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

Scheduling Toolbox for MATLAB

User's Guide

Version 1.0

Ing. H.J. Lincklaen Arriëns
March 2006

Scheduling Toolbox for MATLAB *User's Guide*

Table of Contents

This page intentionally left blank

Introduction.

This toolbox has been primarily developed to support the lectures on "Methods and algorithms for system design (et4054)", which is a compulsory course for MSc students in Computer Engineering and a specialization course for Electrical Engineering MSc students at the Delft University of Technology. Course material is based on the book *Synthesis and Optimization of Digital Circuits* by G. De Micheli.

With this toolbox it is now possible to create a synthesizable VHDL description for a (signal processing) circuit, or a mathematical function, that can be described in a *data-flow graph* (DFG). This DFG is internally reduced to a *sequence graph* (SG), and this one can be scheduled –i.e. the start time of each operation is determined according to a specific algorithm– to obtain a *scheduled sequence graph* (SSG).

At this moment the toolbox supports the following scheduling algorithms:

- As Soon As Possible Scheduling algorithm (ASAP),
- As Late As Possible Scheduling algorithm (ALAP),
- Forced-directed Scheduling (ForceD), and
- List Scheduling.

Resource sharing, register sharing and binding algorithms are then used to create both a MATLAB and a VHDL description of the SSG in a fixed-point data representation.
The MATLAB files can be used e.g. for fast testing the (fixed-point arithmetic) implementation in a larger environment or simply as a reference.
In the VHDL architectures which are focused on implementation on Xilinx FPGAs, registers and multiplexors are used for storing intermediate data values and connecting the available resources. This generally will result in more hardware then when dedicated RAM is used but avoids the time taken by read and write actions.

The setup implies that –apart from the 'data clock' that takes care for providing the input data and reading of the output data– a (high speed) 'SSG clock' that takes care for the internal states in the SSG should be available. If a scheduled implementation should need x SSG-clock 'states' (= cycles), the SSG-clock should have at least an x times higher frequency then the highest expected data rate. There is no need that the data clock is derived from the SSG clock (or vice versa) and they can be completely asynchronous, provided that there is some kind of handshaking mechanism. The only constraint is that the output data from the SSG should be stable when the next data-read clock edge arrives.

The DFG that has to be realized has to be written in a proprietary ".cir"-format (ASCII file) which is extensively described in the Reference Guide. In this User Guide we will use an example to explain the complete design trajectory. The example is based on a Lattice Wave Digital Filter implementation of a 5^{th} order Cauer low-pass filter that has been created with the aid of our (L)WDF Design Toolbox.
In Appendix B of this manual, the MATLAB script is given that can be used to verify the data in the following chapters.

Installation.

The Toolbox will be made available as a zip-file, that should be unzipped to a directory of your choice, thereby preserving the directory structure of the zip-file ('Use folder names').

Assuming that you have chosen to unzip to <$STBX_DIR>, you will find

all the toolbox' MATLAB-files in <$STBX_DIR>

and the new subdirectories

<$STBX_DIR>\bin , which contains an executable (dot.exe) and some dll-files,

<$STBX_DIR>\vhdl , which contains two .vhd-files, and

<$STBX_DIR>\example_cirs, which contains some .cir examples files.

Adjust the Properties of the MATLAB icon on your desktop, so that MATLAB starts in <$STBX_DIR>, or add <$STBX_DIR> to your MATLAB path.

If you had chosen for a customized directory structure, be sure to edit the file fConfig.m (default in <$STBX_DIR>), such that the string variables

dotDir contains the absolute path to the directory where dot.exe and the dll-files reside,

and

vhdDir contains the absolute path to the directory where the .vhd-files have been written.

The GUI can see the example .cir-files when started with

```
schedGUI( '<$STBX_DIR>\example_cirs' );
```

Note concerning dotDir:

For drawing of the graphs, the toolbox needs some executables from the **Graphviz** (Graph Visualization Software) package, developed by AT&T Research. Graphviz is now open source software that has been extended in several ways, and that is available from http://www.graphviz.org/.
The Graphviz layout programs take descriptions of graphs in a simple text language, and can make diagrams in several useful formats. The program dot.exe (on Windows) that we use, draws directed graphs in a.o. png, jpg or hpgl format and will be located in the dotDir directory.
From Release 14 on, MATLAB contains an own stripped down version of this dot program. To ensure that the correct full blown version is executed, we need the dotDir variable.

A 5th Order Cauer Filter.

As written before, we will introduce you to the capabilities of the toolbox by describing how to obtain the VHDL description of a filter example. We will start with some basic filter theory to clarify the circuit and the test signals that are used in the example.

Cauer or elliptic filters show an equiripple behavior of the magnitude transfer function, both in its pass-band as well as in its stop-band. For this example a discrete-time domain 5th order low-pass filter has been designed with only 0.1 dB ripple in the pass-band, 45 dB ripple in the stop-band and a -3dB cut-off frequency at 0.34 times the sample frequency.
This low-pass filter can be mathematically described with its transfer function in the frequency domain

$$H(z) = \frac{\sum_{i=0}^{5} b_i \, z^{-i}}{\sum_{j=0}^{5} a_j \, z^{-j}} \qquad \text{..... (1a)}$$

in which $z = e^{j\theta}$ and $\theta = 2\pi \frac{\text{frequency}}{\text{sample frequency}}$,

or alternatively with its output time function

$$y[n] = \sum_{i=0}^{5} b_i \cdot x[n-i] \; - \sum_{j=1}^{5} a_j \cdot y[n-j] \qquad \text{..... (1b)}$$

Table 1 shows the coefficient values for the design parameters mentioned above, while the resulting magnitude transfer function is given in Figure 1a. Remind that this characteristic is relative to the sample frequency that is chosen, i.e. for a sample frequency of 44.1 kHz the –3dB cut-off frequency point will be at about 15 kHz.

Table 1. *coefficienst for the 5th order Cauer filter of Figure 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

Instead of measuring the complete transfer function, it is often easier –at least for a first test– to characterize 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

$$\delta[n] = \begin{cases} 1, & \text{for } n = 0 \\ 0, & \text{otherwise.} \end{cases} \qquad \text{..... (2a)}$$

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

$$u[n] = \begin{cases} 0, & \text{for } n < 0 \\ 1, & \text{for } n \geq 0. \end{cases} \qquad \text{..... (2b)}$$

for this purpose.

Our filter, when subjected to a unit impulse function or a unit step function, will result in the output response values given in Table 2. For completeness, Figures 1b and c show these responses in graphical format.

Table 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

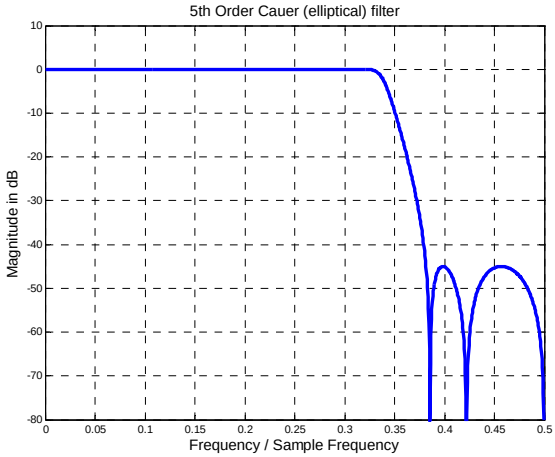


Figure 1a. *Magnitude transfer function of the 5th order Cauer filter.*

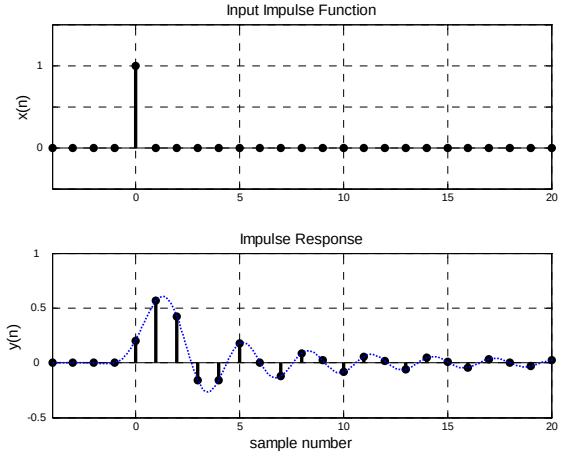


Figure 1b. *Impulse response of the 5th order Cauer filter.*

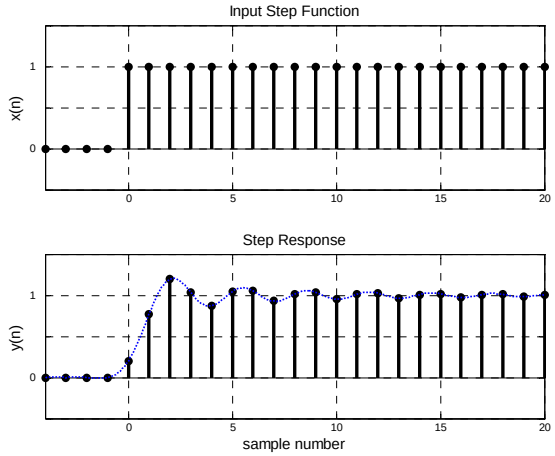


Figure 1c. *Step response of the 5th order Cauer filter.*

Lattice Wave Digital Filter implementation.

There are a lot of possible structures which can be used to realize the previously mentioned transfer functions with. Here, we will opt for an implementation as a Lattice Wave Digital Filter. In the User's Guide for the (Lattice) Wave Digital Toolbox this type of structures has been described in detail. Figure 2 shows a block diagram of the resulting structure (see the LWDF Toolbox's Reference Guide for what is inside the rectangular blocks, it will also be clarified later on).

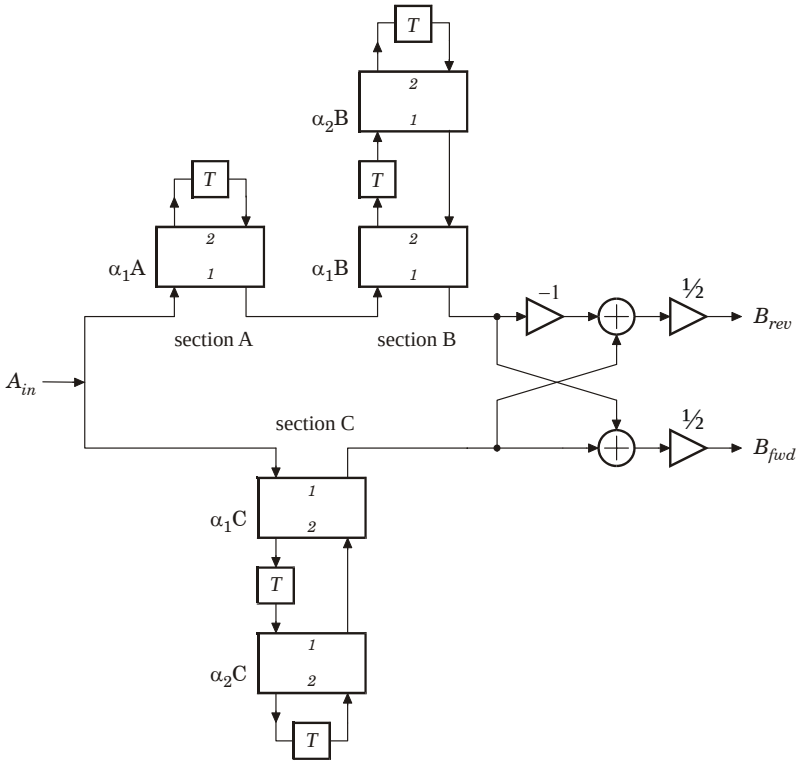


Figure 2. *Block diagram of a Lattice Wave Digital Filter that can be used to realize our 5th order Cauer functions.*

By nature, wave digital filters will show 2 outputs, here denoted with B_{fwd} and $B_{rev,}$ of which B_{fwd} will show the specified low-pass transfer function and B_{rev} a complementary high-pass function. Although we only defined the low-pass transfer function, for the sake of this example the high-pass function will be calculated as well.

