Scheduling Toolbox
for
MATLAB

Reference Guide

version MSc_08

Ing. H.J. Lincklaen Arriëns
November 2008
November 2007: No changes between versions msclab2006 and MSc_07
November 2008: No changes between versions msclab2006 and MSc_08

The author assumes no responsibility whatsoever for use of the software by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. Acknowledgement if the software is used is appreciated.

MATLAB is a registered trademark of The MathWorks, Inc.
Graphviz - Graph Visualization Software has originally been developed by AT&T Research, and is licensed on an open source basis under The Common Public License. See http://www.graphviz.org/
# Table of Contents

Structure and syntax of the .cir file ............................................................................................................ 2
The VHDL component ............................................................................................................................... 5
VHDL Testbench Simulation ...................................................................................................................... 6
VHDL resources ....................................................................................................................................... 7
Signed fixed-point notation ......................................................................................................................... 8
Setup of the .INP- and .OUT-files............................................................................................................ 11
Overview of the Software Environment.................................................................................................... 13
Categorical Listing of Functions ............................................................................................................... 14
Alphabetical Listing of Functions ............................................................................................................. 16
   ALAP.................................................................................................................................................. 18
   ALU................................................................................................................................................... 19
   ASAP................................................................................................................................................ 20
cirInfo................................................................................................................................................... 21
   fixp2hex.......................................................................................................................................... 22
   forceD................................................................................................................................................ 23
gen_INP................................................................................................................................................. 24
gen_mTB.............................................................................................................................................. 26
gen_VHD............................................................................................................................................ 28
hex2fixp............................................................................................................................................... 30
listSched............................................................................................................................................... 31
MUL.................................................................................................................................................... 32
parse.................................................................................................................................................... 33
read_OUT ........................................................................................................................................... 34
ret ime_minCycles.................................................................................................................................. 35
showDistrib......................................................................................................................................... 36
showGraph......................................................................................................................................... 37
toAdjMat .......................................................................................................................................... 38
toSFixp............................................................................................................................................... 39
toUFixp............................................................................................................................................. 40
view_cir_IO....................................................................................................................................... 41
view_DFG ......................................................................................................................................... 42
xplore ............................................................................................................................................... 43
schedGUI .......................................................................................................................................... 44
Structure and syntax of the .cir file

The input for the scheduling software is an ASCII text file (distinguishable by the extension '.cir'), in which basically all assignments (e.g. the operations to be performed) are written line by line. In the cir-file, one has to describe the operations that can be scheduled (a textual representation of a sequencing graph (SG)), and how this SG is connected to the outer world. Here, we will use an additional layer –the 'feedback and external I/O layer'– around the SG (which after scheduling will be an SSG) in which all feedback operations and connections have to be described: this information is thus a part of the cir-file, although not subjected to the scheduling process.

In fact, the cir-file is a description of a complete circuit (or circuit module).

![Diagram of a circuit](image)

**Figure 1.** Concept of the cir-file description.
The concept of this setup is shown in Figure 1, for an arbitrary imaginable circuit with 2 external inputs, 1 external output and 3 coefficients.

The operations that are supported in the (S)SG are multiplications (integer shifts, see later), addition and subtractions. The feedback layer is intended for delay elements (registers) and connections. Connections between feedback layer and (S)SG vice versa, and connections to the outside world are established through **Input Ports** and **Output Ports**.

This hierarchy will be maintained when the cir-file is translated into MATLAB m-files and/or when translated into VHDL-entities and architectures.

There are some simple rules that should apply for the .cir-file to be valid:

- **Operations** – which perform a multiplication, addition or subtraction – should be identified by a unique 'identifier' name. Such identifiers are also required for all input ports, output ports and coefficient ports. Delays are not part of the (S)SG and are connected to the (S)SG through (internal) input and output ports.

- **Identifiers** are case sensitive and may consist only of characters and figures. The first character of an identifier indicates whether the identifier applies to an operation, input port, etc.

By default, identifiers starting with an

- ‘i’ are reserved for input ports,
- ‘o’ are reserved for output ports,
- ‘a’ or ‘c’ are referring to coefficients (a special kind of inputs),

while all remaining characters can be used to indicate an operation (diacritic characters, underscores, etc. are not allowed).

Each operation will have two (2) inputs and one (1) output:

- **Inputs** are either connected to a previously terminated operation or an input port.
- An input which only appears at the right hand side(s) of (an) assignment(s), and never at a left hand side, is considered to be connected to an external input port.
- The **output** can be either connected to a next-in-line operation or to an output port.
- The result (e.g. the output) of an operation will have the same identifier name as the operation itself.
- Floating input or outputs, as well as floating input ports and output ports, are considered erroneous.

An **Assignment** should have the form

identifier = identifier operator identifier     with possible operators ‘∗’, ‘+’ and ‘−’

or

output port = identifier

or

input port = Toutput port    which involves a delayed feedback register

and can be optionally terminated with a ‘;’

**e.g.**

\[ v1 = v2 + v3; \]
\[ i2 = T01; \]

Only one assignment per line is allowed.

Note that an input of an operation can be implicitly connected to an input port, as in

\[ v1 = v2 \cdot i1; \]

but that outputs have to be explicitly connected to (only one) operation, e.g.

\[ o3 = v1; \]
although an operation can be connected to more than one outputs, e.g. an external and a feedback output as in the following fragment:

```
... 
outp = m1; 
of1 = m1; 
if1 = Tof1; 
....
```

Multiplications that are powers of 2 can be handled more efficiently by using a ‘shift’ operation. In hardware, such a shift is just a change in bit-line interconnect.

A shift operation in the cir-file can be specified using the ‘>>’ or ‘<<’ operators.

A shift operation should consist of

- the identifier of the operation, the result of which should be shifted, or an input that should be shifted,
- the shift operator, indicating the direction of the shift: ‘<<’ means a shift to the left (output value is larger than original value), while ‘>>’ means a shift to the right (decreased output value, power of negative value),
- the integer number of bits (powers of two) to be shifted,

this all surrounded by parenthesis.

E.g.

```
v3 = v1 + (v2 >> 2);  % v3 = v1 + v2/4
v1 = (i0 >> 1) - (i0 >> 3);  % v1 = 0.375 * i0;
outp = (v2 << 1);  % outp = 2 * v2;
```

**Note:** The shift operation performs an arithmetic shift to the right, so the sign of the originating value is preserved. For shifts to the left, it’s the user’s responsibility that the shifted value is handled correctly.

All characters on a line following a ‘%’ are considered to be a comment. The ‘%’-character can be the first one on a line, or can be following an assignment.

If for some reason you are not satisfied with the default start characters, it is possible to define your own identifiers by starting the cir-file (in any case before the first assignment) with the string

```
ioDef = 'xxxx';
```

where

- the first ‘x’ is replaced with the new character to identify the inputs with,
- the second ‘x’ with the new character for outputs, and
- the third and fourth ‘x’s with new characters for recognizing the coefficients.

E.g.

```
ioDef = 'XYab';
```

Remember that identifiers are **case-sensitive**.

Notice that both ‘in’ and ‘out’ are reserved words in VHDL (used for defining the direction of port signals), so avoid using these exact words for identifying inputs and outputs in the .cir-file.

In case `gen_VHD` finds them in the .cir-file, it replaces them with respectively ‘inp’ and ‘outp’ throughout the circuit and issues a warning.

Also note that it is advisable, given the fact that the names are displayed in output plots, to use relatively short identifiers.
The VHDL component

Except for the connections for the inputs(s), output(s) and coefficient(s), the VHDL code that will be generated for the circuit will also define 5 external control connections, viz.

- input ports for Clk and Reset,
- ports for the activation and completion signals: Start and Done, as well as
- an additional Error signal that will go high in case of overflow errors during calculations.

