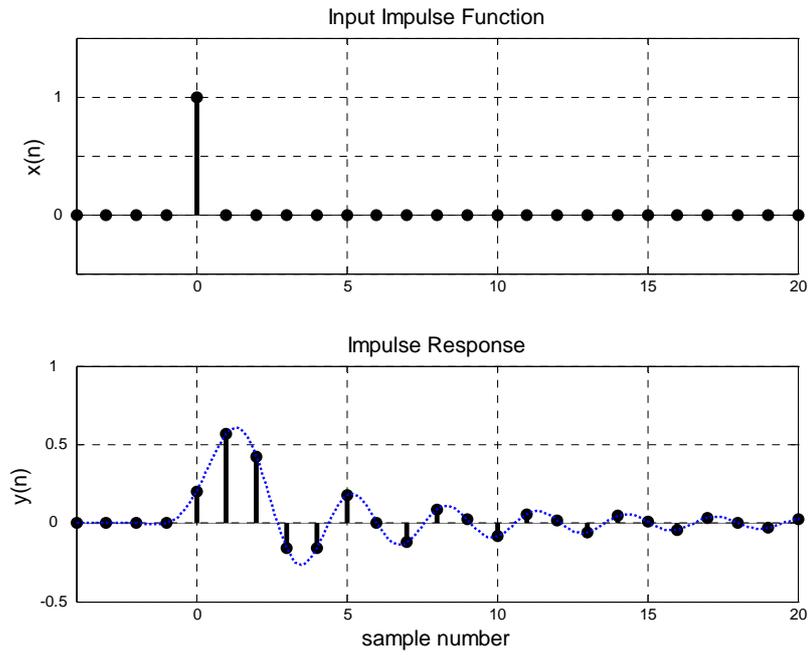
When this filter is described in the time domain, again with a unit impulse response and a unit step response, we obtain the output sample values given in Table 2.2.

For completeness, Figures 2.1b and c show these responses in graphical format.

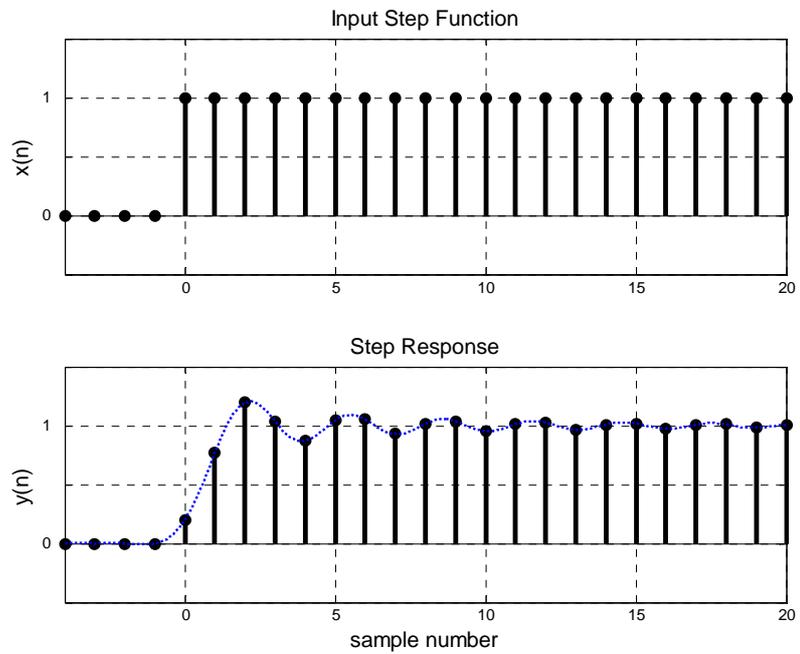
**Table 2.2** unit impulse response and unit step response

First 17 output samples, when at $n = 0$ a unit impulse function is applied to the input:	$n$	First 17 output samples, when at $n = 0$ a unit step function is applied to the input:
0.20391160908339	0	0.20391160908339
0.56921811723610	1	0.77312972631949
0.42265347541487	2	1.19578320173436
-0.15981719677043	3	1.03596600496393
-0.16248578323008	4	0.87348022173384
0.17493475418864	5	1.04841497592248
0.00429756663652	6	1.05271254255900
-0.12393053401583	7	0.92878200854316
0.08823847261738	8	1.01702048116054
0.02201969023977	9	1.03904017140032
-0.08576118081758	10	0.95327899058273
0.05805258511653	11	1.01133157569926
0.01526273551051	12	1.02659431120977
-0.06035714314809	13	0.96623716806168
0.04391746724945	14	1.01015463531113
0.00777806140658	15	1.01793269671771
-0.04326483749048	16	0.97466785922723

It can be seen that the step response is the cumulative sum of the impulse response values. This, in fact, should be the case, since the unit step function is also the cumulative sum of the unit impulse function, and our filter is a linear system.



**Figure 2.1b** Impulse response of the 5<sup>th</sup> order Cauer filter.



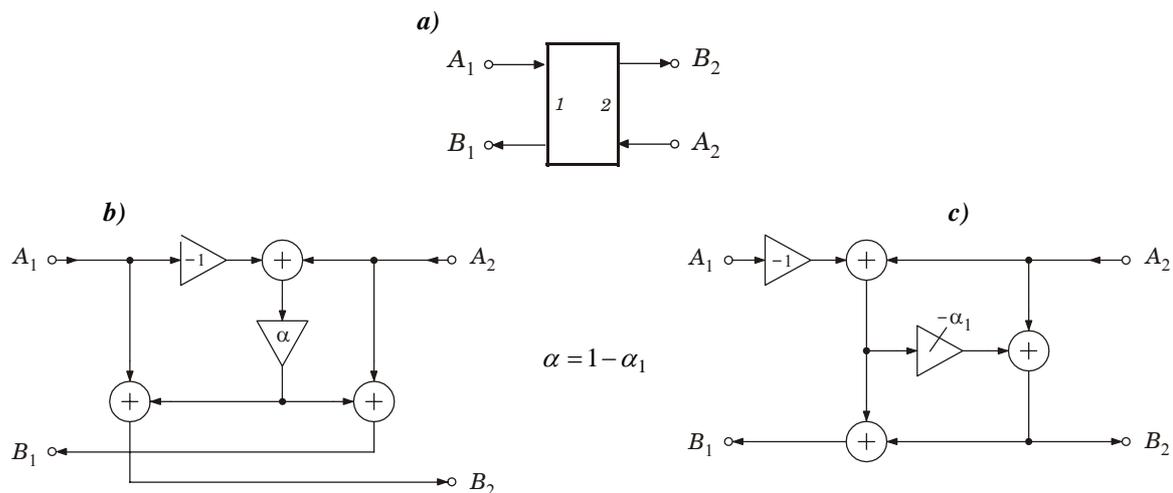
**Figure 2.1c** Step response of the 5<sup>th</sup> order Cauer filter.

## 2-port and 3-port adapters

The different WDF filter structures that we will investigate are all constructed with the aid of so-called 2-port and 3-port adapters, combined with delay elements.

The symbols that are commonly used, and the block schemes what is really inside of them, are shown in Figures 2.2 and 2.4a to d. Notice that they are again build up using multipliers and adders. In the block schemes, we also notice the presence of negations ( $* -1$ ), but in almost any case these can be combined with an adder to form a subtractor. This reduces the number of resources.

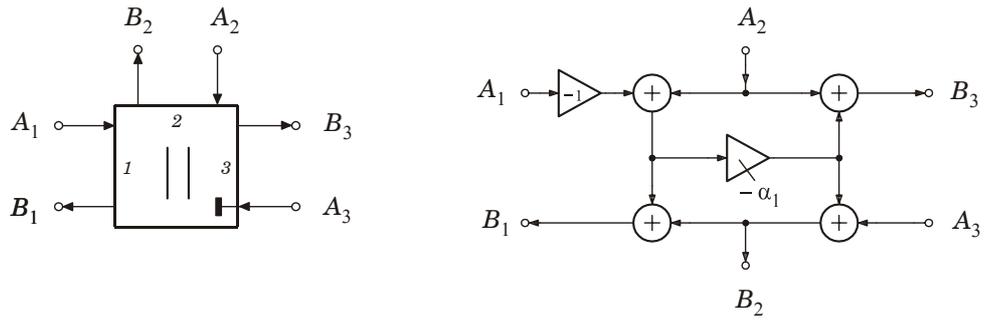
Also, each adaptor shows to comprise at least one constant coefficient multipliers, with a coefficient  $\alpha$ ,  $\alpha_1$  or  $\alpha_3$ .



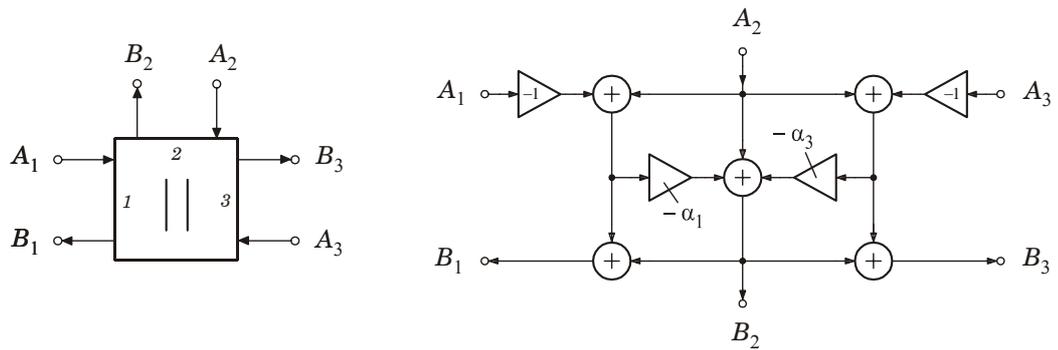
**Figure 2.2** Symbol a), and two functionally equivalent realization schemes b) and c) for a ‘2 port adaptor’

In Figures 2.5a and b, 2.6a and b, 2.7a and b and 2.8a and b, twelve different structures are given that all realize exactly the same transfer function. In fact, each of these Wave Digital Filters not only realizes the low-pass transfer function of Figure 2.1a from input  $A_{in}$  ( input data  $x[n]$  ) to output  $B_{fwd}$  ( output samples  $y[n]$  ), but also a ‘reflection’ function from  $A_{in}$  to output  $B_{rev}$  that will show a high-pass magnitude transfer function. Here, we will focus ourselves to the forward output only.

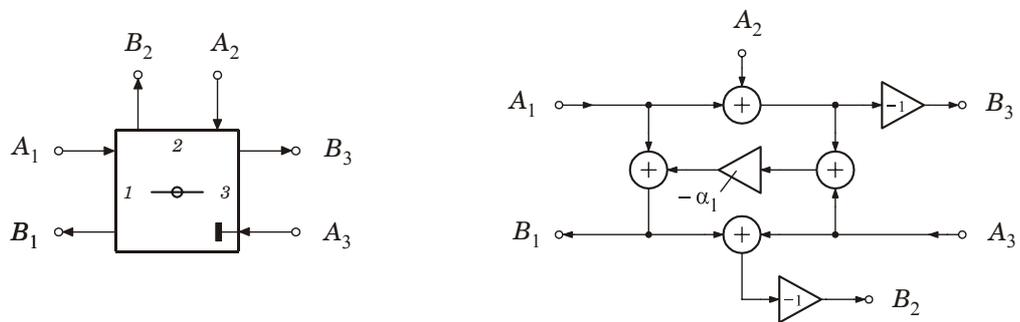
Figures 2.9a and b show even four more implementations, again of the same transfer functions and also derived using the wave digital filter theory, but with a completely different architecture. These implementations are called Lattice Wave Digital Filters (LWDFs) and are realized using only 2-port adapters, again delay elements, two more adders/subtractors and two divide-by-2 functions at the outputs.



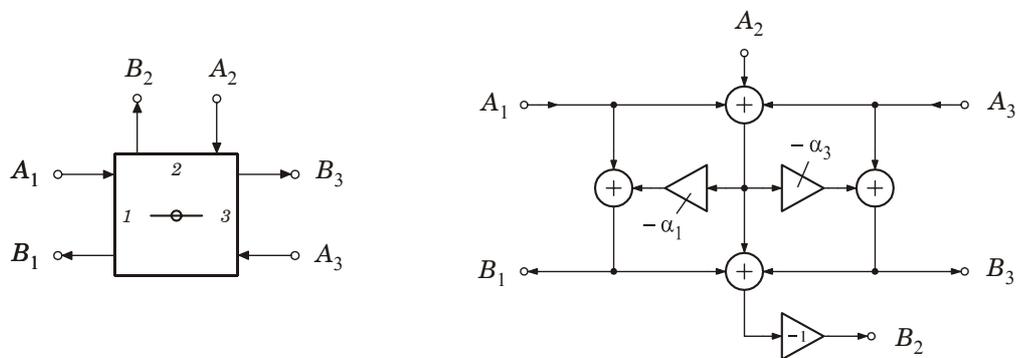
**Figure 2.3a** Symbol and Block scheme for a ‘matched 3 port parallel adaptor’



**Figure 2.3b** Symbol and Block scheme for a ‘3 port parallel adaptor’

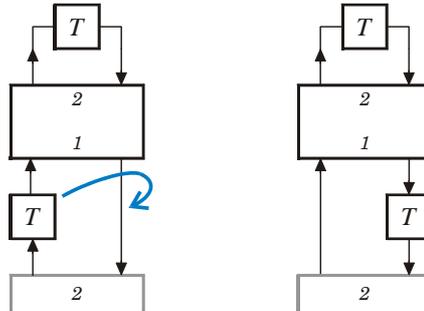


**Figure 2.3c** Symbol and Block scheme for a ‘matched 3 port serial adaptor’



**Figure 2.3d** Symbol and Block scheme for a ‘3 port serial adaptor’

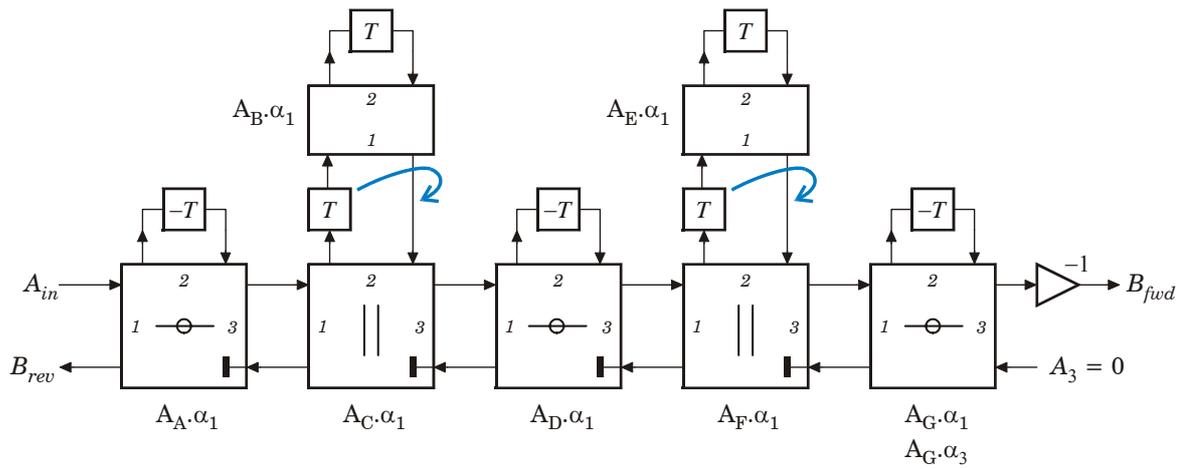
In the block schemes to follow, when 2-ports are present, there are actually two distinct possibilities to locate the interconnecting delay element. The transfer function from input to output(s) stays exactly the same. Of course the positioning does influence the data flow paths, so from the scheduling point of view there can be a preference to choose one or the other. In the block schemes, the two circuits are simply differentiated using the postfix `_L` or `_R` at the end of the (file)name.



