TABLE OF CONTENTS

CHAPTER 1

TAYLOR DECOMPOSITION SYSTEM MANUAL

```
Taylor Decomposition System
Compiled at: 15:56:30(GMT), Sep  9 2012
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 just above node ONE.
 – bottomdcse   [CSE Dynamic] Move candidates just above node ONE.
 – bottomscse   [CSE Static] Move candidates just above node ONE.
 – candidate    Show the candidates expression for CSE.
 – cluster      Create partitions from multiple output TEDs.
 – clusterexe   Execute a command on a TED paritions.
 – clusterinfo  Print out information from the clusters
 – compute      Annotate required bitwidth for exact computation.
 – cost         Print 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 Evaluates explicit DFG constants
 – dfgflatten   Smooths out all DFG outputs used as DFG inputs
 – dfgschedule  Perform the scheduling of the DFG.
 – dfgstrash    Perform a structural hash of the DFG.
 – erase        Erase a primary output from the TED.
 – eval         Evaluate a TED node.
 – exchange     Exchange the position of two variables.
 – extract      Extract primary outputs from the TED or Netlist.
 – fixorder     Fix the order broken by a retime operation.
 – flip         Flip the order of a linearized variable.
 – info         Print out TED information: statistic, etc.
 – jumpAbove    Move a variable above another one.
 – jumpBelow    Move a variable below another one.
 – lcse         CSE for linearized TED.
 – linearize    Transform a non linear TED into a linear one.
 – listvars     List all variables according to its ordering.
 – load          Load the environment.
 – ntl2ted       Extract from a Netlist all TEDs.
 – optimize      Minimize DFG bitwidth subject to an 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.
 – pushshifter   Perform a structural hash of the DFG.
 – quartus       Generate, compile and report Quartus project.
 – read          Read a script, a CDFG, a TED or a DFG.
 – recluster     Reclusters paritions according to an objective.
 – reloc         Relocate the given variable to the desired position.
 – remapshift    Remap multipliers and additions by shifters.
 – reorder       Reorder the variables in the TED (Pre–fixed cost)
 – reorder*      Reorder the variables in the TED. (User cost)
 – retime        Performs (forward/backward) re–timing in TED.
 – save          Save the environment.
 – scse          [CSE Static] Extract candidates, one at a time.
 – set           Set the variable bitwidth and other options.
 – setenv        Set a environment variable.
 – shifter       Replace edge weight by constant nodes.
 – 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 root.
 – tr            Construct a TED from predefined DSP transforms.
 – vars          Preset the order of variables.
 – verify        Verifies that two TED outputs are the same.
 – write         Write the existing NTL|DFG|TED into a file.
 ————— GENERAL
 – ![bin]          System call to execute bin.
 – h[elp]          Print this help.
 – e[xit]          Exit the shell.
 – q[uit]          Quit the shell.
 – history         Print all executed commands with execution times.
 – time            Show elapsed time for the last command executed.
 – man             Print the manual of the given command.
 – Use <TAB> for command completion. i.e. type fl<TAB> for flip.
```

1.1 Taylor Decomposition System Data Structure

The Taylor Decomposition System (TDS) is composed of a shell integrating three data structures as seen in figure 1.1, the Taylor Expansion Diagram (TED), the Data Flow Graph (DFG) and Netlist (NTL).

1. TED captures the functionality of an algebraic data path and performs optimizations on the behavioral level.
2. DFG provides a mechanism to visualize the data-path being implemented by TED.
3. NTL provides a mechanism to communicate with a high level synthesis tool (GAUT).

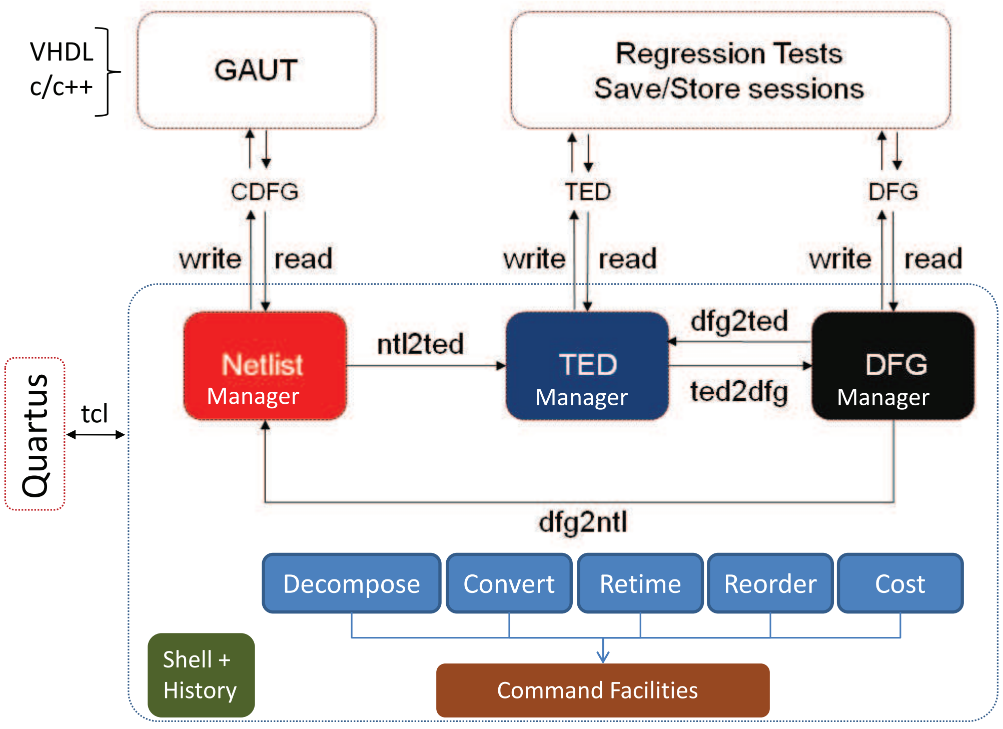


Figure 1.1. Taylor Decomposition System's internal data structures.