An entity, automatically generated with gen_VHD will look like

```vhdl
entity design_name is
  generic ( N_g     : positive := value defined for N;
            M_g     : positive := value defined for M;
            NX_g    : positive := value of N + x;
            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;
         list of all constant coefficients :  in std_logic_vector(N_g-1 downto 0);
         list of all external inputs :  in std_logic_vector(N_g-1 downto 0);
         list of all external outputs : out std_logic_vector(N_g-1 downto 0);
         Done  : out std_logic;
         Error : out std_logic
       );
end design_name;
```

All signals are active high. With Reset, all memory elements will be cleared to zero. The SSG starts calculating at the first positive going clock edge of Clk, following a low-to-high transition of the Start-signal. At each following positive going Clk-edge, the next STATE is executed. Finally the Done signal is set high when the output value(s) are valid. Then the process can start all over again (see also the chapter about VHDL Simulation).

In the set-up used, it is expected that the coefficients are passed to the circuit by means of the .INP-file (exactly the same file can be used for the MATLAB simulation). Changing the transfer characteristic of the filter by using a different set of coefficients –and limited only by the structure used– is then very simple.

It is certainly possible, if the coefficients don’t change during a simulation, to have them stored or hardwired in the feedback and ext I/O layer. This implies manually editing the VHDL-files or rewriting the MATLAB function.
VHDL Testbench Simulation

The simulation starts each run 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/system) Clk when Reset is high, all coefficients from the .INP-file are read and the Done bit of the Component-Under-Test (CUT) goes high to signal that the SSG is ready and awaiting.

Each time we want to start the computations with a new set of input data, we have to generate a positive going edge of the testbench signal Nxt_InData. A new input data value (or as many input values as there are external inputs) is (are) read from the .INP-file. This Nxt_InData is internally connected to the Start-port of the CUT, so at the first positive going edge of the clock, the SSG process sets Done low and starts with its first STATE. STATE is advanced each positive going Clk-edge. Finally, the CUT’s Done signal goes high again, and everything is halted until a new Nxt_InData is detected. When this actually happens is, of course, determined by the external data-input frequency and, when this data-input frequency is relatively low, can take a large number of SSG clock cycles. In the simulation testbench, the time that a new Nxt_InData 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 here).

The .OUT-file is written each time that Done goes low with the same number of output values as there are external output ports, eventually complemented with user-defined SSG-outputs.
VHDL resources

If synthesized with Synplify Pro, the architectures described in `resources_reg.vhd` with their generic buswidths [17 15], viz. ALU_R, MUL_R and REG_R, look as shown below.
Signed fixed-point notation

In most of the Digital Signal Processing designs, calculations are performed with the aid of fixed-point hardware instead of floating-point hardware. The added complexity and hardware resources that are needed by floating-point solutions are always more expensive than fixed-point solutions, and are usually only justifiable for systems with high dynamic ranges. In our filter design, we will also use a fixed-point implementation.

Surely, everyone is acquainted with the all integer representation, i.e. where we are dealing with only positive whole numbers. Since this is a severe limitation in calculations, it is a necessity to also be able to represent negative numbers. For this purpose, the 2’s-complement method is the most widely used representation. In this representation, the MSB is assigned a negative weight (see Table 1). The most appealing advantage of this representation is the fact that no additional hardware is needed to perform additions and subtractions; the drawback is that negating a number is not merely an inversion of each bit since the number representation is asymmetrical.

<table>
<thead>
<tr>
<th>Table 1.</th>
<th>( N )-bits integer representations (2’s complement for signed)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>value</td>
</tr>
<tr>
<td>unsigned</td>
<td>( N_{\text{UINT}} = b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \cdots + b_1s^1 + b_0s^0 )</td>
</tr>
<tr>
<td>signed</td>
<td>( N_{\text{SINT}} = -b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \cdots + b_1s^1 + b_0s^0 )</td>
</tr>
</tbody>
</table>

However, there is no reason why we should not insert a virtual binary point somewhere between the bits. In fact, for integer representations this point is just assumed to be to the right of the least significant bit. Inserting a binary point influences the weights of each bit, as is indicated in Figure 2, again for an \( N \) bits fixed point number. We distinguish a ‘whole part’ and a ‘fraction part’.

The binary point itself is generally not coded, but its position has to be known by the software and/or the user: in our notation between the \((M+1)\)th and the \(M\)th bits from the right.

![Fixed-point notation.](image)
The relation between binary bits and decimal values, and the attainable ranges for these values, are tabulated in Table 2, given an \( N \) bit binary number of which \( M \) bits are fraction bits.

**Table 2.** \([N M]\)-bits fixed point representations (2’s complement for signed)

<table>
<thead>
<tr>
<th></th>
<th>value</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed</td>
<td>( N_{S\text{MIX}} = -b_{N-1}2^{N-M-1} + \sum_{i=M}^{N-2}b_i2^{i-M} + \sum_{j=0}^{M-1}b_j2^{j-M} )</td>
<td>(-2^{N-M-1} \leq N_{S\text{MIX}} \leq \left(2^{N-M-1} - \frac{1}{2^M}\right))</td>
</tr>
<tr>
<td>unsigned</td>
<td>( N_{UM\text{IX}} = \sum_{i=M}^{N-1}b_i2^{i-M} + \sum_{j=0}^{M-1}b_j2^{j-M} )</td>
<td>( 0 \leq N_{UM\text{IX}} \leq \left(2^{N-M} - \frac{1}{2^M}\right))</td>
</tr>
</tbody>
</table>

The difference between two consecutive numbers, the resolution, is thus \( \frac{1}{2^M} \).

A special case is obtained when \( N = M + 1 \). There is only one bit left of the binary point: the weighted sign bit which is either 0 or -1. All other bits are used to represent a fractional number less than 1. The exact range in this case can be calculated with

\[
-1.0 \leq N_{S\text{FRAC}} \leq \left(1 - \frac{1}{2^M}\right)
\]

Because of the usually lower number of bits compared with e.g. the number of bits used in MATLAB calculations, the conversion from MATLAB’s “doubles” into fixed-point values will most certainly result in quantization errors (worst case errors amounts to \( 2^{-M} \) in case the translation is by truncation, or maximally \( 2^{N-M-1} \) in case of rounding).

Together, all these quantization errors determine the final resulting computational accuracy.

In our software we use the notation \([N M]\) to indicate a signed value with totally \( N \) bits, of which \( M \) are used for representing the fraction part: exactly as has been described above.

During the calculations, it is possible that intermediate computational results could only be represented using more than \( N \) bits. This can be the case e.g. when summing a number of positive and negative values, where the resulting value is known to always fit in the \( N \) bits. In such a case, we can increase the number of bits inside the SSG to the left of the MSB with \( x \) bits and denote it as \([N M x]\). Instead of \( N-M \) bits for the whole part, we now use \( N+x-M \) bits for the whole part. The software takes care for an appropriate handling of the sign-bit.

In the ALU.m, MUL.m and resources_reg.vhd files, one can see how the computations are exactly implemented.

Usually, we will use the hexadecimal (hex) format (also without a visual binary point) instead of the pure binary representation.

From a computational point of view, it should not be assumed that the data busses in the (S)SG need to have the same width in the feedback layer and the outside world. If intermediate computational
results in the (S)SG would need additional bits, the bus inside the (S)SG can be made wider than the external bus. This, however, is not influencing the cir-file and the scheduling process. It does become crucial when simulating the circuit in MATLAB or VHDL. Indeed, it is with these functions, that this information has to be passed in the form of the fixed-point variables \([N \ M]\) and \([N \ M \ x]\). Here \([N \ M]\) defines the externally used fixed-point data representation, e.g. a bus width of \(N\) bits (see Figure 3), while inside the (S)SG a bus width of \((N + x)\) bits will be maintained. It is the designer’s responsibility that the final output values of the (S)SG fit again in the \(N\) bits external bus without corrupting the data. See the descriptions of \texttt{gen_mTB} and \texttt{gen_VHD} for the syntax to be used to pass the data format definition.
Setup of the .INP- and .OUT-files

The .INP file is an ASCII text file, that can be created with any plain text editor or with the aid of the gen_INP MATLAB utility.