**Figure 2.4** *Two possible positions for the interconnecting delay element.*

The names given here to the WDF structures are intended to be self-explanatory:

- an ‘s’ or ‘p’ is used to indicate that the input 3-port adapter is a serial or parallel type,
- if this is followed by ‘sym’, the setup of the 3-port adapters in the lower row is symmetric, i.e. the 2-multiplier-adapter is in the middle (possible here, because we have an odd order filter). If not symmetric, this adapter is positioned as the forward-output one.
- a ‘2p’ or ‘3p’ indicates whether the structure is built using respectively a combination of 2-port and 3-port adapters, or with only 3-port adapters.
- of course, if 2-port adapters are being used, the ‘2p’ is followed by an indication about the location of the interconnecting delay element (see above, either ‘L’ or ‘R’).

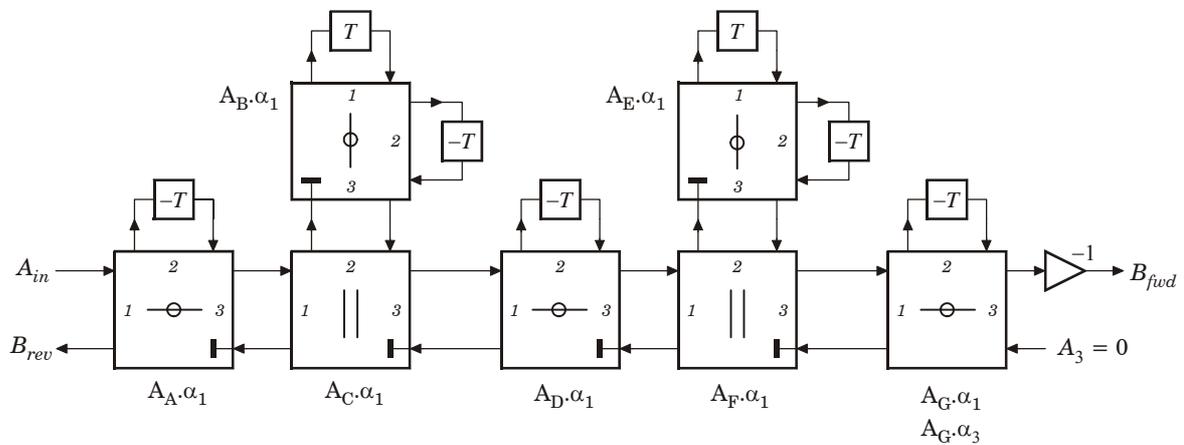


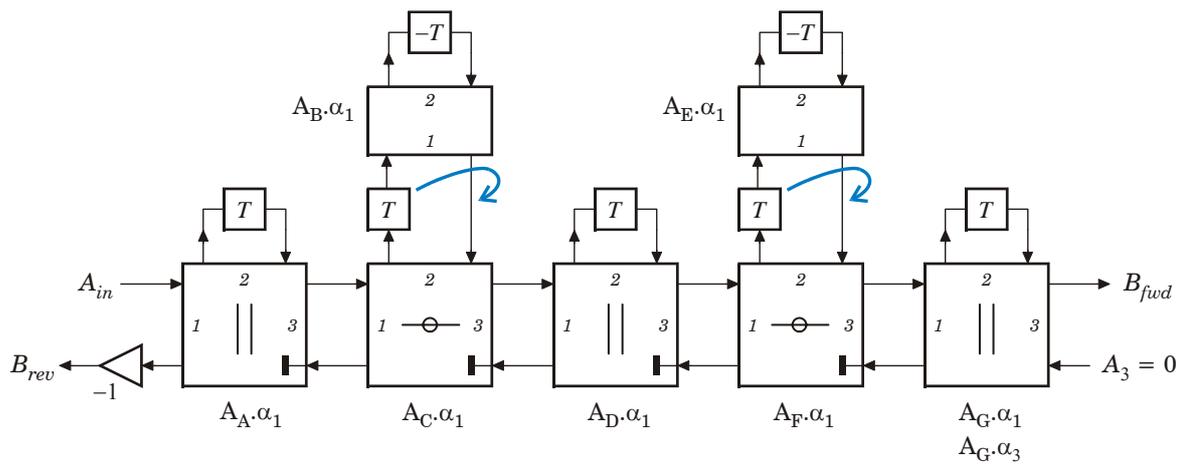
$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	1.88239554055869
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.43378869238454
$\alpha_1 E$	1.75205223583238
$\alpha_1 F$	0.52818356937584
$\alpha_1 G$	0.76350613851909
$\alpha_3 G$	0.82195774820984

Figure 2.5a ‘WDF\_C5\_s\_2p\_L’ and ‘WDF\_C5\_s\_2p\_R’

$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	0.94119777027934
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.43378869238454
$\alpha_1 E$	0.87602611791619
$\alpha_1 F$	0.52818356937584
$\alpha_1 G$	0.76350613851909
$\alpha_3 G$	0.82195774820984

Figure 2.5b ‘WDF\_C5\_s\_3p’



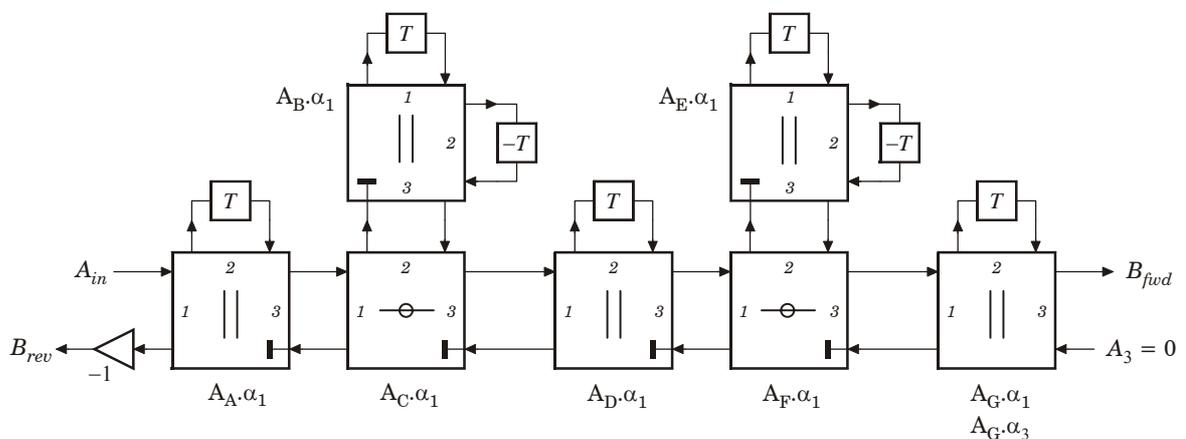


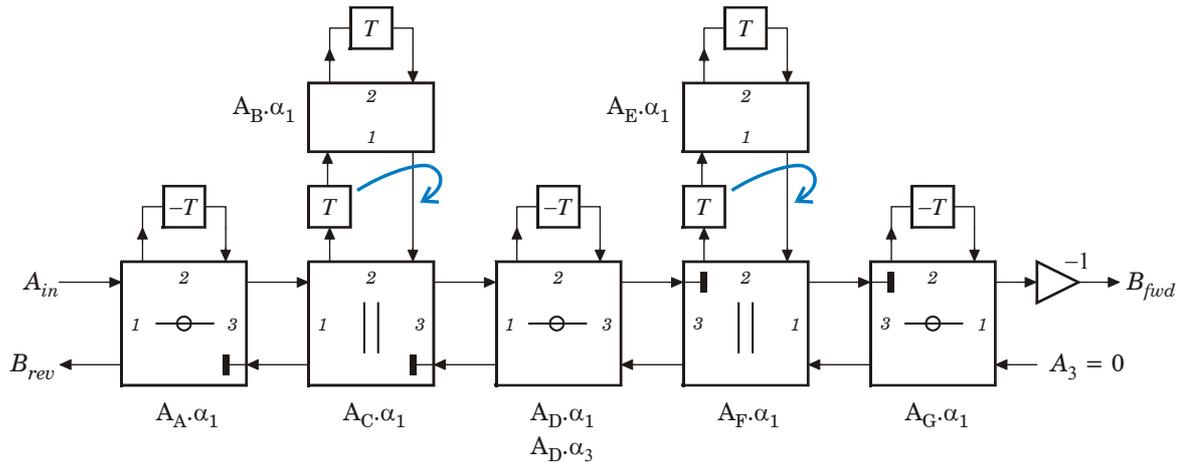
$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	0.11760445944132
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.43378869238454
$\alpha_1 E$	0.24794776416762
$\alpha_1 F$	0.52818356937584
$\alpha_1 G$	0.76350613851909
$\alpha_3 G$	0.82195774820984

Figure 2.6a 'WDF\_C5\_p\_2p\_L' and 'WDF\_C5\_p\_2p\_R'

$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	0.05880222972066
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.43378869238454
$\alpha_1 E$	0.12397388208381
$\alpha_1 F$	0.52818356937584
$\alpha_1 G$	0.76350613851909
$\alpha_3 G$	0.82195774820984

Figure 2.6b 'WDF\_C5\_p\_3p'



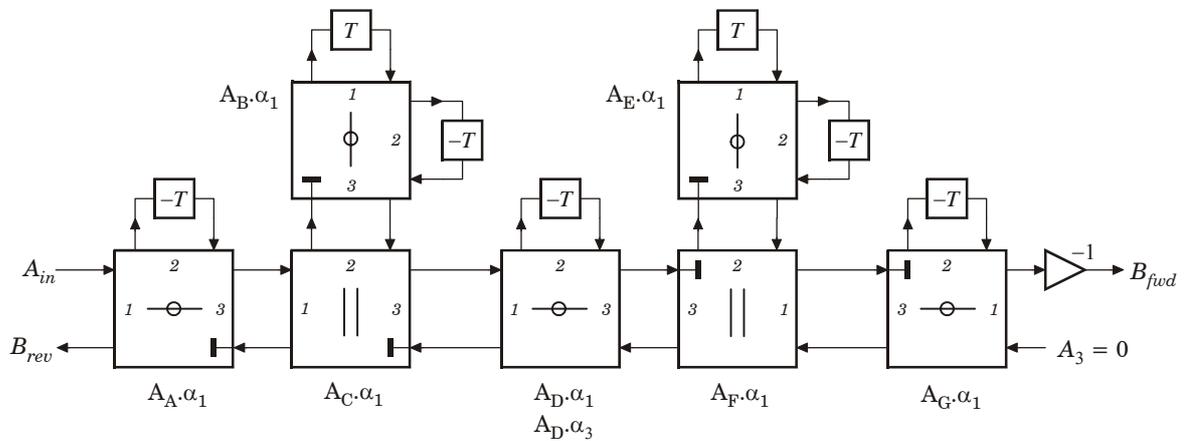


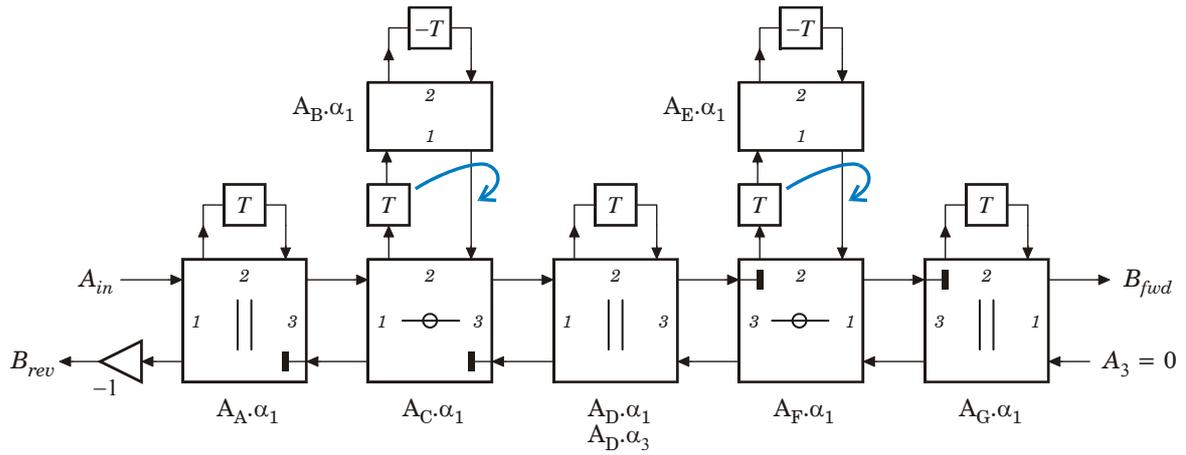
$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	1.88239554055869
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.58427183716240
$\alpha_3 D$	0.65309574126830
$\alpha_1 E$	1.75205223583238
$\alpha_1 F$	0.56685999288327
$\alpha_1 G$	0.66474874952101

**Figure 2.7a** ‘WDF\_C5\_s\_sym\_2p\_L’ and ‘WDF\_C5\_s\_sym\_2p\_R’

$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	0.94119777027934
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.58427183716240
$\alpha_3 D$	0.65309574126830
$\alpha_1 E$	0.87602611791619
$\alpha_1 F$	0.56685999288327
$\alpha_1 G$	0.66474874952101

**Figure 2.7b** ‘WDF\_C5\_s\_sym\_3p’



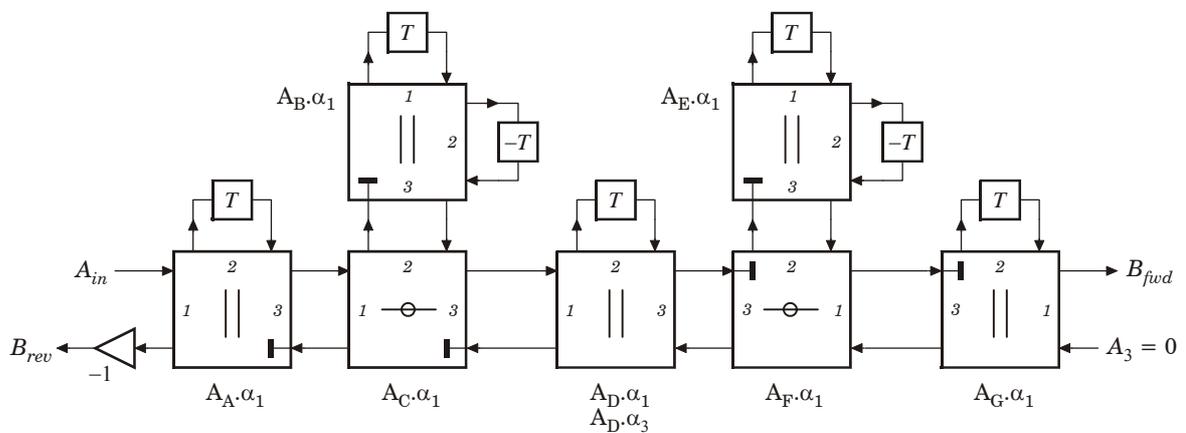


$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	0.11760445944132
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.58427183716240
$\alpha_3 D$	0.65309574126830
$\alpha_1 E$	0.24794776416762
$\alpha_1 F$	0.56685999288327
$\alpha_1 G$	0.66474874952101