The TED manager is the main data structure containing the canonical TED graph. Most of the algorithms described related to TED are implemented within the TED manager by visitor classes as shown in Figure 1.2.

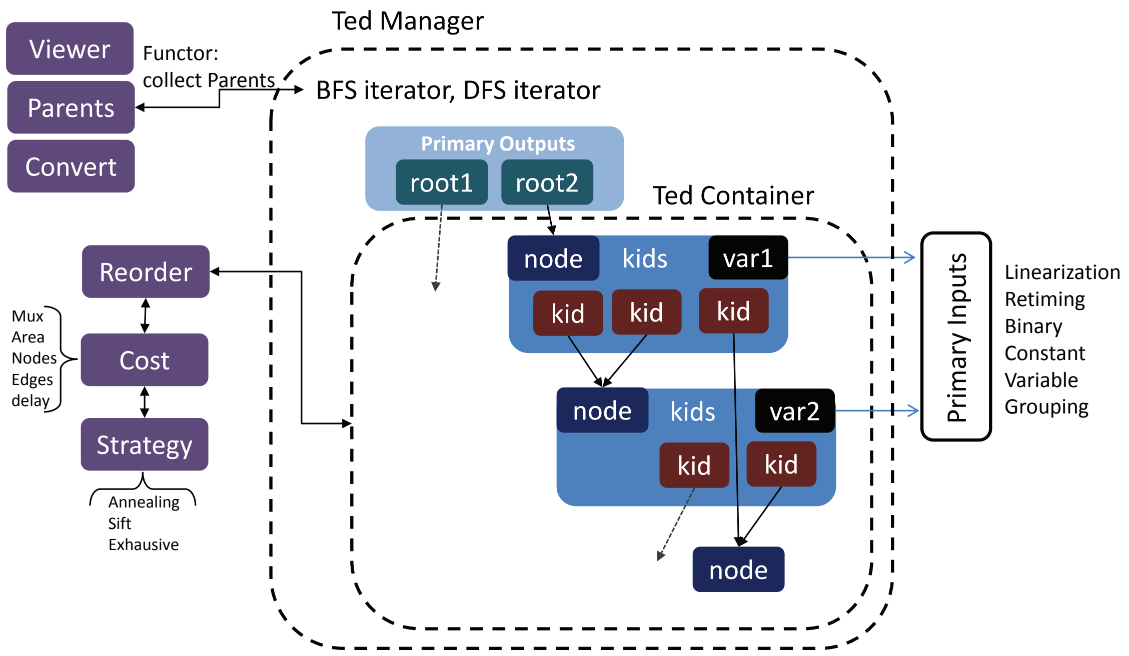


Figure 1.2. Taylor Expansion Diagram's internal data strcuture.

Figure 1.3 provides an additional view of how the different data structures in TDS are integrated. The NTL data structure is used to import and export Control Data Flow Graphs (CDFG) generated by high level synthesis tool. The TDS system has been interfaced to GAUT (which can be obtained for free at http://hls-labsticc.univ-ubs.fr/) as its primary high level synthesis engine. Additionally, the TDS system also supports importing and exporting control data flow graphs in eXtendded Markup Language (XML) format. The high level synthesis tool parses C/C++ designs, compiles them, and traslate them into CDFG or XML formats which can then used by TDS to perform different optimizations. In the TDS system optimizations are encoded into scripts which can be executed as file batch or command by command through the shell console.

