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

Scheduling Toolbox for MATLAB

Reference Guide

Version 0.9

Ing. H.J. Lincklaen Arriëns
January 2006

Table of Contents

Categorical Listing of Functions

MATLAB resource descriptions simulating their VHDL behavior	
ALU	Simulate the behavior of the VHDL description of a registered ALU.
MUL	Simulate the behavior of the VHDL description of a MULtiplier.

Scheduling functions	
ALAP	Find the SSG using the ALAP method.
ASAP	Find the SSG using the ASAP method.
forceD	Find the SSG using the Force Directed Scheduling method.
listSched	Find the SSG using a List Scheduling method.

Fixed-point translation and manipulation functions	
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 SFxd bits.
toUFixp	Converts a signed fractional value to fit in UFxd bits.

Info, Graph and schedule viewers	
cirInfo	Extract and display some information from a .cir-file.
read_OUT	Convert the hex data in an .OUT-file into decimal format.
schedGUI	M-file for schedGUI.fig
showDistrib	Plot resource usage versus clock STATEs.
showGraph	Converts 'graph.dot' to an image and opens a viewer to show it.
view_cir_IO	Graphical view of input circuit description.
xplore	Plot design evaluation space information for a cir-file description.

Testbench and MATLAB / VHDL file generators	
gen_mTB	Create MATLAB reference testbench files.
gen_VHD	Create testbench, wrapper and SSG-module VHDL-files.

Utilities, mostly for internal use only	
`axDrag2`	Pan and zoom with mouse and simple keystrokes.
`groupIOs`	For internal use only.
`hpgPlot`	Plot a .hpg-file in schedGUI's preview window.
`lifeTimes`	For internal use only (registered outputs in resources assumed).
`mapResources`	For internal use only (registered outputs in resources assumed).
`parse`	Read and convert a .cir-file into internal data format.
`schedule`	Determines a time schedule (SSG).
`showREGs`	Plot REGister usage.
`startup`	Setup paths for msclab_et4054.
`toAdjMat`	Converts an interconnection table into an adjacency matrix.

Graphical User Interface	
`schedGUI`	GUI that combines most of the above mentioned functions.

Alphabetical Listing of Functions

for general use

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

for internal use only

`axDrag2`	Pan and zoom with mouse and simple keystrokes.
`groupIOs`	For internal use only.
`hpgPlot`	Plot a .hpg-file in schedGUI's preview window.
`lifeTimes`	For internal use only (registered outputs in resources assumed).
`mapResources`	For internal use only (registered outputs in resources assumed).
`schedule`	Determines a time schedule (SSG).
`showREGs`	Plot REGister usage.
`startup`	Setup paths for msclab_et4054.

ALAP

Purpose Find the SSG using the ALAP method.

Syntax

```
tFrames = ALAP(adjMat,delayVec)
```

Description `tFrames = ALAP(adjMat,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 `adjMat` with the clock delays per resource type (e.g. `MUL, ALU`) as integer number of clock cycles specified in `delayVec.`

Examples

See Also

`ASAP`	Find the SSG using the ASAP method.
`forceD`	Find the SSG using the Force Directed Scheduling method.
`listSched`	Find the SSG using the List Schedu;ing method.
`parse`	Read and convert a .cir-file into internal data format.
`toAdjMat`	Converts an interconnection table into an adjacency matrix.

ALU

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

Syntax `result = ALU(op1,op2,fxd, opcode)`

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

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

Examples

See Also `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 Schedu;ing 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

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.
`listSched`	Find the SSG using the List Schedu;ing method.
`parse`	Read and convert a .cir-file into internal data format.
`toAdjMat`	Converts an interconnection table into an adjacency matrix.

Reference

Force-Directed Scheduling for the Behavioral Synthesis of ASIC's,
Pierre G. Paulin and John P. Knight,
IEEE Trans on Computer-Aided Design, Vol. 8, No. 6, JUNE 1989, pages 661-679

gen_INP

Purpose Write data into the correct format for an .INP-file.

Syntax `gen_INP(inpFilename,fxd,coeffs,inpSig)`

Description `gen_INP(inpFilename,fxd,coeffs,inpSig)` creates the file `inpFilename` and writes sequentially first 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 and the start of the coefficients and the input sections will be marked with comment lines. All data values are 'rounded' before they are converted to fixed-point.