Description of the filter in a .cir file.

We have to describe the structure of this LWDF filter in a way that it can be understood by the scheduling software, i.e. the circuit has to be translated into operations and calculations that can be implemented on real multipliers and adders/subtractors. The elements that can be used –functional block-elements as they are often used in signal processing literature– are depicted in Figure 3.

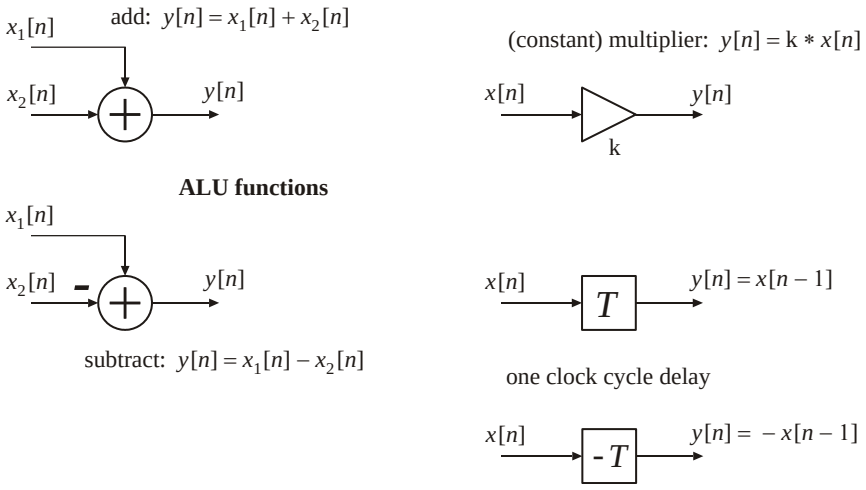


Figure 3. *The available basis elements for scheduling.*

The scheduling software expects 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'. An extensive description of the syntax, structure and the thoughts behind a .cir-file can be found in the Reference Guide

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 4 can help in recognizing the parts to be scheduled (the shaded area) and the input and output ports.

Now '*inp*' is the overall input port of the filter ($x[n]$) and '*o_lp*' ($y[n]$) and '*o_hp*' the output ports. Using the correct syntax and assignment rules, may result in the cir-file shown in Figure 5. This .cir-file, let's call it '`LWDF_5.cir`', will be the input for the scheduling software.

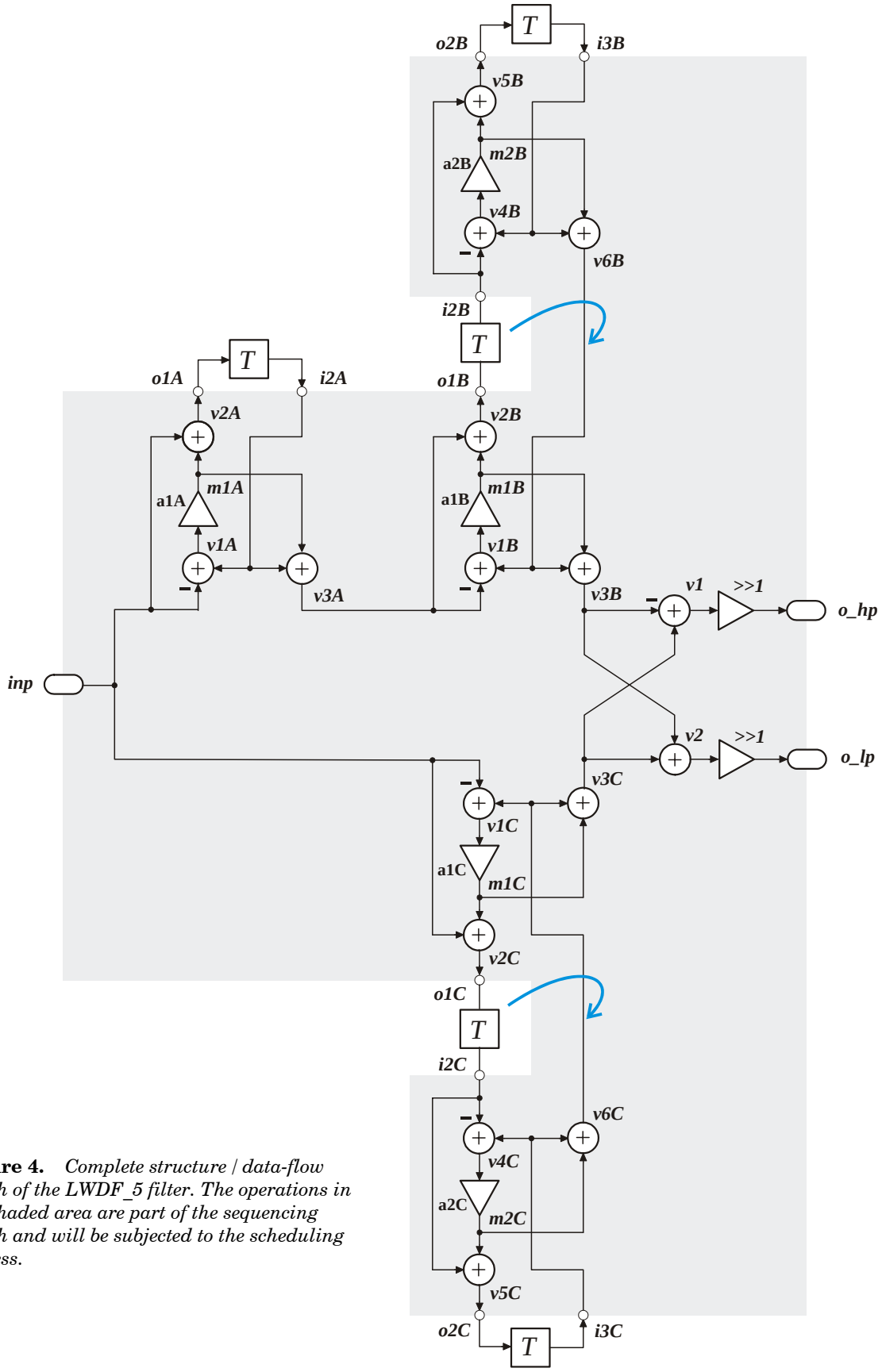


Figure 4. *Complete structure / data-flow graph of the LWDF_5 filter. The operations in the shaded area are part of the sequencing graph and will be subjected to the scheduling process.*

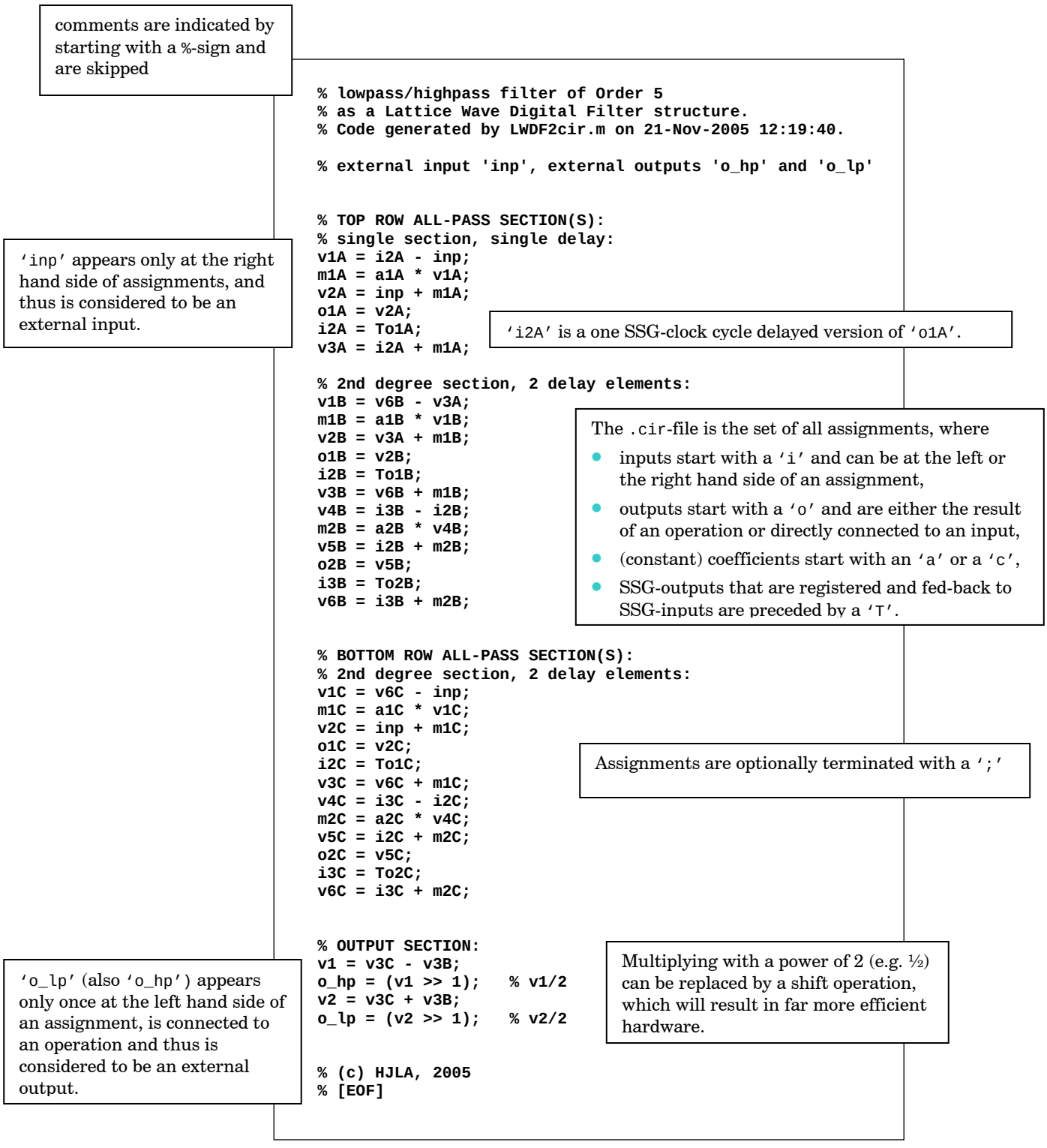


Figure 5. *Listing of the text file* `LWDF_5.cir` *that will be the input for the scheduling software.*

The Graphical User Interface (schedGUI).

The Toolbox consists of a set of MATLAB functions that can be used from the MATLAB command window. All functions are supplied with help comments, while a complete description of all functions can be found in the Reference Guide.
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 `<$OUR_CIR_DIR>`. **OR PATH and** `schedGUI('<$OUR_CIR_DIR>')`.

Figure 6 shows a screen shot of this schedGUI, with displayed the result when our LWDF_5 filter should be scheduled according to the 'List' method given the resource constraint that only 2 MULtipliers and 2 ALUs (which can perform both the add and the subtract calculations) are available.
In the Figure, the MULtipliers are supposed to perform with a latency of two SSG-clock cycles, while the ALUs will be ready in only one SSG-clock cycle.

