Taylor **D**ecomposition **S**ystem (TDS)
Taylor **E**xpansion **D**iagrams (TED)

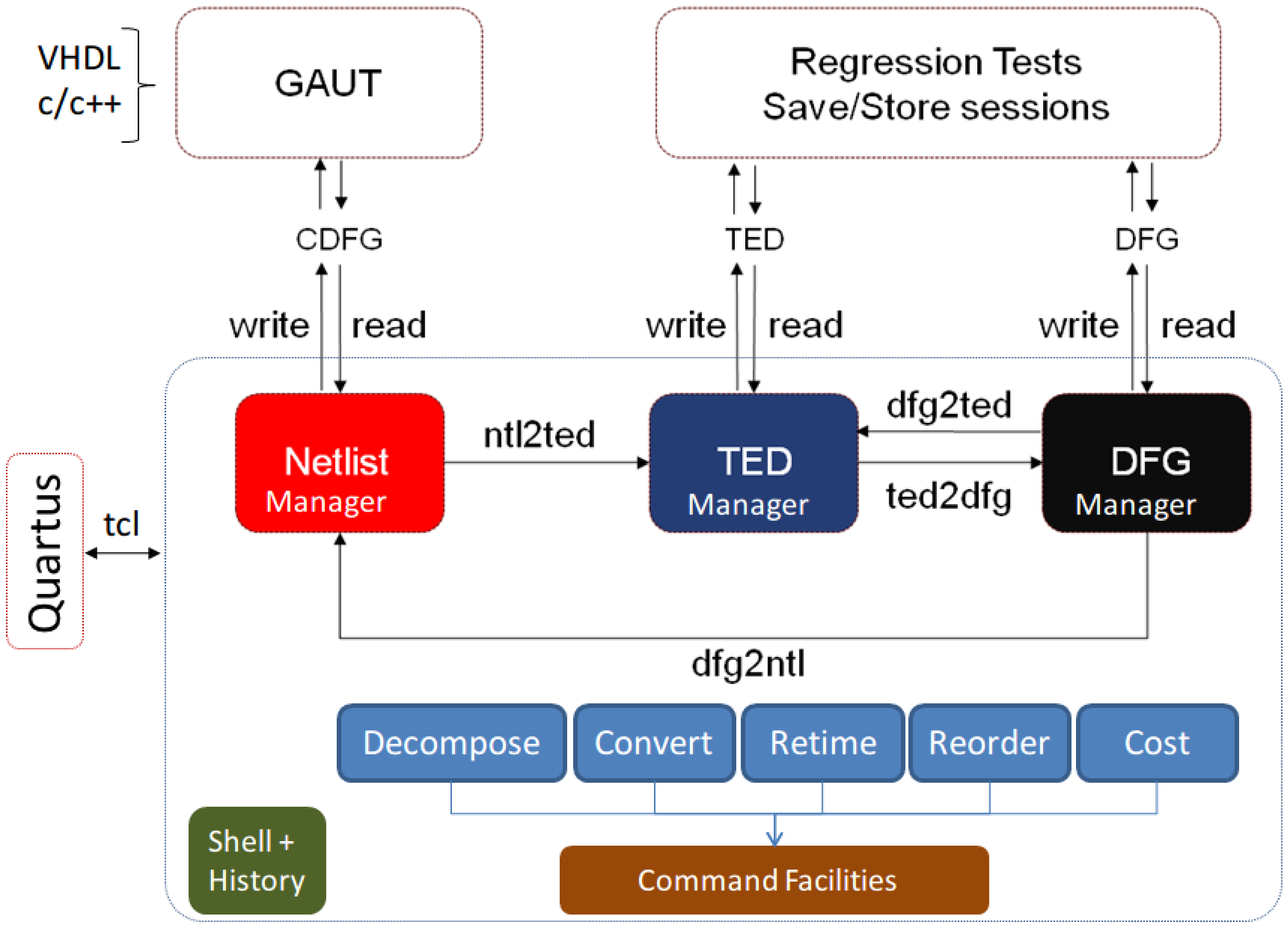
```
Taylor Decomposition System
Compiled at: 10:35:27(GMT), Aug 27 2011
Tds 01> help
 - balance              Balance the DFG or Netlist to minimize latency.
 - bbldown              Move down the given variable one position.
 - bblup                Move up the given variable one position.
 - bottom               Move the given variable to the bottom.
 - bottomdcse           CSE Dynamic, move candidates to bottom.
 - bottomscse           CSE Static, move candidates to bottom.
 - candidate            Show the candidates expression for CSE.
 - compute              Annotate the bitwidths required for exact computation.
 - cost                 Prints out the cost associated to this TED.
 - dcse                 CSE Dynamic, extract all candidates available.
 - decompose            Decompose the TED in its Normal Factor Form.
 - dfactor              Dynamic factorization.
 - dfg2ntl              Generate a Netlist from the DFG.
 - dfg2ted              Generate a TED from the DFG.
 - dfgarea              Balance the DFG to minimize the area.
 - dfgevalconst         Replace constant multipliers by shifters.
 - dfgschedule          Perform the scheduling of the DFG.
 - erase                Erase a primary output from the TED.
 - eval                 Relocate the given variable to the desired position.
 - exchange             Exchange the position of two variables.
 - explore              Performs a space exploration of the architecture, on development.
 - extract              Extract primary outputs from the TED or Netlist.
 - fixorder             Fixes the order broken by a retime operation.
 - flip                 Flip the order of a linearized variable.
 - jumpAbove            Moves a variable above another one.
 - jumpBelow            Moves a variable below another one.
 - lcse                 CSE for linearized TED.
 - linearize            Transform a non linear TED into a linear one.
 - listvars             List the variables in a top to bottom order.
 - load                 Load the environment.
 - ntl2ted              Extract from a Netlist all parts representable by TED.
 - optimize             Minimizes the bitwidth of a DFG keeping the maximum error bound.
 - poly                 Construct a TED from a polynomial expression.
 - print                Print out TED information: statistic, etc.
 - printenv             Print the environment variables.
 - printntl             Print out statistics of the Netlist.
 - purge                Purge the TED, DFG and/or Netlist.
 - quartus              Generates a quartus project, compiles it and report its freq and # of LE.
 - read                 Read a script, a CDFG, a TED or a DFG.
 - reloc                Relocate the given variable to the desired position.
 - remapshift           Remap the shifters to <<.
 - reorder              Reorder the variables in the TED (Pre-fixed cost).
 - reorder*             Reorder the variables in the TED. (User defined cost)
 - retime               Performs(forward/backward)retiming in TED.
 - save                 Save the environment.
 - scse                 CSE Static, extract one candidate at a time.
 - set                  Set the variable bitwidth and other options.
 - setenv               Set a environment variable.
 - shifter              Replace constant multipliers by shifters.
 - show                 Show the TED, DFG or Netlist graph.
 - sift                 Heuristically optimize the level of the variable.
 - sub                  Substitute an arithmetic expression by a variable
 - ted2dfg              Generate a DFG from the TED.
 - top                  Move the given variable to the top.
 - tr                   Construct a TED from a set of DSP transforms.
 - vars                 Preset the order of variables, before any TED is given.
 - write                Write the existing NTL|DFG|TED into a*.[cdfg|dfg|ted] file.
 -------- GENERAL
 - ![bin]                 System call to execute bin
 - h[elp]                 Print this help
 - e[xit]                 Exit the shell
 - q[uit]                 Quit the shell
 - time                   Show the elapsed time for the last command executed
 - man                    Prints the manual of the given command
 - the shell accepts command completion. i.e. type dec<TAB> for decompose

Tds 01> _
```

Additional documentation for individual commands can be obtained by typing “man command_name”

TDS Data Structure:

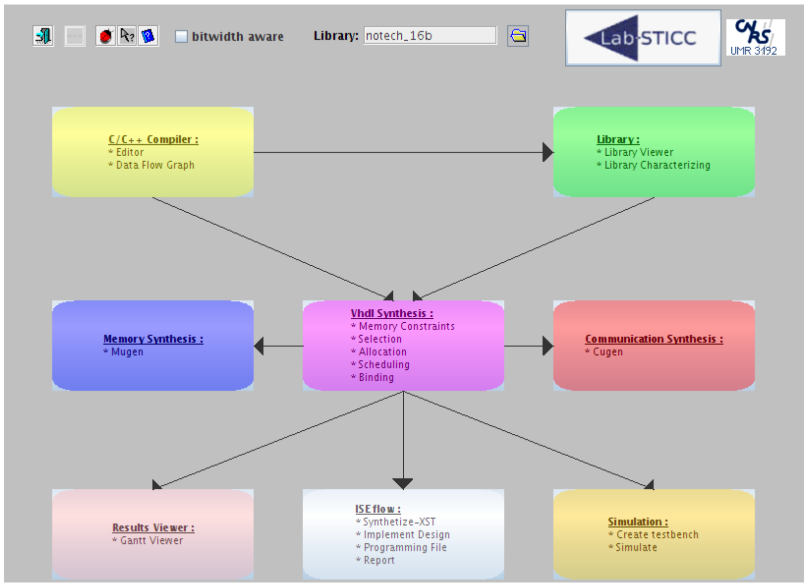
The TDS System is composed of three data structures, the Taylor Expansion Diagram (TED) and two helper data structures: the Data Flow Graph (DFG) and Netlist (NTL).