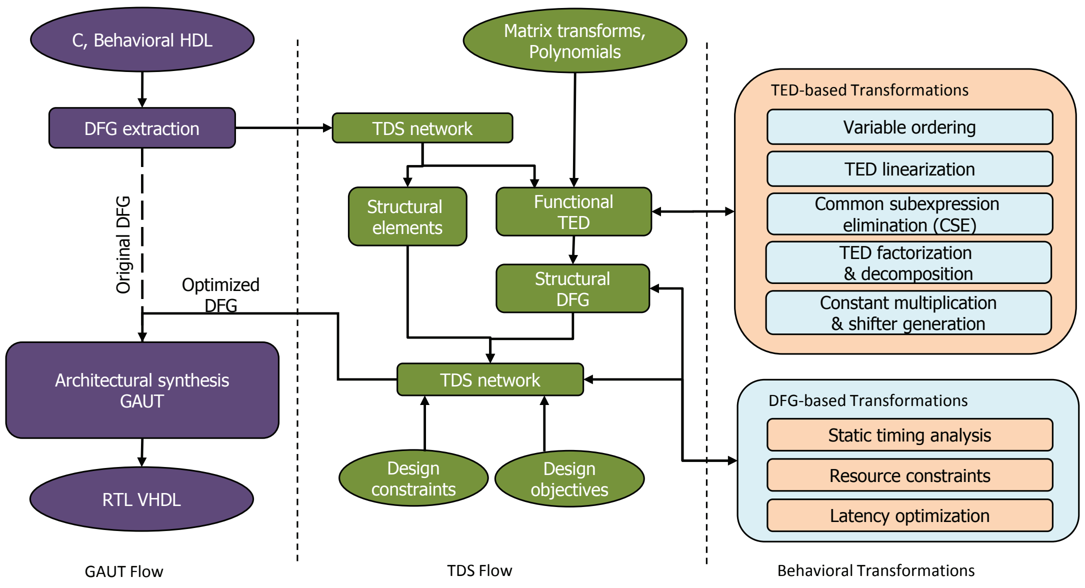
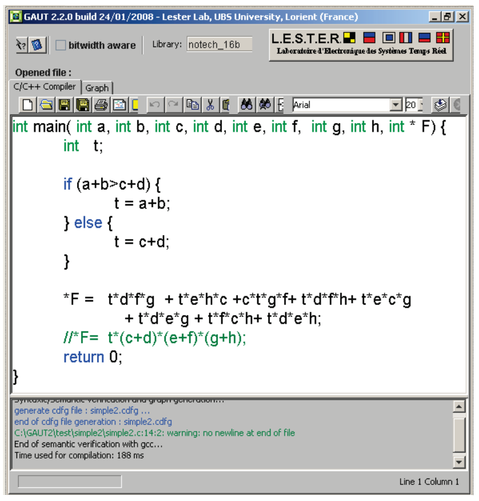


Figure 1.3. Taylor Decomposition System flow.

Although CDFGs 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 design netlist to be represented by a set of TEDs, in which the input of some TEDs are the outputs of other TEDs. Figure 1.4(a) shows a C design in GAUT, and figure 1.4(b) shows the NTL data structure with structural elements imported into TDS.

The optimized CDFG provided by TDS can afterwards given back to the high level synthesis tool to finish the synthesis process and generate a Register Transfer Level (RTL). The final hardware cost after logic synthesis, map and routing can be obtained withi the TDS shell by compiling the design into an Altera project and running the Altera Quartus tool (which can be downloaded for free at http://www.altera.com/products/software/sfw-index.jsp).

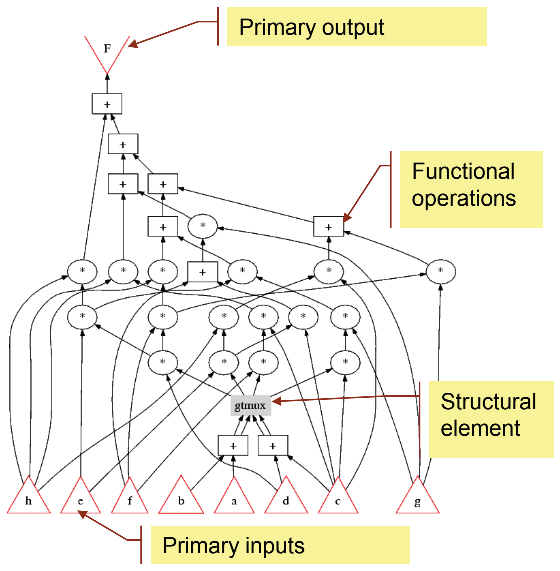

GAUT 2.2.0 build 24/01/2008 - Lester Lab, UBS University, Lorient (France)
bitwidth aware
Library: notech_16b
L.E.S.T.E.R.
Laboratoire d'Electronique des Systèmes Temps Réel
Opened file :
C/C++ Compiler
Graph
Arial
20
Line 1 Column 1

```
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

(a)



Initial TDS network

(b)

Figure 1.4. (a) GAUT system. (b) The control data flow graph imported into TDS.

1.1.1 TDS interface with high level synthesis tools

Besides importing and exporting CDFGs, the comamnds read and write allow to store and load the TED and DFG internal data structures. These commands, as shown in listings 1.1 and 1.2, recognize the input and output format based in the filename extension.

Listing 1.1. Import Facility

```
Tds 01> read -h
NAME
  read - Read a script, a CDFG, a TED or a DFG.
SYNOPSIS
  read inputfile.[c|cpp|scr|poly|mx|ted|cdfg|xml]
OPTIONS
 -h, --help
    Print this message.
  input file
    The file to read could be any of the following extensions:
     - c|cpp      for c files
     - poly|scr   for script files
     - mx         for matrix files
     - ted        for ted data structure files
     - cdfg       for files generated from GAUT
     - xml        for files generated from GECO
 [--no_dff]
     disables register "DFF" discovering when reading the cdfg
SEE ALSO
  write, show, purge
```

Listing 1.2. Export Facility

```
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 [DEFAULT BEHAVIOR]
  outputfile
    The desired output file name. The extension defines the format:
     - c          c language.
     - 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.
     - xml       XML format for the DFG data structure.
     - scr|poly  generates a script file from the command history.
NOTE
  By default the file format determines the data structure from which
  the file will be writed: 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 its into a cdfg file
  write poly1.dfg
    ... writes the DFG in file poly1.dfg
SEE ALSO
  read, purge
```

1.1.2 Transforming Internal Data Structures

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 shown in figure 1.3 by options –normal and –factor

Listing 1.3. Transforming TED into DFG