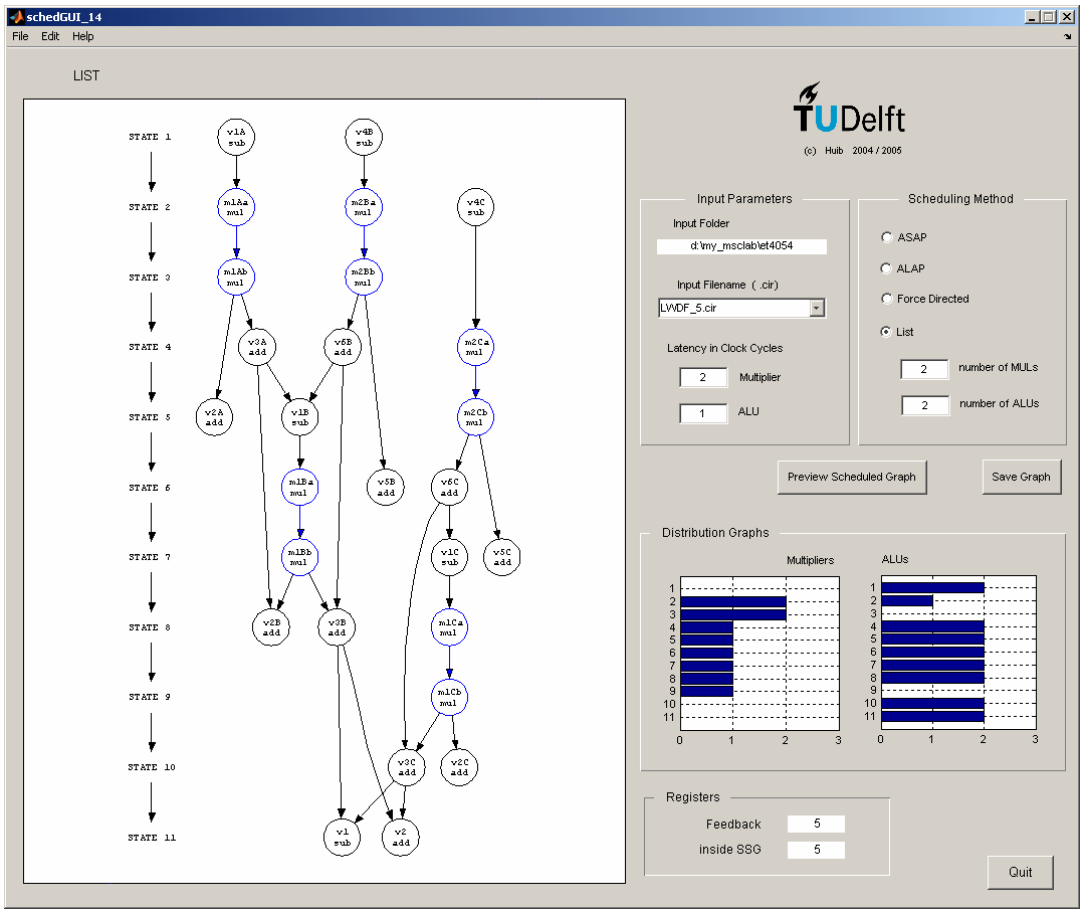


Figure 6. *Screen-shot of* `schedGUI` *working with the* `LWDF_5.cir` *file.*

It can be seen that the result of the calculations will be valid after 11 SSG-clock cycles. In the lower-right corner also the distribution of the resources over the clock cycles is shown, and the number of additional registers needed for life-time extension of intermediate variables. Obviously, the GUI expects that 6 additional REGisters will be needed here. It is important to note that this will only be the case if **registered MULtipliers** and **registered ALUs** are used, as have been described in the architectures in the `resource_reg.vhd` file in this toolbox.
In other words, `schedGUI` expects that the VHDL descriptions of the MULtiplier and ALU components already provide for registers immediately following the MULtiplier core and the ALU core, that can store the results of the computation done in the 'core' when they are clocked at the end of an SSG-cycle. Clock-enable signals control when and which computational results should be registered.

To get an impression of the resources (chip area) vs speed that result from the scheduling methods, the function `xplore` can be used. Figure 7 shows the design space for the LWDF-5 circuit for the arbitrary (and unrealistic) assumption that a MULtiplier would take up twice the area, and an ALU 1.5 times the area of a REGister. Also latencies of 1 clock-cycle are assumed for MULs and ALUs.

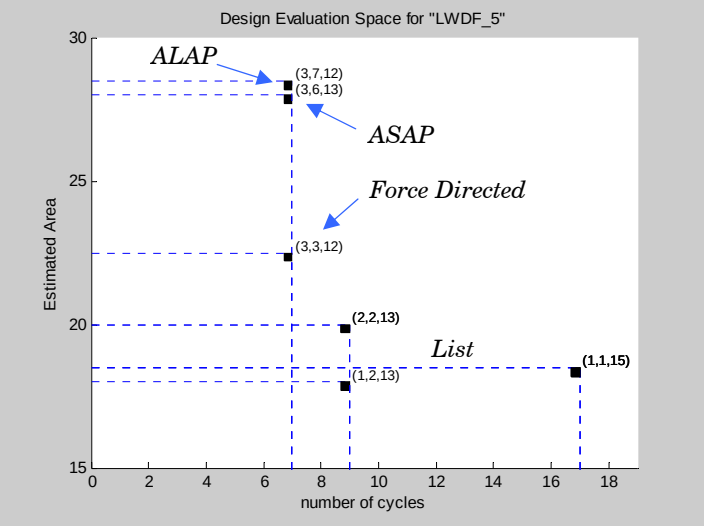
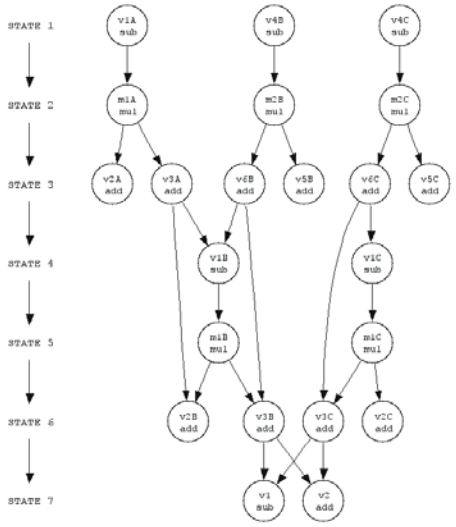


Figure 7. *The result of* `xplore('LWDF_5.cir', [2 1.5 1])`. *In the picture,* `(a,b,c)` *indicates the need for* `a` *MULtipliers,* `b` *ALUs and* `c` *REGisters, where* `c` *is the sum of all registers, i.e. including the feedback, an input and output registers.*

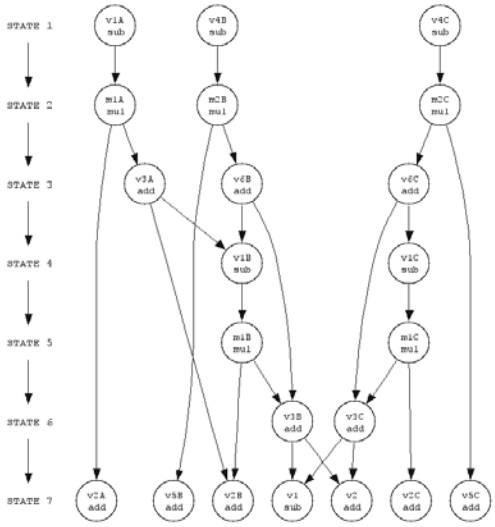
In Figure 8 four possible scheduling schemes are presented, while the one chosen for the implementation as a 2's complement fixed-point architecture is shown explicitly in Figure 9.
For this schedule we will first generate MATLAB files and a MATLAB testbench for easy debugging and enabling the incorporation of this circuit in a larger MATLAB test-environment if needed. Generally, debugging and determining the optimal sizes of the fixed-point parameters is much easier with MATLAB then in a VHDL environment.

ASAP



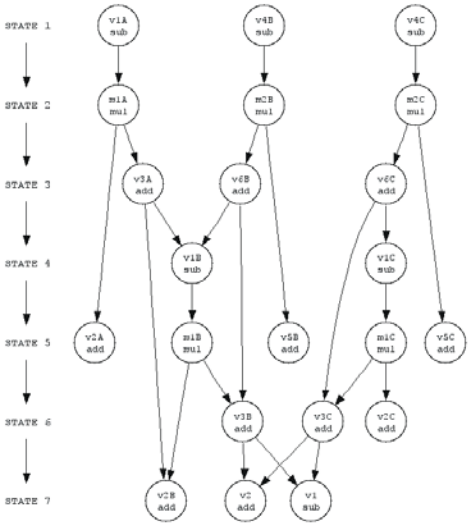
7 States, 3 MULs, 6 ALUs, 5 Additional REGs

ALAP



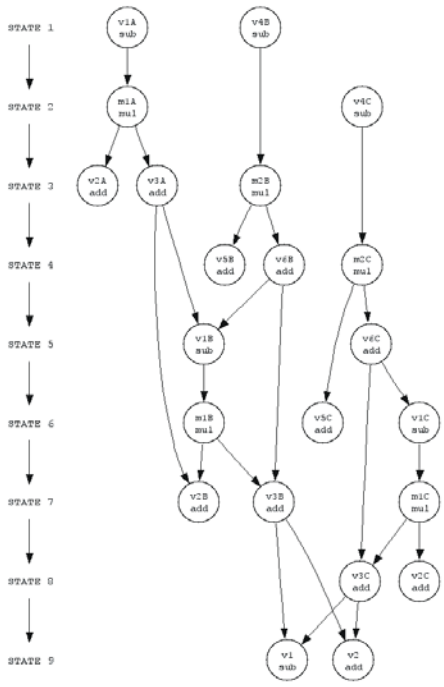
7 States, 3 MULs, 7 ALUs, 4 Additional REGs

Force Directed



7 States, 3 MULs, 3 ALUs, 4 Additional REGs

List 1, 2



9 States, 1 MUL, 2 ALUs, 5 Additional REGs

Figure 8. *Several possible scheduling schemes for LWDF_5.*

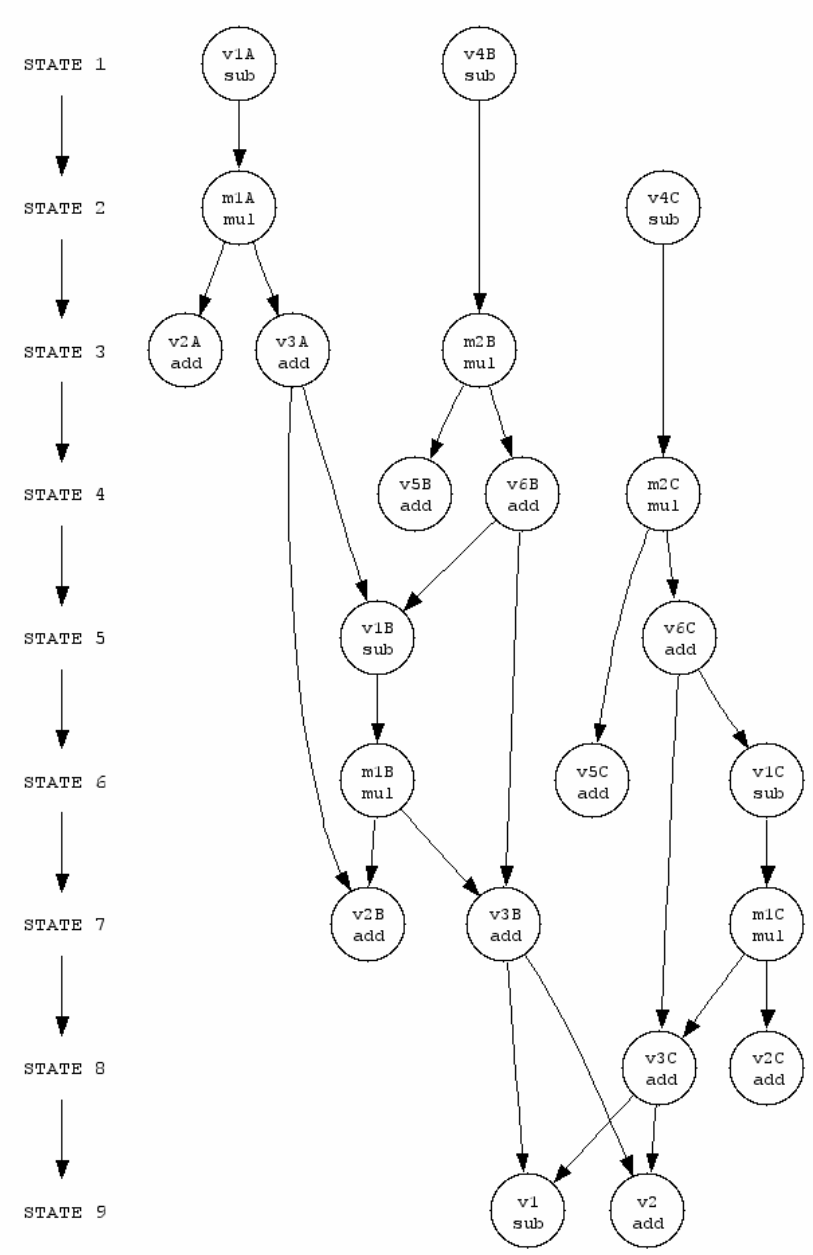


Figure 9. *The LWDF_5 schedule that will be implemented ('List' with 1 MULtiplier and 2 ALUs, all resources finished within one SSG-clock cycle).*

The MATLAB testbench.