- TED captures the functionality of an algebraic data path and performs optimizations on the behavioral level
- DFG provides a mechanism to visualize the data-path being implemented by TED
- NTL provides a mechanism to communicate with the GAUT front-end.



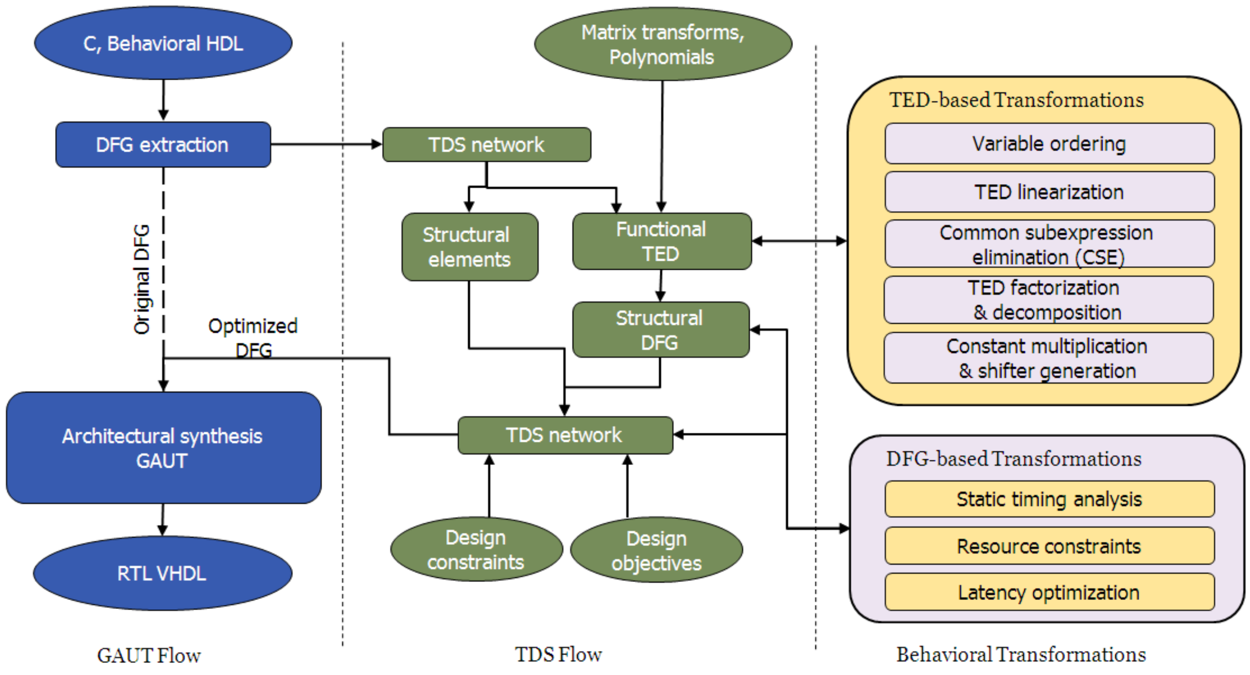
The main interface of TDS software with is the high level synthesis tool GAUT (which can be obtained for free from http://www-labsticc.univ-ubs.fr/www-gaut/). Upon start of GAUT the C/C++ compiler will translate a C source design into an intermediate parse-tree structure called CDFG, this format is used by TDS as input/output format to communicate with GAUT. The modified CDFG provided by TDS can be given back to the GAUT Vhdl synthesis tool to perform its synthesis into RT. The final hardware statistics provided by TDS are derived from after physical synthesis from the Altera Quartus tool (which can be downloaded for free from http://www.altera.com/products/software/sfw-index.jsp)

A snapshot of the front end of GAUT tool is provided below:

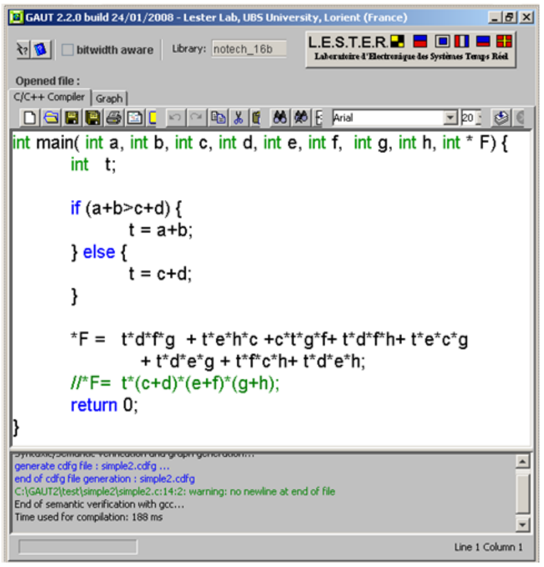

bitwidth aware

Library: notech_16b

The interface between TDS and GAUT is also depicted in flow chart below. The DFG extraction is obtained by reading the CDFG file produced by GAUT into TDS, and the optimized DFG is the resulting CDFG obtained with TDS provided to GAUT.



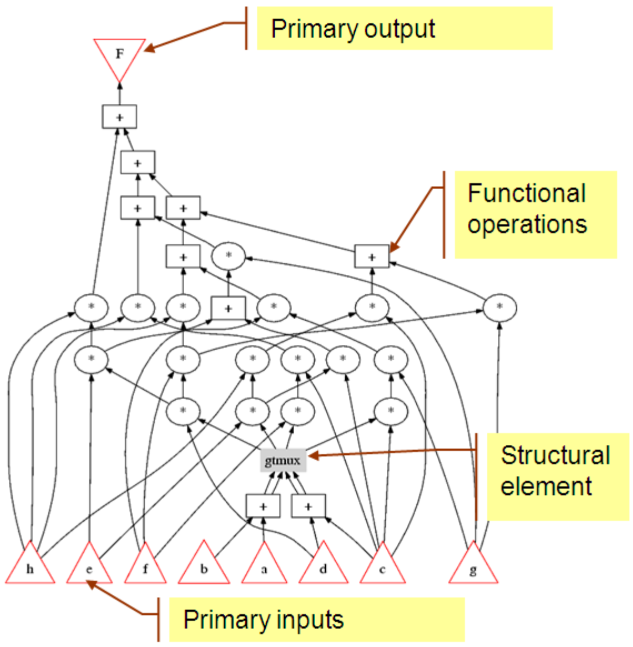
Although the CDFG file can be read into TDS and plot into its NTL data structure, there are structural elements in the NTL that cannot be transform into TED, these structural elements force a single NTL to be represented by a set of TEDs, in which the input of some TEDs are the outputs of other TEDs.