Examples

```
>> load coeffs_FIR5.mat
>> coeffs
coeffs =
    'c0'    [-0.07556556070608]
    'c1'    [ 0.09129209297815]
    'c2'    [ 0.47697917208036]
    'c3'    [ 0.47697917208036]
    'c4'    [ 0.09129209297815]
    'c5'    [-0.07556556070608]
>> gen_INP( 'FIR5.INP', [17 15], coeffs, -1:0.4:1 )
>> type FIR5.INP

-- [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"18000"
x"1B333"
x"1E666"
x"0199A"
x"04CCD"
x"08000"
>>
```

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

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,toScreen,schedMethod,varargin)
coeffsSeq = GEN_MTB(cirFilename,toScreen,schedMethod,varargin)
```

Description

`gen_mTB(cirFilename,toScreen,schedMethod,varargin)` creates two MATLAB files:
a description of the SSG given in `cirFilename`, and a testbench-file which calls this SSG.
If `toScreen` is 1, these are written to the screen; if 0, they are written to disk.
If `cirFilename` should 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.

`coeffsSeq = gen_mTB(cirFilename,toScreen,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

```
gen_mTB( 'my_file.cir', 1, 'asap' )
gen_mTB( 'my_file.cir', 0, 'alap', 2,1 )
gen_mTB( 'my_file.cir', 0, 'forceD' )
coeffsSeq = gen_mTB( 'my_file.cir', 0, 'list', 1,1, 3,2 )

>> help testbench_my_file_auto
 Syntax: TESTBENCH_ my_file_AUTO(SFxd3,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 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_VHD`	Create testbench, wrapper and SSG VHDL-files.

gen_VHD

Purpose

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

Syntax

```
gen_VHD (cirFilename,SFxd3,toScreen,schedMethod,varargin)
```

Description

`gen_VHD(cirFilename,SFxd3,toScreen,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 `toScreen` is 1, the files are written to the screen; if 0, they are written to disk. If `cirFilename` should 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.

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

The command

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

will 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;
             New_Sample :  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_mTB`	Create MATLAB reference testbench files.

hex2fixp

Purpose

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

Syntax

```
decVal = hex2fixp(hexStr,fxd)
```

Description

`decVal = hex2fixp(hexStr,fxd)` converts `hexStr` into a (possibly fractional) signed decimal value in `fxd` bits.
`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 `decVal` cannot be represented in the given `fxd` bits.
`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

`fixp2hex`	Convert a signed value to a hex string represented by fxd-bits.
`toSFixp`	Converts a signed fractional value to fit in SFxd bits.
`toUFixp`	Converts an unsigned fractional value to fit in UFxd bits.

listSched

Purpose

Find the SSG using a List Scheduling method.

Syntax

```
tFrames = listSched(adjMat,delayVec,mulOps,nMULs,nALUs)
```

Description

`tFrames = listSched(adjMat,delayVec,mulOps,nMULs,nALUs)` 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

`ALAP`	Find the SSG using the ALAP method.
`ASAP`	Find the SSG using the ASAP method.
`listSched`	Find the SSG using the List Schedu;ing method.
`parse`	Read and convert a .cir-file into internal data format.
`toAdjMat`	Converts an interconnection table into an adjacency matrix.

MUL

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

Syntax `result = MUL(op1,op2,fxd)`

Description `result = MUL(op1,op2,fxd)` returns the (signed) result of the multiplication of `op1` and `op2` (both signed).
The binary fixed-point format of `result`, as well as that of `op1` and `op2` should be given in the vector `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 = parse(inFilename)`

Description `CIRC_Data = parse(inFilename)` extracts data from a textual circuit description in a .cir-file. The data returned in the structure `CIRC_Data` contains the following fields.

```
        CIRC_Data.iocDef
        CIRC_Data.iconnTbl
        CIRC_Data.xconnTbl
        CIRC_Data.opTypes
        CIRC_Data.opNames
        CIRC_Data.inpNames
        CIRC_Data.outNames
        CIRC_Data.dlydInps
```

`CIRC_Data.iocDef` is a string defining the characters that are used as the first character in an `opName` to identify inputs, outputs and constant coefficients with.
`CIRC_Data.iconnTbl` is a (number of operations x 4) array that describes the interconnections between the operations. The first column list the operation identification numbers (`opid#`), where the corresponding type and name of an `opid#` can be found in `CIRC_Data.opTypes` and `CIRC_Data.opNames` respectively. Columns 2 and 3 list the `opid#`s of the operations that are connected to this `opid#`'s inputs. A negative sign is used to indicate that the input is connected to an input port (`inp#`), the name of which can be found as entry `inp#` in the cell-vector `CIRC_Data.inpNames`.
Output ports (outp#) are given in column 4, while their names are listed in `CIRC_Data.outNames`.
If no shift operations (<< or >>) are present, `iconnTbl` is a one dimensional (number_of_operations,4) array, else a (number_of_operations,4,2) array.

`opTypes: mul, add, sub, …`

`CIRC_Data.xconnTbl`

`CIRC_Data.dlydInps`

Examples

See Also `GroupIOs` …

read_OUT

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

Syntax

```
y = READ_OUT(outFilename,fxd2)
y = READ_OUT(outFilename,fxd2,nOutputs)
y = READ_OUT(outFilename,fxd2,nOutputs,nFracDigits)
```

Description `y = READ_OUT(outFilename,fxd2)` returns the hex data as read from `outFilename` in a decimal format specified by the vector `fxd2` (`[N M]` fixed-point format). Output is also displayed in the console window.