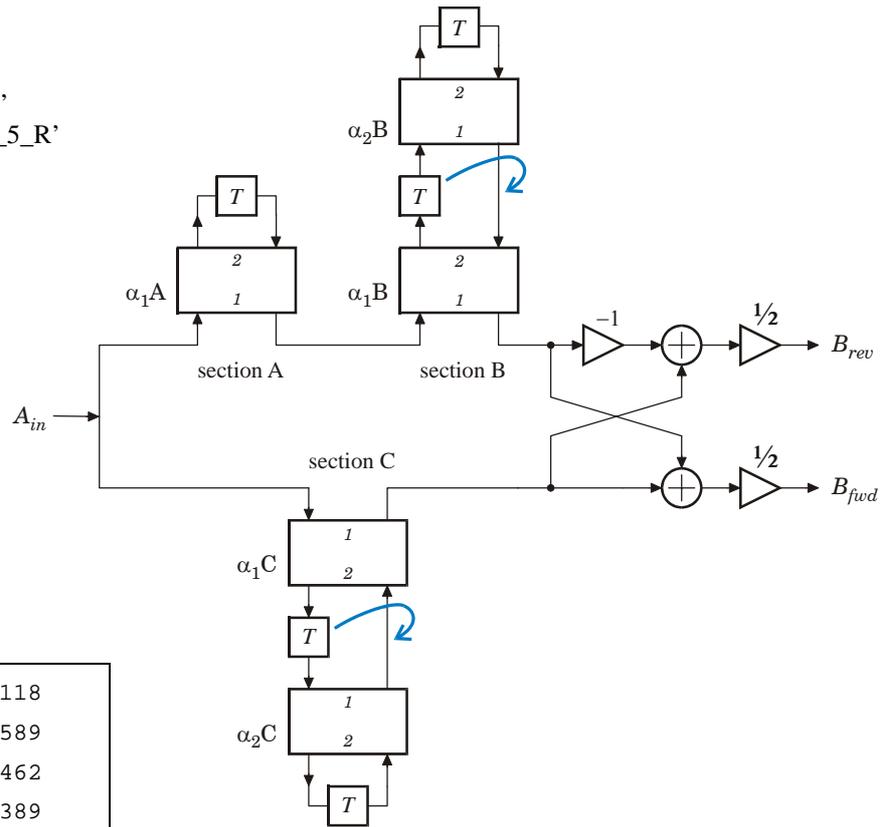
Figure 2.8a ‘WDF\_C5\_p\_sym\_2p\_L’ and ‘WDF\_C5\_p\_sym\_2p\_R’

$\alpha_1 A$	0.61567453576615
$\alpha_1 B$	0.05880222972066
$\alpha_1 C$	0.46968595194661
$\alpha_1 D$	0.58427183716240
$\alpha_3 D$	0.65309574126830
$\alpha_1 E$	0.12397388208381
$\alpha_1 F$	0.56685999288327
$\alpha_1 G$	0.66474874952101

Figure 2.8b ‘WDF\_C5\_p\_sym\_3p’



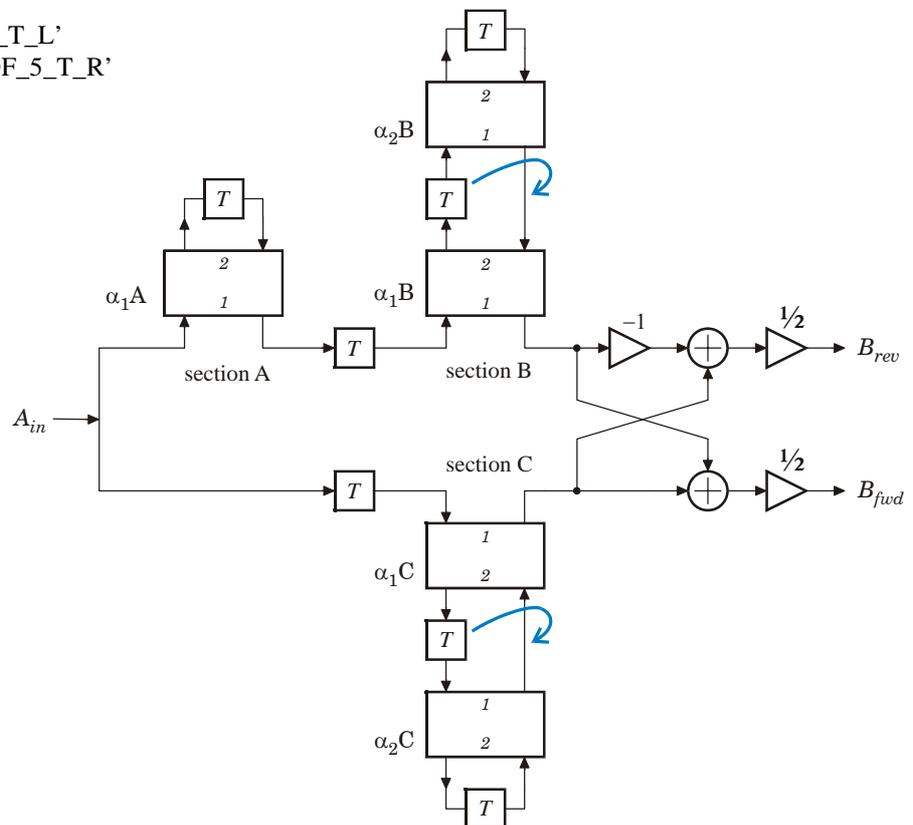
**Figure 2.9a** ‘LWDF\_5\_L’ and ‘LWDF\_5\_R’



Coefficients for both  
‘LWDF\_5\_L|R’ and  
‘LWDF\_5\_T\_L|R’.

$\alpha_1A$	-0.03119210537118
$\alpha_1B$	-0.82935592147589
$\alpha_2B$	-0.52095844720462
$\alpha_1C$	-0.38195386087389
$\alpha_2C$	-0.35687921966298

**Figure 2.9b** ‘LWDF\_5\_T\_L’ and ‘LWDF\_5\_T\_R’



## Assignments 2006/2007.

For this Laboratory Exercise, you are expected to investigate and compare a number of computational circuits with respect to scheduling and retiming possibilities. For one of them you also have to create and run the MATLAB and VHDL testbenches.

If you got time to spare, you can synthesize the VHDL code with Synplify\_Pro and test the retiming function available inside Synplify\_Pro.

All files that have been used for the FIR5 example in this manual will be available to you.

In more detail:

Each one of you will be assigned a selection of 5 circuits/structures from the Figures 2.5a/b to 2.9a/b. The structures from 2.9a and 2.9b (which is a pipelined version of 2.9a) are always part of this set, the other 3 are chosen more or less randomly. For all structures, the .cir-files have been created beforehand, as are the coefficients-data. This data will be present in your start-up directory. The files can be identified by the names given in the Figure captions.

As mentioned before, the .cir-files are plain ASCII text files that can be read or edited with the text editor of your choice. The given .cir-files are optimized in such a way that negations are avoided.

- We want you to investigate and compare the structures assigned to you in terms of scheduling properties using e.g. the `schedGUI` program, and to formulate your own conclusions. Determine and plot a number of Pareto points using these designs (viz. time or clock cycles vs. area plots). For simplicity, assume that all resource types need the same amount of chip area. Figure out which of the circuits (only those that are assigned to you) can be retimed (use `delayMUL = delayALU = 1`) and find out why (or why not). What is to be preferred: a `_L` or a `_R` realization? Optionally take a look at the effects of different latencies of the resources.
- Next, select one of the assigned circuits, substantially based on your conclusions, and generate a MATLAB testbench for it. Run the testbench with different .INP-files, where each .INP-file represents a different fixed-point format. Be sure to use the correct `coeffs` data format and sequence in the .INP-file.

The coefficient data is available as .mat-files that you can import in MATLAB with the command

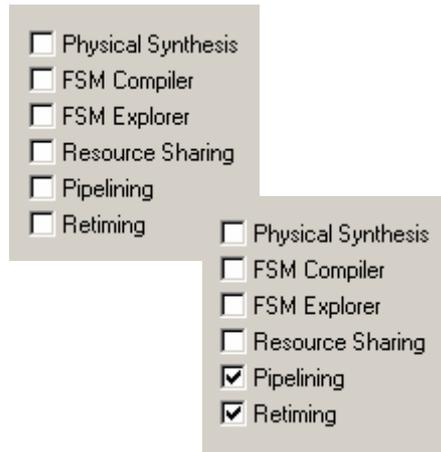
```
>> load coeffs_NAME
```

The result will be that you can access a cell-array variable `coeffs` from your MATLAB console that has in its first column the names of the variables and in its second column the corresponding values.

- If you are satisfied with a certain format and the resulting accuracy, generate the VHDL files and testbench for this structure.
- Then, use the *ModelSim* simulator to check your code and to verify that this VHDL testbench returns the same output values as you found with the MATLAB testbench. In case of discrepancy problems, be reminded that the MATLAB testbench can be run in a “debug”-mode to show all intermediate results, which you can compare with intermediate results returned by the simulator. In the Reference Guide, a description of the timing used for the stimuli in the VHDL testbench can be found.

- If all goes well, and you have some time left or are just curious, run the appropriate (see the Reference Guide) VHDL files with Synplify\_Pro and identify the various components in the RTL view results.

Run the synthesis with and without enabling the internal Retiming (you get Pipelining for free) option and determine from the log-files whether this will have effect on the estimated maximum frequency that the design is expected to work with and explain your findings (what has been changed by this option, why is that possible, etc.)



Already mentioned in the introduction, but repeated here for your convenience:

For the exam you will have to write a short report, which at least covers the following content:

- prints of the Data Flow Graphs assigned to you, including operations, delays and interconnect (Also print the values of the input and calculated output variables),
- plots of Pareto curves of the Sequencing Graphs, including scheduling parameters,
- prints of feasible retimed circuits, with Pareto points and resulting Sequencing Graphs,
- a short motivation (textual) for the Scheduled Sequencing Graph selected by you for simulation,
- a print-out of the simulation results (wave viewer), and
- (optionally) a print-out of the schematic diagram of your design.

-----

## Appendix.

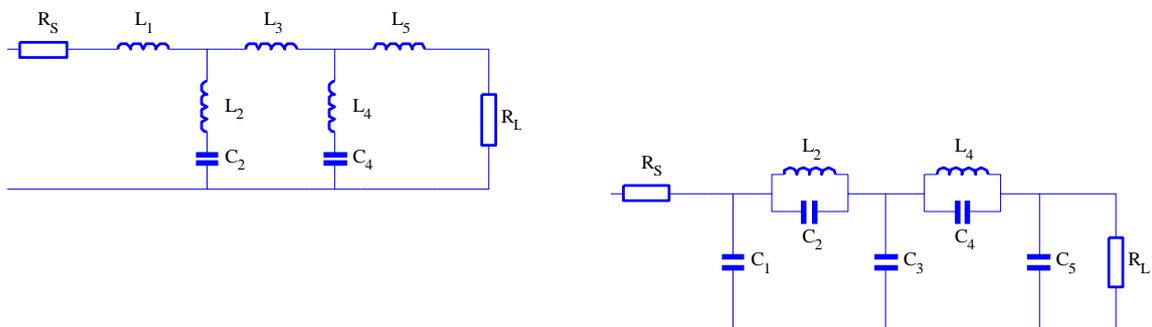
### Architecture descriptions for the Wave Digital Filter implementations.

For each of the structures in Figs. 2.5 – 2.9, a flow diagram and the cir-file that describes that flow diagram are given:

<a href="#">WDF_C5_s_2p_L</a>	.....	pages 36–37
<a href="#">WDF_C5_s_2p_R</a>	.....	38–39
<a href="#">WDF5_C_s_3p</a>	.....	40–41
<a href="#">WDF_C5_p_2p_L</a>	.....	42–43
<a href="#">WDF_C5_p_2p_R</a>	.....	44–45
<a href="#">WDF_C5_p_3p</a>	.....	46–47
<a href="#">WDF_C5_s_sym_2p_L</a>	.....	48–49
<a href="#">WDF_C5_s_sym_2p_R</a>	.....	50–51
<a href="#">WDF_C5_s_sym_3p</a>	.....	52–53
<a href="#">WDF_C5_p_sym_2p_L</a>	.....	54–55
<a href="#">WDF_C5_p_sym_2p_R</a>	.....	56–57
<a href="#">WDF_C5_p_sym_3p</a>	.....	58–59
<a href="#">LWDF_5_L</a>	.....	60–61
<a href="#">LWDF_5_R</a>	.....	62–63
<a href="#">LWDF_5_T_L</a>	.....	64–65
<a href="#">LWDF_5_T_R</a>	.....	66–67

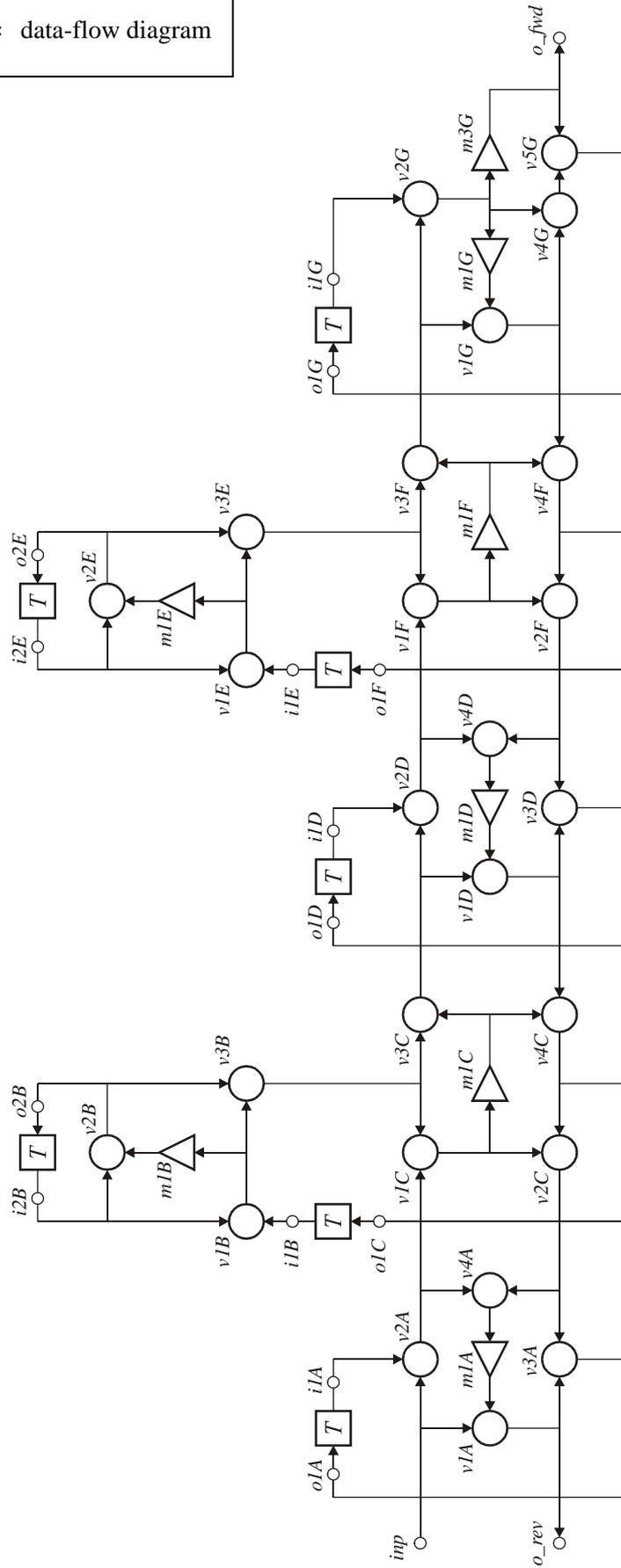
All data that follows has been created with the (L)WDF Toolbox for MATLAB (developed at the TU Delft). The interested reader can find more about this toolbox at [http://ens.ewi.tudelft.nl/~huib/mtbx/more\\_wdfs.php](http://ens.ewi.tudelft.nl/~huib/mtbx/more_wdfs.php)

The WD-filters are translations in the time-discrete domain of the dual, time-continuous low-pass realizations shown below.