GAUT 2.2.0 build 24/01/2008 - Lester Lab, UBS University, Lorient (France)
bitwidth aware
Library: notech_16b
L.E.S.T.E.R.
Laboratoire d'Electronique des Systèmes Temps Réel
Opened file :
C/C++ Compiler
Graph
Arial
20
Line 1 Column 1

```
int main( int a, int b, int c, int d, int e, int f,  int g, int h, int * F) {
        int   t;

        if (a+b>c+d) {
                t = a+b;
        } else {
                t = c+d;
        }

        *F =   t*d*f*g  + t*e*h*c +c*t*g*f+ t*d*f*h+ t*e*c*g
                  + t*d*e*g + t*f*c*h+ t*d*e*h;
        //*F=  t*(c+d)*(e+f)*(g+h);
        return 0;
}
```

```
generate cdfg file : simple2.cdfg ...
end of cdfg file generation : simple2.cdfg
C:\GAUT2\test\simple2\simple2.c:14:2: warning: no newline at end of file
End of semantic verification with gcc...
Time used for compilation: 188 ms
```

Example behavioral design in C



Initial TDS network

Similar to the CDFG read/write facilities, a snapshot of the internal data structures TED or CDFG can also be taken through the commands `read` and `write`, these commands as explained below recognize the input and output format by the filename extension.

```
Tds 01> write -h
NAME
  write - Write the existing NTL|DFG|TED into a *.[cdfg|dfg|ted] file.
SYNOPSIS
  write [cdfg options] outputfile.[cdfg|dfg|ted|scr]
OPTIONS
  -h,--help
    Print this message.
[cdfg options]
  -d,--dfg
    Uses the DFG data structure as starting point
  -t,--ted
    Uses the TED data structure as starting point
  -n,--ntl
    Uses the NTL data structure as starting point
  outputfile
    The desired output file name. The extension defines the format:
     - cdfg  current GAUT format.
     - dfg   internal DFG data structure.
     - ted   internal TED data structure.
     - gappa GAPPA script for computing the accuracy of the DFG data structure.
     - scr   generates a script file from the command history.
NOTE
  By default the file format determines the data structure from which
  the file will be writen: cdfg->NTL, dfg->DFG, ted->TED
EXAMPLE
  write poly2.cdfg
    ... writes the NTL in a cdfg file format
  write --ted poly1.cdfg
    ... converts the TED into NTL and then writes it into a cdfg file
  write poly1.dfg
    ... writes the DFG in file poly1.dfg
SEE ALSO
  read,purge
```

NOTE:

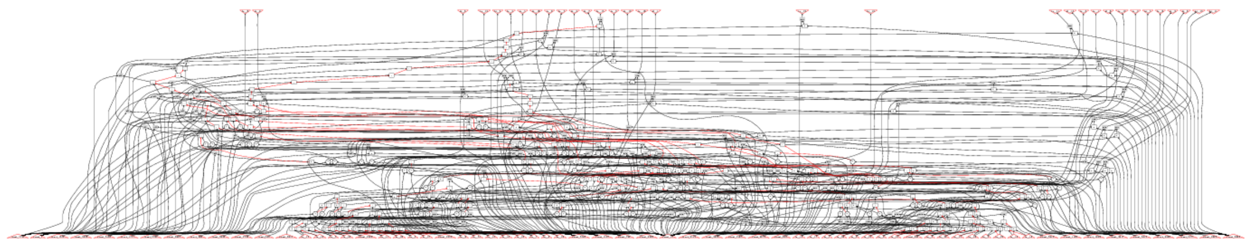
The transformations `ntl2ted, dfg2ted, dfg2ntl` are a one to one transformation. This is not the case with the `ted2dfg` transform which might generate different DFG graphs from the same TED, depending on how the TED is traversed. This is depicted on the ted2dfg command by options `--normal` and `--factor`

```
Tds 01> ted2dfg -h
NAME
  ted2dfg - Generate a DFG from the TED.
SYNOPSIS
  ted2dfg [method]
OPTIONS
  -h,--help
    Print this message.
  [method]
  -n,--normal
    Generate the DFG by a NFF traversal of the TED [DEFAULT BEHAVIOR].
  -f,--factor [--show]
    Force the factorization of common terms in the DFG graph.
    Sub option: --show
    Treat each factor found as a pseudo output in the DFG graph.
DETAILS
  Most of the times this construction is made implicit. For instance when
  an operation in a DFG is requested (i.e. show -d) and no DFG exist yet
  an implicit conversion occurs. If a DFG already exists, this command will
  overwrite it.
SEE ALSO
  ted2ntl,ntl2ted,dfg2ted,dfg2ntl
```

TDS Input: [related commands `read, poly, ntl2ted, ted2dfg, dfg2ted, dfg2ntl, write, extract`]

Besides reading CDFG files such as

```
Tds 01> read dct32.cdfg
```



and then transforming the NTL into TEDs by:

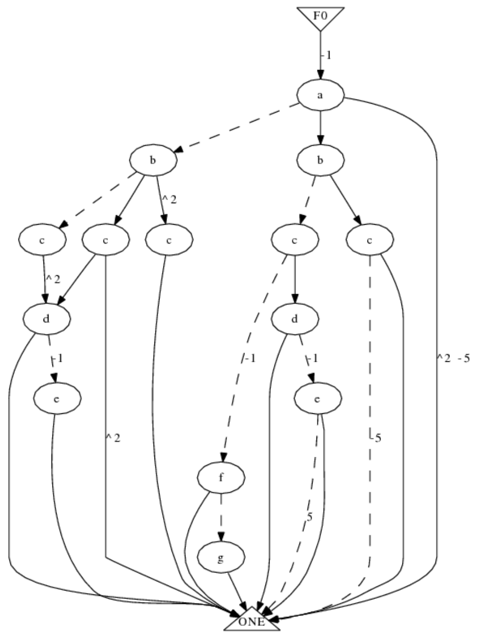
```
Tds 01> ntl2ted
```

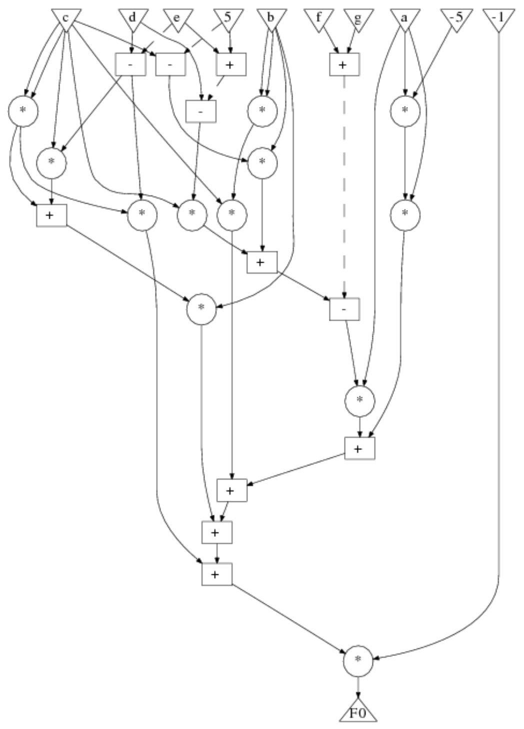
One can elaborate the datapath directly onto TED by using the command:

```
Tds 01> poly (a+b+c)*(5*a-c*(b+d-e))+(f+g)*a
```

And then visualize its TED using the command `show`, and its DFG using `show -d`

```
Tds 02> show --ted
Tds 03> ted2dfg
Tds 04> show --dfg
```





Once the TED data structure has been built, different optimizations can be performed on it.

TDS Environment: [Related commands `load, save, printenv, setenv`]

There is a set of environments in TDS that can be customized. The complete list of environments is stored on a file named "*tds.env*" and reproduced below:

```
#######################################
# Environment file generated by TDS.  #
# http://incascout.ecs.umass.edu/main #
#######################################
# Environment variable  |  Value
#-----------------------|---------------------
# TED Related
    bitwidth_fixedpoint | 4,12
       bitwidth_integer | 16
    default_design_name | tds2ntl
               delayADD | 1
               delayLSH | 1
               delayMPY | 2
               delayREG | 1
               delayRSH | 1
               delaySUB | 1
                dot_bin | dot
                 ps_bin | evince
                   rADD | 0
                   rMPY | 0
                   rSUB | 0
           reorder_type | proper
           show_bigfont | false
         show_directory | ./dotfiles
             show_level | true
           show_verbose | false
```

```
# GAUT Related
          const_as_vars | false
        const_cdfg_eval | false
           const_prefix | const_
        negative_prefix | moins_
gaut_allocate_strategy | -distributed_th_lb
           gaut_cadency | 200
             gaut_clock | 10
    gaut_cost_extension | .gcost
  gaut_gantt_generation | false
               gaut_mem | 10
    gaut_mem_generation | false
 gaut_optimize_operator | true
 gaut_register_strategy | 0
 gaut_schedule_strategy | force_no_pipeline
 gaut_soclib_generation | false
              gaut_tech | notech_16b.lib
               gaut_bin | ~/Code/tds/workingTds/src/3rdparty/gaut/src
               gaut_lib | ~/Code/tds/workingTds/src/3rdparty/gaut/lib
               cdfg_bin | ~/Gaut/ver2_4_2/GautC/cdfgcompiler/bin/cdfgcompiler
#---------------------------------------------
# Other possible values |
#-----------------------
# gaut_schedule_strategy  { "", "force_no_pipeline", "force_no_mobility", \
#                            "no_more_stage", ""}
# gaut_allocate_strategy  {"-distributed_th_lb", "-distributed_reuse_th", \
#                            "-distributed_reuse_pr_ub", "-distributed_reuse_pr", \
#                            "-global_pr_lb", "-global_pr_ub"}
# gaut_register_strategy  {"0", "1", "2", "3"}
#                            0 MWBM [default]
#                            1 MLEA
#                            2 Left edge
#                            3 None
#                 ps_bin  {"evince", "gv", "gsview32.exe"}
#           reorder_type  {"proper", "swap", "reloc"}
```