Suppose that the .cir-file(s) are in a directory indicated with <$OUR_CIR_DIR> and that the toolbox files are either in the same directory or that they are incorporated in MATLABPATH.
Then the schedule from Figure 9 will be translated into a fixed-point description with

```
>> gen_mTB( 'LWDF_5.cir', 0, 'List', 1,1, 1,2 );
```

See the Reference Guide for an explanation of the syntax and all possibilities.
gen_mTB will create a sub-directory <$OUR_CIR_DIR>\LWDF_5\matlab in which it will write its m-files, viz. testbench_LWDF_5_auto.m and TB_LWDF_5_auto.m
(all names are derived from the .cir-file name).
Of these, TB_LWDF_5_auto.m simulates the SSG (of course, without a real SSG-clock).

The testbenches (both the MATLAB as the VHDL ones) read their input values from a .INP-file in which data should be written in VHDL hex-format on a one variable per line basis.
Notice that the sequence in which the constant coefficient should be listed in this .INP-file is reported by gen_mTB in the command window, viz. a1A, a1B, a1C, a2B, a2C.
In this example the coefficients are generated with the (L)WDF Design Toolbox, and are in the correct format, but not in the correct sequence, so a little reshuffling will be needed.

Quantizing the floating-point coefficients into their correct fixed-point counterparts can be done with the toolbox functions toSFixp and hex2fixp, or with gen_INP which is specifically meant to create .INP-files. Here, we will opt for a representation with 17 bits, of which 15 are reserved for the fraction part (see Figure 10 and the Reference Guide).

Let the coefficients be given as a MATLAB structure coeffs, with

```
coeffs =
    'a1A'    [-0.03119210537118]
    'a1B'    [-0.82935592147589]
    'a1C'    [-0.38195386087389]
    'a2B'    [-0.52095844720462]
    'a2C'    [-0.35687921966298]
```

then a .INP-file for a unit step input function $u[n-2]$ can be created with

```
>> gen_INP( 'LWDF_5\matlab \LWDF_5.INP', [17 15], coeffs, [0 0 ones(1,10)] );
>> cd LWDF_5\matlab
```

or alternatively, with

```
>> gen_INP( 'LWDF_5.INP', [17 15], coeffs, [0 0 ones(1,10)] );
>> addpath( '<$OUR_CIR_DIR>\LWDF_5\matlab' )
```

The difference will be the directory where the .INP and .OUT-files will be written
The user should find out for her/himself whether she/he prefers to switch between directories or to end up with a somewhat cluttered single directory.
In any case, the LWDF_5.INP file should look like Table 3.

Table 3. *The* `LWDF_5.INP` *file for our Cauer filter.*

```
-- [17 15] fixed-point format
-- coefficients: a1A,a1B,a1C,a2B,a2C
x"1FC02"
x"195D8"
x"1CF1C"
x"1BD51"
x"1D252"
-- input function
x"00000"
x"00000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
```

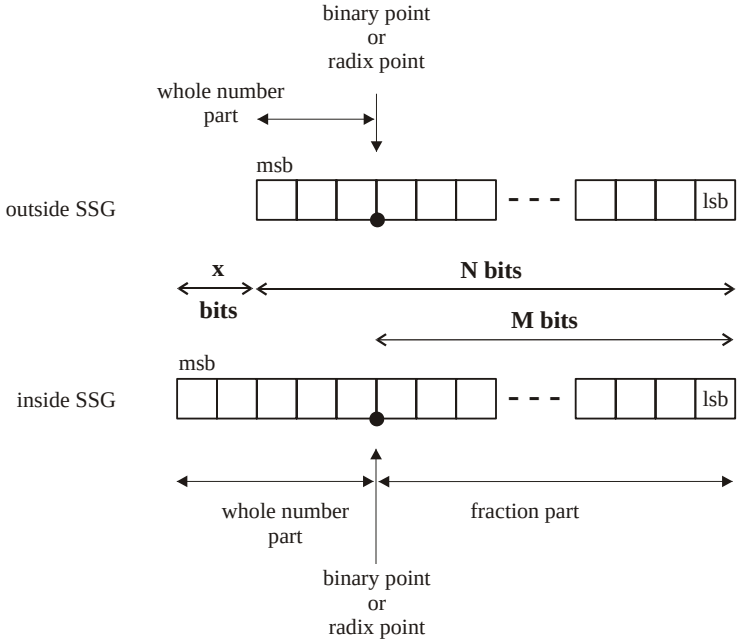


Figure 10. *Fixed-point bus width (signed 2's complement) can be (left) extended for the computations inside the SSG.*

The next thing is to determine whether the total number of bits (17) will be sufficient to represent all intermediate results (for this input signal only!). With cumulative additions or subtractions inside the SSG, it can certainly be expected that some of the intermediate results will grow larger than, or more negative than, the values that can be represented with the 2 integer bits that we defined.
We can expand the integer part of the fixed-point number by passing a 3rd number in the fixed-point parameter to the testbench file (see again Figure 10).

Let's start by not expanding any bus and see what happens:

```
>> testbench_LWDF_5_auto( [17 15 0], 0 );
Reading file 'LWDF_5.INP' ...
o_hp:  x"00000"    0.000000
o_lp:  x"00000"    0.000000
o_hp:  x"00000"    0.000000
o_lp:  x"00000"    0.000000
o_hp:  x"016CA"    0.178040
o_lp:  x"01A19"    0.203888
o_hp:  x"1F4F2"   -0.0863647
o_lp:  x"062F5"    0.773102
Warning: overflow ...
> In ALU at 91
  In TB_LWDF_5_auto>lfDFG at 38
  In TB_LWDF_5_auto at 9
  In testbench_LWDF_5_auto at 57
Warning: overflow ...
> In ALU at 91
  In TB_LWDF_5_auto>lfDFG at 62
  In TB_LWDF_5_auto at 9
  In testbench_LWDF_5_auto at 57
o_hp:  x"018EE"    0.194763
o_lp:  x"1990E"   -0.804260
Warning: overflow ...
> In ALU at 91
  In TB_LWDF_5_auto>lfDFG at 26
  In TB_LWDF_5_auto at 9
  In testbench_LWDF_5_a…..
```

---- executation aborted ----

The text above indicates that the testbench calculates and writes the output data for both `o_hp` and `o_lp` in hex and in decimal format, but that at certain moment in time overflow errors occur which will invalidate one or maybe both of all following output values (there is feedback in the system).

We can pinpoint the error more accurately by running the testbench in 'debug'-mode, which will show all internal results for each PASS (i.e. for each complete loop through the SSG due to a new input data value) and for each STATE:

```
>> testbench_LWDF_5_auto( [17 15 0], 1 );
```

The following text is a part of the total output data starting at the offending PASS:

```
PASS 5:
  === AFTER STATE 1:
v1A = i2A - inp:   1FFE0 = 07FE0 - 08000
v4B = i3B - i2B:   11D06 = 00B1A - 0EE14
  === AFTER STATE 2:
m1A = a1A * v1A:   00000 = 1FC02 * 1FFE0
v4C = i3C - i2C:   0573D = 0F004 - 098C7
  === AFTER STATE 3:
v2A = inp + m1A:   08000 = 08000 + 00000
v3A = i2A + m1A:   07FE0 = 07FE0 + 00000
m2B = a2B * v4B:   0763F = 1BD51 * 11D06
  === AFTER STATE 4:
Warning: overflow ...
> In ALU at 91
  In TB_LWDF_5_auto>lfDFG at 38
  In TB_LWDF_5_auto at 9
  In testbench_LWDF_5_auto at 57
v5B = i2B + m2B:   16453 = 0EE14 + 0763F
v6B = i3B + m2B:   08159 = 00B1A + 0763F

----  executation aborted ----
```

We can see now that the addition in operation `v5B` causes the error, due to the fact that in *this* number format the addition of two positive values results in a negative sum.
Indeed, the addition $0EE14h + 0763F_h = 0\_1110\_1110\_0001\_0100 + 0\_0111\_0110\_0011\_1111$
results in the binary number $1\_0110\_0100\_0101\_0011 = 16453_h$,
which in a 2's complement notation is a negative number since the msb is '1'.
If we had been working with 18 bits (3 integer bits), then

```
      00_1110_1110_0001_0100
      00_0111_0110_0011_1111
    ------------------------  +
      01_0110_0100_0101_0011
```

which is still 16453_h but in another number format with a zero msb.

Again, the toolbox functions `hex2fixp`, `fixp2hex` and MATLAB's `hex2dec` and `dec2bin` can be helpful for a closer look:

```
>> NM2 = [17 15];
>> NM3 = [18 15];
>> hex2fixp('0ee14',NM2) + hex2fixp('0763F',NM2)
ans =
    2.7838
>> fixp2hex( hex2fixp('0ee14',NM2) + hex2fixp('0763F',NM2), NM2 )
??? Error using ==> fixp2hex
FIXP2HEX: signed value "2.78378" doesn't fit in 2 integer bits ...

>> fixp2hex( hex2fixp('0ee14',NM2) + hex2fixp('0763F',NM2), NM3 )
ans =
16453
```

Indeed, the command

```
>> testbench_LWDF_5_auto( [17 15 1], 0 );
```

returns without errors for the given input signal, and the output file `LWDF_5.OUT` will have been created in the current directory.
`FIR5.OUT` is a file in a specific hex-format, which can be translated into decimal form using the function `read_OUT.m` (since we have an `o_lp` and an `o_hp`, be sure to select 2 outputs):

```
>> y = read_OUT('LWDF_5',[17 15],2);
  1:  x"00000" =  0.000000000000000    x"00000" =  0.000000000000000
  2:  x"00000" =  0.000000000000000    x"00000" =  0.000000000000000
  3:  x"00000" =  0.000000000000000    x"00000" =  0.000000000000000
  4:  x"016CA" =  0.178039550781250    x"01A19" =  0.203887939453125
  5:  x"1F4F2" = -0.086364746093750    x"062F5" =  0.773101806640625
  6:  x"018EE" =  0.194763183593750    x"0990E" =  1.195739746093750
  7:  x"1F20E" = -0.108947753906250    x"08499" =  1.035919189453125
  8:  x"001B5" =  0.013336181640625    x"06FCD" =  0.873443603515625
  9:  x"00484" =  0.035278320312500    x"08632" =  1.048400878906250
 10:  x"1F980" = -0.050781250000000    x"086BE" =  1.052673339843750
 11:  x"004E6" =  0.038269042968750    x"076E1" =  0.928741455078125
 12:  x"1FFCD" = -0.001556396484375    x"0822C" =  1.016967773437500
 13:  x"1FBA2" = -0.034118652343750    x"084FE" =  1.039001464843750
```

In here, the 2nd column represents B_{fwd} a.k.a. `o_lp` or the wanted low-pass output.
It can be checked that the maximum absolute difference between these output values and the reference values for the low-pass output is about $5.3 \cdot 10^{-5}$ (Again, this holds true only for the investigated samples).

Notice that using the setup with only one sample signal for both supplying new input values, and reading the resulting output from the previous sample (see the timing diagram in the Reference Guide), an additional one cycle delay in the output values will occur.

> **Note:** Be sure to always use the same `[N M]`-fixed-point representation for the `.INP`-file, for the testbench calculations and for interpretation of the `.OUT`-results.