In the LWDF-filters, the transfer function to be realized is obtained by combining (adding for low-pass, subtracting for high-pass) two all-pass transfer functions that differ in phase in the correct way.

WDF\_C5\_s\_2p\_L: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:36:19.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "-" in top row
o1A = v3A;
i1A = To1A;
% bottom row 3p adaptor, type s1m_ppp_pii
v1A = inp - m1A;
v2A = inp + i1A;
v3A = v1A + v2C;
v4A = v2A + v2C;
m1A = a1A * v4A;

o_rev = v1A;

% top row 2p adaptor, serial resonator
o1C = v4C;
i1B = To1C;
v1B = i2B - i1B;
m1B = a1B * v1B;
v2B = i2B - m1B;
o2B = v2B;
i2B = To2B;
v3B = v1B + v2B;
% bottom row 3p adaptor, type plm_ipp_ppp
v1C = v2A + v3B;
v2C = v1C + v4C;
m1C = a1C * v1C;
v3C = v3B - m1C;
v4C = v1D - m1C;

% "-" in top row
o1D = v3D;
i1D = To1D;
% bottom row 3p adaptor, type s1m_ppp_pii
v1D = v3C - m1D;
v2D = v3C + i1D;
v3D = v1D + v2F;
v4D = v2D + v2F;
m1D = a1D * v4D;

% top row 2p adaptor, serial resonator
o1F = v4F;
i1E = To1F;
v1E = i2E - i1E;
m1E = a1E * v1E;
v2E = i2E - m1E;
o2E = v2E;
i2E = To2E;
v3E = v1E + v2E;
% bottom row 3p adaptor, type plm_ipp_ppp
v1F = v2D + v3E;
v2F = v1F + v4F;
m1F = a1F * v1F;
v3F = v3E - m1F;
v4F = v1G - m1F;

% "-" in top row
o1G = v5G;
i1G = To1G;
% last bottom row 3p adaptor with A3=0, type S2m_pp0_pii
v1G = v3F - m1G;
v2G = v3F + i1G;
m1G = a1G * v2G;
m3G = a3G * v2G;
v4G = v2G + v1G;
v5G = v4G - m3G;

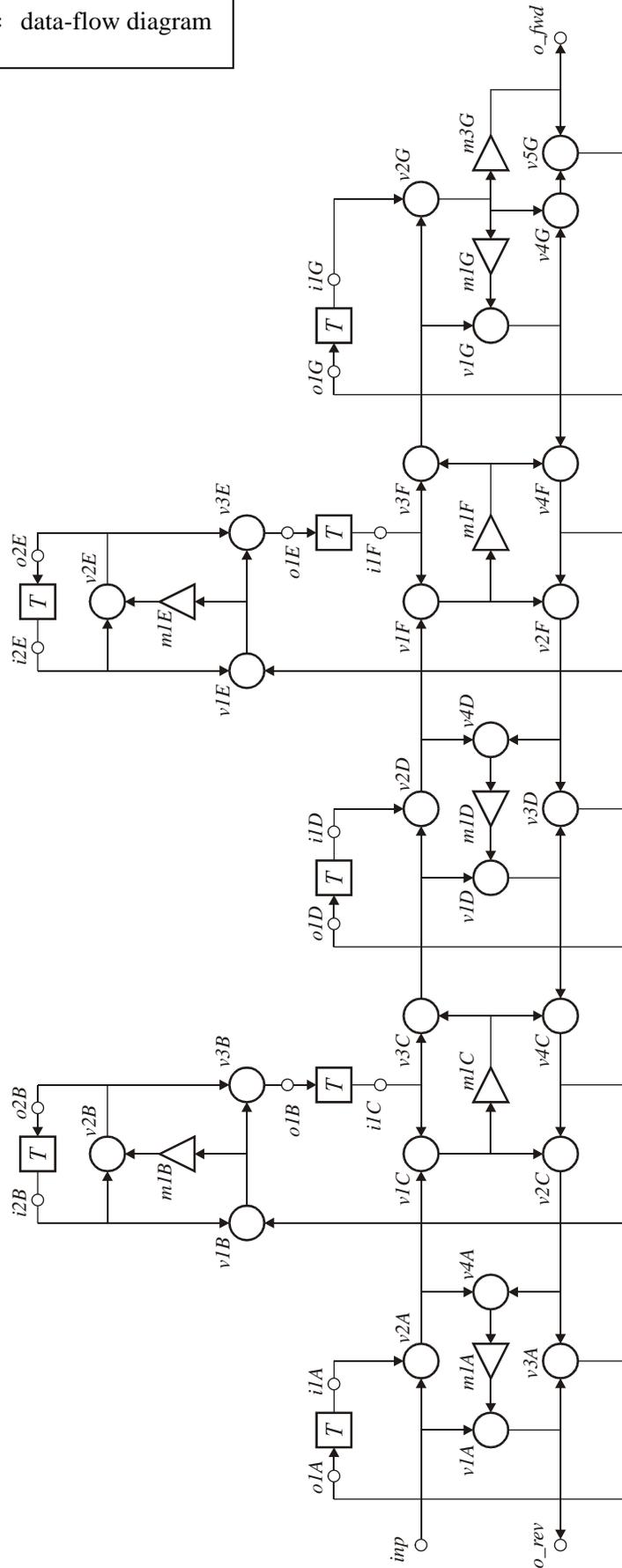
o_fwd = m3G;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_s\_2p\_L.cir

WDF\_C5\_s\_2p\_R: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:36:38.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "-" in top row
o1A = v3A;
i1A = To1A;
% bottom row 3p adaptor, type s1m_ppp_pii
v1A = inp - m1A;
v2A = inp + i1A;
v3A = v1A + v2A;
v4A = v2A + v2C;
m1A = a1A * v4A;

o_rev = v1A;

% top row 2p adaptor, serial resonator
v1B = i2B - v4C;
m1B = a1B * v1B;
v2B = i2B - m1B;
o2B = v2B;
i2B = To2B;
v3B = v1B + v2B;
o1B = v3B;
i1C = To1B;
% bottom row 3p adaptor, type plm_ipp_ppp
v1C = v2A + i1C;
v2C = v1C + v4C;
m1C = a1C * v1C;
v3C = i1C - m1C;
v4C = v1D - m1C;

% "-" in top row
o1D = v3D;
i1D = To1D;
% bottom row 3p adaptor, type s1m_ppp_pii
v1D = v3C - m1D;
v2D = v3C + i1D;
v3D = v1D + v2D;
v4D = v2D + v2F;
m1D = a1D * v4D;

% top row 2p adaptor, serial resonator
v1E = i2E - v4F;
m1E = a1E * v1E;
v2E = i2E - m1E;
o2E = v2E;
i2E = To2E;
v3E = v1E + v2E;
o1E = v3E;
i1F = To1E;
% bottom row 3p adaptor, type plm_ipp_ppp
v1F = v2D + i1F;
v2F = v1F + v4F;
m1F = a1F * v1F;
v3F = i1F - m1F;
v4F = v1G - m1F;

% "-" in top row
o1G = v5G;
i1G = To1G;
% last bottom row 3p adaptor with A3=0, type S2m_pp0_pii
v1G = v3F - m1G;
v2G = v3F + i1G;
m1G = a1G * v2G;
m3G = a3G * v2G;
v4G = v2G + v1G;
v5G = v4G - m3G;

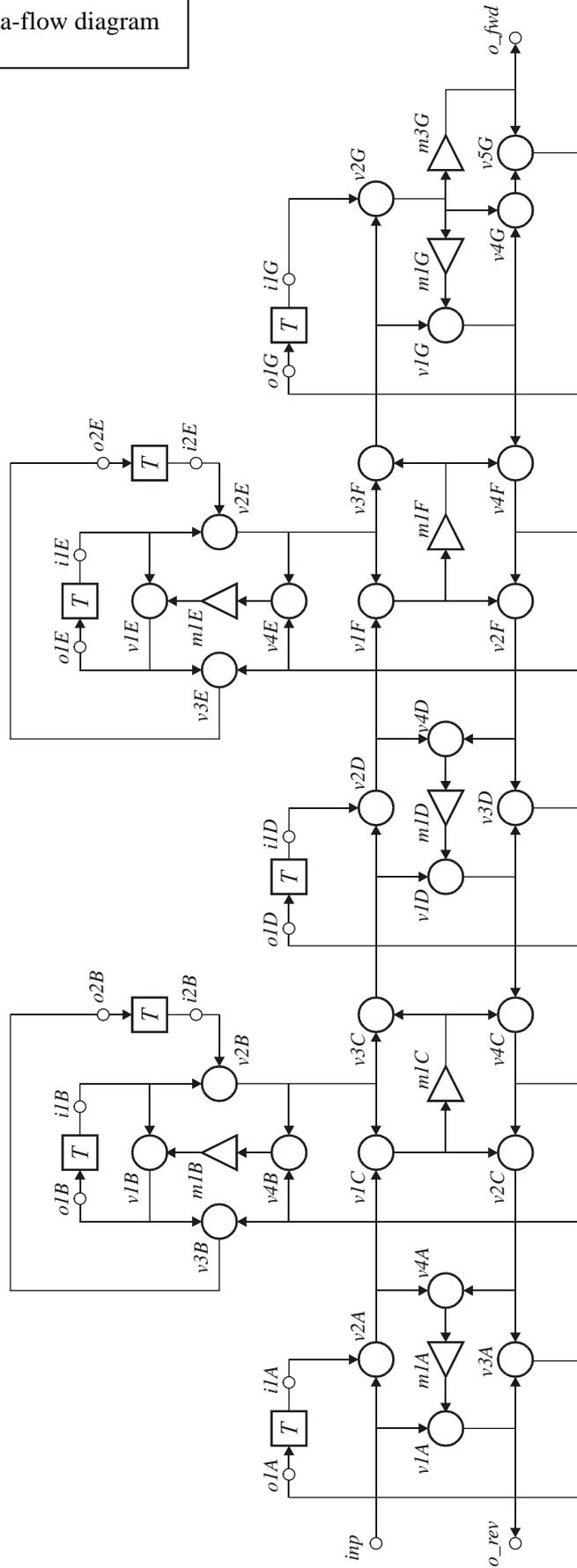
o_fwd = m3G;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_s\_2p\_R.cir

WDF\_C5\_s\_3p: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:36:04.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "-" in top row
o1A = v3A;
i1A = To1A;
% bottom row 3p adaptor, type slm_ppp_pii
v1A = inp - m1A;
v2A = inp + i1A;
v3A = v1A + v2C;
v4A = v2A + v2C;
m1A = a1A * v4A;

o_rev = v1A;

% top row 3p adaptor, reflection free, with T and -T
v1B = i1B - m1B;
v2B = i1B + i2B;
v3B = v1B - v4C;
v4B = v2B - v4C;
m1B = a1B * v4B;
o1B = v1B;
o2B = v3B;
i1B = To1B;
i2B = To2B;
% bottom row 3p adaptor, type plm_ipp_ppp
v1C = v2A + v2B;
v2C = v1C + v4C;
m1C = a1C * v1C;
v3C = v2B - m1C;
v4C = v1D - m1C;

% "-" in top row
o1D = v3D;
i1D = To1D;
% bottom row 3p adaptor, type slm_ppp_pii
v1D = v3C - m1D;
v2D = v3C + i1D;
v3D = v1D + v2F;
v4D = v2D + v2F;
m1D = a1D * v4D;

% top row 3p adaptor, reflection free, with T and -T
v1E = i1E - m1E;
v2E = i1E + i2E;
v3E = v1E - v4F;
v4E = v2E - v4F;
m1E = a1E * v4E;
o1E = v1E;
o2E = v3E;
i1E = To1E;
i2E = To2E;
% bottom row 3p adaptor, type plm_ipp_ppp
v1F = v2D + v2E;
v2F = v1F + v4F;
m1F = a1F * v1F;
v3F = v2E - m1F;
v4F = v1G - m1F;

% "-" in top row
o1G = v5G;
i1G = To1G;
% last bottom row 3p adaptor with A3=0, type S2m_pp0_pii
v1G = v3F - m1G;
v2G = v3F + i1G;
m1G = a1G * v2G;
m3G = a3G * v2G;
v4G = v2G + v1G;
v5G = v4G - m3G;

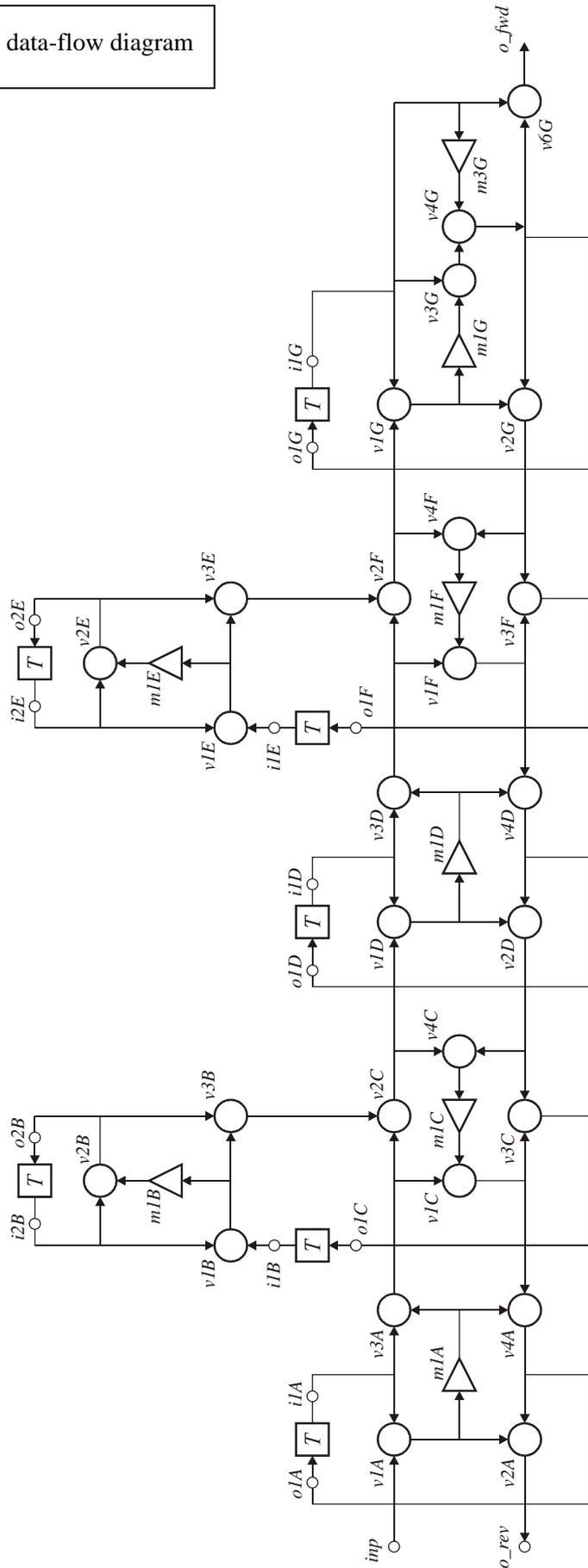
o_fwd = m3G;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_s\_3p.cir