The current list of environment settings can be obtained in TDS with the command print environment:

```
Tds 01> printenv
```

Similarly, a particular setting can be modified with the command set environment:

```
Tds 01> setenv reorder_type = proper
```

The current environment can be saved:

```
Tds 01> save [optional filename argument, default filename is tds.env]
```

or load and re-load:

```
Tds 01> load [optional filename argument, default filename is tds.env]
```

NOTE:

These environment variables have a direct impact on how different commands work, for instance the environment variable "`const_prefix`" determines the string expected by the CDFG parser to identify a constant value, if this value is modified all constants read from the CDFG file will fail to be identify and will be treated as variables.

TDS Command's help:

The correct syntax of any command can be checked using the command `man`:

```
Tds 01> man setenv
NAME
  setenv - Set an environment variable.
SYNOPSIS
  setenv variable = condition
OPTIONS
  -h,--help
    Print this message.
```

The same result can also be obtained by using the option `-h` or `--help`

```
Tds 01> setenv --help
```

TDS Alias Commands:

Additional to the commands provided by TDS, one can use aliases to refer to a particular command or to a group of commands. The list of aliases should be stored in a file named "tds.aliases".

```
##############################################################
#reserved word    alias name    semi-colon separated commands#
##############################################################
alias             flatten       ted2dfg -f; dfg2ted
alias             stats         print -s
```

This file is uploaded by TDS on startup, so any modification on this file requires restarting the TDS system. It is worth noting that this alias commands do not accept arguments, therefore no help can be invoked on these alias-commands.

TED Ordering: [Related commands `bblup, bbldown, bottom, exchange, flip, fixorder,jumpAbove, jumpBelow, reloc, sift, top`]

Moving an individual variable in the TED data structure can be achieved by the following commands: `bblup, bbldown, bottom, top, flip, reloc`. While reordering the TED to optimize a specific metric is achieved by the commands `reorder` and `reorder*`.

All these commands except by "`reorder*`" are subject to the following three reordering methods:

```
[reorder algorithm]
  -r,--reloc
    Reconstructs the TED with the desired order of variables. This is
    slow, but serves as golden reference.
  -s,--swap
    A Hybrid implementation, that re-structures the TED with the desired
    order, but internally uses reconstructs and operations on the bottom
    graph [DEFAULT BEHAVIOR].
  -p,--proper
    The proper algorithm to reorder the TED as described in the paper
    http://doi.ieeecomputersociety.org/10.1109/HLDVT.2004.1431235.
```

In this particular setting the default algorithm is the “swap”; but this setting can always be changed on the environment settings of the TDS. The reason for these many implementations is best understood with the following recap:

Brief history:

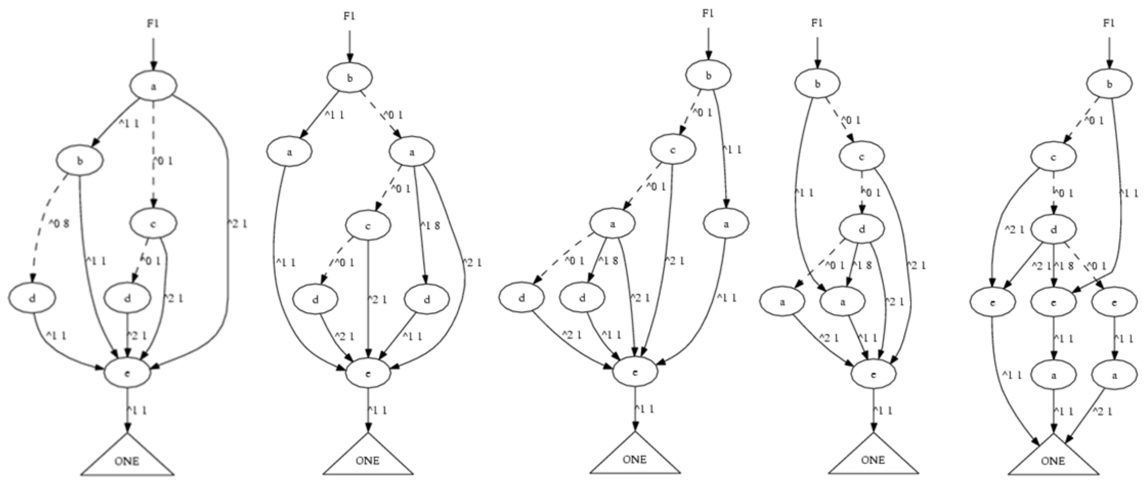
There have been 4 different implementations of the TED package.

1. The first one used internally the ccud package and was started on 2002 and never finished.
2. The second one was a re-write of the package called TED in 2004 - 2005, and implemented the construction of the TED and variable ordering.
3. The third version named TEDify was an optimized version of the second package built from scratch to cope with memory problems and efficiency 2006 - 2007.
4. The forth version was also started in 2006 and was named TDS. The development of the third and forth version overlapped in time, and because other people started working on the forth version, the TEDify was abandon and its algorithms ported into TDS. Since 2008 substantial changes and improvements have been made to the TDS package.

Coding the variable ordering algorithm is most likely one of the most troublesome parts to write on the TED data structure. And although the “proper” algorithm built for TEDify has been completely ported to TDS, new requirements on the data structure (retiming, bitwith, error) require new modification to this algorithm. Therefore a quick and dirt implementation called “swap” has been left, this implementation disregards any information other than the variable name in a TED. The “proper” implementation performs a bit faster than the “swap” and produces the same results, but currently it is being modified to take into account the register limitations imposed by retiming in TED.

Ted Reordering [Related commands `reorder, reorder*, cost, quartus`]

Ordering the TED to optimize a particular cost function is possible. The commands reorder and reorder* permit to evaluate the variable ordering of a TED to a certain cost function. For instance, each of the TEDs shown below correspond to the same TED but with different orderings as to minimize the number of nodes, number of multipliers, latency, etc.



Searching for the best TED ordering for a particular cost function is in the worst case exponential in the number of nodes (an exhaustive search is not recommended), therefore one can specify other heuristics with the command reorder and reorder*:

```
  [number of iterations]
  -a, --annealing
    It minimizes the cost function by using the annealing algorithm
  -e, --exhaustive [--end]
    Tries all possible orders by doing permutation, is O(N!)
    Sub option: --end
    Prevents the abortion of the permutation when 'ESC' is pressed
  -s, --sift [-g, --group]
    Moves each variable at a time throughout the height of the TED
    graph till its best position is found, is O(N^2)[DEFAULT BEHAVIOR].
    Sub option: -g, --group
    This option only affects the SIFT algorithm. If grouping is selected,
    the sifting is done preserving the relative grouping of all variables
    that were linearized. If not every single variable is moved regardless
    of its grouping.
```