There is a second reason which may justify the choice for a `[17 15 1]` representation.
In the (L)WDF Design Toolbox was mentioned that for this class of filters, the magnitude transfer function in the stop-band is extremely sensitive to deviations in the calculated phase characteristics in the upper and lower branches of the structure.
Figure 11 shows the stop-band of the magnitude transfer function for two different fixed-point representations, viz. `[23 21 1]` and `[15 13 1]`.
These transfer functions in the frequency domain have been obtained by calculating the impulse response of the filter with the MATLAB testbench and subjecting this to an FFT. Care has to be taken of course that the sequence is long enough to avoid seeing artifacts of the FFT (4096 points are used).

As expected, it can be seen that the resolution used for the fixed-point calculations directly affects the transfer function due to quantizing effects. Our `[17 15 1]` representation will show a reasonable, all-be-it not perfect approximation of the ideal stop-band behavior.

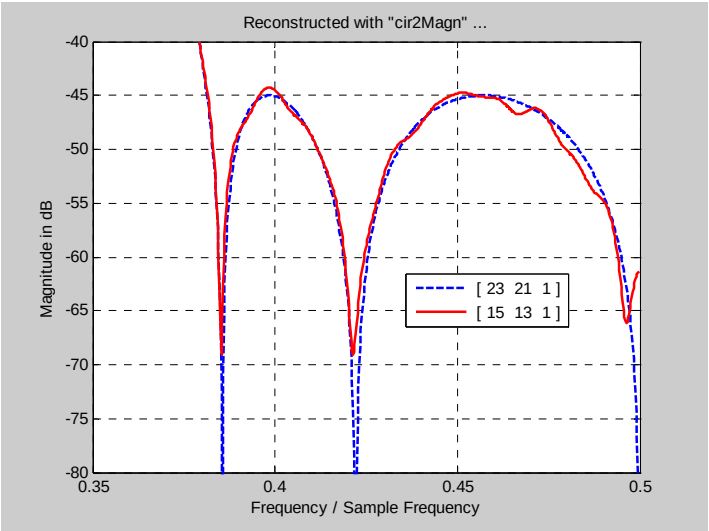


Figure 11. *Stop-band reconstruction from the impulse response with a 4096 point FFT.*

Generating the VHDL files and testbench.

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

```
>> gen_VHD( 'LWDF_5.cir', [17 15 1], 0, 'List', 1,1, 1,2 );
```

to generate the VHDL design files and the VHDL testbench file.
In the newly created sub-directory <$OUR_CIR_DIR>\FIR5\vhdl\ will be written

- the generated files LWDF_5_SSG_auto.vhd, LWDF_5_ auto.vhd, testbench_LWDF_5_ auto.vhd,
- and the files resources_reg.vhd and txt_util2.vhd that are copied from the toolbox' installation tree.

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.

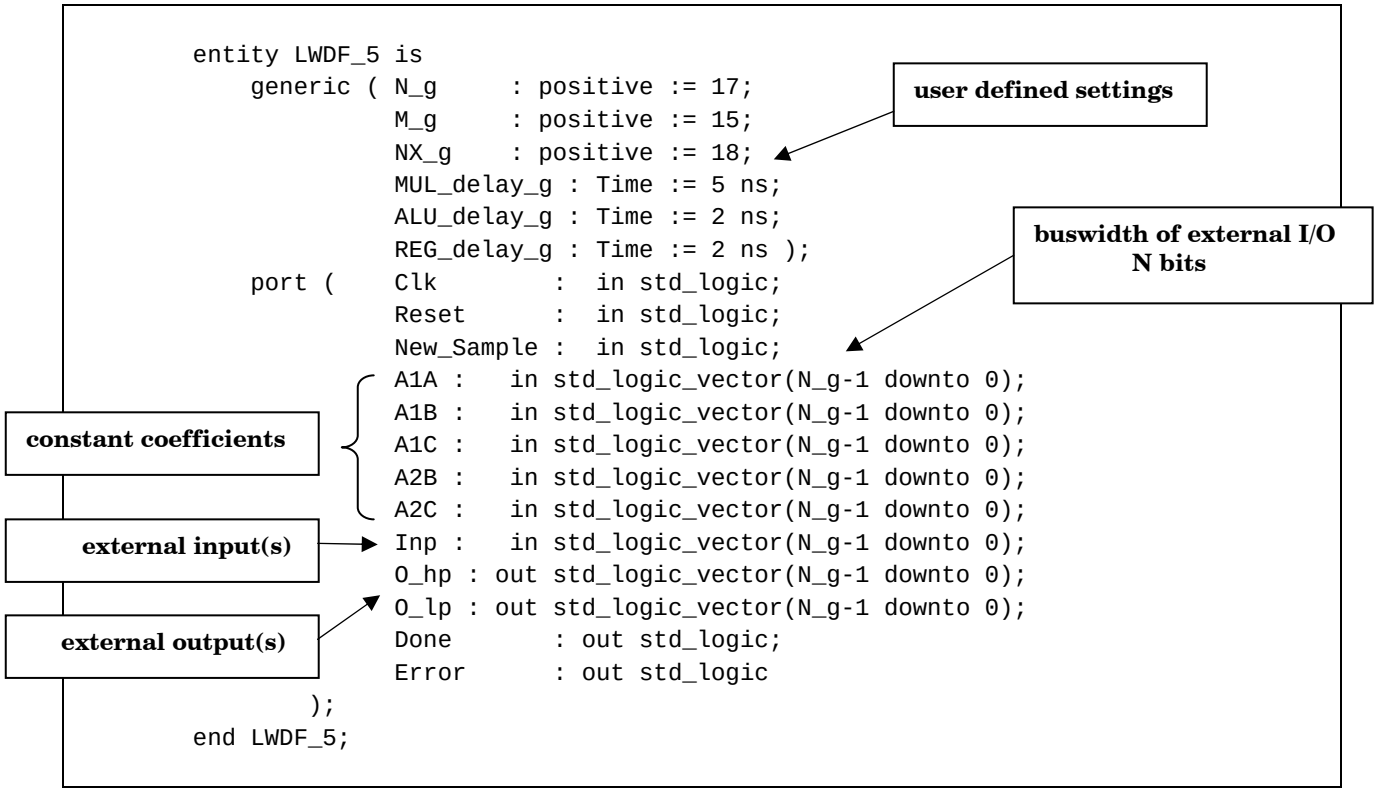
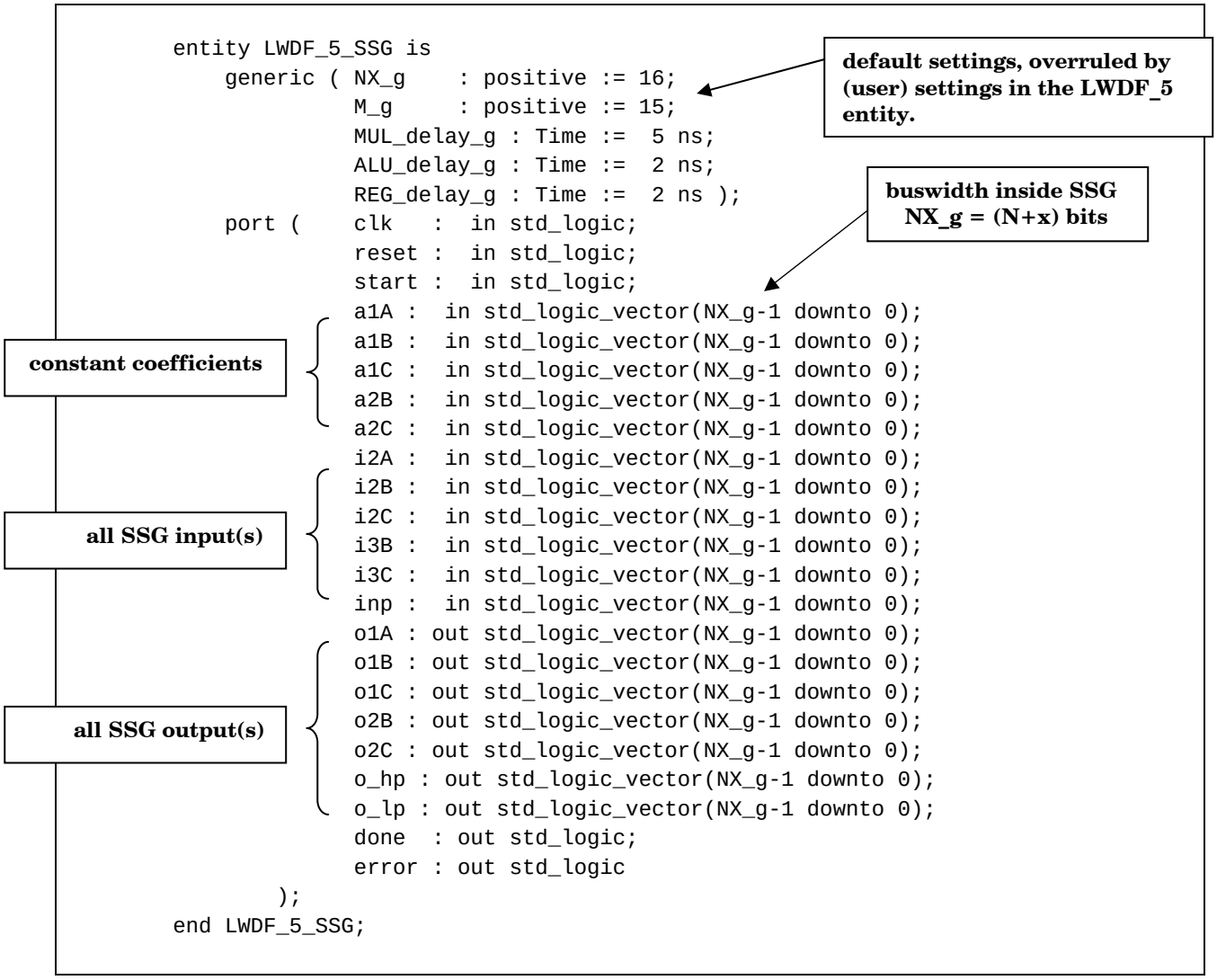


Figure 12a. *The definition of the entity* LWDF_5 *in the file* LWDF_5_auto.vhd.

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

Finally, `LWDF_5_SSG_auto.vhd`, `LWDF_5_auto.vhd` and `testbench_ LWDF_5_auto.vhd` are the descriptions of the circuit and its test environment in increasing level of hierarchy.
Figure 12a and b list the entity definitions of `LWDF_5_SSG` and `LWDF_5`. `LWDF_5_SSG` is instantiated as a component inside `LWDF_5`'s architecture.
Notice that the feedback registers in `LWDF_5_auto.vhd` are inferred automatically (instead of by inserting `REG` components) by the way in which the VHDL code is written.




```
entity LWDF_5_SSG is
    generic ( NX_g    : positive := 16;
              M_g     : positive := 15;
              MUL_delay_g : Time :=  5 ns;
              ALU_delay_g : Time :=  2 ns;
              REG_delay_g : Time :=  2 ns );
    port (    clk   :  in std_logic;
              reset :  in std_logic;
              start :  in std_logic;
              a1A :  in std_logic_vector(NX_g-1 downto 0);
              a1B :  in std_logic_vector(NX_g-1 downto 0);
              a1C :  in std_logic_vector(NX_g-1 downto 0);
              a2B :  in std_logic_vector(NX_g-1 downto 0);
              a2C :  in std_logic_vector(NX_g-1 downto 0);
              i2A :  in std_logic_vector(NX_g-1 downto 0);
              i2B :  in std_logic_vector(NX_g-1 downto 0);
              i2C :  in std_logic_vector(NX_g-1 downto 0);
              i3B :  in std_logic_vector(NX_g-1 downto 0);
              i3C :  in std_logic_vector(NX_g-1 downto 0);
              inp :  in std_logic_vector(NX_g-1 downto 0);
              o1A : out std_logic_vector(NX_g-1 downto 0);
              o1B : out std_logic_vector(NX_g-1 downto 0);
              o1C : out std_logic_vector(NX_g-1 downto 0);
              o2B : out std_logic_vector(NX_g-1 downto 0);
              o2C : out std_logic_vector(NX_g-1 downto 0);
              o_hp : out std_logic_vector(NX_g-1 downto 0);
              o_lp : out std_logic_vector(NX_g-1 downto 0);
              done  : out std_logic;
              error : out std_logic
          );
end LWDF_5_SSG;
```

Figure 12b. *The definition of the entity* `LWDF_5_SSG` *in the file* `LWDF_5_SSG_auto.vhd`.