```
Tds 01> ted2dfg -h
NAME
  ted2dfg - Generate a DFG from the TED.
SYNOPSIS
  ted2dfg [method]
OPTIONS
  -h, --help
    Print this message.
  [method]
  -l, --latency [--cluster, --clusterA, --clusterD] [--level]
    Generate a DFG by balancing all operations on it
    Sub option: --clusterA
    Generate the DFG using only TEDs in clusterA.
    Sub option: --clusterD
    Generate the DFG using only TEDs in clusterD.
    Sub option: --cluster
    Generate the DFG using TEDs in both clusterA and clusterD.
    Sub option: --level
    Maintain the delay level of extracted boxes in the Netlist.
  -n, --normal [--cluster, --clusterA, --clusterD]
    Generate a one to one translation of the DFG through a TED NFF traversal [DEFAULT BEHAVIOR].
  -f, --factor, --flatten [--show]
    Flatten the DFG by factorizing common terms in the TED 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 exist this command will
  overwrite it.
EXAMPLE
  poly X = a-b+c
  poly Y = a+b-c
  poly F = X+Y
  dfg2ted --normal
  show --dfg
  echo produces a DFG with outputs X,Y and F
  purge --dfg
  dfg2ted --factor
  show --dfg
  echo polynomials X and Y disappear in the DFG as evaluation of F,
  echo the resulting polynomial F = 2*a, has no record of X or Y
SEE ALSO
  ted2ntl, ntl2ted, dfg2ted, dfg2ntl
```

1.2 TDS Environment

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 shown in listing 1.4.

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

```
Tds 01> printenv
```

Listing 1.4. TDS Environment

```
######################################
# Environment file generated by TDS.  #
# http://incascout.ecs.umass.edu/main #
######################################
# Environment variable  |  Value
#-----------------------|-----------------------
     bitwidth_fixedpoint | 4,28
        bitwidth_integer | 32
           cdfg_bin_path | /home/daniel/Gaut/GAUT_2_4_2/GautC/cdfgcompiler/bin/
          const_as_vars  | false
         const_cdfg_eval | false
            const_prefix | const_
     default_design_name | tds2ntl
                delayADD | 1
                delayLSH | 1
                delayMPY | 2
                delayREG | 1
                delayRSH | 1
                delaySUB | 1
                 dot_bin | dot
             fpga_device | AUTO
             fpga_family | "Stratix II"
  gaut_allocate_strategy | -distributed_th_lb
           gaut_bin_path | /home/daniel/Gaut/GAUT_2_4_2/GautC/bin/
            gaut_cadency | 200
              gaut_clock | 10
     gaut_cost_extension | .gcost
   gaut_gantt_generation | false
           gaut_lib_path | /home/daniel/Gaut/GAUT_2_4_2/GautC/lib/
                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_lib | notech_16b.lib
       gaut_tech_lib_vhd | notech_lib.vhd
           gaut_tech_vhd | notech.vhd
         negative_prefix | moins_
                  ps_bin | evince
       quartus_bin_path  |
                    rADD | 0
                    rMPY | 0
                    rSUB | 0
            reorder_type | proper
            show_bigfont | false
          show_directory | ./dotfiles
              show_level | true
            show_verbose | false
#-----------------------------------------------
# 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"}
```

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

```
Tds 02> setenv reorder_type = proper
```

The current environment can be saved:

```
Tds 03> save [optional filename argument, default is tds.env]
```

or load and re-load:

```
Tds 04> load [optional filename argument, default is tds.env]
```

The 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.

1.3 ALIASES

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 with the format shown in listing 1.5. 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.

Listing 1.5. Alias Commands

```
################################################################
#reserved word      alias name       semi-colon separated commands#
################################################################
alias               flatten          ted2dfg -n; dfgflatten; dfg2ted
alias               stats            print -s
alias               shifter*         shifter --ted; remapshift
alias               gautn            cost -n
alias               gautd            cost -d
alias               gautt            cost -t
```

1.4 Commands

1.4.1 Variable Ordering

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 three reordering algorithms as shown in listing 1.6.

Listing 1.6. Alias Commands

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

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.

1.4.2 Optimization

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*** as shown in listing 1.7.

Listing 1.7. Heuristic number of iterations

```
[iteration strategy]
 -a, --annealing [--no_stride] [--stride_backtrack] [--pJumpAbove]
                 [--pJumpBelow] [--beta] [--alpha] [--ratio] [--adjustment]
   It minimizes the cost function by using the annealing algorithm
   Sub option: --no_stride
                   Prevents (groups of a) multiple output TEDs to be treated as individual TEDs
   Sub option: --stride_backtrack
                   Enables backtracking after annealing each single stride
   Sub option: --probAnnealingJumpAbove,-paja double_number_between_0_and_1
                   Defines the probability that one variable will jump above another one
   Sub option: --probAnnealingJumpBelow, -pajb double_number_between_0_and_1
                   Defines the probability that one variable will jump below another one
   Sub option: --beta double_number_between_0_and_1
                   Defines a scale factor for the initial temperature of the annealing algorithm
   Sub option: --alpha double_number_between_0_and_1
                   Defines a scale factor for the new temperature of the annealing algorithm
   Sub option: --ratio integer
                   Defines a scale factor for the number of levels
   Sub option: --adjustment integer
                   Defines an adjustment to the initial cost of the annealing algorithm
 -e, --exhaustive [--end] [--no_stride]
   Tries all possible orders by doing permutation, is O(N!)
   Sub option: --end
                   Prevents the abortion of the permutation when 'ESC' is pressed
   Sub option: --no_stride
                   Prevents clustering multiple output TEDs
 -s, --sift [-g, --group]
   Moves each variable at a time through out 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 (only one cost function at a time) as shown in listing 1.8.