The .INP file contains

- optionally a number of comment lines (indicated by starting with -- ),
- all coefficients in hexadecimal format,
- optionally a number of comment lines,
- all input data in hexadecimal format.

The sequence in which the values of the coefficients should be entered, has been written in the MATLAB command window when gen_mTB had been run (normally, this is according to the identifiers of the coefficients sorted in ascending order).

Each data input line represents an input value for an (external) input port. If the circuit contains more than one input, the data lines represent the data for every input port for a particular sample.

If all inputs have been handled, the data for the next sample follows, again for each input port.

An example of a .INP - listing for a 5th order FIR-filter (6 coefficients) with one input (i0) and one output (o5) is shown below.

```
-- [17 15] fixed point format
-- coefficients: c0,c1,c2,c3,c4,c5
x"1F654"
x"00BAF"
x"03D0E"
x"03D0E"
x"00BAF"
x"1F654"
-- input function
x"00000"
x"00000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
```

. INP-file format  ( both for MATLAB and VHDL)  

- start with the constant coefficients,
- followed by the input data sequence to be investigated
- comment lines are allowed before the coefficients block and before the inputs block ( in ‘--’ VHDL format )
- hex VHDL format should be used for all data

```
input values ( for i0 in this example ) are read one by one, while each of them is followed by one pass through the SSG and result in an output value ( from o5 ) in ‘FIR5.OUT’
```

The hex values above hold true for a [17 15] signed 2’s complement fixed-point format, e.g. 2 bits for the integer part (of which the MSB reflects the weighted sign-bit) and 15 bits for the fractional part. 

So x"08000" means 0_1.000_0000_0000_0000, which is a '1' (unit step function as an input, starting at n = 2). It will be clear that x"1F654" (the first and last coefficients) represents a negative number since its sign bit is set.
In the .OUT-file, the same hex format will be used. Output is also written line by line, e.g. one data line for each output resulting from the same input sample, then the same procedure for the next sample, and so on. In case only one output is used, consecutive outputs thus are written on consecutive lines. When each sample generates more outputs, these output values are each time followed by a ‘---’ separation line to ease readability. The .OUT-files are started with two comment lines, which a.o. lists the format in which the hex data has been written.

SFxd3 = [32 30 2]
Outputs resp.: o_hp, o_lp
x"00000000"
---
x"1C0B62A2"
x"01CB05A7"
---
x"F533735D"
x"081B98CA"
---
x"FB4CD13E"
x"151FFD5E"
---
x"09A3A9B6"
x"28272F0D"
---

-- VHDL testbench output: 
-- Format [32 30 (2)]
-- Outputs resp.: o_hp, o_lp
x"00000000"
---
x"1C0B62A2"
x"01CB05A7"
---
x"F533735D"
x"081B98CA"
---
x"FB4CD13E"
x"151FFD5E"
---
x"09A3A9B6"
x"28272F0D"
---

Figure 3. Comparable .OUT-files created with a MATLAB (left) and a VHDL (right) testbench for a circuit with two outputs (o_hp and o_lp).

By default, only external outputs are shown. If it is desired to also show one or more of the other SSG-outputs, this needs running gen_mTB and/or gen_VHD with these additional outputs given as the very last argument. See the descriptions of gen_mTB and gen_VHD for more details.
Overview of the Software Environment

Both `gen_mTB` and `gen_VHD` will write a number of files in specific directories that will be created automatically when the programs are used for the first time.

Suppose that you are working with a .cir-file called ‘NAME.cir’ and that you saved this file in the directory ‘$YOUR_CIR_DIR’, then after running `gen_mTB`, a new directory will have been created with the following files in it:

```
<$YOUR_CIR_DIR>/NAME/matlab / testbench_NAME_auto.m
TB_NAME_auto.m
NAME.INP
```

`gen_VHD` will result in the following set-up:

```
<$YOUR_CIR_DIR>/NAME/vhdl / testbench_NAME_auto.vhd
NAME_auto.vhd
NAME_SSG_auto.vhd
resources_reg.vhd
txt_util2.vhd
NAME.INP
```

All 6 files are required for simulation

These 3 files are needed for synthesis

Both testbenches will write a file NAME.OUT in the directory from where they are run.
## Categorical Listing of Functions

**MATLAB resource descriptions simulating their VHDL behavior**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Simulate the behavior of the VHDL description of a registered ALU.</td>
</tr>
<tr>
<td>MUL</td>
<td>Simulate the behavior of the VHDL description of a MULtiplier.</td>
</tr>
</tbody>
</table>

**Scheduling functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALAP</td>
<td>Find the SSG using the ALAP method.</td>
</tr>
<tr>
<td>ASAP</td>
<td>Find the SSG using the ASAP method.</td>
</tr>
<tr>
<td>forced</td>
<td>Find the SSG using the Force Directed Scheduling method.</td>
</tr>
<tr>
<td>listSched</td>
<td>Find the SSG using a List Scheduling method.</td>
</tr>
</tbody>
</table>

**Transformation functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>retime_minCycles</td>
<td>Find circuit descriptions that need less computation cycles.</td>
</tr>
</tbody>
</table>

**Fixed-point translation and manipulation functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixp2hex</td>
<td>Convert a signed value to a hex string represented by fxd-bits.</td>
</tr>
<tr>
<td>hex2fixp</td>
<td>Convert a hex string given by fxd-bits into its signed decimal equivalent.</td>
</tr>
<tr>
<td>toSFixp</td>
<td>Converts a signed fractional value to fit in SFxd bits.</td>
</tr>
<tr>
<td>toUFixp</td>
<td>Converts a signed fractional value to fit in UFxd bits.</td>
</tr>
</tbody>
</table>

**Info, Graph and schedule viewers**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cirInfo</td>
<td>Extract and display some information from a .cir-file.</td>
</tr>
<tr>
<td>read_OUT</td>
<td>Convert the hex data in an .OUT-file into decimal format.</td>
</tr>
<tr>
<td>schedGUI</td>
<td>M-file for schedGUI.fig</td>
</tr>
<tr>
<td>showDistrib</td>
<td>Plot resource usage versus clock STATEs.</td>
</tr>
<tr>
<td>showGraph</td>
<td>Converts 'graph.dot' to an image and opens a viewer to show it.</td>
</tr>
<tr>
<td>view_cir_IO</td>
<td>Graphical view of input circuit description.</td>
</tr>
<tr>
<td>view_DFG</td>
<td>Graphical view of a Data Flow Graph.</td>
</tr>
<tr>
<td>xplore</td>
<td>Plot design evaluation space information for a cir-file description.</td>
</tr>
</tbody>
</table>
## Testbench and MATLAB / VHDL file generators

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen_mTB</td>
<td>Create MATLAB reference testbench files.</td>
</tr>
<tr>
<td>gen_VHD</td>
<td>Create testbench, wrapper and SSG-module VHDL-files.</td>
</tr>
</tbody>
</table>