`y = READ_OUT(outFilename,fxd2,nOutputs)` should be used if the circuit for which `outFilename` is valid, shows more then one outputs. `y` will become a column vector with as many rows as given by `nOutputs` .

`y = READ_OUT(outFilename,fxd2,nOutputs,nFracDigits)` can be used to limit the number of fractional digits that are displayed in the console window (default the sum of the numbers of digits for the whole part, the fractional point and for the fractional part together is 17).

Examples

```
>> o5 = read_OUT('FIR5\vhdl\FIR5.OUT',[17 15],1,5);
  1:  x"00000" =  0.00000
  2:  x"00000" =  0.00000
  3:  x"00000" =  0.00000
  4:  x"1F654" = -0.07556
  5:  x"00203" =  0.01572
  6:  x"03F11" =  0.49271
  7:  x"07C1F" =  0.96970
  8:  x"087CE" =  1.06097
  9:  x"07E22" =  0.98541
 10:  x"07E22" =  0.98541
 11:  x"07E22" =  0.98541
 12:  x"07E22" =  0.98541
 13:  x"07E22" =  0.98541
```

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

See Also

`gen_INP`	Write data into the correct format for an .INP-file.

showDistrib

Purpose Plot resource usage versus clock STATEs.

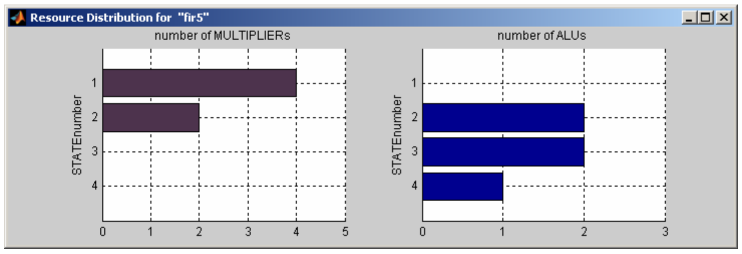
Syntax `showDistrib(cirFilename,schedMethod,varargin)`

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 `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_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(iconnTbl)
```

Description `adjMat = TOADJMAT(iconnTbl)`, where `iconnTbl` usually will be the `CIRC_Data.iconnTbl` 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 `SFixValue = toSFixp(decVal,SFxd)`

Description `SFixValue = toSFixp(decVal,SFxd)` truncates (= towards minus infinity), if needed, the signed decimal fractional `decVal` to fit in the given `SFxd` bits and returns it in `SFixValue`.
`SFxd` 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.
`decVal` can be a vector of signed fractional decimals.

`SFixValue = toSFixp(decVal,SFxd,'round')` 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

```
UFixValue = toUFixp(decVal,UFxd)
```

Description

`UFixValue = toUFixp(decVal,UFxd)` truncates (= towards minus infinity), if needed, the unsigned decimal fractional `decVal` to fit in the given `UFxd` bits and returns it in `UFixValue`.
`UFxd` 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.
`decVal` 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