Listing 1.8. Cost functions for command **reorder**

```
 [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
      tLatency               = the latency computed from 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"
 -tl, --tLatency
   Minimizes the TED latency, its critical path, subject to the resources specified in the environment variables
  -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
```

1.4.3 Gaut and Quartus

For example, the estimated latency and area of implementing polynomial $F = fbh + a + cb + gfb + edb$ without TED optimization in GAUT is 100ns and 91 units. In Altera the frequency obtained is 167Mhz with 287 ALUs. See listing 1.9 for the commands used.

Listing 1.9. Design $F = fbh + a + cb + gfb + edb$ without optimization

```
Tds 01> poly F=f*b*h+a+c*b+g*f*b+e*d*b
Tds 02> cost
Cadency is 200
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|-----------------------------------------------|
|                      TED                      |
|-----------------------------------------------|
|  node | edge0 |   edgeN   | factor |   width  |
|-----------------------------------------------|
|     9 |     4 |         4 |      0 |        0 |
|-----------------------------------------------|
|         DFG           |        Schedule       |
|-----------------------|-----------------------|
| nMUL | nADD | nSUB | Latency | rMPY | rADD |
|    4 |    4 |    0 |       6 |    2 |    1 |
|-----------------------------------------------|
|                      Gaut                     |
|-----------------------------------------------|
|   Muxes  |  Latency  |  Register  |   Area   |
|       96 |       100 |          6 |       91 |
design name: ted
Tds 03> setenv gaut_cadency=100
Tds 04> cost
Cadency is 100
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295
|-----------------------------------------------|
|                      TED                      |
|-----------------------------------------------|
|  node | edge0 |   edgeN   | factor |   width  |
|-----------------------------------------------|
|     9 |     4 |         4 |      0 |        0 |
|-----------------------------------------------|
|         DFG           |        Schedule       |
|-----------------------|-----------------------|
| nMUL | nADD | nSUB | Latency | rMPY | rADD |
|    4 |    4 |    0 |       6 |    2 |    1 |
|-----------------------------------------------|
|                      Gaut                     |
|-----------------------------------------------|
|   Muxes  |  Latency  |  Register  |   Area   |
|       96 |       100 |          6 |       91 |
design name: ted
Tds 05> ted2dfg
Tds 06> dfg2ntl
Tds 07> cost -n
Cadency is 100
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 |
|     0 |       0 |        0 |    0 |    0 |    0 |       0 |    0 |    0 |    48 |     100 |        5 |   91 |
design name: dfg_2ntl
Tds 08> quartus -p
Tds 09> quartus -c
Info: Report file saved in "dfg_2ntl.quartus.report"
Info: Loading report file "dfg_2ntl.quartus.report"

clock name | target freq | design freq
-----------------------------------------
clk        | 1000.0 MHz  | 167.76 MHz
Resources                         | Synthesis | Fitter
---------------------------------------------------------------------
Combinational ALUTs               | 263       | 287 / 12,480 ( 2 % )
-- 7 input functions              | 0         | 0
-- 6 input functions              | 70        | 71
-- 5 input functions              | 115       | 114
-- 4 input functions              | 26        | 26
-- <=3 input functions            | 52        | 76
Dedicated logic registers         | 90        | 90 / 12,480 ( 1 % )
---------------------------------------------------------------------
Assignment to |: PARTITION_HIERARCHY = root_partition
There are 510 ADDs in the design
There are 510 SUBs in the design
There are 0 shifts in the design
```

Optimizing the design using the command **reorder** as shown in listing 1.10 gives an estimated latency of 70ns with an area of 348 units in GAUT, and a frequency of 182.58Mhz with 720 ALUs in Quartus.