## Utilities, mostly for internal use only

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axDrag2</td>
<td>Pan and zoom with mouse and simple keystrokes.</td>
</tr>
<tr>
<td>change_CirFile_m2nT</td>
<td>Part of the retiming routine to write an updated cir-file.</td>
</tr>
<tr>
<td>CIRC2cir</td>
<td>Reconstruct a cir-file from the internal data structures.</td>
</tr>
<tr>
<td>CIRC2t_G</td>
<td>Derive retiming information from the internal data structures.</td>
</tr>
<tr>
<td>fConfig</td>
<td>Directory information concerning the Graphviz executable(s)</td>
</tr>
<tr>
<td>hpgPlot</td>
<td>Plot a .hpg-file in schedGUI's preview window.</td>
</tr>
<tr>
<td>lifeTimes</td>
<td>For internal use only (registered outputs in resources assumed).</td>
</tr>
<tr>
<td>mapResources</td>
<td>For internal use only (registered outputs in resources assumed).</td>
</tr>
<tr>
<td>parse</td>
<td>Read and convert a .cir-file into internal data format.</td>
</tr>
<tr>
<td>schedule</td>
<td>Determines a time schedule (SSG).</td>
</tr>
<tr>
<td>showREGs</td>
<td>Plot REGister usage.</td>
</tr>
<tr>
<td>toAdjMat</td>
<td>Converts an interconnection table into an adjacency matrix.</td>
</tr>
</tbody>
</table>

## Graphical User Interface

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedGUI</td>
<td>GUI that combines most of the above mentioned functions.</td>
</tr>
</tbody>
</table>
# Alphabetical Listing of Functions

**Alphabetical Listing of Functions**

*for general use*

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALAP</td>
<td>Find the SSG using the ALAP method.</td>
</tr>
<tr>
<td>ALU</td>
<td>Simulate the behavior of the VHDL description of a registered ALU.</td>
</tr>
<tr>
<td>ASAP</td>
<td>Find the SSG using the ASAP method.</td>
</tr>
<tr>
<td>cirInfo</td>
<td>Extract and display some information from a .cir-file.</td>
</tr>
<tr>
<td>fixp2hex</td>
<td>Convert a signed value to a hex string represented by fxd-bits.</td>
</tr>
<tr>
<td>forceD</td>
<td>Find the SSG using the Force Directed Scheduling method.</td>
</tr>
<tr>
<td>gen_INP</td>
<td>Write data into the correct format for an .INP-file.</td>
</tr>
<tr>
<td>gen_mTB</td>
<td>Create MATLAB reference testbench files.</td>
</tr>
<tr>
<td>gen_VHD</td>
<td>Create testbench, wrapper and SSG-module VHDL-files.</td>
</tr>
<tr>
<td>hex2fixp</td>
<td>Convert a hex string given by fxd-bits into its signed decimal equivalent.</td>
</tr>
<tr>
<td>listSched</td>
<td>Find the SSG using a List Scheduling method.</td>
</tr>
<tr>
<td>MUL</td>
<td>Simulate the behavior of the VHDL description of a MULtiplier.</td>
</tr>
<tr>
<td>parse</td>
<td>Read and convert a .cir-file into internal data format.</td>
</tr>
<tr>
<td>read_OUT</td>
<td>Convert the hex data in an .OUT-file into decimal format.</td>
</tr>
<tr>
<td>retime_minCycles</td>
<td>Find circuit descriptions that need less computation cycles.</td>
</tr>
<tr>
<td>schedGUI</td>
<td>M-file for schedGUI.fig</td>
</tr>
<tr>
<td>showDistrib</td>
<td>Plot resource usage versus clock STATEs.</td>
</tr>
<tr>
<td>showGraph</td>
<td>Converts 'graph.dot' to an image and opens a viewer to show it.</td>
</tr>
<tr>
<td>toAdjMat</td>
<td>Converts an interconnection table into an adjacency matrix.</td>
</tr>
<tr>
<td>toSFixp</td>
<td>Converts a signed fractional value to fit in SFxd bits.</td>
</tr>
<tr>
<td>toUFixp</td>
<td>Converts a signed fractional value to fit in UFxd bits.</td>
</tr>
<tr>
<td>view_cir_IO</td>
<td>Graphical view of input circuit description.</td>
</tr>
<tr>
<td>view_DFG</td>
<td>Graphical view of a Data Flow Graph.</td>
</tr>
<tr>
<td>xplore</td>
<td>Plot design evaluation space information for a cir-file description.</td>
</tr>
</tbody>
</table>
for internal use only

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axDrag2</td>
<td>Pan and zoom with mouse and simple keystrokes.</td>
</tr>
<tr>
<td>change_CirFile_m2nT</td>
<td>Part of the retiming routine to write an updated cir-file.</td>
</tr>
<tr>
<td>CIRC2cir</td>
<td>Reconstruct a cir-file from the internal data structures.</td>
</tr>
<tr>
<td>CIRC2t_G</td>
<td>Derive retiming information from the internal data structures.</td>
</tr>
<tr>
<td>fConfig</td>
<td>Directory information concerning the Graphviz executable(s)</td>
</tr>
<tr>
<td>hpgPlot</td>
<td>Plot a .hpg-file in schedGUI's preview window.</td>
</tr>
<tr>
<td>lifeTimes</td>
<td>For internal use only (registered outputs in resources assumed).</td>
</tr>
<tr>
<td>mapResources</td>
<td>For internal use only (registered outputs in resources assumed).</td>
</tr>
<tr>
<td>schedule</td>
<td>Determines a time schedule (SSG).</td>
</tr>
<tr>
<td>showREGs</td>
<td>Plot REGister usage.</td>
</tr>
</tbody>
</table>
Purpose

Find the SSG using the ALAP method.

Syntax

\[ tFrames = \text{ALAP}(\text{adjMat}, \text{delayVec}) \]

Description

\[ tFrames = \text{ALAP}(\text{adjMat}, \text{delayVec}) \]
returns a two-column vector \( tFrames \) describing the SSG (Scheduled Sequencing Graph) of the operations when scheduled according to the ALAP (As Late As Possible) method (no resource constraints).

The unscheduled flow has to be given in \( \text{adjMat} \) with the clock delays per resource type (e.g. MUL, ALU) as integer number of clock cycles specified in \( \text{delayVec} \).

Examples

See Also

ASAP
forceD
listSched
parse
toAdjMat

Find the SSG using the ASAP method.

Find the SSG using the Force Directed Scheduling method.

Find the SSG using the List Scheduling method.

Read and convert a .cir-file into internal data format.

Converts an interconnection table into an adjacency matrix.
ALU

Purpose

Simulate the behavior of the VHDL description of a registered ALU.

Syntax

\[
\text{result} = \text{ALU}(\text{op1}, \text{op2}, \text{fxd}, \text{opcode})
\]

Description

\[
\text{result} = \text{ALU}(\text{op1}, \text{op2}, \text{fxd}, \text{opcode})
\]
returns the result of the operation as specified in \text{opcode}. Here \text{op1} and \text{op2} are two signed fixed-point input variables, while \text{result} is the output in the same signed fixed-point format. This format has to be given in the vector \text{fxd} as \([N M]\), where \(N\) denotes the total number of bits, while \(M\) defines the number of fractional bits. Valid opcodes are \text{'add'} and \text{'sub'}.

\[
[\text{result}, \text{overflow}] = \text{ALU}(\text{op1}, \text{op2}, \text{fxd}, \text{opcode}, \text{ovMode})
\]
also signals the correctness of the result in \text{'overflow'}. A zero means that the result is unaltered and correct, \text{overflow} = 1 means that the result has been 'wrapped' (e.g. bits left of msb has been skipped) if \text{ovMode} = \text{'wrap'} (the default choice), or has been saturated to its highest positive value or lowest negative value when \text{ovMode} = \text{'sat'} has been specified.

Examples

See Also

\text{MUL} 

simulate the behavior of the VHDL description of a registered multiplier.
ASAP

Purpose

Find the SSG using the ASAP method.

Syntax

tFrames = ASAP(adjMat, delayVec)

Description

tFrames = ASAP(adjMat, delayVec) returns a two-column vector tFrames describing the SSG (Scheduled Sequencing Graph) of the operations when scheduled according to the ASAP (As Soon As Possible) method (no resource constraints).

The unscheduled flow has to be given in adjMat with the clock delays per resource type (e.g. MUL, ALU) as integer number of clock cycles specified in delayVec.