During its run, gen_VHD generates a couple of plots, in which it shows the mapping of the operations on the available MULs and ALUs, and where exactly it needs the additional registers. Figure 13 holds true for our LWDF_5 example: 1 MUL, 2 ALUs and 5 REGs.

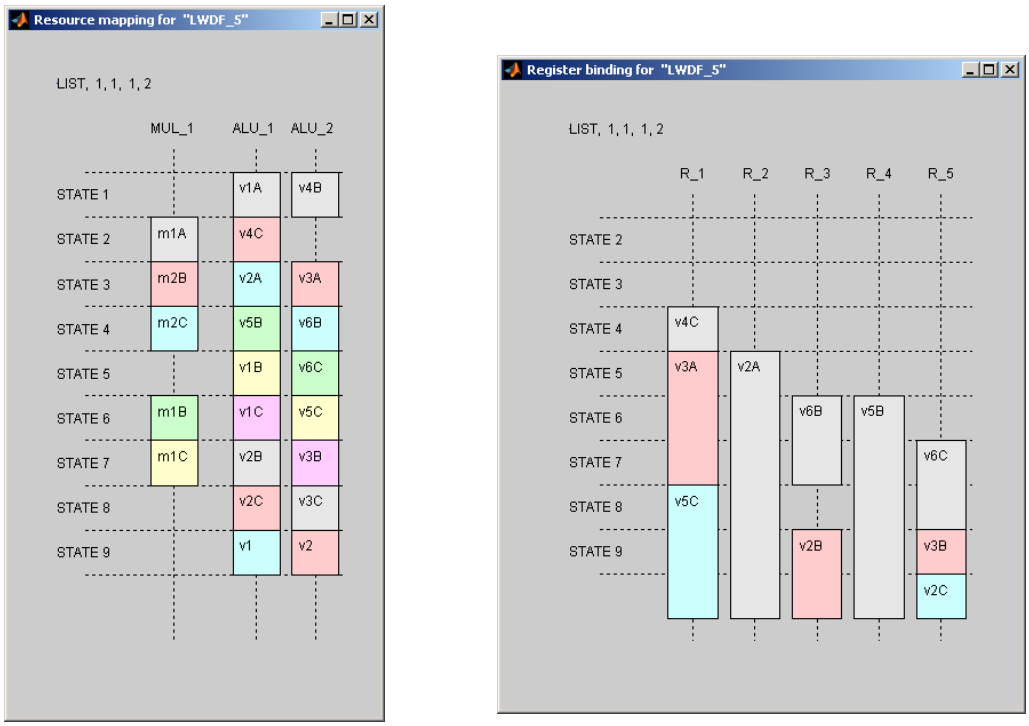


Figure 13. *The Resource mapping and Registry binding as reported by* gen_VHD.

Simulation results.

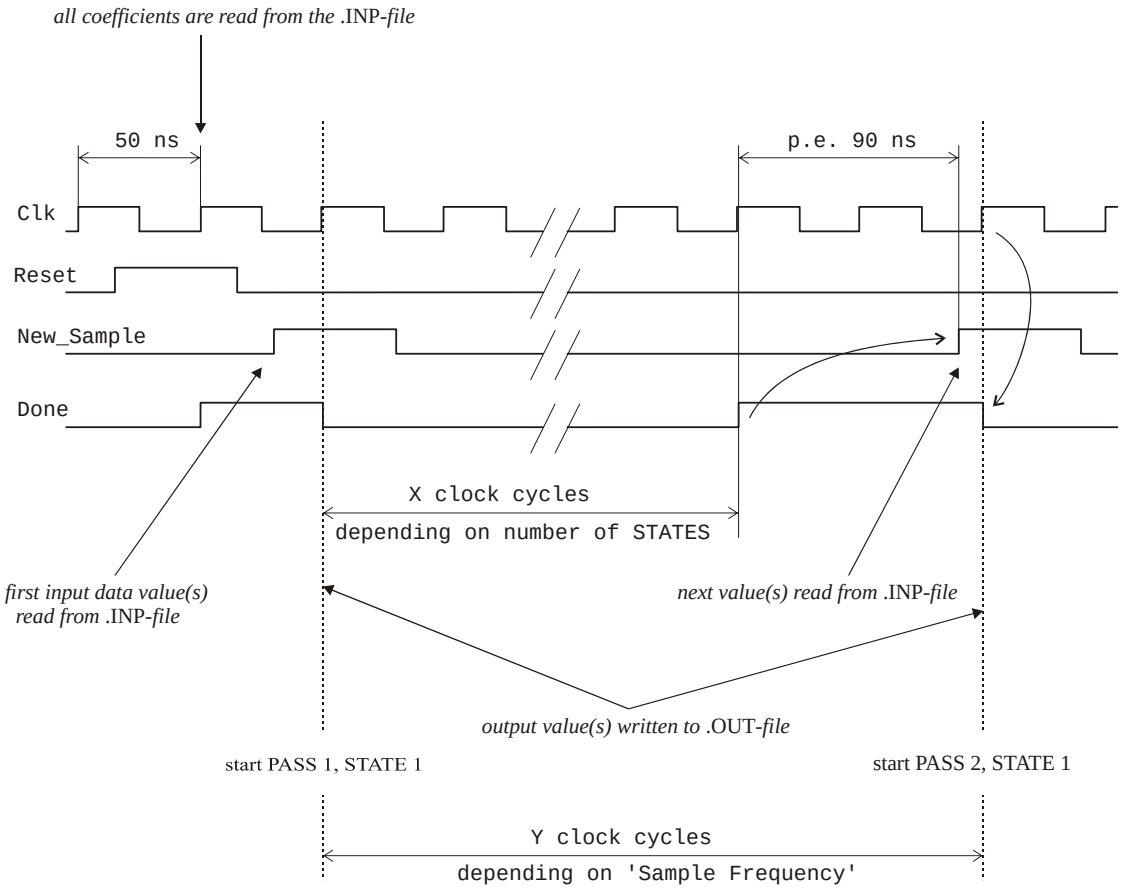


Figure 14. *Timing diagram of the simulation sequence..*

The testbench starts each run of the simulation (see Figure 14) with an initialization phase in which an asynchronous `Reset` is issued that clears all internal registers. At the first positive going edge of the (SSG-) `Clk` when `Reset` is high, all coefficients from the `.INP`-file are read and the `Done` bit goes high to signal that the SSG is ready and awaiting.

At each `New_Sample` (= `Start`) going high, a new input data value (or as many input values as there are external inputs) is (are) read from the `.INP`-file. At the first positive going edge of the clock, the SSG process sets `Done` low and starts with its first STATE. The STATES are advanced each clock period. Finally, its `Done` signal goes high again, and everything is halted until the next sample arrives. When this actually happens is, of course, determined by the sample frequency to SSG clock ratio and, when the sample frequency is relatively low, can take a large number of SSG clock cycles. In the simulation testbench, the time that a new sample pulse trails the `Done` edge is fixed and set to a value of 90 ns, slightly less then 2 clock periods (one `Clk`-period is set to 50 ns).

The `.OUT`-file is written each time that `Done` goes low with the same number of output values as there are external output ports.

All VHDL files in `<$OUR_CIR_DIR>\FIR5\vhdl\` together with the `.INP`-file, are needed for the simulation. Usually this means that the `.INP`-file used with MATLAB has to be duplicated in the directory where the simulator's project file has been created.

Figure 15 shows a screen-shot of the Wave window that can be obtained by running the VHDL files through the Model*Sim* simulator. All in- and output signals of the `LWDF_5` block (i.e. 'Signals in Region') are shown here. See the Reference Guide for an explanation of the stimuli signals and time steps that are used for the simulation.

Just like the MATLAB testbench, the VHDL testbench writes it results to a `LWDF_5.OUT`-file.
Comparison of both `LWDF_5.OUT`-files 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.

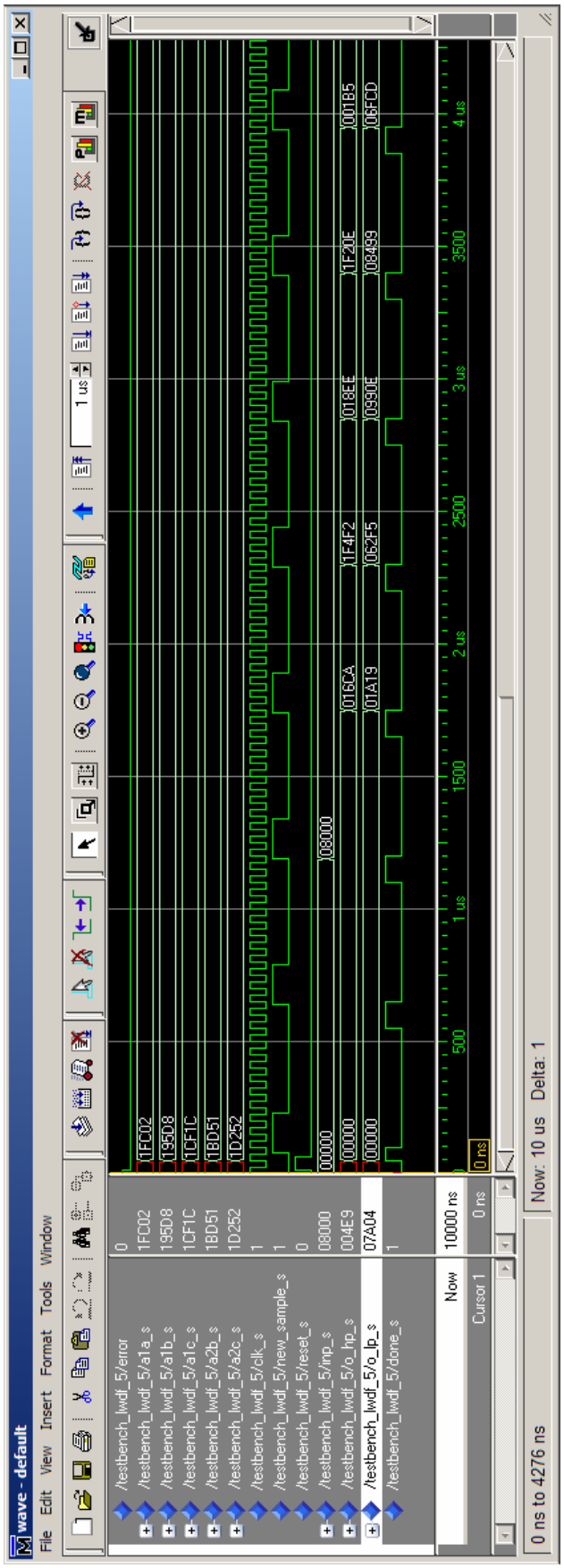


Figure 15. *Part of th e waveform output of the* `LWDF_5` *VHDL design after simulation with* Model*Sim*.

Figure 16 shows the simulation results corresponding to our previously described errors when using a `[17 15 0]` fixed-point setup.
From the Resource Map in Figure 13 can be seen that operation `V5B` is mapped on `ALU_1` (and indeed in STATE 4 as was also concluded from the MATLAB error). If we zoom in in the Wave window to PASS 5 (following the 5th `new_sample_s` pulse), we notice the `error` signal going high and the erroneous values of `o_lp` following this PASS. In the timing setup of our testbench this will occur after about 2500 ns.
If we select 'Add to Wave: Signals in Design' and search for the signals in STATE 4, we recognize the same values as reported by MATLAB. Notice that the result of the addition is *registered* in STATE 5.