```
view_cir_IO(cirFilename)
```

Description

`view_cir_IO(cirFilename)` shows the Data Flow Graph that is extracted from the circuit-file `cirFilename` 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 seperate top and bottom lines outside the graph (`topBottom = 1`).

Examples

See Also

`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).

`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).

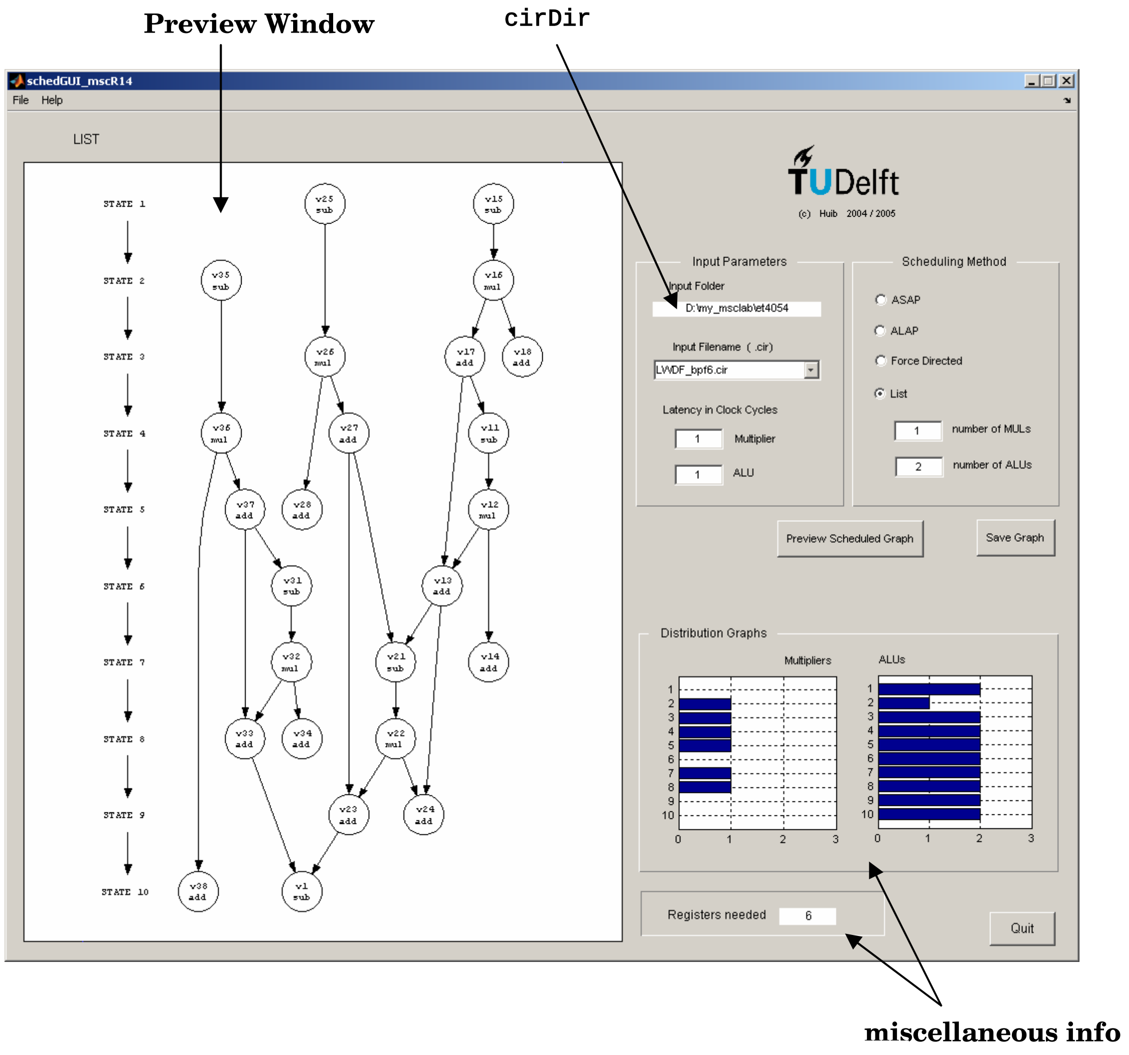
Examples

See Also

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.



Input Parameters
Input Folder: D:\my_msclab\et4054
Input Filename (.cir): LWDF_bpf6.cir
Latency in Clock Cycles
1 Multiplier
1 ALU

Scheduling Method
ASAP
ALAP
Force Directed
[✓ List]
1 number of MULs
2 number of ALUs

Registers needed: 6

Figure 1. *Screen-shot of the SchedGUI.*

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/

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).

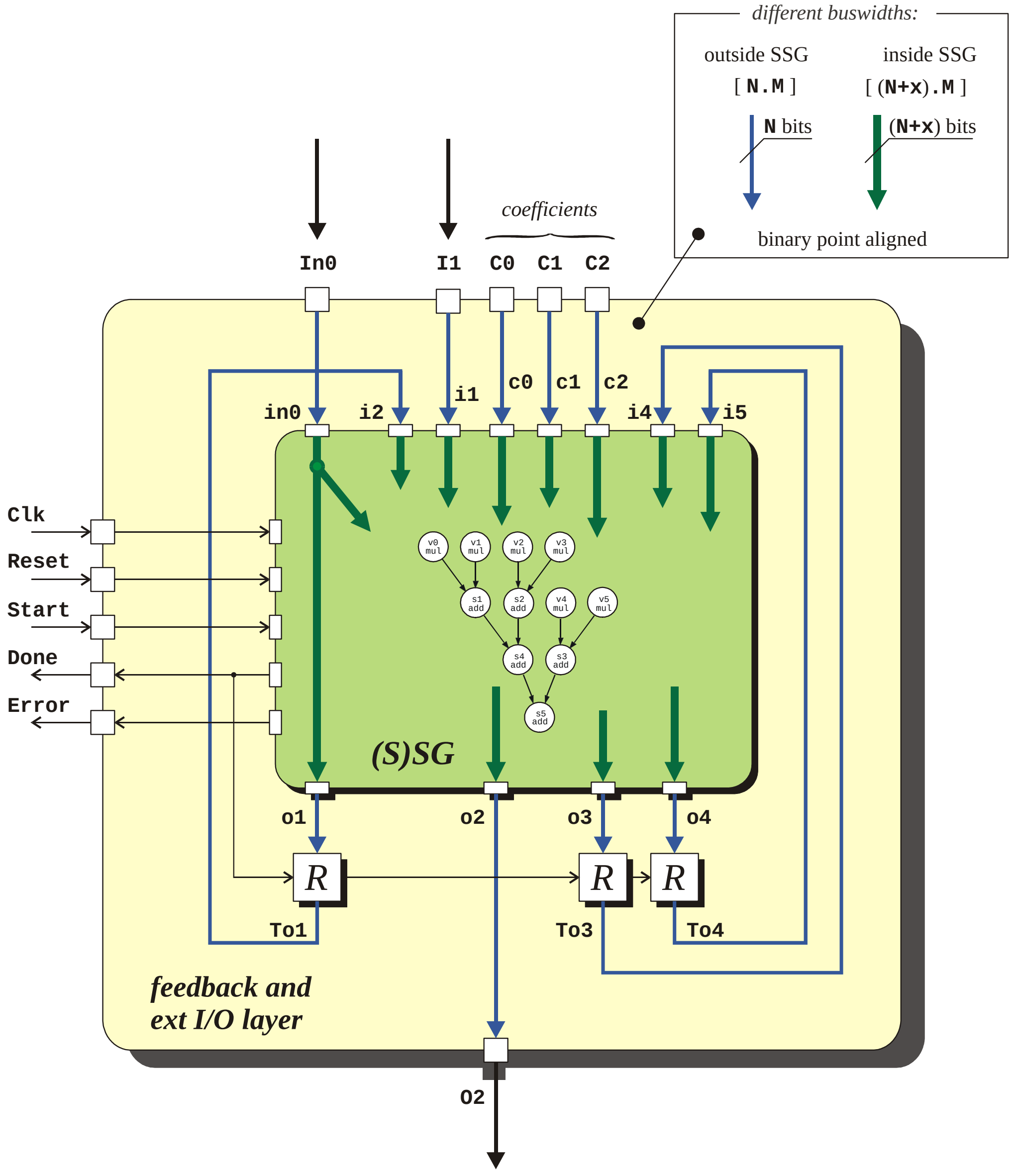


Figure 2. *Concept of the cir-file description.*

The concept of this setup is shown in Figure 2, 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 = **T**output port   which involves a delayed feedback register

and can be optionally terminated with a '`;`'

e.g.
```
        v1 = v2 + v3;
        i2 = To1;
```
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 - i1;
```
but that outputs have to be explicitly connected to (only one) operation, e.g.
```
        o3 = v1;
```