Examples

See Also

ALAP
Find the SSG using the ALAP method.

forceD
Find the SSG using the Force Directed Scheduling method.

listSched
Find the SSG using the List Scheduling method.

parse
Read and convert a .cir-file into internal data format.

toAdjMat
Converts an interconnection table into an adjacency matrix.
**cirInfo**

**Purpose**
Extract and display some information from a .cir-file.

**Syntax**
`cirInfo(cirFilename)`

**Description**
cirInfo lists in the console window the total number of operations, and the numbers of multiplications, ALU operations, coefficients, and delay elements. Also the number of external inputs and outputs are listed, followed by the input and output identifier names. cirInfo can be used as a first test to check a .cir-file for the absence of errors.

**Examples**
```
>> cirInfo('FIR5.cir')
totally 11 operations, of which
    6 multiplications, and
    5 ALU operations.
    6 constant coefficients
    5 delay elements
1 input(s) : i0
1 output(s): o5
```

**See Also**

Scheduling Toolbox for MATLAB  Reference Guide
fixp2hex

Purpose
Convert a signed value to a hex string represented by fxd-bits.

Syntax
hexStr = fixp2hex(decVal, fxd)

Description
hexStr = fixp2hex(decVal, fxd) checks whether the decimal figure decVal
fits in the given fxd bits and if so, returns its hexadecimal representation as a
string.
fxd is supposed to be a two-element vector [N M], where N defines the total
number of available bits and M defines the number of bits to the right of the
binary point.
If M is too low to exactly represent decVal, decVal will be truncated.
An error message is issued when N - M should be too low to represent the signed
integer part of decVal.
decVal itself can be a vector of fractional decimals.

Examples

See Also
hex2fixp     Convert a hex string given by fxd-bits into its signed decimal
equivalent.
toSFixp     Converts a signed fractional value to fit in SFxd bits.
toUFixp     Converts an unsigned fractional value to fit in UFxd bits.
forceD

Purpose
Find the SSG using the Force Directed Scheduling method.

Syntax
```
tFrames = forceD(adjMat, mulOps)
```

Description
```
tFrames = forceD(adjMat, mulOps)  returns a two-column vector tFrames describing the SSG (Scheduled Sequencing Graph) of the operations when scheduled according to the Force Directed Scheduling method with optimal resource distribution.
```

NOTE: At the moment, the latency of the resources (MUL, ALU) is expected to be one clock cycle.

The unscheduled flow has to be given in adjMat, while in the vector mulOps the resource type of the operations should be specified (a 1 means a multiplication).

Examples

See Also
```
ALAP          Find the SSG using the ALAP method.
ASAP          Find the SSG using the ASAP method.
lstSched      Find the SSG using the List Scheduling method.
pars          Read and convert a .cir-file into internal data format.
toAdjMat      Converts an interconnection table into an adjacency matrix.
```

Reference
```
```
Purpose
Create a .INP-file and fill it with data in the correct format.

Syntax
\[
gen_INP(inpFilename,fxd,coeffs,inpSig)
\]

Description
\[
gen_INP(inpFilename,fxd,coeffs,inpSig) creates the file inpFilename and writes sequentially all coefficients line by line, and next inpSig line by line in a VHDL hex representation of the fxd format to this file.

The format used, the start of the coefficients section and the start of the input section will be marked with comment lines (VHDL style).

All data values are ‘rounded’ before they are converted to fixed-point.

coeffs should be a cell array giving the coefficients' names and values, i.e.
\[
cell array: \{coeff_name\}  \{coeff_value\}
\]

(this is the same format as returned by the (L)WDF Filter Toolbox functions \[wdf2cir.m \] and/or \[LWDF2cir.m \] )

Be sure to use the same sequence as is expected by gen_mTB and gen_VHD (the correct sequence is also listed in \[InOuts.constCoeffs\], returned by \[parse.m\]).

Examples
\[
>> \text{load coeffs_FIR5.mat} \\
>> \text{coeffs} \\
\text{coeffs =} \\
\begin{align*}
'c0' & [-0.07556556070608] \\
'c1' & [ 0.09129209297815] \\
'c2' & [ 0.47697917208036] \\
'c3' & [ 0.47697917208036] \\
'c4' & [ 0.09129209297815] \\
'c5' & [-0.07556556070608]
\end{align*}
\]
\[
>> \text{gen_INP(} 'FIR5.INP', [17 15], coeffs, -1:0.4:1 ) \\
>> \text{type FIR5.INP} \\
\text{-- [17 15] fixed-point format} \\
\text{-- coefficients: c0,c1,c2,c3,c4,c5} \\
\begin{align*}
x'1F654' \\
x'00BAF' \\
x'03D0E' \\
x'03D0E' \\
x'00BAF' \\
x'1F654' \\
\end{align*}
\text{-- input function} \\
\begin{align*}
x'18000' \\
x'1B333' \\
x'1E666' \\
x'0199A' \\
x'04CCD' \\
x'08000'
\end{align*}
\]

Scheduling Toolbox for MATLAB Reference Guide 24
.INP-file format  See Chapter “INP and OUT-files”.
and usage

See Also  read_OUT  Convert the hex data in an .OUT-file into decimal format.
**gen_mTB**

**Purpose**
Create MATLAB reference testbench files.

**Syntax**
```
gen_mTB(cirFilename, toFile, schedMethod, varargin)
coeffsSeq = GEN_MTB(cirFilename, toFile, schedMethod, varargin)
```

**Description**
`gen_mTB(cirFilename, toFile, schedMethod, varargin)` creates two MATLAB files:
a description of the SSG given in `cirFilename`, and a testbench-file which calls this SSG.
If `toFile` is 1, these are written to files on disk; if 0, they are only listed in the MATLAB command window. If `cirFilename` would be `NAME.cir`, they can be found in (a newly created directory) `NAME\matlab`.

- The (top-level) testbench-file will be named `testbench_NAME_auto.m` (see its HELP function), the SSG-file `TB_NAME_auto.m`. Input data for the testbench is expected to be found (line by line) in a (user created) file named `NAME.INP`, and output will be written to a file `NAME.OUT`, both in the directory `NAME\matlab`.
- `schedMethod` specifies the scheduling method to be used, and can be 'ASAP', 'ALAP', 'forceD' or 'LIST'.
- The parameters that follow `schedMethod` (indicated with `varargin` here) are dependant on the method chosen, and may be values for `delayMUL`, `delayALU`, `nMULs` and/or `nALUs`:
  - `delayMUL` and `delayALU` are the latencies of resp. MULTIpliers and ALUs in integer multiples of a clock cycle. They are optional for 'ASAP' and 'ALAP' (default, if not specified, is 1 cycle each). At this moment they are not needed for 'forceD' (both fixed to 1). For the 'LIST' method, each of the delays needs to be specified.
  - `nMULs` and `nALUs` are the number of available MULTIpliers and ALUs, and have to be specified for the 'LIST' method only.

As the last (or the only) argument following `schedMethod`, one or more feedback output names may be defined that should also be listed in the .OUT-file, next to the external outputs that are written by default. This argument should be in the form of a cell array of strings (or just as a string if only one output is added).

`coeffsSeq = gen_mTB(cirFilename, toFile, schedMethod, varargin)` also returns the names and the order of the coefficients that are expected in the .INP-file.