Listing 1.10. Design F optimized with: reorder –annealing -gl

```
Tds 01> poly F=f*b*h+a+c*b+g*f*b+e*d*b
Tds 02> setenv gaut_cadency=100
Tds 03> reorder --annealing -gl
strade=111111110
[========================================] 100% Cost=80.00
Tds 04> setenv gaut_cadency=80
Tds 05> reorder --annealing -gl
strade=111111110
[========================================] 100% Cost=70.00
Tds 06> setenv gaut_cadency=70
Tds 07> reorder --annealing -gl
strade=111111110
[                                        ]   0% T=39.90 Window=49   Cost=70.00 \too strong constraints
[=====                                   ]  14% T=33.91 Window=47   Cost=70.00 \too strong constraints
[===========                             ]  27% T=28.83 Window=45   Cost=70.00 \too strong constraints
[===============                         ]  38% T=24.50 Window=43   Cost=70.00 \too strong constraints
[===================                     ]  47% T=20.83 Window=41   Cost=70.00 \too strong constraints
[=================================       ]  83% T=6.68  Window=27   Cost=70.00 \too strong constraints
[=====================================   ]  94% T=2.14  Window=13   Cost=70.00 \too strong constraints
[========================================] 100%
Tds 08> ted2dfg
Tds 09> balance -d
Tds 10> dfg2ntl
Tds 11> cost -n
Cadency is 70
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 |
|     0 |       0 |        0 |    0 |    0 |    0 |       0 |    0 |    0 |    48 |      70 |        8 |  348 |
design name: dfg_2ntl
Tds 12> quartus -p
Tds 13> quartus -c
Info: Report file saved in "dfg_2ntl.quartus.report"
Info: Loading report file "dfg_2ntl.quartus.report"

clock name | target freq | design freq
---------------------------------------
clk        | 1000.0 MHz  | 182.58 MHz
Resources                       | Synthesis | Fitter
---------------------------------------------------------------------
Combinational ALUTs             | 696       | 720 / 12,480 ( 6 % )
-- 7 input functions            | 0         | 0
-- 6 input functions            | 67        | 67
-- 5 input functions            | 339       | 339
-- 4 input functions            | 100       | 102
-- <=3 input functions          | 190       | 212
Dedicated logic registers       | 135       | 135 / 12,480 ( 1 % )
---------------------------------------------------------------------
Assignment to |: PARTITION_HIERARCHY = root_partition
There are 510 ADDs in the design
There are 510 SUBs in the design
There are 0 shifts in the design
```

The command **reorder*** differs from command **reorder** in that it accept a list of cost functions to be optimized. The least of cost functions should be entered in such that the least important cost function is given first, and the most important cost function is given last. The cost functions available are shown in listing 1.11.

Listing 1.11. Cost functions for command **Reorder***

```
 [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
 -tl, --tLatency
   Minimizes the critical path computed in the TED graph
 --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
```

Optimizing the design using the command **reorder*** as shown in listing 1.12 gives a latency of 70ns in GAUT; and a frequency of 193.12Mhz with 411 ALUs in Quartus.

Listing 1.12. Design F optimized with: reorder –annealing -gr -gm -ga -gl

```
Tds 01> poly F=f*b*h+a+c*b+g*f*b+e*d*b
Tds 02> setenv gaut_cadency=100
Tds 03> reorder* --annealing -gr -gm -ga -gl
strade=111111110
[=====================================] 100% Cost 80.00
Info: backtracking previous result with cost 80
Tds 04> setenv gaut_cadency=80
Tds 05> reorder* --annealing -gr -gm -ga -gl
strade=111111110
[=====================================] 100% Cost 70.00
Tds 06> setenv gaut_cadency=70
Tds 07> ted2dfg
Tds 08> balance -d
Tds 09> dfg2ntl
Tds 10> cost -n
Cadency is 70
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 |
|     0 |       0 |        0 |    0 |    0 |    0 |       0 |    0 |    0 |    80 |      70 |        7 |  174 |
design name: dfg_2ntl
Tds 11> quartus -p
Tds 12> quartus -c
Info: Report file saved in "dfg_2ntl.quartus.report"
Info: Loading report file "dfg_2ntl.quartus.report"

clock name | target freq | design freq
--------------------------------------------
clk        | 1000.0 MHz  | 193.12 MHz
Resources                          | Synthesis | Fitter
---------------------------------------------------------------
Combinational ALUTs                | 409       | 411 / 12,480 ( 3 % )
-- 7 input functions               | 0         | 0
-- 6 input functions               | 66        | 66
-- 5 input functions               | 226       | 226
-- 4 input functions               | 25        | 27
-- <=3 input functions             | 92        | 92
Dedicated logic registers          | 119       | 119 / 12,480 ( 1 % )
---------------------------------------------------------------
Assignment to |: PARTITION_HIERARCHY = root_partition
There are 510 ADDs in the design
There are 510 SUBs in the design
There are 0 shifts in the design
```

1.4.4 Registers

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
```

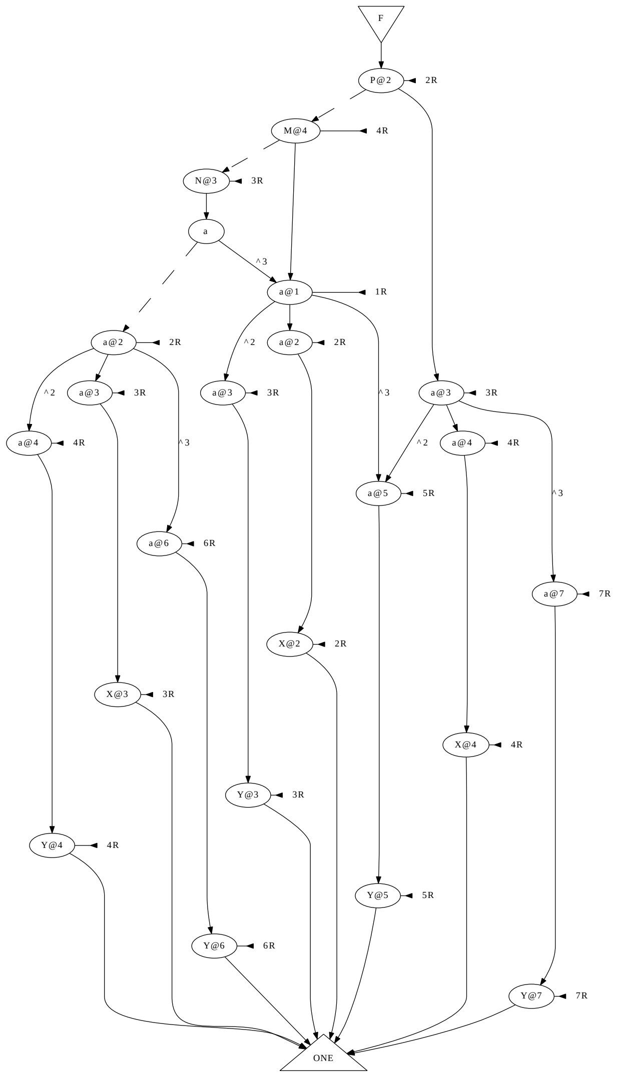


Figure 1.5. Unretimed TED.