Also note that it is common sense, given the fact that the names are displayed in output plots, to use relatively short identifiers.

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^(-2)
      v1 = (i0 >> 1) - (i0 >> 3);      % v1 = 0.375 * i0;
```

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

```
      iocDef = '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.

```
      iocDef = 'XYab';
```

Remember that identifiers are case-sensitive.

Some remarks concerning the VHDL code for simulation and implementation.

Although not mentioned in the cir file, the VHDL code that will be generated for the circuit will define 5 other 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.

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 **Simulation timing setup**).

In the set-up used here, it is expected that the coefficients are passed to the circuit by means of a **.INP**-file. Changing the transfer characteristic of the filter –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.

From a computational point of view, it cannot be assumed that the data busses in the (S)SG should have the same width as 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]`. See the next Chapter, **Signed fixed-point notation**, for a description of this notation.

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 then 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.

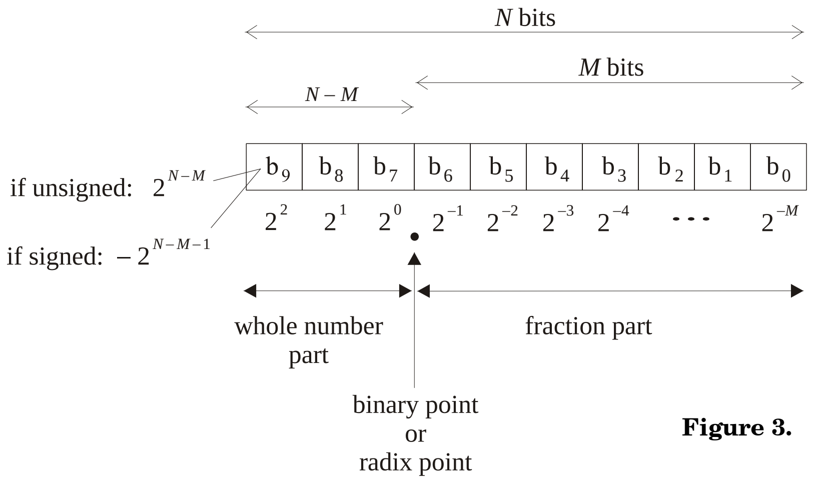
Surely, everyone is acquainted with the all integer representation, i.e. where we are dealing with only positive whole numbers. Since this is a severe limitation in calculations, it is a necessity to also be able to represent negative numbers. For this purpose, the 2’s-complement method is the most widely used representation. In this representation, the MSB is assigned a negative weight (see Table B1). 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 1. *N-bits integer representations (2’s complement for signed)*