**NOTE 1:** `gen_mTB` expects to be run from the directory in which the .cir-file resides.

**NOTE 2:** The fixed-point format of the .INP-file should always match the format that is passed to the testbench.
Examples

```matlab
gen_mTB( 'my_file.cir', 0, 'asap' )
gen_mTB( 'my_file.cir', 1, 'alap', 2, 1 )
gen_mTB( 'my_file.cir', 1, 'alap', 2, 1, {'o1' 'o2'} )
gen_mTB( 'my_file.cir', 1, 'forceD' )
gen_mTB( 'my_file.cir', 1, 'forceD', 'of1' )
coeffsSeq = gen_mTB( 'my_file.cir', 1, 'list', 1, 1, 3, 2 )
```

```matlab
>> help testbench_my_file_auto
Syntax: TESTBENCH_my_file_AUTO(SFxd3,withPrint,DBG)
SFxd3 defines width of signed fractional fixed-point databus
in vector [N M x], where N is the total number of bits of the
external I/O bus from which M are fractional bits.
x defines an additional number of bits with which N should be
extended inside the SSG.
If withPrint = 1, output data is printed to screen.
If DBG = 1, intermediate results are printed.
Needs 'MY_FILE.INP' in current directory to read input data from,
writes 'MY_FILE.OUT' with results.
```

.INP-file format  See Chapter “INP and OUT-files”.

See Also  
- gen_INP Create a .INP-file and fill it with data
- gen_VHD Create testbench, wrapper and SSG VHDL-files.
- read_OUT Convert the hex data in an .OUT-file into decimal format.
**gen_VHD**

**Purpose**  
Create testbench, wrapper and SSG-module VHDL-files.

**Syntax**  
```matlab
gen_VHD(cirFilename, SFxd3, toFile, schedMethod, varargin)
```

**Description**  
`gen_VHD(cirFilename, SFxd3, toFile, schedMethod, varargin)` creates the VHDL files needed for simulation and synthesis (a testbench, a wrapper and the SSG-module VHDL-files).  
`SFxd3` should be a 3-element vector `[N M x]`, in which `N` indicates the external buswidth (signed fixed-point), `M` the number of bits of `N` to be used for the fraction part, and `x` an additional number of bits for extending (the whole-number part of) `N` inside the SSG (to allow for intermediate results greater or less than can be represented with `N` bits).  
If `toFile` is 1, these are written to files on disk; if 0, they are only listed in the MATLAB command window.  
If `cirFilename` would be `NAME.cir`, they can be found in (a newly created directory) `NAME\vhdl`.  
The (top-level) testbench-file will be named `testbench_NAME_auto.vhd`, the wrapper file `NAME_auto.vhd`, and the SSG-file `NAME_SSG_auto.vhd`.  
For simulation all vhd-files placed in the directory are needed, e.g. also the files `resources_regd.vhd` and `txt_util2.vhd`.  
Input data for the testbench is expected to be found (line by line) in a (user created) file named `NAME.INP`, and output will be written to a file `NAME.OUT`.  
Synthesis needs the files `resources_reg.vhd`, `NAME_SSG_auto.vhd` and as top level file `NAME_auto.vhd`.  

`schedMethod` specifies the scheduling method to be used, and can be 'ASAP', 'ALAP', 'forced' or 'LIST'.  
The parameters that follow `schedMethod` (indicated with `varargin` here) are dependant on the method chosen, and may be values for `delayMUL`, `delayALU`, `nMULs` and/or `nALUs`:  
- `delayMUL` and `delayALU` are the latencies of resp. MULTipliers and ALUs in integer multiples of a clock cycle. They are optional for 'ASAP' and 'ALAP' (default, if not specified, is 1 cycle each). At this moment they are not needed for 'forced' (both fixed to 1). For the 'LIST' method, each of the delays needs to be specified.  
- `nMULs` and `nALUs` are the number of available MULTipliers and ALUs, and have to be specified for the 'LIST' method only.

As the last (or the only) argument following `schedMethod`, one or more feedback output names may be defined that should also be listed in the .OUT-file, next to the external outputs that are written by default.  
This argument should be in the form of a cell array of strings (or just as a string if only one output is added).

**NOTE 1:** `gen_VHD` expects to be run from the directory in which the .cir-file resides.  
**NOTE 2:** The fixed-point format of the .INP-file should always match the format that is passed to the testbench.
Examples

```
    gen_VHD( 'my_file.cir', [16 15 0], 0, 'asap' )
gen_VHD( 'my_file.cir', [16 15 2], 1, 'alap', 2, 1 )
gen_VHD( 'my_file.cir', [16 15 2], 1, 'alap', 2, 1, {'o1'; 'o2'} )
gen_VHD( 'my_file.cir', [32 30 4], 1, 'forceD' )
gen_VHD( 'my_file.cir', [32 30 1], 1, 'forceD', 'ofl' )
gen_VHD( 'my_file.cir', [24 20 8], 1, 'list', 1, 1, 3, 2 )
```

A command

```
    >> gen_VHD( 'FIR5.cir', [17 15 1], 1, 'asap' )
```

may result in an entity definition in `FIR5_auto.vhd` that looks like:

```
entity FIR5 is
    generic ( N_g : positive := 17;
              M_g : positive := 15;
              NX_g : positive := 18;
              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;
           C0 : in std_logic_vector(N_g-1 downto 0);
           C1 : in std_logic_vector(N_g-1 downto 0);
           C2 : in std_logic_vector(N_g-1 downto 0);
           C3 : in std_logic_vector(N_g-1 downto 0);
           C4 : in std_logic_vector(N_g-1 downto 0);
           C5 : in std_logic_vector(N_g-1 downto 0);
           I0 : in std_logic_vector(N_g-1 downto 0);
           O5 : out std_logic_vector(N_g-1 downto 0);
           Done : out std_logic;
           Error : out std_logic
    );
end FIR5;
```

See Also

- `gen_INP` - Create a `.INP-file and fill it with data`
- `gen_mTB` - Create MATLAB reference testbench files.
- `read_OUT` - Convert the hex data in an `.OUT-file into decimal format.`
hex2fixp

Purpose
Convert a hex string given by fxd-bits into its signed decimal equivalent.

Syntax
\[ \text{decVal} = \text{hex2fixp}(\text{hexStr}, \text{fxd}) \]

Description
\[ \text{decVal} = \text{hex2fixp}(\text{hexStr}, \text{fxd}) \]
converts \text{hexStr} into a (possibly fractional) signed decimal value in \text{fxd} bits.
\text{fxd} is supposed to be a two-element vector \([N M]\), where \(N\) defines the total number of available bits and \(M\) defines the number of bits to the right of the binary point.
An error message is issued when \text{decVal} cannot be represented in the given \text{fxd} bits.
\text{hexStr} can be an array of strings.

**NOTE:** usually MATLAB’s ‘format long’ will be necessary to represent the result with enough decimals (avoid rounding to ‘short’ format).

Examples

See Also
- \text{fix2hex} Convert a signed value to a hex string represented by fxd-bits.
- \text{toSFfixp} Converts a signed fractional value to fit in SFxd bits.
- \text{toUFfixp} Converts an unsigned fractional value to fit in UFxd bits.
listSched

Purpose

Find the SSG using a List Scheduling method.

Syntax

\[ t_{\text{Frames}} = \text{listSched}(\text{adjMat}, \text{delayVec}, \text{mulOps}, \text{nMULs}, \text{nALUs}) \]

Description

Returns a two-column vector 'tFrames' describing the SSG (Scheduled Sequencing Graph) of the operations when scheduled according to a specified List Scheduling method under resource constraints. The unscheduled flow has to be given in adjMat with the latency per resource type (e.g. MUL, ALU) as integer number of clock cycles specified in delayVec. In the vector mulOps the resource type of the operations should be given (a 1 means a multiplication), and the available numbers of resources in nMULs and nALUs.