The command **`reorder`** can optimize one of the following cost functions, and only one cost function at a time.

```
  [cost functions]
     Definitions:
       ted_subexpr_candidates = # of product terms in the TED with 2+ parents connecting to ONE
       ted_nodes              = # of nodes in the TED graph
       nMUL                   = # of multiplications in the DFG graph
       nADD                   = # of additions in the DFG graph
       nSUB                   = # of substractions in the DFG graph
       rMPY                   = # of multipliers after scheduling the DFG
       rADD                   = # of adders and subtractors after scheduling the DFG
       dLatency               = the latency of the DFG
       gLatency               = the latency of the Gaut implementation
     Environment variables used to set the:
       1) delay of the DFG operators:
          delayADD [Default value = 1]
          delaySUB [Default value = 1]
          delayMPY [Default value = 2]
          delayREG [Default value = 1]
       2) maximum number of resources used by the DFG scheduler:
          rMPY [Default value = 4294967295]
          rADD [Default value = 4294967295]
          rSUB [Default value = 4294967295]
  --node
    Minimizes the function "10*ted_nodes - ted_subexpr_candidates"
   -nm, --nMUL {legacy -m, --mul}
    Minimizes the function "10*nMUL - ted_subexpr_candidates" [DEFAULT BEHAVIOR]
  --op
    Minimizes the function "10*(nMUL + nADD + nSUB)- ted_subexpr_candidates"
  --opscheduled
    Minimizes nMUL, followed by(nADD+nSUB), dfg latency, rMPY, rADD
  -dl,--dLatency {legacy --latency}
    Minimizes the DFG latency subject to the resources specified in the environment variables
  --bitwidth
    Minimizes the bitwidth of the HW implementation subject to unlimited latency|resources
  -gm, --gMUX {legacy --gmux}
    Minimizes the Gaut mux count in the Gaut implementation(each mux is considered 2 to 1)
  -gl, --gLatency {legacy --glatency}
    Minimizes the Gaut latency in the Gaut implementation
  -gr, --gREG {legacy --garch}
    Minimizes the Gaut register count in the Gaut implementation
  --gappa
    Minimizes the upper tighter bound found trough Gappa
```

Example: Let’s take the polynomial f*b*h+a+c*b+g*f*b+e*d*b and let’s find out the cost of implemented this polynomial as reported by Gaut and by Altera

```
Tds 08> poly f*b*h+a+c*b+g*f*b+e*d*b
Tds 09> cost
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|          TED                          |       DFG          |       Schedule        |           Gaut                    |
|---------------------------------------|--------------------|-----------------------|-----------------------------------|
| node | edge0 | edgeN | factor | width | nMUL | nADD | nSUB | Latency | rMPY | rADD | Muxes | Latency | Register | Area |
|    9 |     4 |     4 |      0 |     0 |    4 |    4 |    0 |       7 |    2 |    1 |    80 |       9 |       80 |   91 |
design name: ted
Tds 10> listvars
List of TED variables in the Container(from top to bottom):
f b h a c g e d
Primary Outputs
F0 PrimaryOutput 10
Tds 11> ted2dfg
Tds 12> dfg2ntl
Tds 13> cost -n
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|          TED               |       DFG          |       Schedule        |           Gaut                    |
|----------------------------|--------------------|-----------------------|-----------------------------------|
| nodes | factors | bitwidth | nMUL | nADD | nSUB | latency | rMPY | rADD | Muxes | latency | Register | Area |
| 2147483647 | 2147483647 |        0 |    0 |    0 |    0 |       0 |    0 |    0 |    80 |       9 |       80 |   91 |
design name: dfg_2ntl
Tds 14> quartus
Info: Report file saved in "dfg_2ntl.quartus.report"

clock name | target freq | design freq
--------------------------------------
clk        | 1000.0 MHz  | 84.98 MHz

Resources                          | Synthesis | Fitter
---------------------------------------------------------------------
Maximum fan-out                    | 105       | 105
Average fan-out                    | 3.38      | 3.53
Total LABs                         | 0         | 48 / 291 ( 16 % )
Total registers                    | 100       | 100 / 3,210 ( 3 % )
Total logic elements               | 420       | 403 / 2,910 ( 14 % )
-- Combinational with no register  | 320       | 303
-- Register only                   | 16        | 0
-- Combinational with a register   | 84        | 100
-- 4 input functions               | 201       | 202
-- 3 input functions               | 139       | 140
-- 2 input functions               | 63        | 60
-- 1 input functions               | 1         | 1
-- 0 input functions               | 0         | 0
```

```
Tds 29> poly f*b*h+a+c*b+g*f*b+e*d*b
Tds 30> reorder --annealing -gl
strade=111111110
[========================================] 100%
Tds 31> cost
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|          TED                          |       DFG          |       Schedule        |           Gaut                    |
|---------------------------------------|--------------------|-----------------------|-----------------------------------|
| node | edge0 | edgeN | factor | width | nMUL | nADD | nSUB | Latency | rMPY | rADD | Muxes | Latency | Register | Area |
|    9 |     4 |     6 |      1 |     0 |    5 |    4 |    0 |       7 |    3 |    1 |    96 |       9 |       80 |   91 |
design name: ted
Tds 32> reorder --annealing -gl
strade=111111110
[========================================] 100%
Tds 33> cost
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|          TED                          |       DFG          |       Schedule        |           Gaut                    |
|---------------------------------------|--------------------|-----------------------|-----------------------------------|
| node | edge0 | edgeN | factor | width | nMUL | nADD | nSUB | Latency | rMPY | rADD | Muxes | Latency | Register | Area |
|    9 |     4 |     6 |      1 |     0 |    5 |    4 |    0 |       7 |    3 |    1 |    96 |       9 |       80 |   91 |
design name: ted
Tds 34> ted2dfg
Tds 35> balance -d
Tds 36> cost -d
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
```

```
|           TED              |        DFG         |        Schedule       |           Gaut                     |
|----------------------------|--------------------|-----------------------|------------------------------------|
| nodes | factors | bitwidth | nMUL | nADD | nSUB | latency | rMPY | rADD | Muxes | latency | Register | Area |
| 2147483647 | 2147483647 |        0 |    5 |    4 |    0 |       6 |    3 |    1 |    64 |       8 |       96 |   91 |
design name: dfg2ntl
Tds 37> dfg2ntl
Tds 38> cost -n
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|           TED              |        DFG         |        Schedule       |           Gaut                     |
|----------------------------|--------------------|-----------------------|------------------------------------|
| nodes | factors | bitwidth | nMUL | nADD | nSUB | latency | rMPY | rADD | Muxes | latency | Register | Area |
| 2147483647 | 2147483647 |        0 |    0 |    0 |    0 |       0 |    0 |    0 |    64 |       8 |       96 |   91 |
design name: dfg_2ntl
Tds 38> quartus
Info: Report file saved in "dfg_2ntl.quartus.report"

clock name | target freq | design freq
--------------------------------------
clk        | 1000.0 MHz  | 90.2 MHz
Resources                       | Synthesis | Fitter
--------------------------------------------------------------------
Maximum fan-out                 | 122       | 122
Average fan-out                 | 3.40      | 3.49
Total LABs                      | 0         | 51 / 291 ( 18 % )
Total registers                 | 116       | 116 / 3,210 ( 4 % )
Total logic elements            | 469       | 467 / 2,910 ( 16 % )
-- Combinational with no register | 353       | 351
-- Register only                | 0         | 0
-- Combinational with a register  | 116       | 116
-- 4 input functions            | 199       | 203
-- 3 input functions            | 127       | 124
-- 2 input functions            | 142       | 139
-- 1 input functions            | 1         | 1
-- 0 input functions            | 0         | 0
```