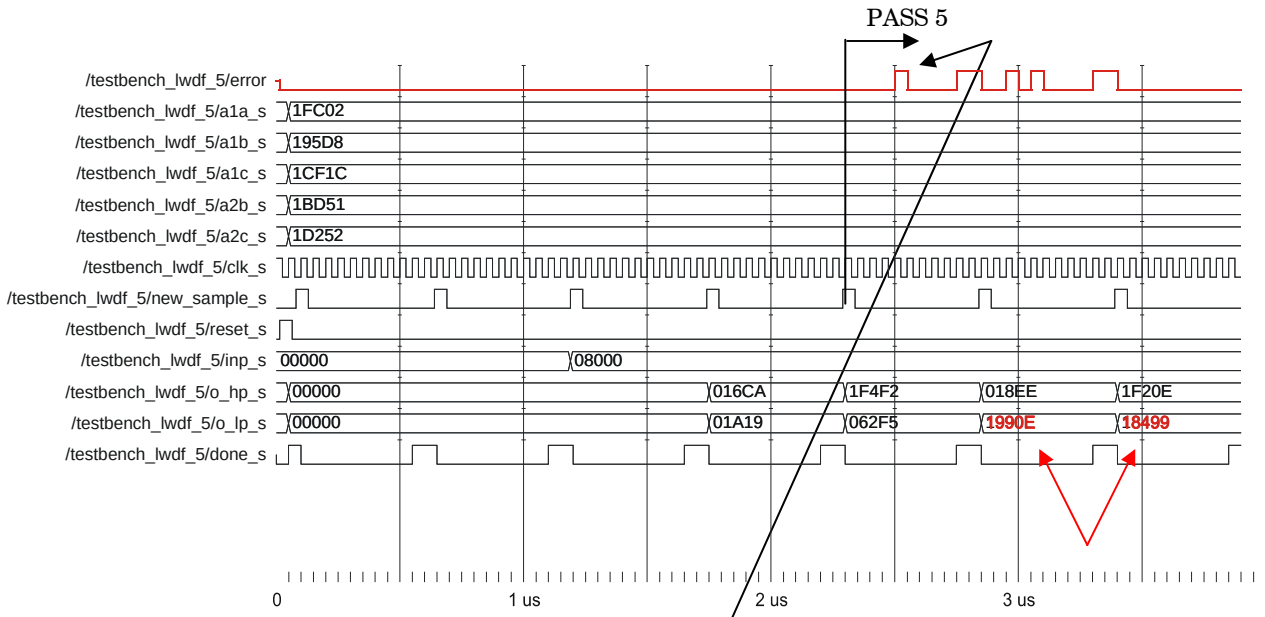
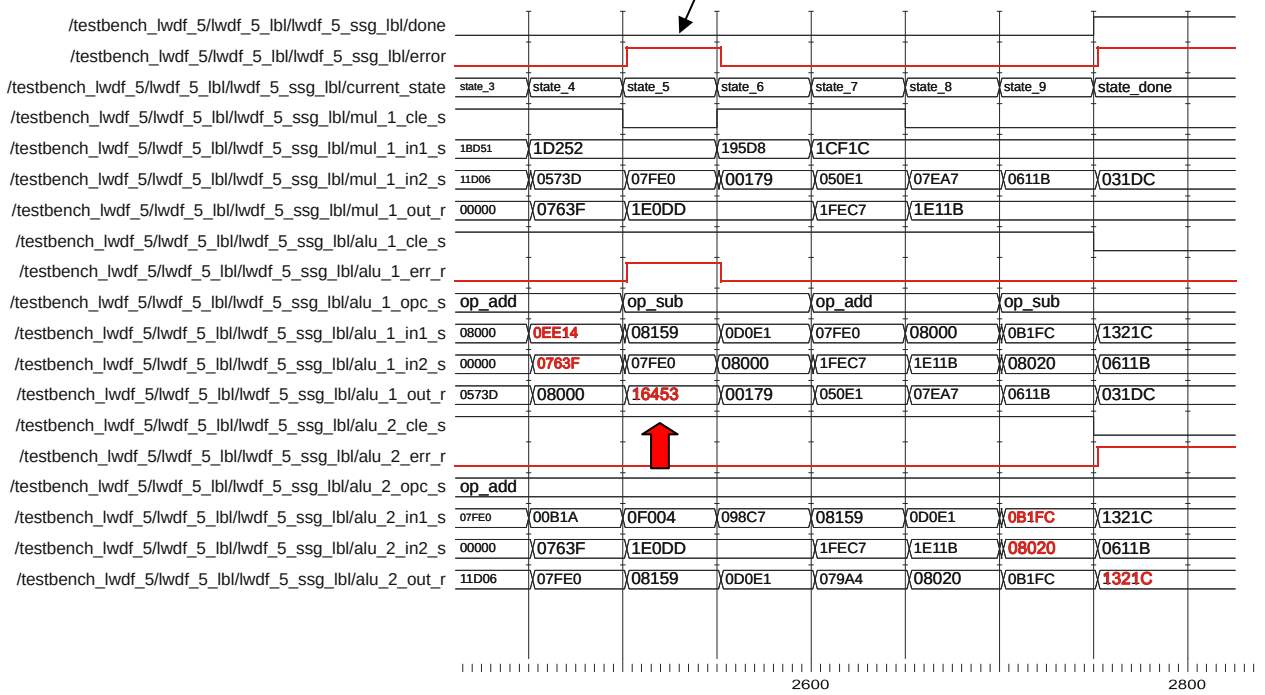


Figure 16. *Simulation with a* `[17 15 0]` *bus setup.*



Synthesis results.

For synthesis, we need only the files `resources_reg.vhd`, `LWDF_5_SSG_auto.vhd` and `LWDF_5_auto.vhd` .
In Figure 17a to e, the results 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, resetable registered output, exactly as had been described before. The VHDL definition of `MUL_R` in `resources_reg.vhd` is such, that these registered versions of the MULT18x18 component will be selected by the synthesizer (i.e. by the synthesis tools from Synplicity Inc.).

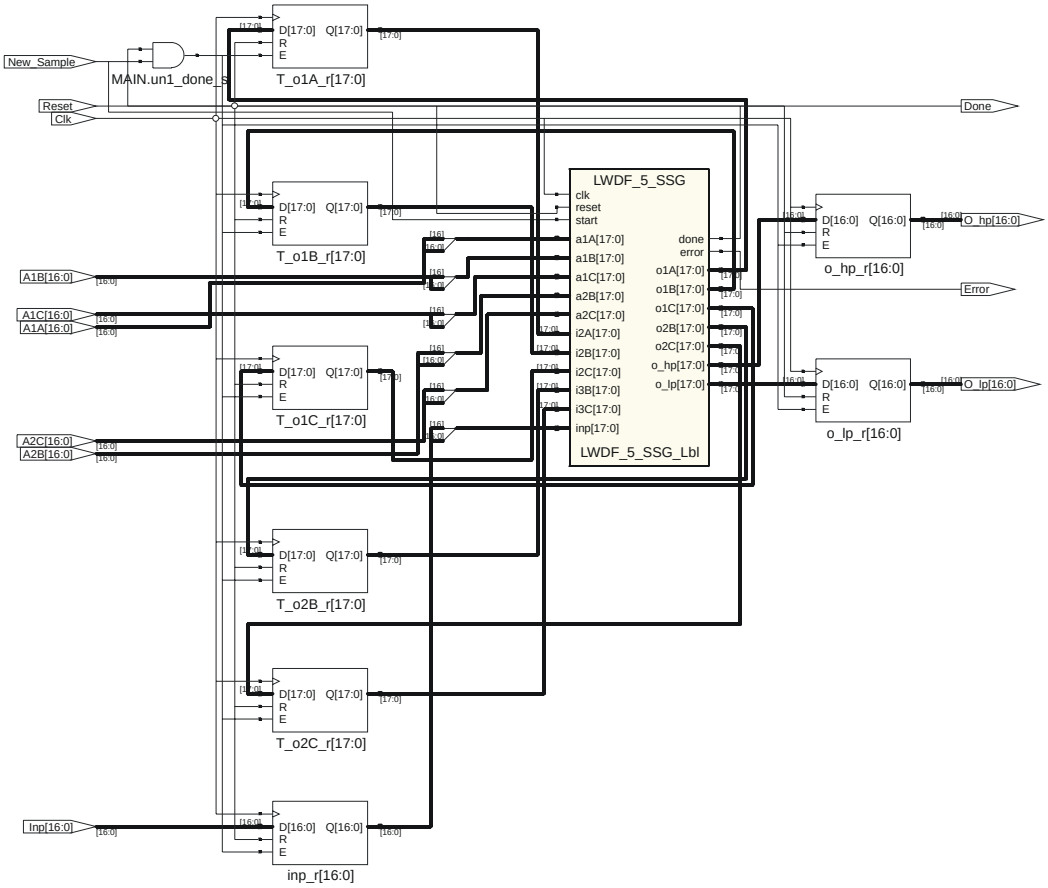


Figure 17a. *RTL view of the synthesized* `LWDF_5` *top level as drawn by Synplify_Pro.*

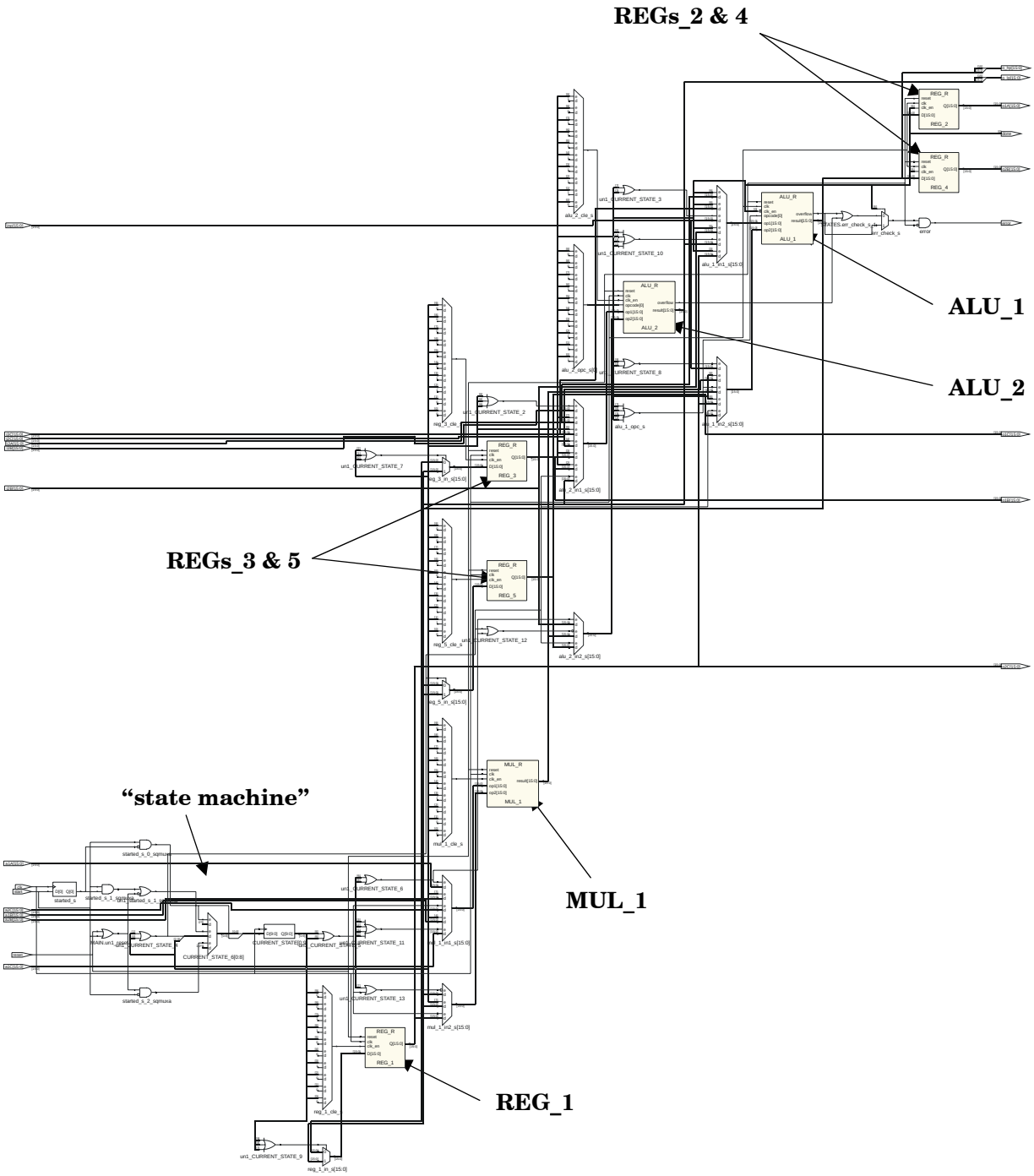


Figure 17b. *RTL view of the* LWDF_5_SSG*-block.*

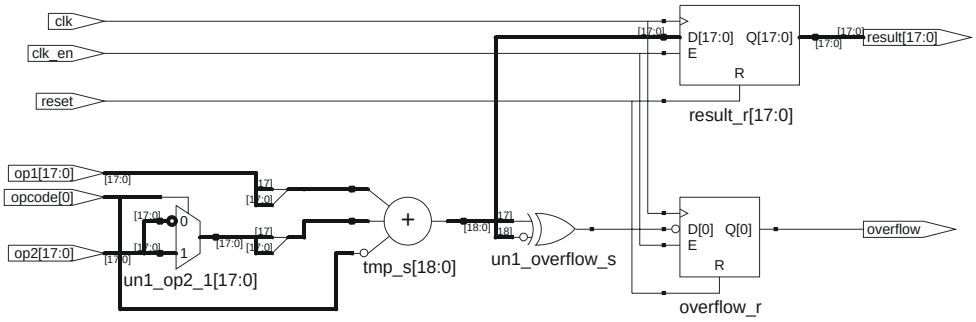


Figure 17c. *RTL view of* ALU_R.