WDF\_C5\_p\_2p\_L: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:37:19.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "+" in top row
o1A = v4A;
i1A = To1A;
% bottom row 3p adaptor, type plm_ppp_ipp
v1A = inp - i1A;
v2A = v1A - v4A;
m1A = a1A * v1A;
v3A = i1A + m1A;
v4A = v1C + m1A;

o_rev = v2A;

% top row 2p adaptor, parallel resonator
o1C = v3C;
i1B = To1C;
v1B = i2B - i1B;
m1B = a1B * v1B;
v2B = m1B - i2B;
o2B = v2B;
i2B = To2B;
v3B = v1B - v2B;
% bottom row 3p adaptor, type slm_pip_pii
v1C = v3A - m1C;
v2C = v3A - v3B;
v3C = v1C + v2D;
v4C = v2C + v2D;
m1C = a1C * v4C;

% "+" in top row
o1D = v4D;
i1D = To1D;
% bottom row 3p adaptor, type plm_ipp_ppp
v1D = v2C + i1D;
v2D = v1D + v4D;
m1D = a1D * v1D;
v3D = i1D - m1D;
v4D = v1F - m1D;

% top row 2p adaptor, parallel resonator
o1F = v3F;
i1E = To1F;
v1E = i2E - i1E;
m1E = a1E * v1E;
v2E = m1E - i2E;
o2E = v2E;
i2E = To2E;
v3E = v1E - v2E;
% bottom row 3p adaptor, type slm_pip_pii
v1F = v3D - m1F;
v2F = v3D - v3E;
v3F = v1F + v2G;
v4F = v2F + v2G;
m1F = a1F * v4F;

% "+" in top row
o1G = v4G;
i1G = To1G;
% last bottom row 3p adaptor with A3=0, type P2m_ip0_ppp
v1G = v2F + i1G;
m1G = a1G * v1G;
m3G = a3G * i1G;
v3G = i1G - m1G;
v4G = v3G - m3G;
v2G = v1G + v4G;
v6G = v4G + i1G;

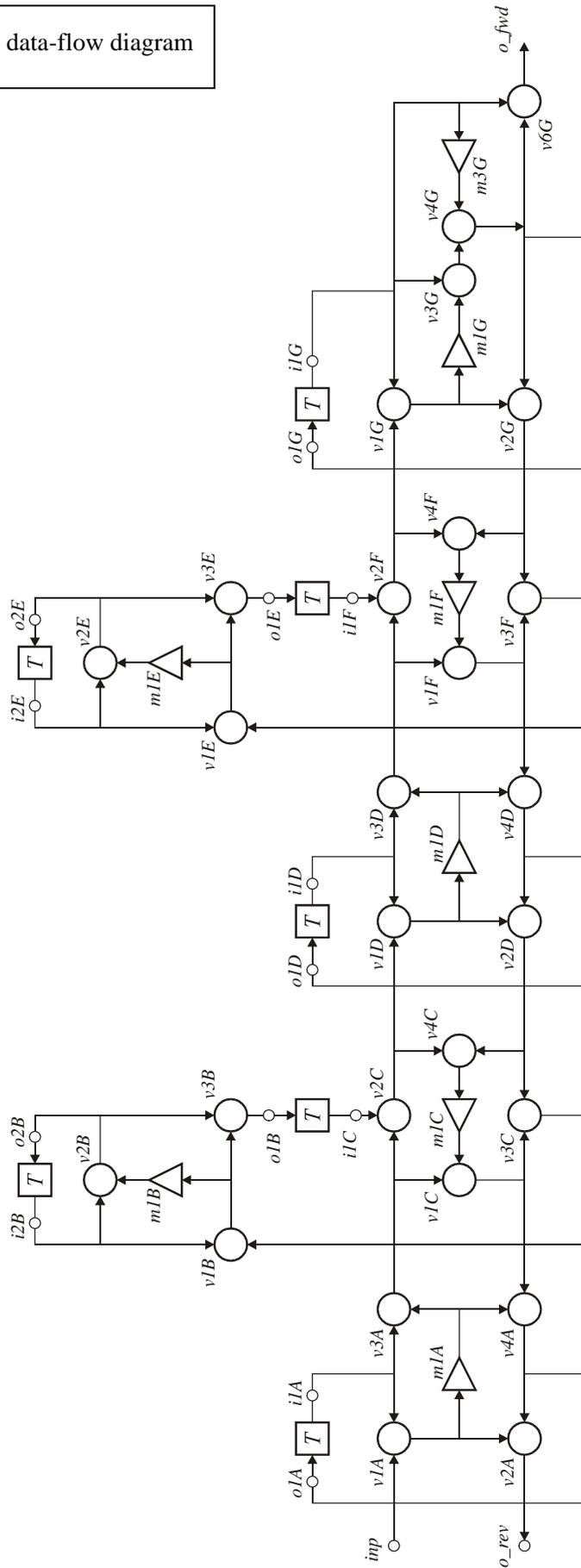
o_fwd = v6G;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_p\_2p\_L.cir

WDF\_C5\_p\_2p\_R: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:37:29.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "+" in top row
o1A = v4A;
i1A = To1A;
% bottom row 3p adaptor, type plm_ppp_ipp
v1A = inp - i1A;
v2A = v1A - v4A;
m1A = a1A * v1A;
v3A = i1A + m1A;
v4A = v1C + m1A;

o_rev = v2A;

% top row 2p adaptor, parallel resonator
v1B = i2B - v3C;
m1B = a1B * v1B;
v2B = m1B - i2B;
o2B = v2B;
i2B = To2B;
v3B = v1B - v2B;
o1B = v3B;
i1C = To1B;
% bottom row 3p adaptor, type slm_pip_pii
v1C = v3A - m1C;
v2C = v3A - i1C;
v3C = v1C + v2D;
v4C = v2C + v2D;
m1C = a1C * v4C;

% "+" in top row
o1D = v4D;
i1D = To1D;
% bottom row 3p adaptor, type plm_ipp_ppp
v1D = v2C + i1D;
v2D = v1D + v4D;
m1D = a1D * v1D;
v3D = i1D - m1D;
v4D = v1F - m1D;

% top row 2p adaptor, parallel resonator
v1E = i2E - v3F;
m1E = a1E * v1E;
v2E = m1E - i2E;
o2E = v2E;
i2E = To2E;
v3E = v1E - v2E;
o1E = v3E;
i1F = To1E;
% bottom row 3p adaptor, type slm_pip_pii
v1F = v3D - m1F;
v2F = v3D - i1F;
v3F = v1F + v2G;
v4F = v2F + v2G;
m1F = a1F * v4F;

% "+" in top row
o1G = v4G;
i1G = To1G;
% last bottom row 3p adaptor with A3=0, type P2m_ip0_ppp
v1G = v2F + i1G;
m1G = a1G * v1G;
m3G = a3G * i1G;
v3G = i1G - m1G;
v4G = v3G - m3G;
v2G = v1G + v4G;
v6G = v4G + i1G;

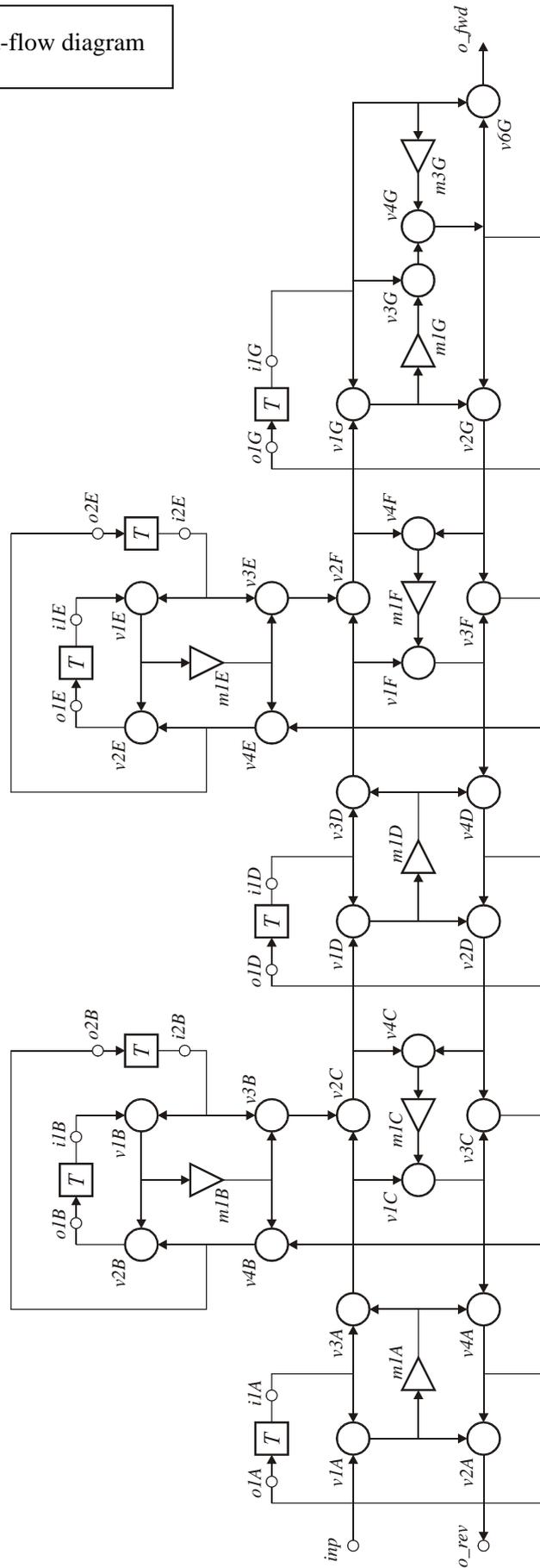
o_fwd = v6G;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_p\_2p\_R.cir

WDF\_C5\_p\_3p: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:37:08.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "+" in top row
o1A = v4A;
i1A = To1A;
% bottom row 3p adaptor, type plm_ppp_ipp
v1A = inp - i1A;
v2A = v1A - v4A;
m1A = a1A * v1A;
v3A = i1A + m1A;
v4A = v1C + m1A;

o_rev = v2A;

% top row 3p adaptor, reflection free, with T and -T
v1B = i1B + i2B;
v2B = v4B - v1B;
v3B = i2B - m1B;
v4B = m1B - v3C;
m1B = a1B * v1B;
o1B = v2B;
o2B = v4B;
i1B = To1B;
i2B = To2B;
% bottom row 3p adaptor, type slm_pip_pii
v1C = v3A - m1C;
v2C = v3A - v3B;
v3C = v1C + v2D;
v4C = v2C + v2D;
m1C = a1C * v4C;

% "+" in top row
o1D = v4D;
i1D = To1D;
% bottom row 3p adaptor, type plm_ipp_ppp
v1D = v2C + i1D;
v2D = v1D + v4D;
m1D = a1D * v1D;
v3D = i1D - m1D;
v4D = v1F - m1D;

% top row 3p adaptor, reflection free, with T and -T
v1E = i1E + i2E;
v2E = v4E - v1E;
v3E = i2E - m1E;
v4E = m1E - v3F;
m1E = a1E * v1E;
o1E = v2E;
o2E = v4E;
i1E = To1E;
i2E = To2E;
% bottom row 3p adaptor, type slm_pip_pii
v1F = v3D - m1F;
v2F = v3D - v3E;
v3F = v1F + v2G;
v4F = v2F + v2G;
m1F = a1F * v4F;

% "+" in top row
o1G = v4G;
i1G = To1G;
% last bottom row 3p adaptor with A3=0, type P2m_ip0_ppp
v1G = v2F + i1G;
m1G = a1G * v1G;
m3G = a3G * i1G;
v3G = i1G - m1G;
v4G = v3G - m3G;
v2G = v1G + v4G;
v6G = v4G + i1G;

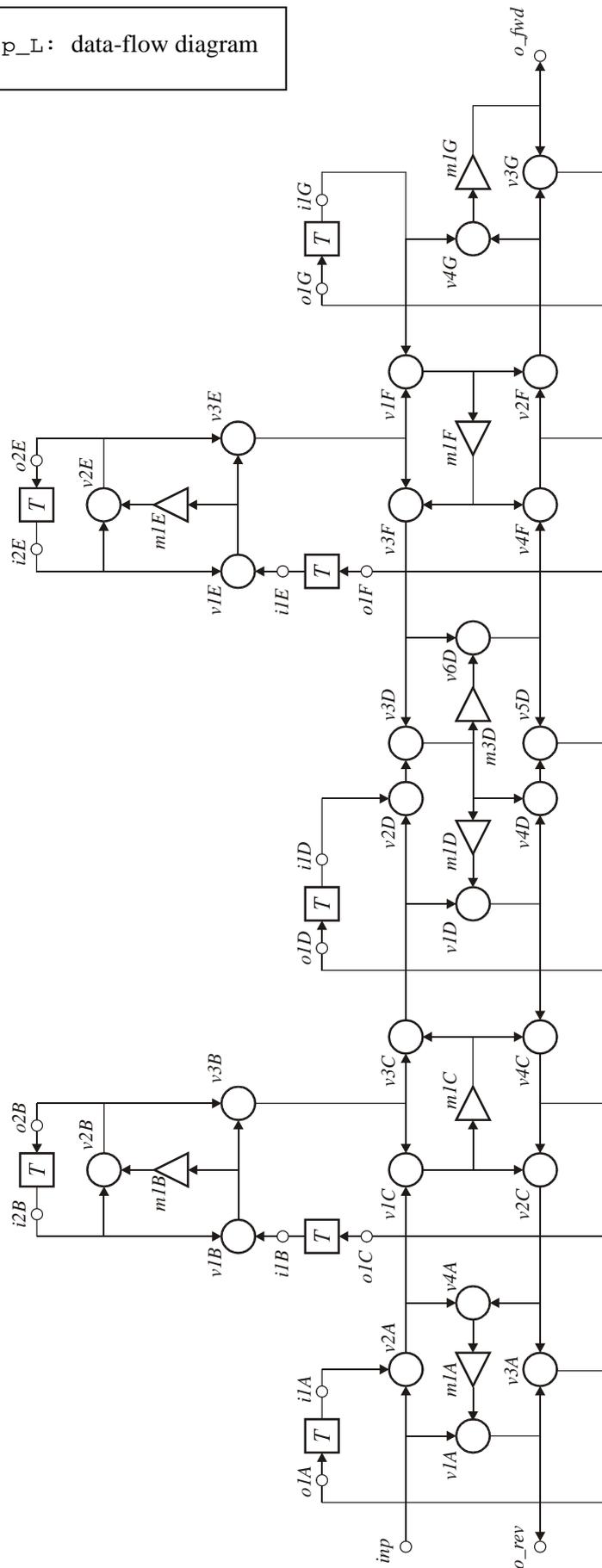
o_fwd = v6G;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_p\_3p.cir