The command **`reorder*`** optimizes a list of cost functions, where the least important cost function is given first, and the most important cost function is given last.

```
  [list of cost functions] {LICF ... MICF}
    Where LICF and MICF stands for the Least/Most Important Cost Function to optimize
  --node
    Minimizes the number of TED nodes
  --edge
    Minimizes the total number of TED edges
  --edge0
    Minimizes the number of additive TED edges
  --edgeN
    Minimizes the number of multiplicative TED edges
  -nm, --nMUL
    Minimizes the number of multiplications in the DFG
  -na, --nADD
    Minimizes the number of additions(and substractions)in the DFG
   -rm, --rMPY
    Minimizes the number of multipliers erOfCandidates" [DEFAULT BEHAVIOR]
  -dl, --dLatency
    Minimizes the lantecy in the DFG
  --bitwidth
    Minimizes the bitwidth of the HW implementation subject to unlimited latency|resources
  --gappa
    Minimizes the upper tighter bound found trough Gappa
  -gm, --gMUX
    Minimizes the number of muxes in the GAUT implementation
  -gl, --gLatency
    Minimizes the latency in the GAUT implementation
  -gr, --gREG
    Minimizes the number of registers in the GAUT implementation
  -ga, --gArea
    Minimizes the total area of the operators in the GAUT implementation
```

```
Tds 01> poly f*b*h+a+c*b+g*f*b+e*d*b
Tds 02> cost
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|           TED                         |        DFG         |        Schedule       |           Gaut                    |
|---------------------------------------|--------------------|-----------------------|-----------------------------------|
| node | edge0 | edgeN | factor | width | nMUL | nADD | nSUB | Latency | rMPY | rADD | Muxes | Latency | Register | Area |
|    9 |     4 |     4 |      0 |     0 |    4 |    4 |    0 |       7 |    2 |    1 |    80 |       9 |       80 |   91 |
design name: ted
Tds 03> reorder* --annealing -gr -ga -gm -gl
strade=111111110
[========================================] 100%
Tds 04> cost
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|           TED                         |        DFG         |        Schedule       |           Gaut                    |
|---------------------------------------|--------------------|-----------------------|-----------------------------------|
| node | edge0 | edgeN | factor | width | nMUL | nADD | nSUB | Latency | rMPY | rADD | Muxes | Latency | Register | Area |
|    9 |     4 |     6 |      1 |     0 |    5 |    4 |    0 |       7 |    3 |    1 |    96 |       9 |       80 |   91 |
design name: ted
Tds 05> ted2dfg
Tds 06> balance -d
Tds 07> cost -d
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|           TED              |        DFG         |       Schedule        |           Gaut                    |
|----------------------------|--------------------|-----------------------|-----------------------------------|
| nodes | factors | bitwidth | nMUL | nADD | nSUB | latency | rMPY | rADD | Muxes | latency | Register | Area |
| 2147483647 | 2147483647 |        0 |    5 |    4 |    0 |       6 |    3 |    1 |   112 |       8 |       96 |   91 |
design name: dfg2ntl
Tds 08> dfg2ntl
Tds 09> quartus
Info: Report file saved in "dfg_2ntl.quartus.report"

clock name | target freq | design freq
--------------------------------------
clk        | 1000.0 MHz  | 87.43 MHz
Resources                          | Synthesis | Fitter
--------------------------------------------------------------------
Maximum fan-out                    | 87        | 87
Average fan-out                    | 3.19      | 3.29
Total LABs                         | 0         | 50 / 291 ( 17 % )
Total registers                    | 84        | 84 / 3,210 ( 3 % )
Total logic elements               | 436       | 434 / 2,910 ( 15 % )
-- Combinational with no register  | 352       | 350
-- Register only                   | 0         | 0
-- Combinational with a register   | 84        | 84
-- 4 input functions               | 168       | 171
-- 3 input functions               | 152       | 152
-- 2 input functions               | 114       | 109
-- 1 input functions               | 2         | 2
-- 0 input functions               | 0         | 0
```

TED Bitwidth: [Related commands `set, optimize, compute`]

The bit width information can be annotated in the TED as post process. That is one can generate a polynomial and afterwards through the command `set` specify the bit width of each variable.

```
Tds 04> set -h
NAME
set - Set the variable bitwidth and other options.
SYNOPSIS
set [bitwidth] [range] [maximal error]
OPTIONS
-h,--help
Print this message.
[bitwidth]
-b,--bitwidth integer|int|fixedpoint|fxp [var1:bitwidth1 var2:bitwidth2 ...]
Set the initial bitwidth of the variables in the TED.
All other variables not specified in the list, take a
default bitwidth depending on its type:
* integer    (int) -> 16
* fixedpoint (fxp) -> 4,12
[range]
-r,--range var1:interval1 [var2:interval2 ...]
Where interval has the syntax: [minval,maxval]
[maximal error]
-e,--error po1:maxerror1 [po2:maxerror2 ...]
Where maxerror1 is the maximal error allowed at the primary output po1
EXAMPLE
poly F1 = a-b+c+d
poly F2 = (a+b)*(c+d)^2
set -b fixedpoint a:4,16 c:2,10
set -r d:[0.128,1.123432] b:[-5.3223,321.32e-3] c:[0,1]
set -e F1:1.324 F2:0.983
SEE ALSO
listvars,compute
```

The command `compute` is used then to compute the bit-width information across the TED data structure.

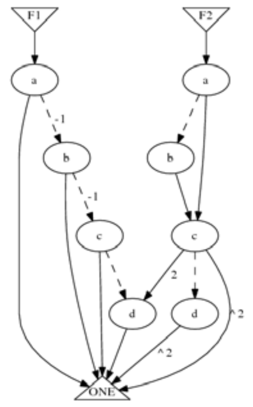
```
Tds 04> compute -h
NAME
  compute - Annotate the bitwidths required for exact computation.
SYNOPSIS
  compute [--ted –b] [--ted --snr] [--dfg –b] [--dfg -g]
OPTIONS
  -h,--help
    Print this message.
  -b,--bitwidth
    Compute the bitwidth at each point in the graph [DEFAULT BEHAVIOR].
  -g,--gappa
    Compute a bound on the maximal error.
  --snr
    Compute the Signal to Noise Ratio of the architecture.
  -t,--ted
    In the TED graph [DEFAULT BEHAVIOR].
  -d,--dfg
    In the DFG graph.
SEE ALSO
  optimize
```

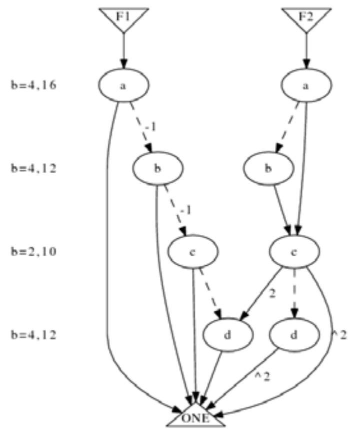
The above means that the bitwidth information can be computed on the TED and DFG data structure, whereas the maximal error bound provided by gappa can only be computed from the DFG graph.

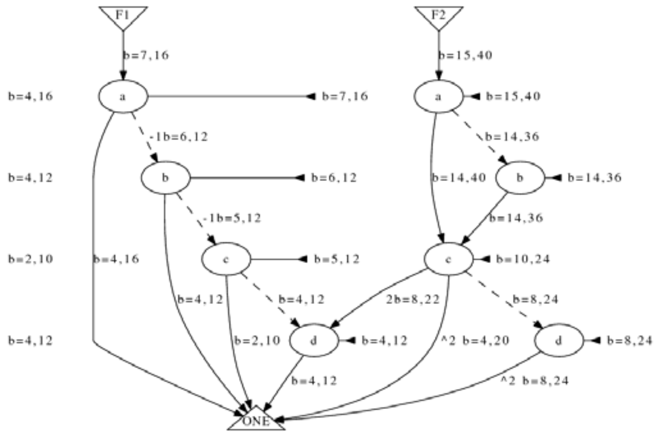
Example

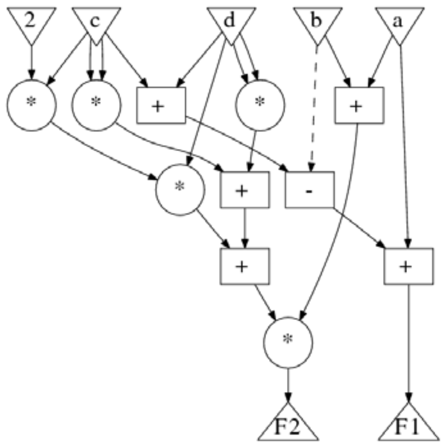
```
Tds 05> poly F1 = a-b+c+d
Tds 06> poly F2 = (a+b)*(c+d)^2
Tds 07> show
Tds 08> show ted1.dot
Tds 09> set -b fixedpoint a:4,16 c:2,10
Tds 10> show ted2.dot
Tds 11> compute -t -b
Tds 12> show ted3.dot
Tds 13> show -d dfg1.dot
Tds 14> compute -b -d
Tds 15> show -d dfg2.dot
Tds 16> set -r d:[0.128,1.123432] b:[-5.3223,321.32e-3] c:[0,1]
Tds 17> set -e F1:1.324 F2:0.983
Tds 18> compute -d –g
The maximal error associated with this DFG is 3.27433e+08
```