Examples

See Also

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALAP</td>
<td>Find the SSG using the ALAP method.</td>
</tr>
<tr>
<td>ASAP</td>
<td>Find the SSG using the ASAP method.</td>
</tr>
<tr>
<td>listSched</td>
<td>Find the SSG using the List Scheduling method.</td>
</tr>
<tr>
<td>parse</td>
<td>Read and convert a .cir-file into internal data format.</td>
</tr>
<tr>
<td>toAdjMat</td>
<td>Converts an interconnection table into an adjacency matrix.</td>
</tr>
</tbody>
</table>
# MUL

**Purpose**  
Simulate the behavior of the VHDL description of a registered MULtiplier.

**Syntax**  
\[ \text{result} = \text{MUL}(\text{op1}, \text{op2}, \text{fxd}) \]

**Description**  
\[ \text{result} = \text{MUL}(\text{op1}, \text{op2}, \text{fxd}) \] returns the (signed) result of the multiplication of \( \text{op1} \) and \( \text{op2} \) (both signed). The binary fixed-point format of \( \text{result} \), as well as that of \( \text{op1} \) and \( \text{op2} \) should be given in the vector \( \text{fxd} \) as \([N M]\), where \( N \) denotes the total number of bits, while \( M \) defines the number of fractional bits.

**Examples**

**See Also**  
**ALU**  
simulate the behavior of the VHDL description of a registered ALU.
parse

Purpose
Read and convert a .cir-file into internal data format.

Syntax

\[
[CIRC\_Data, InOuts] = \text{parse}(\text{inFilename})
\]

Description

\[
[CIRC\_Data, InOuts] = \text{parse}(\text{inFilename})
\]
extracts data from a textual circuit description in a .cir-file. The data returned in the structure CIRC\_Data contains the following fields:

\[
\begin{align*}
\text{CIRC\_Data.iocDef} \\
\text{CIRC\_Data.opNames} \\
\text{CIRC\_Data.opTypes} \\
\text{CIRC\_Data.inpNames} \\
\text{CIRC\_Data.outNames} \\
\text{CIRC\_Data.dlyInps} \\
\text{CIRC\_Data.opsConnTbl} \\
\text{CIRC\_Data.outConnTbl}
\end{align*}
\]

CIRC\_Data.iocDef is the string defining the characters that are used as the first character in an opName to identify inputs, outputs and constant coefficients with. All fields are mainly used for internal purposes.

The data structure InOuts combines the fields:

\[
\begin{align*}
\text{InOuts.constCoeffs} \\
\text{InOuts.extInps} \\
\text{InOuts.ssgInps} \\
\text{InOuts.ssgOutps} \\
\text{InOuts.extOutps}
\end{align*}
\]

The function parse is called by many of the function described in this guide.

Examples

See Also
### read_OUT

**Purpose**
Convert the hex data in an .OUT-file into decimal format.

**Syntax**

```
read_OUT(outFilename)
read_OUT(outFilename, nFracDigits)
read_OUT(outFilename, nFracDigits, withPrint)
y = read_OUT(outFilename, nFracDigits, withPrint)
```

**Description**

- `read_OUT(outFilename, fxd2)` prints the data both in hexadecimal and in decimal format to the MATLAB command window.
- `read_OUT(outFilename, nFracDigits)` can be used to specify the number of decimal fractional digits when printing (default 15).
- `read_OUT(outFilename, nFracDigits, withPrint)` controls the output to the command window: if `withPrint` is 0, output will be suppressed (default is 1).
- `y = read_OUT(outFilename, nFracDigits, withPrint)` also captures the (double precision) decimal data in variable `y` (a column for each output).

**NOTE:** Both .OUT-files created with the MATLAB or the VHDL testbenches can be read. If a v1.0 .OUT-file format is detected (without names of the output signals), this function prompts the user to enter the number of outputs that the .OUT-file represents.

**Examples**

```matlab
>> y = read_OUT('LWDF_5_L\vhdl\LWDF_5_L.OUT', 5);
```

```
Fixed-Point Format: [32 30]
-- Outputs resp.: o_hp, o_lp
1:  x"00000000" =  0.00000  x"00000000" =  0.00000
2:  x"1C0B62A2" =  0.43819  x"01CB05A7" =  0.02802
3:  x"F533735D" = -0.16873  x"081B98CA" =  0.12668
4:  x"FB4CD13E" = -0.07344  x"151FFD5E" =  0.33008
5:  x"09A3A9B6" =  0.15061  x"28272F0D" =  0.62739
```

```matlab
>> y(2:5,:)
```

```
an =
   0.4382   0.0280
  -0.1687   0.1267
 -0.0734   0.3301
  0.1506   0.6274
```

**.OUT-file format**
See Chapter "INP and OUT-files".

**See Also**
- `gen_INP` Write data into the correct format for an .INP-file.
retime_minCycles

Purpose
Find circuit descriptions that need less computation cycles.

Syntax

\[ Gmats,xCIRC\_Data = \text{retime\_minCycles}(\text{cirFilename}, \text{delayMUL}, \text{delayALU}) \]

Description
retime_minCycles(cirFilename, delayMUL, delayALU) reads the .cir-file cirFilename, and uses a retiming algorithm to find –by inserting and/or moving the delay elements– functionally equivalent, adapted circuits which needs less computational cycles to complete its longest path. delayMUL and delayALU are the latencies of resp. MULtipliers and ALUs in integer multiples of a clock cycle. If not specified, default values of 1 are used (if specified, both values should be present).
If retiming turns out to be feasible, retime_minCycles writes actions to be taken to the MATLAB command window and creates a new .cir-file that describes the retimed circuit. The new filename used will be the original .cir-filename, extended with a '_#cyc' postfix (# represents the minimum number of cycles needed).

\[ Gmats,xCIRC\_Data = \text{retime\_minCycles}(\text{cirFilename}, \text{delayMUL}, \text{delayALU}) \]
returns the G-matrices (see the Reference mentioned) belonging to each retimed circuit and the necessary data associated with these matrices (xCIRC_data may differ from the original CIRC_Data).

Examples

See Also

Reference
showDistrib

Purpose
Plot resource usage versus clock STATEs.

Syntax
showDistrib(cirFilename, schedMethod, varargin)

Description
showDistrib(cirFilename, schedMethod, varargin) determines and plot the usage of the MULtipliers and ALUs for the circuit described in cirFilename, when scheduled with the method defined with schedMethod and the optionally additional parameters in varargin.

schedMethod specifies the scheduling method to be used, and can be 'ASAP', 'ALAP', 'forceD' or 'LIST'.

The parameters in varargin are dependant on the method chosen, and may be values for delayMUL, delayALU, nMULs and/or nALUs:
• delayMUL and delayALU are the latencies of resp. MULtipliers and ALUs in integer multiples of a clock cycle. They are optional for 'ASAP' and 'ALAP' (default, if not specified, is 1 cycle each). At this moment they are not needed for 'forceD' (both fixed to 1). For the 'LIST' method, each of the delays needs to be specified.
• nMULs and nALUs are the number of available MULtipliers and ALUs, and have to be specified for the 'LIST' method only.

Examples
>> showDistrib('fir5.cir','forceD')

See Also
**showGraph**

**Purpose**
Converts 'graph.dot' to an image and opens a viewer to show it.

**Syntax**

```matlab
showGraph(gFormat)
```

**Description**

`showGraph(gFormat)` displays the image file that is described in 'graph.dot', using the intermediate file 'graph.xxx' where `xxx` is replaced by the string given in `gFormat`. Valid formats are 'png', 'jpg' and 'hpg'. Usually, 'graph.dot' will have been created by `schedGUI` or by `schedule.m`.

**Examples**

**See Also**
- `view_DFG` Graphical view of a Data Flow Graph.
- `view_cir_IO` Graphical view of cir-file description with In- and Outputs shown.
toAdjMat

Purpose

Converts an interconnection table into an adjacency matrix.

Syntax

adjMat = toAdjMat(opsConnTbl)

Description

adjMat = toAdjMat(opsConnTbl), where opsConnTbl usually will be the CIRC_Data.opsConnTbl output from parse.m. The adjMat is a.o. needed as input for the scheduling routines such as ASAP, ALAP, etc.

Examples

See Also

parse Read and convert a .cir-file into internal data format.
toSFixp

Purpose
Converts a signed fractional value to fit in SFxd bits.