WDF\_C5\_s\_sym\_2p\_L: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:36:43.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "-" in top row
o1A = v3A;
i1A = To1A;
% bottom row 3p adaptor, type s1m_ppp_pii
v1A = inp - m1A;
v2A = inp + i1A;
v3A = v1A + v2C;
v4A = v2A + v2C;
m1A = a1A * v4A;

o_rev = v1A;

% top row 2p adaptor, serial resonator
o1C = v4C;
i1B = To1C;
v1B = i2B - i1B;
m1B = a1B * v1B;
v2B = i2B - m1B;
o2B = v2B;
i2B = To2B;
v3B = v1B + v2B;
% bottom row 3p adaptor, type plm_ipp_ppp
v1C = v2A + v3B;
v2C = v1C + v4C;
m1C = a1C * v1C;
v3C = v3B - m1C;
v4C = v1D - m1C;

% "-" in top row
o1G = v3G;
i1G = To1G;
% last bottom row 3p adaptor with A1=0, type s1m_0pp_iii
v4G = i1G + v2F;
m1G = a1G * v4G;
v3G = v2F - m1G;

o_fwd = m1G;

% top row 2p adaptor, serial resonator
o1F = v4F;
i1E = To1F;
v1E = i2E - i1E;
m1E = a1E * v1E;
v2E = i2E - m1E;
o2E = v2E;
i2E = To2E;
v3E = v1E + v2E;
% bottom row 3p adaptor, type plm_ipp_ppp
v1F = i1G + v3E;
v2F = v1F + v4F;
m1F = a1F * v1F;
v3F = v3E - m1F;
v4F = v6D - m1F;

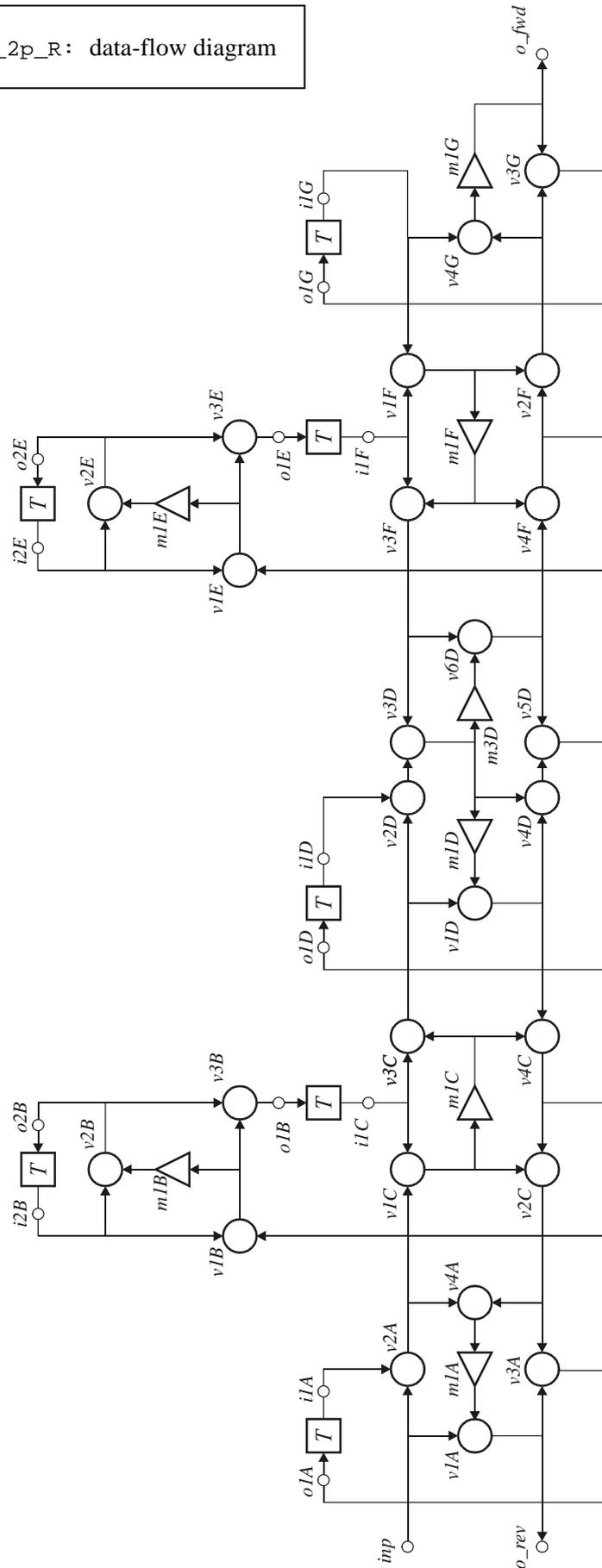
% "-" in top row
o1D = v5D;
i1D = To1D;
% middle bottom row 3p adaptor, type S2m_ppp_pip
v1D = v3C - m1D;
v2D = i1D + v3C;
v3D = v2D + v3F;
v4D = v1D + v3D;
v5D = v4D + v6D;
v6D = v3F - m3D;
m1D = a1D * v3D;
m3D = a3D * v3D;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_s\_sym\_2p\_L.cir

WDF\_C5\_s\_sym\_2p\_R: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:37:02.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "-" in top row
o1A = v3A;
i1A = To1A;
% bottom row 3p adaptor, type s1m_ppp_pii
v1A = inp - m1A;
v2A = inp + i1A;
v3A = v1A + v2C;
v4A = v2A + v2C;
m1A = a1A * v4A;

o_rev = v1A;

% top row 2p adaptor, serial resonator
v1B = i2B - v4C;
m1B = a1B * v1B;
v2B = i2B - m1B;
o2B = v2B;
i2B = To2B;
v3B = v1B + v2B;
o1B = v3B;
i1C = To1B;
% bottom row 3p adaptor, type plm_ipp_ppp
v1C = v2A + i1C;
v2C = v1C + v4C;
m1C = a1C * v1C;
v3C = i1C - m1C;
v4C = v1D - m1C;

% "-" in top row
o1G = v3G;
i1G = To1G;
% last bottom row 3p adaptor with A1=0, type s1m_0pp_iii
v4G = i1G + v2F;
m1G = a1G * v4G;
v3G = v2F - m1G;

o_fwd = m1G;

% top row 2p adaptor, serial resonator
v1E = i2E - v4F;
m1E = a1E * v1E;
v2E = i2E - m1E;
o2E = v2E;
i2E = To2E;
v3E = v1E + v2E;
o1E = v3E;
i1F = To1E;
% bottom row 3p adaptor, type plm_ipp_ppp
v1F = i1G + i1F;
v2F = v1F + v4F;
m1F = a1F * v1F;
v3F = i1F - m1F;
v4F = v6D - m1F;

% "-" in top row
o1D = v5D;
i1D = To1D;
% middle bottom row 3p adaptor, type S2m_ppp_pip
v1D = v3C - m1D;
v2D = i1D + v3C;
v3D = v2D + v3F;
v4D = v1D + v3D;
v5D = v4D + v6D;
v6D = v3F - m3D;
m1D = a1D * v3D;
m3D = a3D * v3D;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_s\_sym\_2p\_R.cir



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:36:13.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "-" in top row
o1A = v3A;
i1A = To1A;
% bottom row 3p adaptor, type s1m_ppp_pii
v1A = inp - m1A;
v2A = inp + i1A;
v3A = v1A + v2C;
v4A = v2A + v2C;
m1A = a1A * v4A;

o_rev = v1A;

% top row 3p adaptor, reflection free, with T and -T
v1B = i1B - m1B;
v2B = i1B + i2B;
v3B = v1B - v4C;
v4B = v2B - v4C;
m1B = a1B * v4B;
o1B = v1B;
o2B = v3B;
i1B = To1B;
i2B = To2B;
% bottom row 3p adaptor, type plm_ipp_ppp
v1C = v2A + v2B;
v2C = v1C + v4C;
m1C = a1C * v1C;
v3C = v2B - m1C;
v4C = v1D - m1C;

% "-" in top row
o1G = v3G;
i1G = To1G;
% last bottom row 3p adaptor with A1=0, type s1m_0pp_iii
v4G = i1G + v2F;
m1G = a1G * v4G;
v3G = v2F - m1G;

o_fwd = m1G;

% top row 3p adaptor, reflection free, with T and -T
v1E = i1E - m1E;
v2E = i1E + i2E;
v3E = v1E - v4F;
v4E = v2E - v4F;
m1E = a1E * v4E;
o1E = v1E;
o2E = v3E;
i1E = To1E;
i2E = To2E;
% bottom row 3p adaptor, type plm_ipp_ppp
v1F = i1G + v2E;
v2F = v1F + v4F;
m1F = a1F * v1F;
v3F = v2E - m1F;
v4F = v6D - m1F;

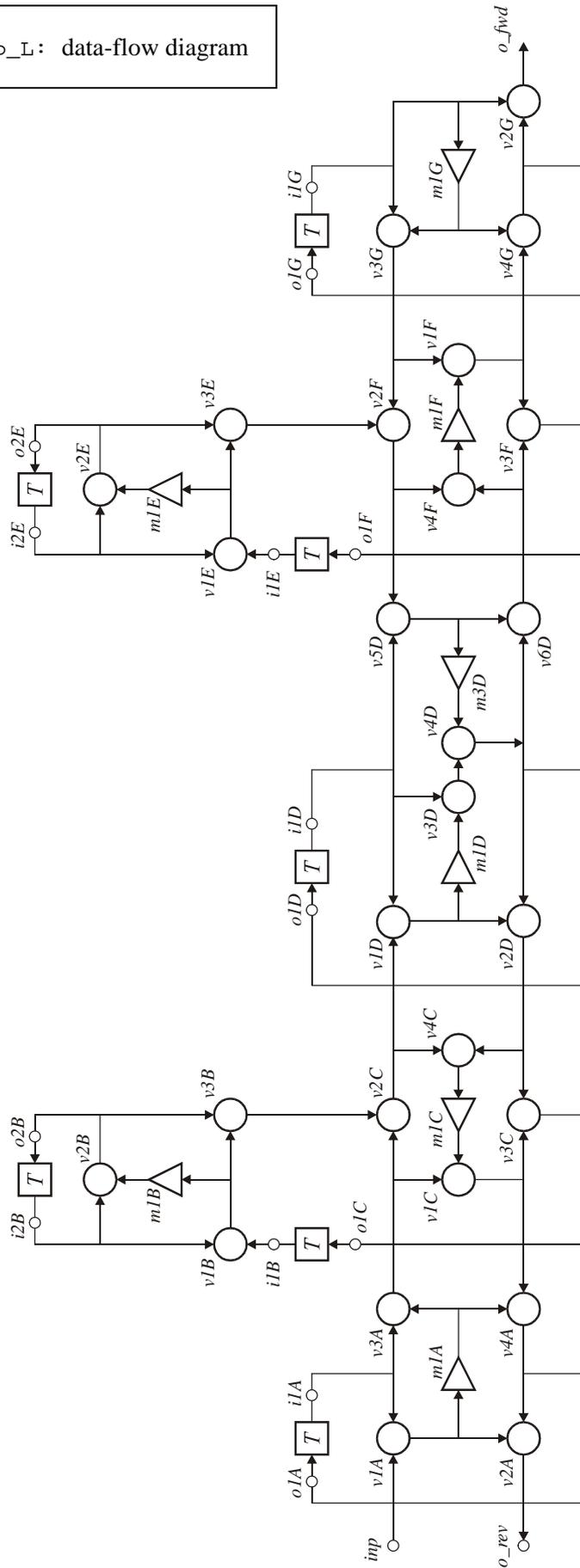
% "-" in top row
o1D = v5D;
i1D = To1D;
% middle bottom row 3p adaptor, type S2m_ppp_pip
v1D = v3C - m1D;
v2D = i1D + v3C;
v3D = v2D + v3F;
v4D = v1D + v3D;
v5D = v4D + v6D;
v6D = v3F - m3D;
m1D = a1D * v3D;
m3D = a3D * v3D;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_s\_sym\_3p.cir

WDF\_C5\_p\_sym\_2p\_L: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:37:48.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "+" in top row
o1A = v4A;
i1A = To1A;
% bottom row 3p adaptor, type plm_ppp_ipp
v1A = inp - i1A;
v2A = v1A - v4A;
m1A = a1A * v1A;
v3A = i1A + m1A;
v4A = v1C + m1A;

o_rev = v2A;

% top row 2p adaptor, parallel resonator
o1C = v3C;
i1B = To1C;
v1B = i2B - i1B;
m1B = a1B * v1B;
v2B = m1B - i2B;
o2B = v2B;
i2B = To2B;
v3B = v1B - v2B;
% bottom row 3p adaptor, type slm_pip_pii
v1C = v3A - m1C;
v2C = v3A - v3B;
v3C = v1C + v2D;
v4C = v2C + v2D;
m1C = a1C * v4C;

% "+" in top row
o1G = v4G;
i1G = To1G;
% last bottom row 3p adaptor with A1=0, type plm_0pp_ppp
m1G = a1G * i1G;
v3G = i1G - m1G;
v4G = v1F - m1G;
v2G = i1G + v4G;

o_fwd = v2G;

% top row 2p adaptor, parallel resonator
o1F = v3F;
i1E = To1F;
v1E = i2E - i1E;
m1E = a1E * v1E;
v2E = m1E - i2E;
o2E = v2E;
i2E = To2E;
v3E = v1E - v2E;
% bottom row 3p adaptor, type slm_pip_pii
v1F = v3G - m1F;
v2F = v3G - v3E;
v3F = v1F + v6D;
v4F = v2F + v6D;
m1F = a1F * v4F;

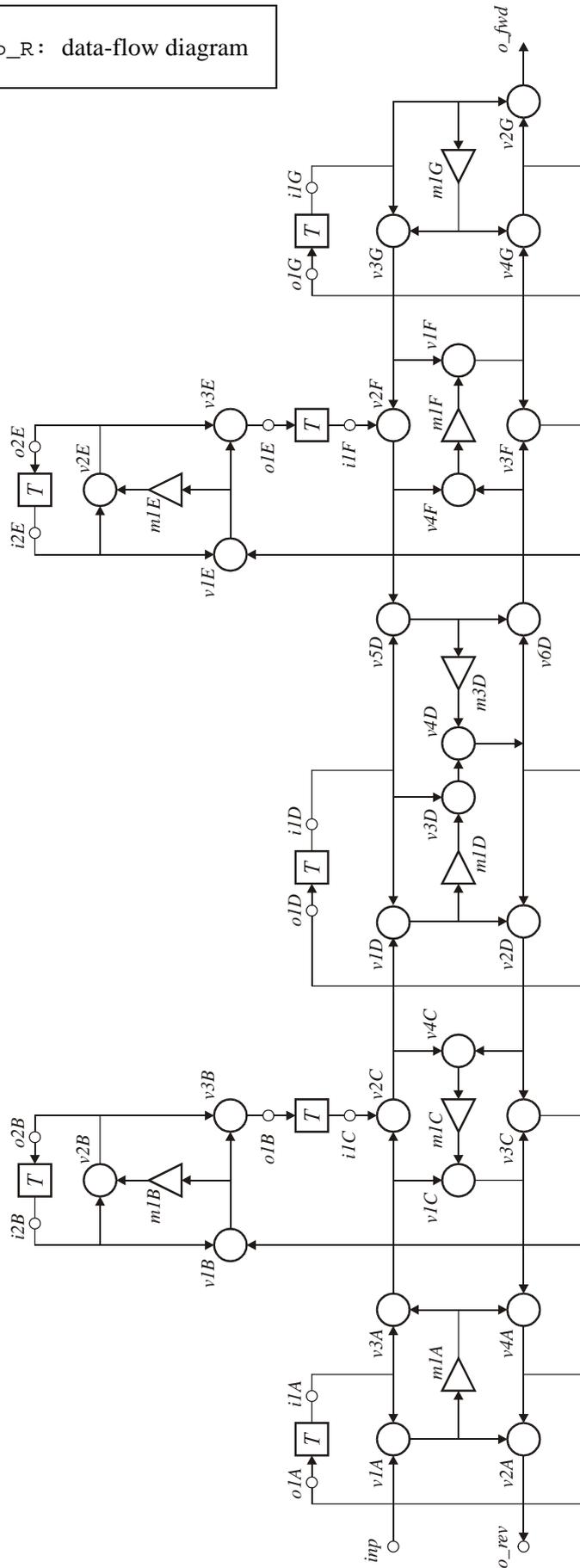
% "+" in top row
o1D = v4D;
i1D = To1D;
% middle bottom row 3p adaptor, type P2m_ipi_ppp
v1D = v2C + i1D;
v2D = v1D + v4D;
v3D = i1D - m1D;
v4D = v3D - m3D;
v5D = i1D + v2F;
v6D = v4D + v5D;
m1D = a1D * v1D;
m3D = a3D * v5D;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_p\_sym\_2p\_L.cir