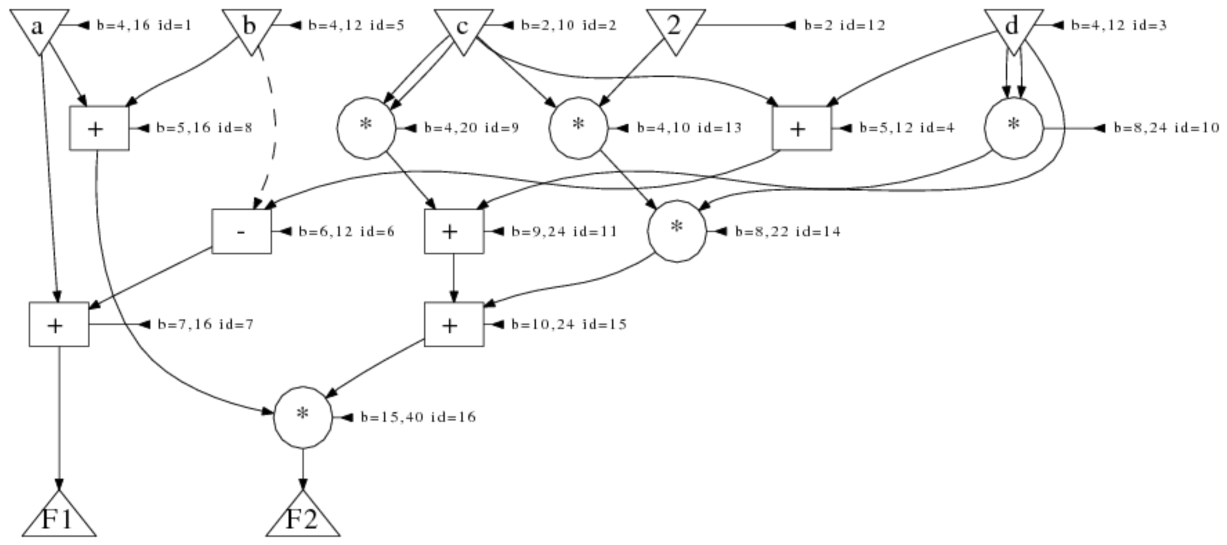
The following represent the TED graphs


ted1.dot


ted2.dot


ted3.dot

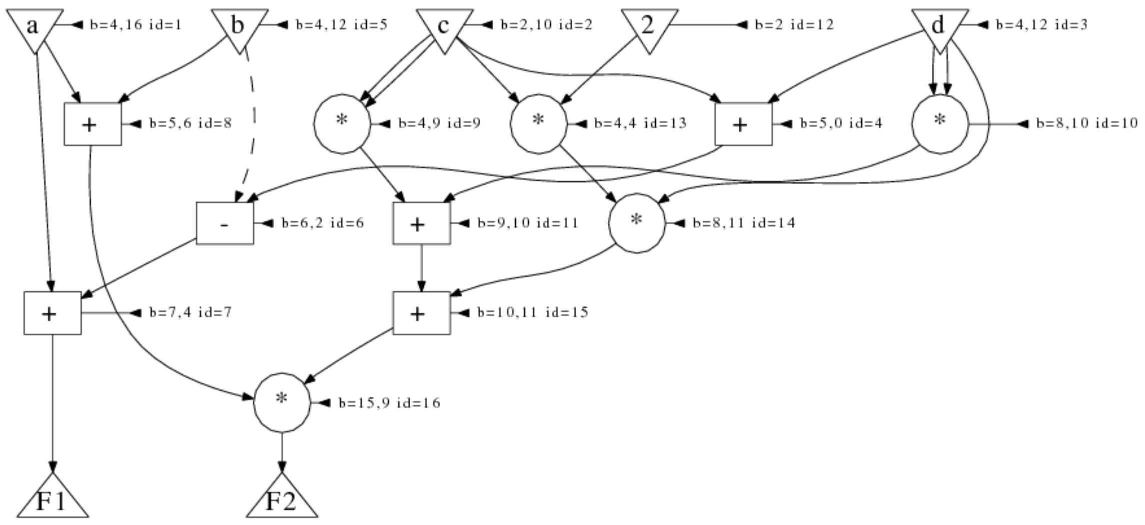

dfg1.dot



dfg2.dot

Now we can optimize the DFG for the given error with the command `optimize`

```
Tds 16> set -r d:[0.128,1.123432] b:[-5.3223,321.32e-3] c:[0,1]
Tds 17> set -e F1:1.324 F2:0.983
:
Tds 19> optimize
[1/4] 2/5 error=2.34713 op=* id=13 bitwidth=4,3 BACKTRACKING prev error=1.20128
[1/4] 4/5 error=1.37001 op=+ id=8 bitwidth=5,5 BACKTRACKING prev error=1.29949
[1/4] 3/5 error=1.33334 op=* id=9 bitwidth=4,8 BACKTRACKING prev error=1.30148
[1/4] 1/5 error=1.32936 op=* id=10 bitwidth=8,9 BACKTRACKING prev error=1.31343
[2/4] 2/3 error=1.32936 op=+ id=11 bitwidth=9,9 BACKTRACKING prev error=1.31343
[2/4] 3/3 error=1.32911 op=* id=14 bitwidth=8,10 BACKTRACKING prev error=1.32114
[2/4] 1/3 error=1.50075 op=- id=6 bitwidth=6,1 BACKTRACKING prev error=1.32114
[3/4] 2/2 error=1.32911 op=+ id=15 bitwidth=10,10 BACKTRACKING prev error=1.32114
[3/4] 1/2 error=1.37573 op=+ id=7 bitwidth=7,3 BACKTRACKING prev error=1.32114
[4/4] 1/1 error=1.32504 op=* id=16 bitwidth=15,8 BACKTRACKING prev error=1.32309
Tds 20> show –d dfg3.dot
```