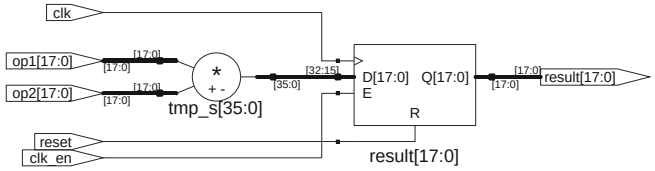


Figure 17d. *RTL view of* MUL_R.

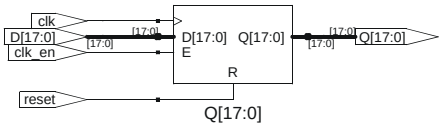


Figure 17e. *RTL view of* REG_R.

In Table 4, the Resource Usage as reported by the synthesizer for the XC3S2000 implementation is given.

Table 4. *listing of a part of the* `LWDF_5.srr` *file*

```
Resource Usage Report for LWDF_5

Mapping to part: xc3s2000fg456-4
Cell usage:
FD              2 uses
FDCE            129 uses
FDR             8 uses
FDRE            141 uses
GND             2 uses
MULT18X18S      1 use
MUXCY_L         36 uses
MUXF5           71 uses
VCC             1 use
XORCY           38 uses
LUT2            22 uses
LUT3            345 uses
LUT4            94 uses

I/O primitives: 141
IBUF           104 uses
IBUFG          1 use
OBUF           36 uses

BUFG           1 use

I/O Register bits:                  0
Register bits not including I/Os:   280 (0%)

Block Multipliers: 1 of 40 (2%)

Global Clock Buffers: 1 of 8 (12%)


Mapping Summary:
Total  LUTs: 461 (1%)
```

Implementation on FPGA.

The VHDL output has been implemented and tested on two Xilinx FPGA platforms, viz.

- as a ‘slave’ connected to a soft-core AVR processor through a Wishbone bus and with data in/output by means of a serial RS2323 connection (AVNET XC3S2000 Development Kit),
- connected straight between a 14-bit ADC and 14-bit DAC on a Xilinx/Nallatech XtremeDSP Kit (Virtex2 family FPGA).

The AVR-Wishbone slave:
The setup for this test consisted of an adapted ATmega103 VHDL-description from opencores.org, which was connected to the filter through a Wishbone bridge, all of them on the XC3S2000 Spartan3 FPGA on the AVNET Development Kit. The AVR was connected to a pc by means of a serial RS2323 connection, which was used to send the coefficients and the ‘signal’ –in our case the step-function again– to the filter and to show the read-back results in a terminal window. These results were exactly as were expected from the previous tests.

The XtremeDSP version:
In this setup a Xilinx XtremeDSP kit, built around a XC2V3000 Virtex2 FPGA was used.
The DSPkit is provided with two 14-bit ADC’s and two 14-bit DAC’s, so the filter can be tested in real-time. Figure 18 shows the measurement results when connected to a HP4195A Network/Spectrum Analyzer. To read the measurement data into the pc, a 82357A GPIB/USB interface from Agilent Technologies was connected between analyzer and pc, while with the aid of a HPGL interpreter m-file the plot was redrawn on the pc from within MATLAB.
The blue line in Figure 18 should be used as the reference transfer of the system without the filter: clearly visible are

- the effect of the sinc-function (sin(x)/x) due to the sample-and-hold function of the DAC (a gradual roll-off from zero to –3.9 dB at half the sample frequency, no “Zero Stuffing”), and
- the low-pass influence of the FIR filter in the DAC with a –3 dB cut-off at about 0.45 times the sample frequency.

The system clock on the DSP kit was set to 40 MHz, and the sample clock had been derived from this clock by dividing it by 100, resulting in a sample frequency of 400 kHz.

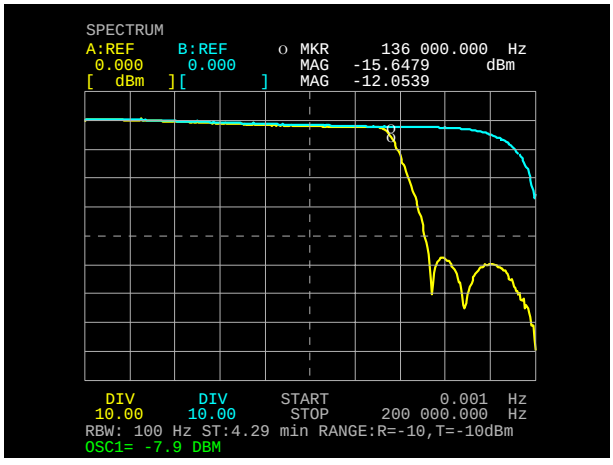


Figure 18. *The XtremeDSP implementation measured on a HP4195A Network/Spectrum analyzer.*

Epilog.

It has been shown that this toolbox can effectively be used to create synthesizable VHDL and realizable hardware implementations, while the MATLAB testbench can be used for system level debugging and to tune parameters like buswidth and fixed-point setup.
In the report Implementation of an 18 point IMDCT on FPGA available from http://ens.ewi.tudelft.nl/~huib/mtbx, this toolbox has been proven to be useful for the design of an imdct in an mp3 decoder.

References.

[DeM94] G. De Micheli
Synthesis and Optimization of Digital Circuits
McGraw-Hill, Inc. (1994)
ISBN 0-07-113271-6

[Par99] K.K. Parhi
VLSI Digital Signal Processing Systems
John Wiley & Sons, Inc. (1999)
ISBN 0-471-24186-5

Available from http://ens.ewi.tudelft.nl/~huib/mtbx/get_files.php

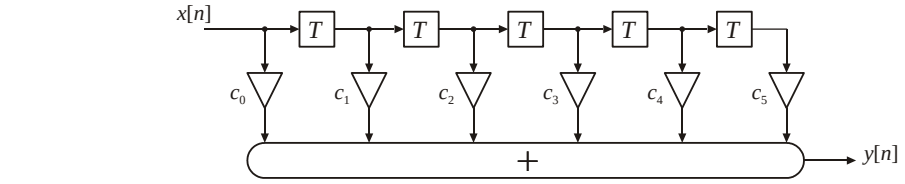
(L)WDF Toolbox User's Guide

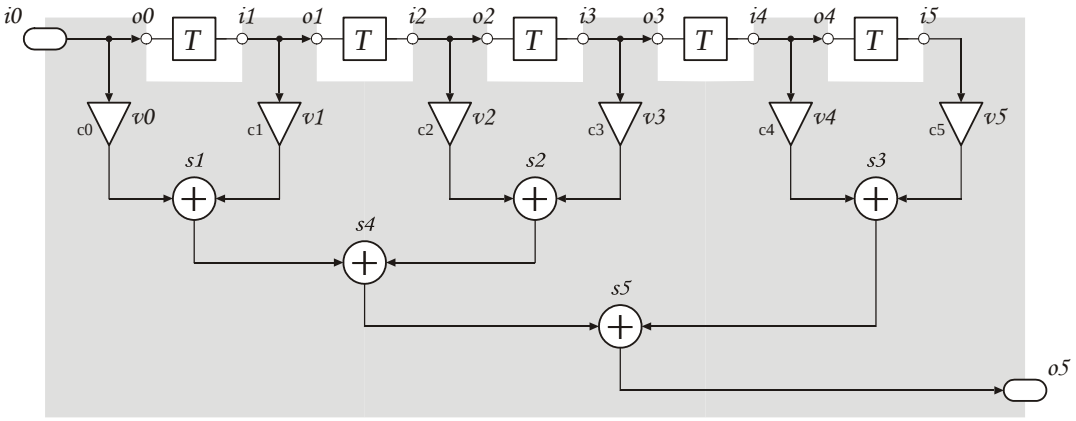
(L)WDF Toolbox Reference Guide

Scheduling Toolbox Reference Guide

Appendix A. In-to-Out connections in .cir-files.

In circuits with a cascade of delay elements (all or not tapped) –for complying to the cir-file rules– it will be necessary to use I/O ports in between the delay elements. The connecting lines between the ports are positioned inside the SSG. This is illustrated in the figures and the (part of the) cir-file listing below.





Assignments to be found in `.cir`-file:

```
    :
    :
o0 = i0;
o1 = i1;
o2 = i2;
o3 = i3;
o4 = i4;
    :
i1 = To0;
i2 = To1;
i3 = To2;
i4 = To3;
i5 = To4;
    :
    :
v0 = c0 * i0;
    :
    :
o5 = s5;
```

'straight-through' connections

definition of delay elements

'other' assignments

Appendix B. MATLAB script for this example …

With both the (L)WDF Design Toolbox and the Scheduling Toolbox installed, this simple script is in fact enough to replay the example described in this User's Guide, up to the VHDL simulation and synthesis.

```
% only needed if both toolboxes not in the same directory
addpath('   <path to L)WDF Design Toolbox>     ');

Hs = Hs_cauer( 5, 0.1, 45, 'a', fz2fs(0.34), 1 );
Hz = Hs2Hz( Hs );
% Table 1:  Hz.poly_fz' and  Hz.poly_gz'

yref = filter( Hz.poly_fz, Hz.poly_gz, ones(1,20) );
% Table 2, step response:  yref'

LWDF = Hs2LWDF( Hs );
coeffs = LWDF2cir(LWDF,'L','LWDF_5.cir');
gen_mTB( 'LWDF_5.cir', 0, 'List', 1,1, 1,2 );

% NOTE THAT the sequence of coefficients needed by gen_mTB differs from the
% one obtained from LWDF2cir, so reshuffling is needed
tmp = coeffs; coeffs(3,:) = tmp(4,:); coeffs(4,:) = tmp(3,:);

gen_INP( 'LWDF_5\matlab\LWDF_5.INP', [17 15], coeffs, [ 0 0 ones(1,10) ] );

addpath('LWDF_5\matlab\');
testbench_LWDF_5_auto( [17 15 1], 0 );
y = read_OUT( 'LWDF_5.OUT', [17 15], 2 );

% Notice that the scheduled circuit introduces one additional data-clock delay
% by using the sample-clock for both reading and writing
maxAbsErr = max( abs(yref(1:10)' - y(4:end,2)) );

gen_VHD( 'LWDF_5.cir', [17 15 1], 0, 'List', 1,1, 1,2 );
!copy LWDF_5\matlab\LWDF_5.INP LWDF_5\vhdl\
```

This page intentionally left blank