WDF\_C5\_p\_sym\_2p\_R: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:37:58.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "+" in top row
o1A = v4A;
i1A = To1A;
% bottom row 3p adaptor, type plm_ppp_ipp
v1A = inp - i1A;
v2A = v1A - v4A;
m1A = a1A * v1A;
v3A = i1A + m1A;
v4A = v1C + m1A;

o_rev = v2A;

% top row 2p adaptor, parallel resonator
v1B = i2B - v3C;
m1B = a1B * v1B;
v2B = m1B - i2B;
o2B = v2B;
i2B = To2B;
v3B = v1B - v2B;
o1B = v3B;
i1C = To1B;
% bottom row 3p adaptor, type slm_pip_pii
v1C = v3A - m1C;
v2C = v3A - i1C;
v3C = v1C + v2D;
v4C = v2C + v2D;
m1C = a1C * v4C;

% "+" in top row
o1G = v4G;
i1G = To1G;
% last bottom row 3p adaptor with A1=0, type plm_0pp_ppp
m1G = a1G * i1G;
v3G = i1G - m1G;
v4G = v1F - m1G;
v2G = i1G + v4G;

o_fwd = v2G;

% top row 2p adaptor, parallel resonator
v1E = i2E - v3F;
m1E = a1E * v1E;
v2E = m1E - i2E;
o2E = v2E;
i2E = To2E;
v3E = v1E - v2E;
o1E = v3E;
i1F = To1E;
% bottom row 3p adaptor, type slm_pip_pii
v1F = v3G - m1F;
v2F = v3G - i1F;
v3F = v1F + v6D;
v4F = v2F + v6D;
m1F = a1F * v4F;

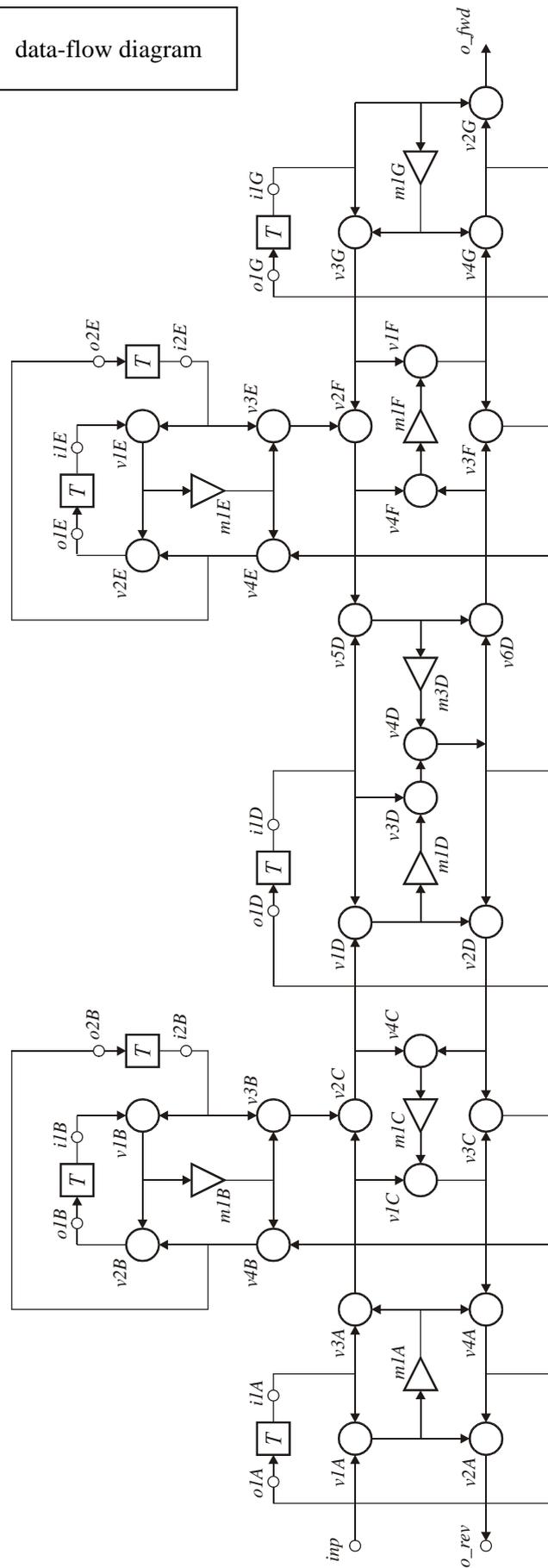
% "+" in top row
o1D = v4D;
i1D = To1D;
% middle bottom row 3p adaptor, type P2m_ipi_ppp
v1D = v2C + i1D;
v2D = v1D + v4D;
v3D = i1D - m1D;
v4D = v3D - m3D;
v5D = i1D + v2F;
v6D = v4D + v5D;
m1D = a1D * v1D;
m3D = a3D * v5D;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_p\_sym\_2p\_R.cir

WDF\_C5\_p\_sym\_3p: data-flow diagram



```

% Wave Digital Filter structure.
% Code generated by WDF2cir.m on 13-Nov-2006 11:37:14.

% external input 'inp', external outputs 'o_fwd' and 'o_rev'

% "+" in top row
o1A = v4A;
i1A = To1A;
% bottom row 3p adaptor, type plm_ppp_ipp
v1A = inp - i1A;
v2A = v1A - v4A;
m1A = a1A * v1A;
v3A = i1A + m1A;
v4A = v1C + m1A;

o_rev = v2A;

% top row 3p adaptor, reflection free, with T and -T
v1B = i1B + i2B;
v2B = v4B - v1B;
v3B = i2B - m1B;
v4B = m1B - v3C;
m1B = a1B * v1B;
o1B = v2B;
o2B = v4B;
i1B = To1B;
i2B = To2B;
% bottom row 3p adaptor, type slm_pip_pii
v1C = v3A - m1C;
v2C = v3A - v3B;
v3C = v1C + v2D;
v4C = v2C + v2D;
m1C = a1C * v4C;

% "+" in top row
o1G = v4G;
i1G = To1G;
% last bottom row 3p adaptor with A1=0, type plm_0pp_ppp
m1G = a1G * i1G;
v3G = i1G - m1G;
v4G = v1F - m1G;
v2G = i1G + v4G;

o_fwd = v2G;

% top row 3p adaptor, reflection free, with T and -T
v1E = i1E + i2E;
v2E = v4E - v1E;
v3E = i2E - m1E;
v4E = m1E - v3F;
m1E = a1E * v1E;
o1E = v2E;
o2E = v4E;
i1E = To1E;
i2E = To2E;
% bottom row 3p adaptor, type slm_pip_pii
v1F = v3G - m1F;
v2F = v3G - v3E;
v3F = v1F + v6D;
v4F = v2F + v6D;
m1F = a1F * v4F;

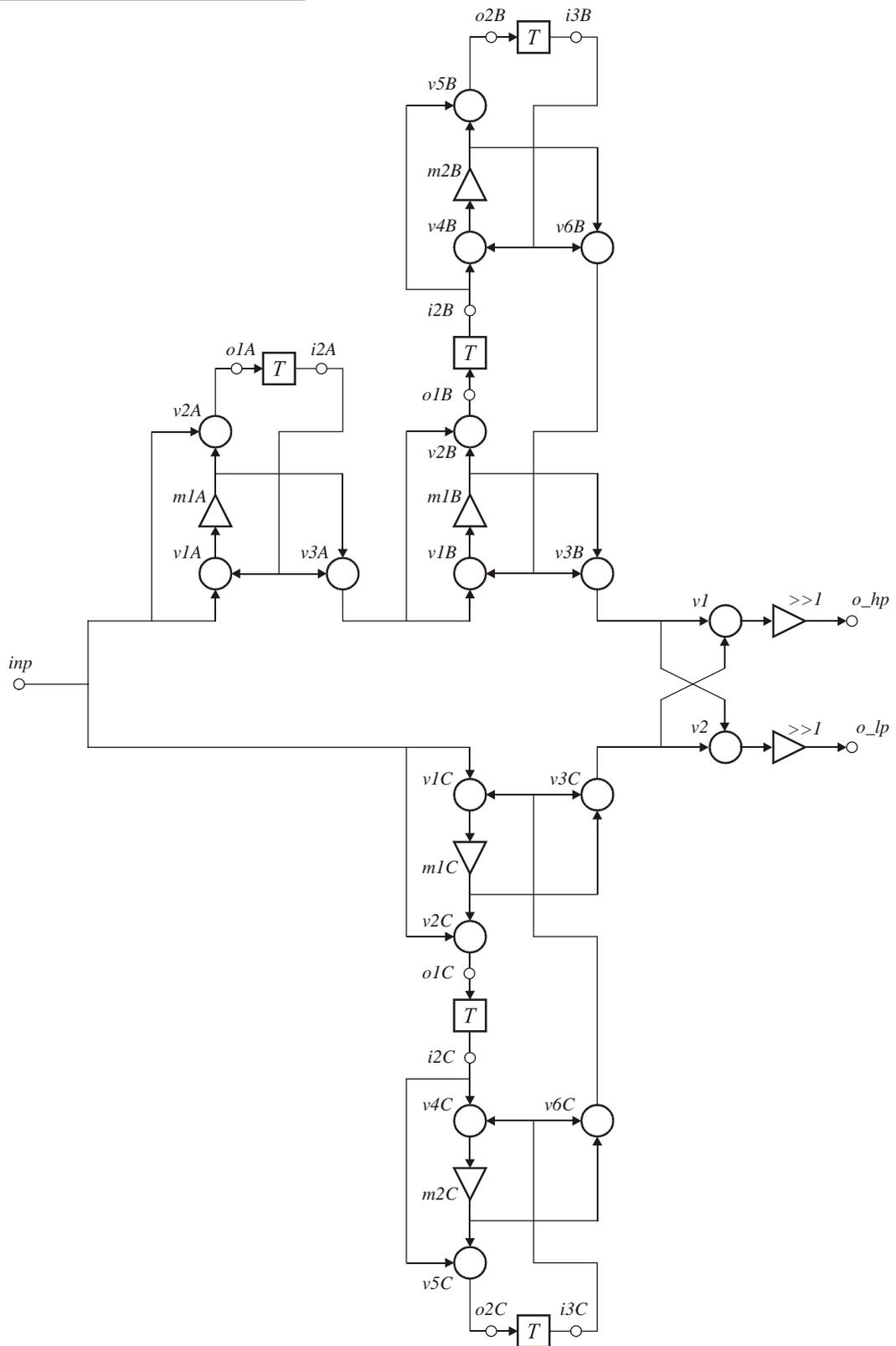
% "+" in top row
o1D = v4D;
i1D = To1D;
% middle bottom row 3p adaptor, type P2m_ipi_ppp
v1D = v2C + i1D;
v2D = v1D + v4D;
v3D = i1D - m1D;
v4D = v3D - m3D;
v5D = i1D + v2F;
v6D = v4D + v5D;
m1D = a1D * v1D;
m3D = a3D * v5D;

% (c) HJLA, 2005
% [EOF]

```

WDF\_C5\_p\_sym\_3p.cir

LWDF\_5\_L: data-flow diagram



```

% lowpass/highpass filter of Order 5
% as a Lattice Wave Digital Filter structure.
% Code generated by LWDF2cir.m on 17-Nov-2006 11:51:49.

% external input 'inp', external outputs 'o_hp' and 'o_lp'

% TOP ROW ALL-PASS SECTION(S):
% single section, single delay:
v1A = i2A - inp;
m1A = a1A * v1A;
v2A = inp + m1A;
o1A = v2A;
i2A = To1A;
v3A = i2A + m1A;

% 2nd degree section, 2 delay elements:
v1B = v6B - v3A;
m1B = a1B * v1B;
v2B = v3A + m1B;
o1B = v2B;
i2B = To1B;
v3B = v6B + m1B;
v4B = i3B - i2B;
m2B = a2B * v4B;
v5B = i2B + m2B;
o2B = v5B;
i3B = To2B;
v6B = i3B + m2B;

% BOTTOM ROW ALL-PASS SECTION(S):
% 2nd degree section, 2 delay elements:
v1C = v6C - inp;
m1C = a1C * v1C;
v2C = inp + m1C;
o1C = v2C;
i2C = To1C;
v3C = v6C + m1C;
v4C = i3C - i2C;
m2C = a2C * v4C;
v5C = i2C + m2C;
o2C = v5C;
i3C = To2C;
v6C = i3C + m2C;

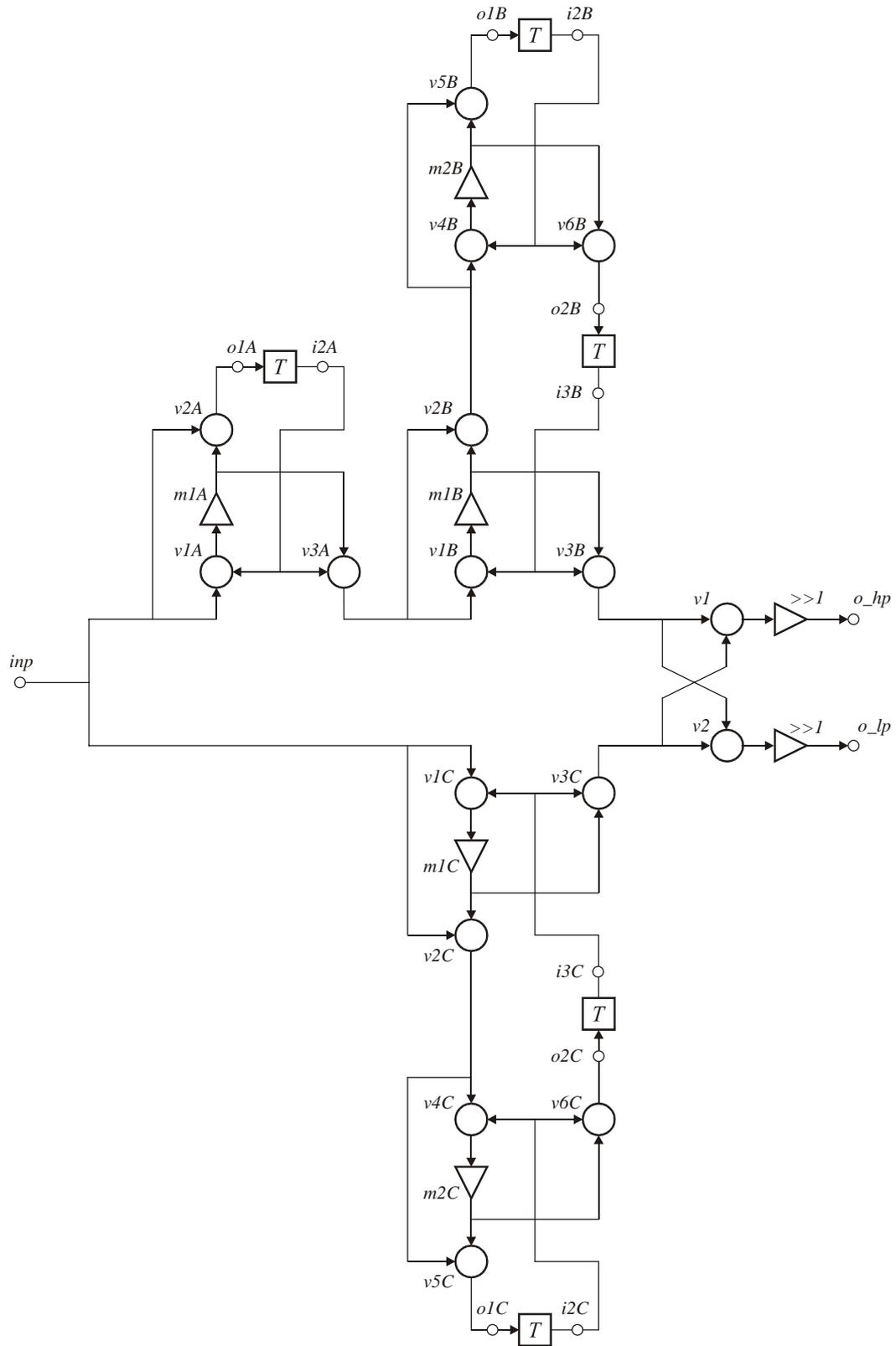
% OUTPUT SECTION:
v1 = v3C - v3B;
o_hp = (v1 >> 1); % v1/2
v2 = v3C + v3B;
o_lp = (v2 >> 1); % v2/2

% (c) HJLA, 2005
% [EOF]