	value	range
unsigned	$N_{UINT} = b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \cdots + b_1 s^1 + b_0 s^0$	$0 \le N_{UINT} \le 2^{N-1}$
signed	$N_{SINT} = -b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \cdots + b_1 s^1 + b_0 s^0$	$-2^{N-1} \le N_{SINT} \le \left(2^{N-1} - 1\right)$

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 3, 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.



N bits
M bits
N – M
b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
if unsigned: 2^{N-M}
if signed: -2^{N-M-1}
2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4} ... 2^{-M}
whole number part
fraction part
binary point or radix point

Figure 3. *Fixed-point notation.*

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)*

	value	range
unsigned	$N_{UMIX} = \sum_{i=M}^{N-1} b_i 2^{i-M} + \sum_{j=0}^{M-1} b_j 2^{j-M}$	$0 \le N_{UMIX} \le \left(2^{N-M} - \frac{1}{2^M}\right)$
signed	$N_{SMIX} = -b_{N-1} 2^{N-M-1} + \sum_{i=M}^{N-2} b_i 2^{i-M} + \sum_{j=0}^{M-1} b_j 2^{j-M}$	$-2^{N-M-1} \le N_{SMIX} \le \left(2^{N-M-1} - \frac{1}{2^M}\right)$

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

$$-1.0 \le N_{SFRAC} \le \left(1 - \frac{1}{2^M}\right)$$

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 $\left|2^{-M}\right|$ in case the translation is by truncation, or maximally $\left|2^{-M-1}\right|$ 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 then `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.

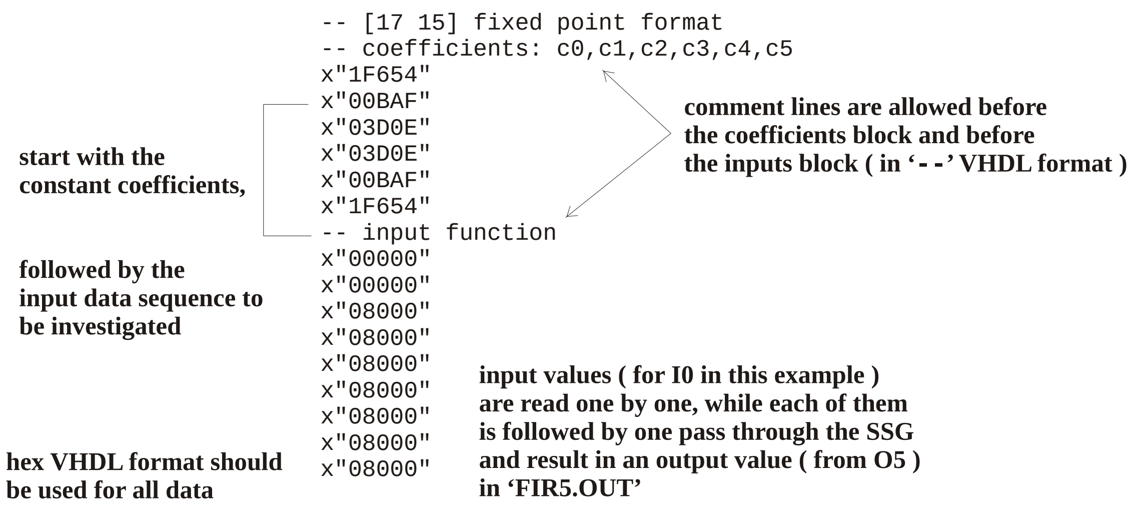
Setup of the .INP- and .OUT-files

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 an input `I0` and an output called `O5` is shown below.