```
Tds 04> ted2dfg -n
Tds 05> show -d unretimed_dfg.dot
```

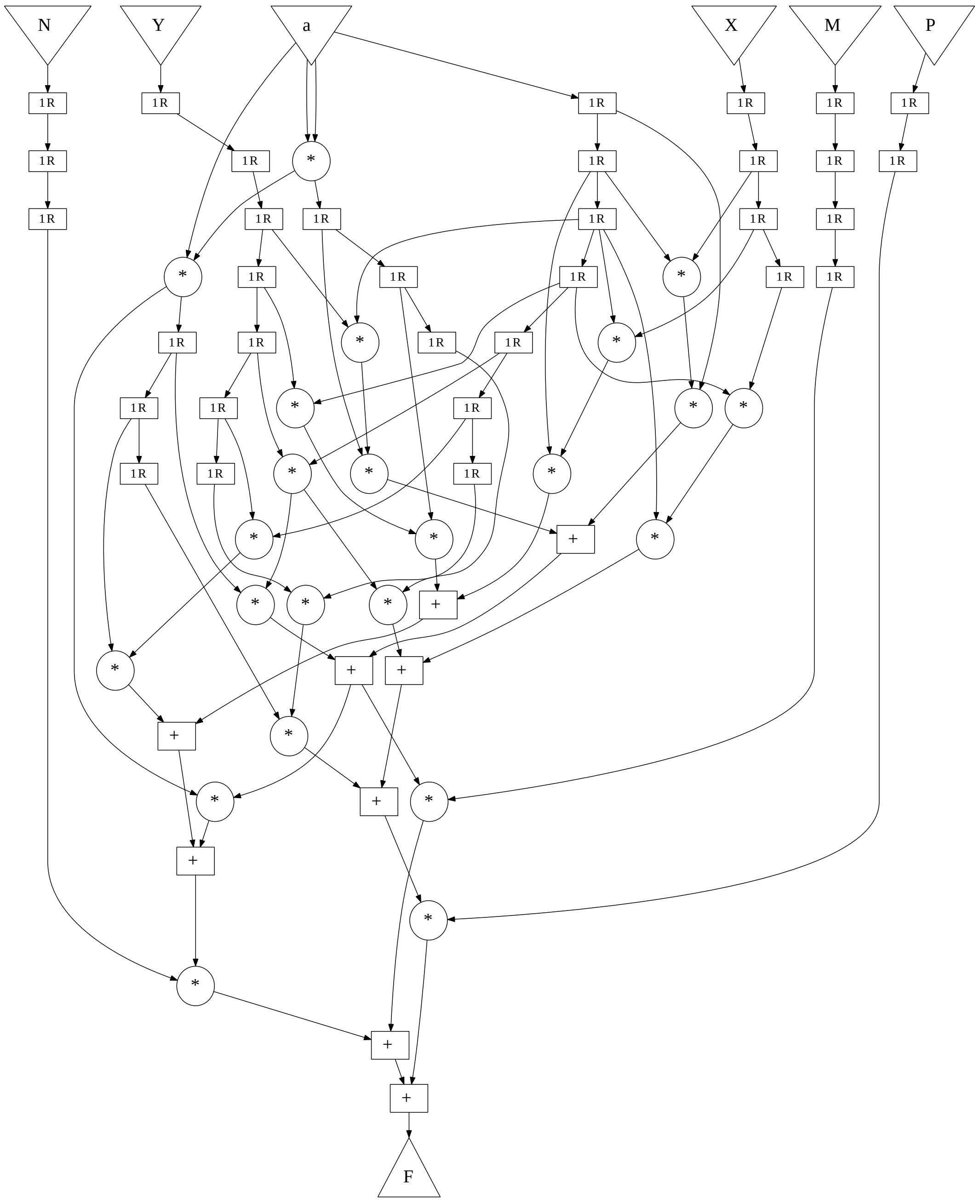


Figure 1.6. DFG corresponding to the unretimed TED.