```

LWDF\_5\_L.cir

LWDF\_5\_R: data-flow diagram



```

% lowpass/highpass filter of Order 5
% as a Lattice Wave Digital Filter structure.
% Code generated by LWDF2cir.m on 13-Nov-2006 11:38:26.

% external input 'inp', external outputs 'o_hp' and 'o_lp'

% TOP ROW ALL-PASS SECTION(S):
% single section, single delay:
v1A = i2A - inp;
m1A = a1A * v1A;
v2A = inp + m1A;
o1A = v2A;
i2A = To1A;
v3A = i2A + m1A;

% 2nd degree section, 2 delay elements:
v1B = i3B - v3A;
m1B = a1B * v1B;
v2B = v3A + m1B;
v4B = i2B - v2B;
m2B = a2B * v4B;
v5B = v2B + m2B;
o1B = v5B;
i2B = To1B;
v6B = i2B + m2B;
o2B = v6B;
i3B = To2B;
v3B = m1B + i3B;

% BOTTOM ROW ALL-PASS SECTION(S):
% 2nd degree section, 2 delay elements:
v1C = i3C - inp;
m1C = a1C * v1C;
v2C = inp + m1C;
v4C = i2C - v2C;
m2C = a2C * v4C;
v5C = v2C + m2C;
o1C = v5C;
i2C = To1C;
v6C = i2C + m2C;
o2C = v6C;
i3C = To2C;
v3C = m1C + i3C;

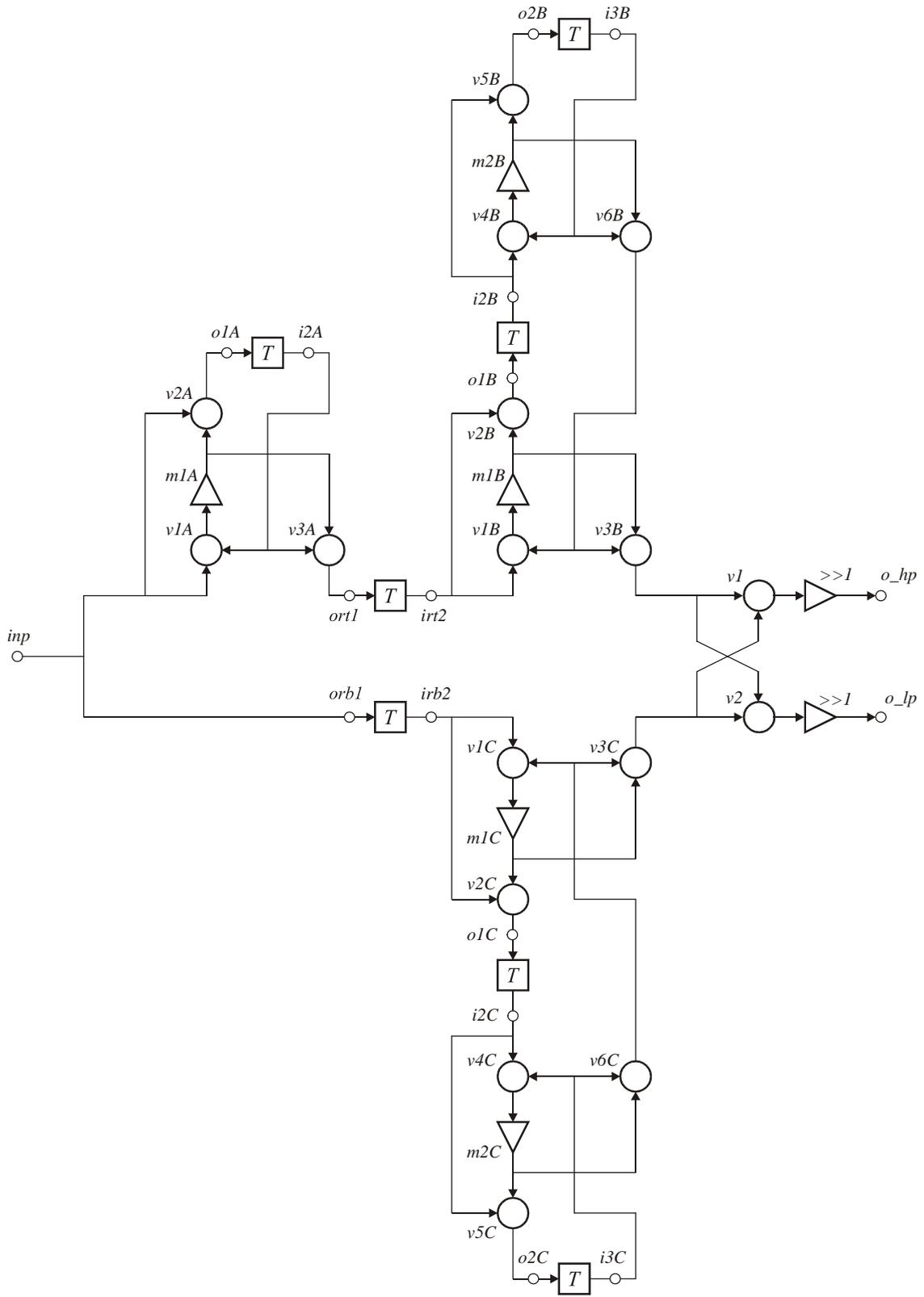
% OUTPUT SECTION:
v1 = v3C - v3B;
o_hp = (v1 >> 1); % v1/2
v2 = v3C + v3B;
o_lp = (v2 >> 1); % v2/2

% (c) HJLA, 2005
% [EOF]

```

LWDF\_5\_R.cir

LWDF\_5\_T\_L: data-flow diagram



```

% lowpass/highpass filter of Order 5
% as a Lattice Wave Digital Filter structure.
% Code generated by LWDF2cir.m on 13-Nov-2006 11:38:44.

% external input 'inp', external outputs 'o_hp' and 'o_lp'

% TOP ROW ALL-PASS SECTION(S):
% single section, single delay:
v1A = i2A - inp;
m1A = a1A * v1A;
v2A = inp + m1A;
o1A = v2A;
i2A = To1A;
v3A = i2A + m1A;

% inserted pipeline register:
ort1 = v3A;
irt2 = Tort1;

% 2nd degree section, 2 delay elements:
v1B = v6B - irt2;
m1B = a1B * v1B;
v2B = irt2 + m1B;
o1B = v2B;
i2B = To1B;
v3B = v6B + m1B;
v4B = i3B - i2B;
m2B = a2B * v4B;
v5B = i2B + m2B;
o2B = v5B;
i3B = To2B;
v6B = i3B + m2B;

% BOTTOM ROW ALL-PASS SECTION(S):
% inserted pipeline register:
orb1 = inp;
irb2 = Torb1;

% 2nd degree section, 2 delay elements:
v1C = v6C - irb2;
m1C = a1C * v1C;
v2C = irb2 + m1C;
o1C = v2C;
i2C = To1C;
v3C = v6C + m1C;
v4C = i3C - i2C;
m2C = a2C * v4C;
v5C = i2C + m2C;
o2C = v5C;
i3C = To2C;
v6C = i3C + m2C;

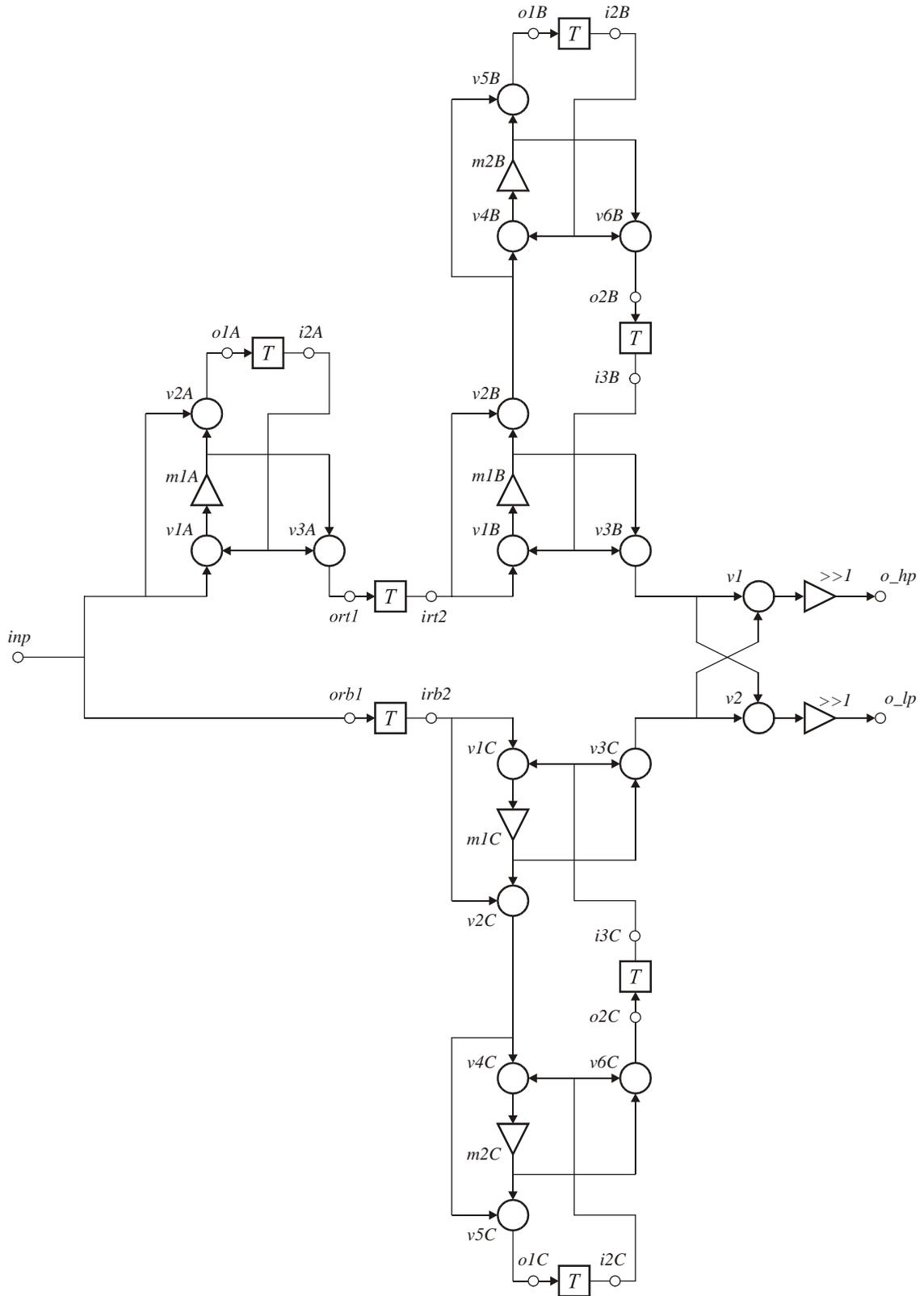
% OUTPUT SECTION:
v1 = v3C - v3B;
o_hp = (v1 >> 1); % v1/2
v2 = v3C + v3B;
o_lp = (v2 >> 1); % v2/2

% (c) HJLA, 2005
% [EOF]

```

LWDF\_5\_T\_L.cir

LWDF\_5\_T\_R: data-flow diagram



```

% lowpass/highpass filter of Order 5
% as a Lattice Wave Digital Filter structure.
% Code generated by LWDF2cir.m on 13-Nov-2006 11:38:53.

% external input 'inp', external outputs 'o_hp' and 'o_lp'

% TOP ROW ALL-PASS SECTION(S):
% single section, single delay:
v1A = i2A - inp;
m1A = a1A * v1A;
v2A = inp + m1A;
o1A = v2A;
i2A = To1A;
v3A = i2A + m1A;

% inserted pipeline register:
ort1 = v3A;
irt2 = Tort1;

% 2nd degree section, 2 delay elements:
v1B = i3B - irt2;
m1B = a1B * v1B;
v2B = irt2 + m1B;
v4B = i2B - v2B;
m2B = a2B * v4B;
v5B = v2B + m2B;
o1B = v5B;
i2B = To1B;
v6B = i2B + m2B;
o2B = v6B;
i3B = To2B;
v3B = m1B + i3B;

% BOTTOM ROW ALL-PASS SECTION(S):
% inserted pipeline register:
orb1 = inp;
irb2 = Torb1;

% 2nd degree section, 2 delay elements:
v1C = i3C - irb2;
m1C = a1C * v1C;
v2C = irb2 + m1C;
v4C = i2C - v2C;
m2C = a2C * v4C;
v5C = v2C + m2C;
o1C = v5C;
i2C = To1C;
v6C = i2C + m2C;
o2C = v6C;
i3C = To2C;
v3C = m1C + i3C;

% OUTPUT SECTION:
v1 = v3C - v3B;
o_hp = (v1 >> 1); % v1/2
v2 = v3C + v3B;
o_lp = (v2 >> 1); % v2/2

% (c) HJLA, 2005
% [EOF]

```

LWDF\_5\_T\_R.cir

This page intentionally left blank