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



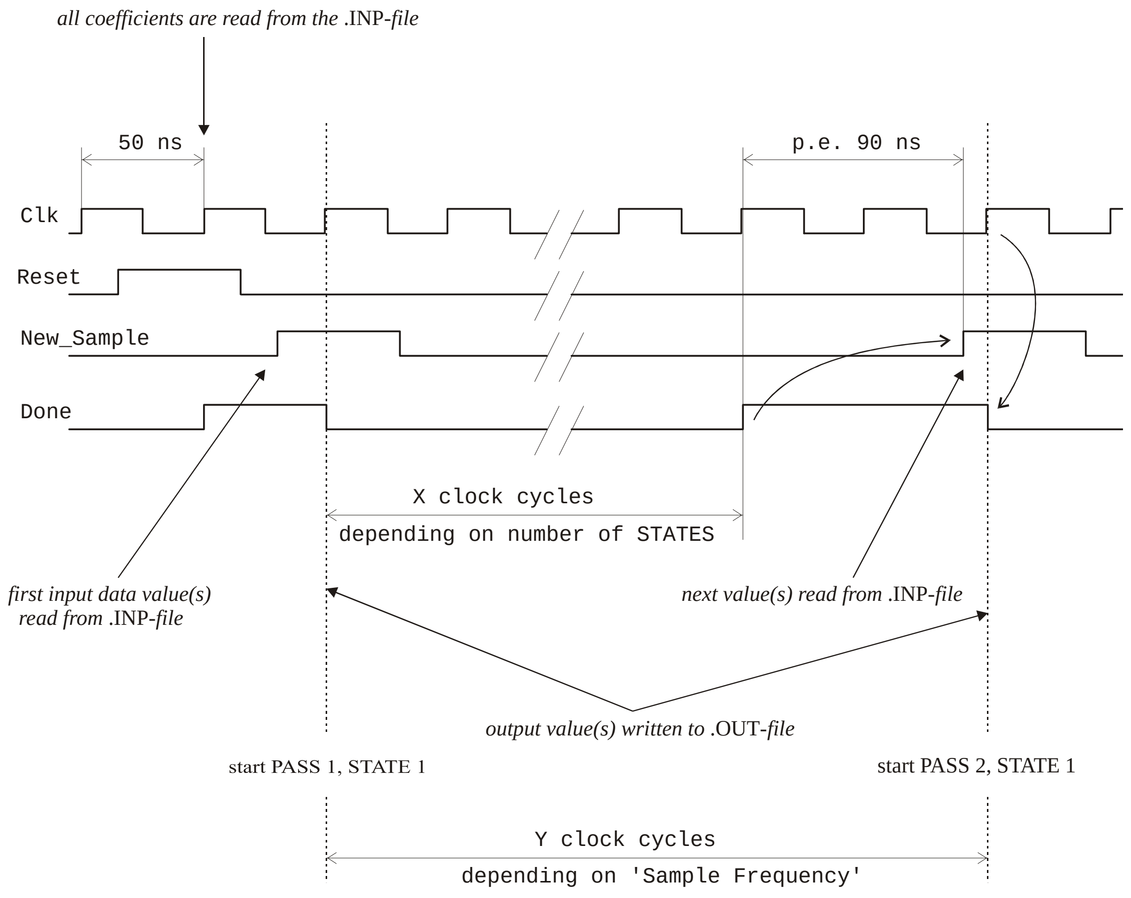
```
-- [17 15] fixed point format
-- coefficients: c0,c1,c2,c3,c4,c5
x"1F654"
x"00BAF"
x"03D0E"
x"03D0E"
x"00BAF"
x"1F654"
-- input function
x"00000"
x"00000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
x"08000"
```

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

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. The .OUT-files are started with two comment lines, which a.o. lists the format in which the hex data has been written.

Simulation timing setup


all coefficients are read from the .INP-file
50 ns
p.e. 90 ns
Clk
Reset
New_Sample
Done
X clock cycles
depending on number of STATES
first input data value(s) read from .INP-file
next value(s) read from .INP-file
output value(s) written to .OUT-file
start PASS 1, STATE 1
start PASS 2, STATE 1
Y clock cycles
depending on 'Sample Frequency'

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 goes high to signal that the SSG is ready and awaiting.

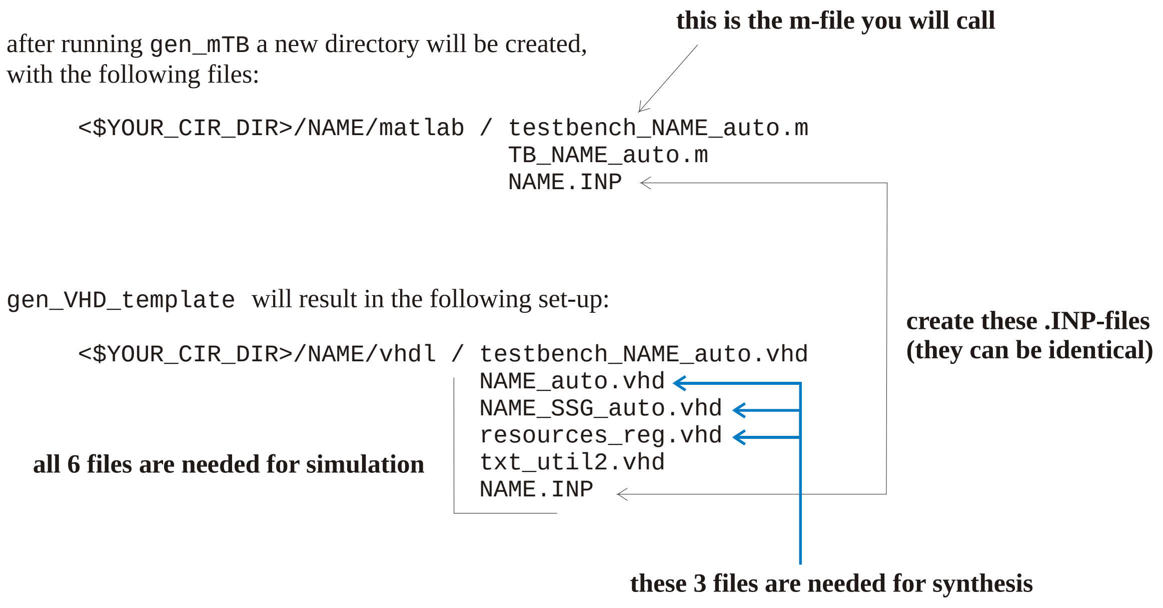
At each `New_Sample` (= `Start`) going high, a new input data value (or as many input values as there are external inputs) is (are) read from the .INP-file. At the first positive going edge of the clock, the SSG process sets `Done` low and starts with its first STATE. The STATES are advanced each clock 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.