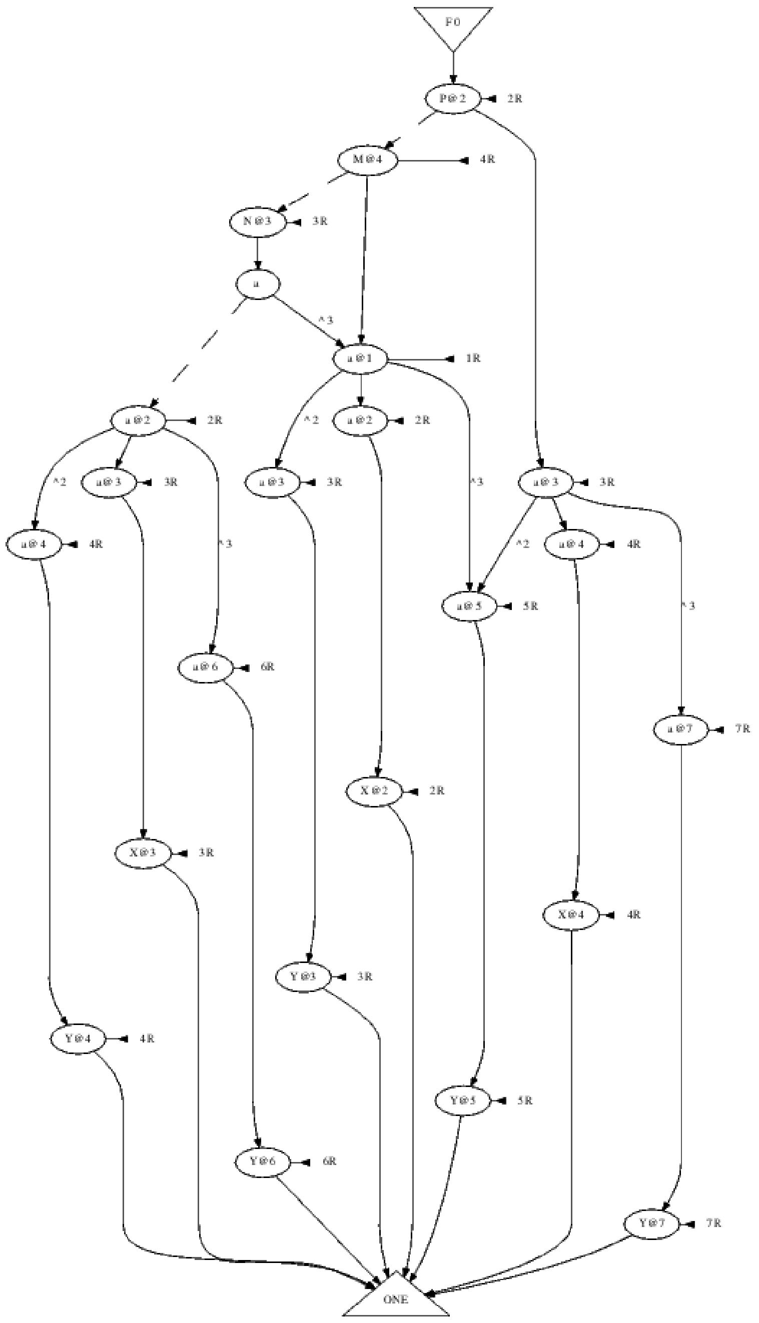


dfg3.dot

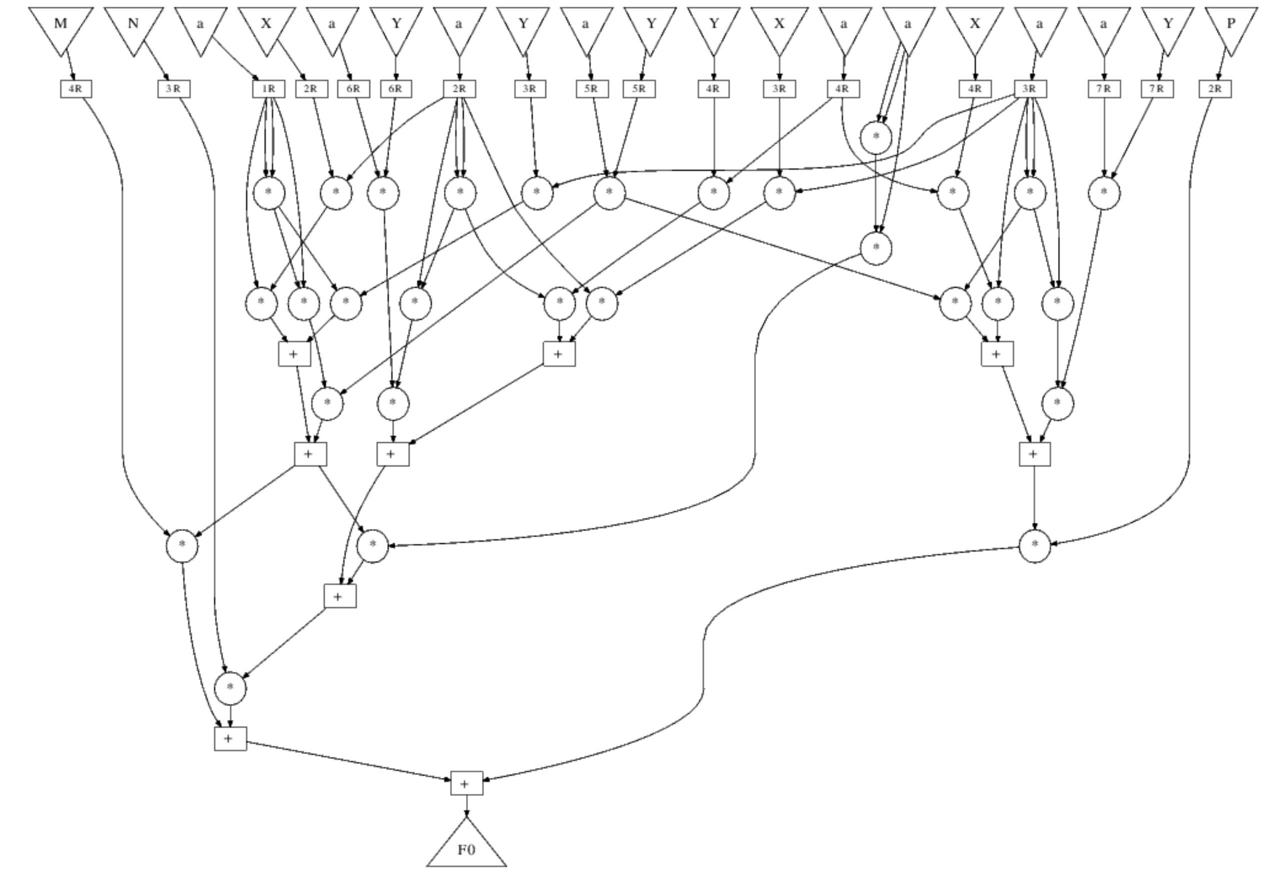
TED Register Annotation:

To annotate registers into the TED, the polynomial operations during construction have been extended to deal with timing information. The operator use to denote time delay is the at sign @.

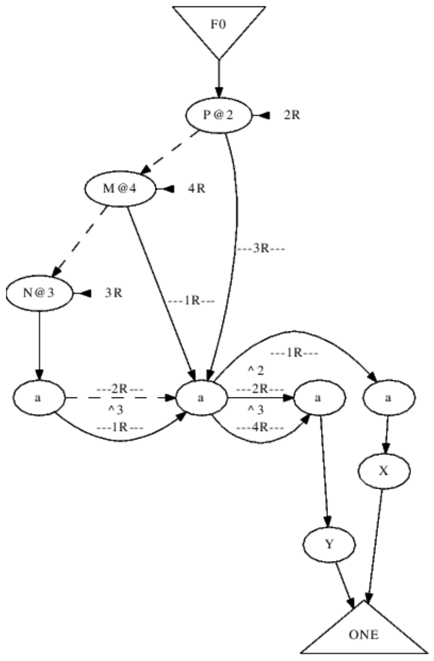
```
Tds 01> vars P M N a
Tds 02> poly
N@3*[{a*(a*X)@1+a^2*(a*Y)@2+a^3*(a*Y)@4}@2+a^3*{a*(a*X)@1+a^2*(a*Y)@2+a^3*(a*Y)@4}@1]+M@4*[{a*(a*
X)@1+a^2*(a*Y)@2+a^3*(a*Y)@4}@1]+P@2*[{a*(a*X)@1+a^2*(a*Y)@2+a^3*(a*Y)@4}@3]
Tds 03> show unretimed_ted
```



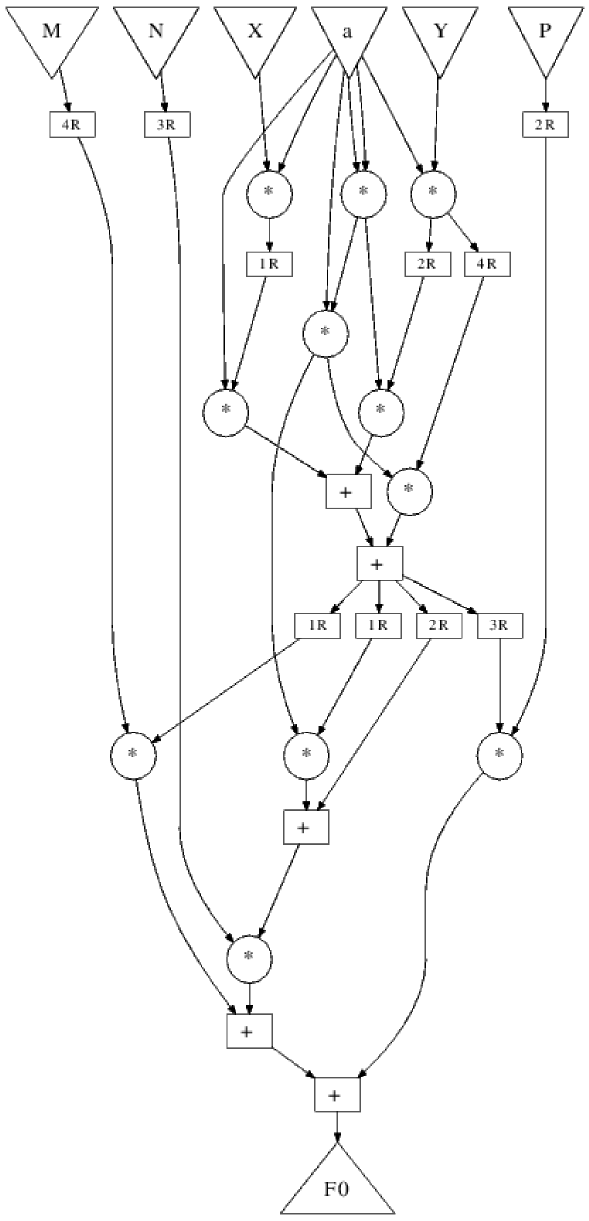
```
Tds 04> ted2dfg –n
Tds 05> show –d unretimed_dfg
```



```
Tds 06> retime –up
Tds 07> show retimed_ted
```



```
Tds 08> ted2dfg –n
Tds 09> show –d retimed_dfg
```


M
N
X
a
Y
P
4R
3R
2R
*
1R
2R
4R
+
1R
1R
2R
3R
F0