Syntax
\[
\text{SFixValue} = \text{toSFixp}(\text{decVal}, \text{SFxd})
\]
\[
\text{SFixValue} = \text{toSFixp}(\text{decVal}, \text{SFxd}, '\text{round}')
\]

Description
\[
\text{SFixValue} = \text{toSFixp}(\text{decVal}, \text{SFxd}) \text{ truncates (\text{=} towards minus infinity), if needed, the signed decimal fractional } \text{decVal} \text{ to fit in the given } \text{SFxd} \text{ bits and returns it in } \text{SFixValue}.
\]
\[
\text{SFxd} \text{ is supposed to be a two-element vector } [N M], \text{ where } N \text{ defines the total number of available bits and } M \text{ defines the number of bits to the right of the binary point.}
\]
\[
\text{decVal} \text{ can be a vector of signed fractional decimals.}
\]
\[
\text{SFixValue} = \text{toSFixp}(\text{decVal}, \text{SFxd}, '\text{round}') \text{ can be used to do a rounding operation instead of just a truncation.}
\]

Examples

See Also
fixp2hex Convert a signed value to a hex string represented by fxd-bits.
hex2fixp Convert a hex string given by fxd-bits into its signed decimal equivalent.
toUFixp Converts an unsigned fractional value to fit in UFxd bits.
**toUFixp**

**Purpose**
Converts an unsigned fractional value to fit in UFxd bits.

**Syntax**

\[
\text{UFixValue} = \text{toUFixp}(\text{decVal}, \text{UFxd})
\]

**Description**
\[
\text{UFixValue} = \text{toUFixp}(\text{decVal}, \text{UFxd}) \text{ truncates (= towards minus infinity), if needed, the unsigned decimal fractional } \text{decVal} \text{ to fit in the given } \text{UFxd} \text{ bits and returns it in } \text{UFixValue}.
\]
\[
\text{UFxd} \text{ is supposed to be a two-element vector } [N M], \text{ where } N \text{ defines the total number of available bits and } M \text{ defines the number of bits to the right of the binary point. } \text{decVal} \text{ can be a vector of signed fractional decimals.}
\]

**Examples**

**See Also**
- `fixp2hex` Convert a signed value to a hex string represented by fxd-bits.
- `hex2fixp` Convert a hex string given by fxd-bits into its signed decimal equivalent.
- `toSFixp` Converts a signed fractional value to fit in UFxd bits.
**view_cir_IO**

**Purpose**  
Graphical view of cir-file description with In- and Outputs shown.

**Syntax**  
```matlab
view_cir_IO(cirFilename)
view_cir_IO(cirFilename, outType)
view_cir_IO(cirFilename, outType, topBottom)
```

**Description**  
`view_cir_IO(cirFilename)` shows the Sequencing Graph that is extracted from the circuit-file `circFilename` in a platform specific viewer. An intermediate file `graph_io.png` will be written in the current directory.

`view_cir_IO(cirFilename, outType)`, where `outType` is a string, can be used to specify different formats of the `graph_io`-file. Valid formats besides `"png"` are: `"hpg"`, `"jpg"`, ...

If `outType` is empty, the `"png"` format will be used.

`view_cir_IO(cirFilename, outType, topBottom)` can be used to specify whether the in and output connections may be displayed inside the graph (`topBottom = 0`, the default value), or should be put on separate top and bottom lines outside the graph (`topBottom = 1`).

**Examples**

**See Also**  
`view_DFG`  
Graphical view of a Data Flow Graph.

`showGraph`  
Opens a viewer to show image-file 'graph.xxx'.
view_DFG

Purpose

Graphical view of a Data Flow Graph.

Syntax

view_DFG(cirFilename)
view_DFG(cirFilename, delayMUL, delayALU)
view_DFG(cirFilename, delayMUL, delayALU, outType)
view_DFG(cirFilename, delayMUL, delayALU, outType, wExtIO)

Description

view_DFG(cirFilename) shows the Data Flow Graph that is extracted from the circuit-file cirFilename in a platform specific viewer.
An intermediate file graph_DFG.png will be written in the current directory.

VIEW_DFG(cirFilename, delayMUL, delayALU) annotates the graph with the latencies of the operations (default delayMUL = delayALU = 1 clock cycle).

view_DFG(cirFilename, delayMUL, delayALU, outType), where outType is a string, can be used to specify different formats of the graph_io-file.
Valid formats besides 'png' are: 'hpg', 'jpg', ...
If outType is empty, the 'png' format will be used.

view_DFG(cirFilename, delayMUL, delayALU, outType, wExtIO) can be used to specify whether the in and output connections should be displayed with the graph. When wExtIO = 1, all these I/O's are shown (with wExtIO = 0 - the default value- inputs are only displayed if strictly necessary for identifying the circuit.

NOTE: when the .cir-file has been created by retime_minCycles, the number of cycles (states) in the filename is only valid for the delayMUL and delayALU values that have been written in the file. The user will be notified that these values will be used.

Examples

See Also

view_cir_IO    Graphical view of cir-file description with In- and Outputs shown.
showGraph      Opens a viewer to show image-file 'graph.xxx'.
xplore

Purpose
Plot design evaluation space information for a cir-file description.

Syntax
xplore(cirFilename)
xplore(cirFilename,costFacs)
xplore(cirFilename,costFacs,delayMUL,delayALU)

Description
xplore(cirFilename) estimates the area or cost involved by different implementations of cirFilename based on the numbers of MULtipliers and ALUs available (and consequently the number of REGisters needed). All registers are taken into account, i.e. the registers needed for life-time extension, but also the feedback register (if present) and registers for each input and each output.

xplore(cirFilename,costFacs) controls the relative areas, that are taken by MULtipliers, ALUs and REGisters with the vector costFacs. This should be a three element vector giving user definable costs factors for [ mulCost aluCost regCost ], e.g. [ 2 1.2 0.8 ].

xplore(cirFilename,costFacs,delayMUL,delayALU) will perform the scheduling computations with the specified delayMUL and delayALU (both given in integer number of clock cycles).

Examples

See Also

Scheduling Toolbox for MATLAB   Reference Guide
**schedGUI**

The Graphical User Interface `schedGUI` can be used for quickly judging and comparing the results of different scheduling methods and scheduling parameters. It assumes a screen resolution of at least 1280 x 1024 to be displayed completely.

The tool can be started with `schedGUI` or (unfortunately not on MATLAB releases before R14) with `schedGUI('cirDir')`. In case `cirDir` is specified, this will be `schedGUI`'s startup directory, otherwise `schedGUI` is started from the current directory. One of the `.cir`-files present in the startup directory can be selected for a scheduling operation.

Available scheduling methods are ASAP, ALAP, Force Directed and a List method. The List method is the only resource constrained method and needs to be informed how many multipliers and ALUs may be used at the same time.

With the ASAP, ALAP and List method, the latencies of the multipliers and ALUs can be specified (in numbers of integer clock cycles). By default, all resources will have a latency of only one clock cycle.

![Figure 4. Screen-shot of the SchedGUI.](image-url)
The result of the scheduling process, the SSG, (executed after pressing the Preview Scheduled Graph button) will be made visible in the preview window. It is possible to pan and zoom out or in on this SSG with the mouse or some predefined keys (see the Menu item Help --> PreviewPane Options). Together with the SSG, additional information about the distribution of the multipliers and ALUs and the additional number of memory registers is listed. The resulting SSG can then be written to an image file by pressing the Save Graph button. Supported formats in this case are 'png' and 'jpg'. Switching to another cirDir is possible by selecting the Menu item File --> Set Input Folder. There is also a Menu item File --> Print Preview, but this one is not yet fully functional.

Drawing of the graphs is accomplished with the aid of software provided by the Graphviz (Graph Visualization Software) package. This software has originally been developed by AT&T Research, and is extended with additional tools in the course of time. Nowadays, it is licensed on an open source basis under The Common Public License. See also http://www.graphviz.org/
This page intentionally left blank