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 be created, with the following files:

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

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

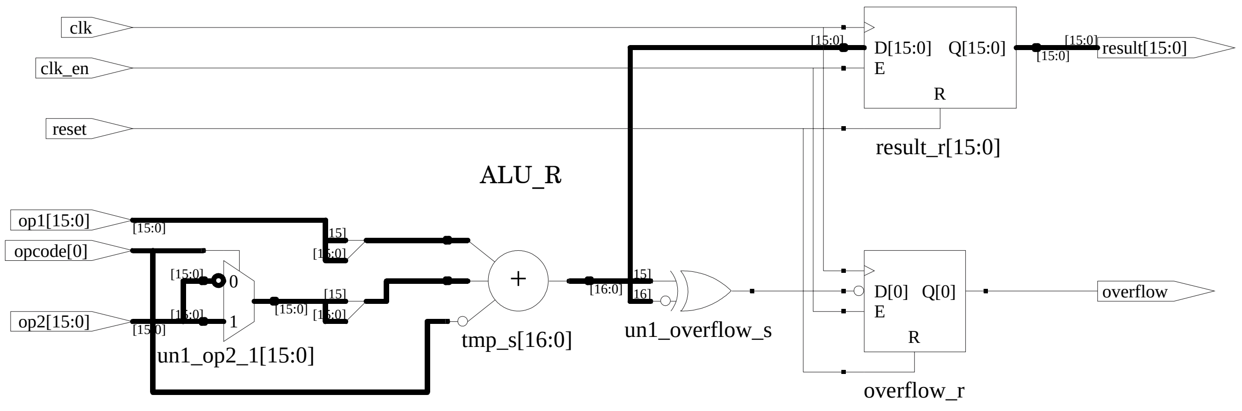
```
     <$YOUR_CIR_DIR>/NAME/vhdl / testbench_NAME_auto.vhd
                                 NAME_auto.vhd
                                 NAME_SSG_auto.vhd
                                 resources_reg.vhd
                                 txt_util2.vhd
                                 NAME.INP
```

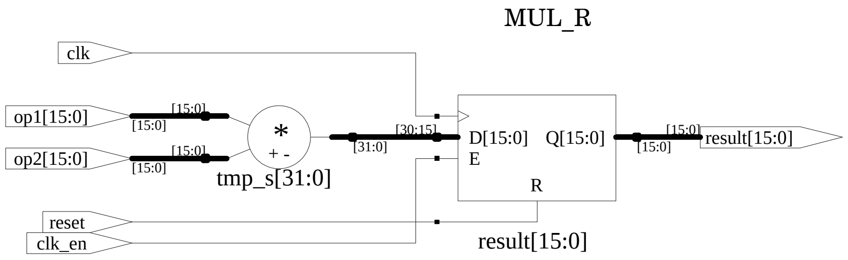
this is the m-file you will call
create these .INP-files (they can be identical)
all 6 files are needed for simulation
these 3 files are needed for synthesis

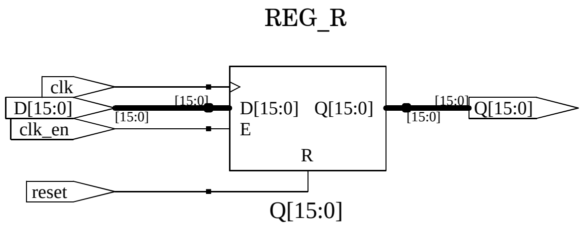
Both testbenches will write a file NAME.OUT in the directory from where they are run.

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.







This page intentionally left blank