```
Tds 06> retime -up
Suggestion to restore ordering visualization:
jumpAbove -p a a
Tds 07> show retimed_ted.dot
```

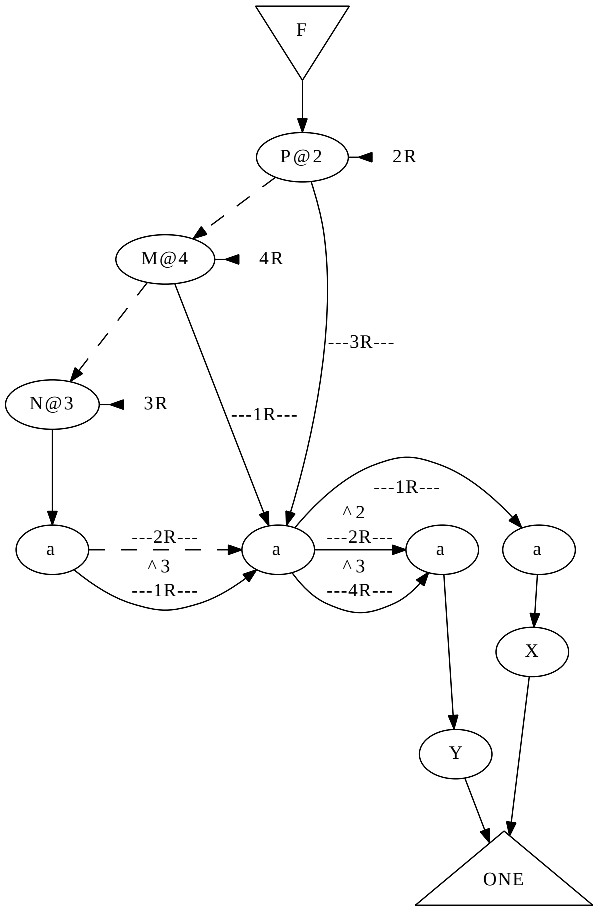


Figure 1.7. Retimed TED.

```
Tds 08> ted2dfg -n
Tds 09> show -d retimed_dfg.dot
```

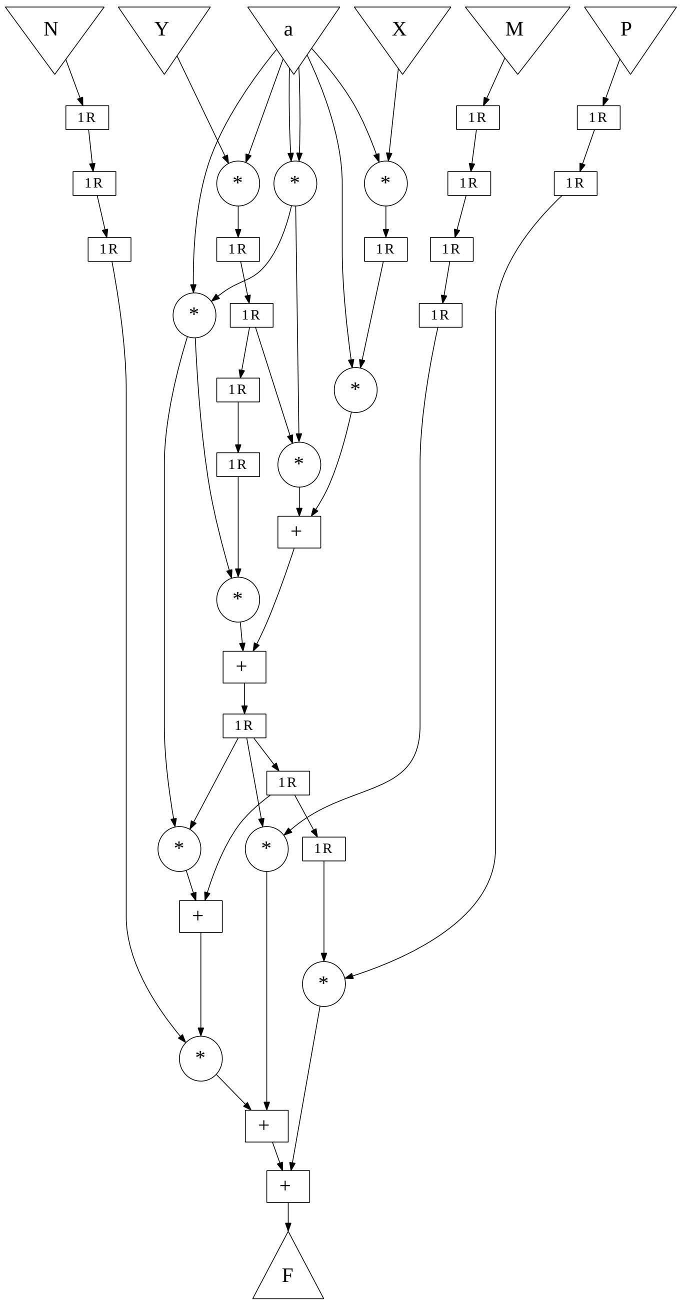


Figure 1.8. DFG corresponding to the retimed TED.

1.4.5 Precission

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 bitwidth of each variable.

Listing 1.13. Annotating bitwidth in TED

```
Tds 01> 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)-> 32
* fixedpoint(fxp)-> 4,28
  [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.

Listing 1.14. Compute bitwidth required for exact computation

```
Tds 01> compute -h
NAME
   compute - Annotate the bit-widths required for exact computation.
SYNOPSIS
   compute [-t -b --snr] | [-d -b -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.

Let's look at the following synthetic example to see how bitwidth optimization works.

```
Tds 01> poly F1=a-b+c+d
Tds 02> poly F2=(a+b)*(c+d)^2
Tds 03> show precision_not_annotated.dot
Tds 04> set -b fixedpoint a:4,16 c:2,10 b:4,12 d:4,12
Tds 05> show precision_ted.dot
```

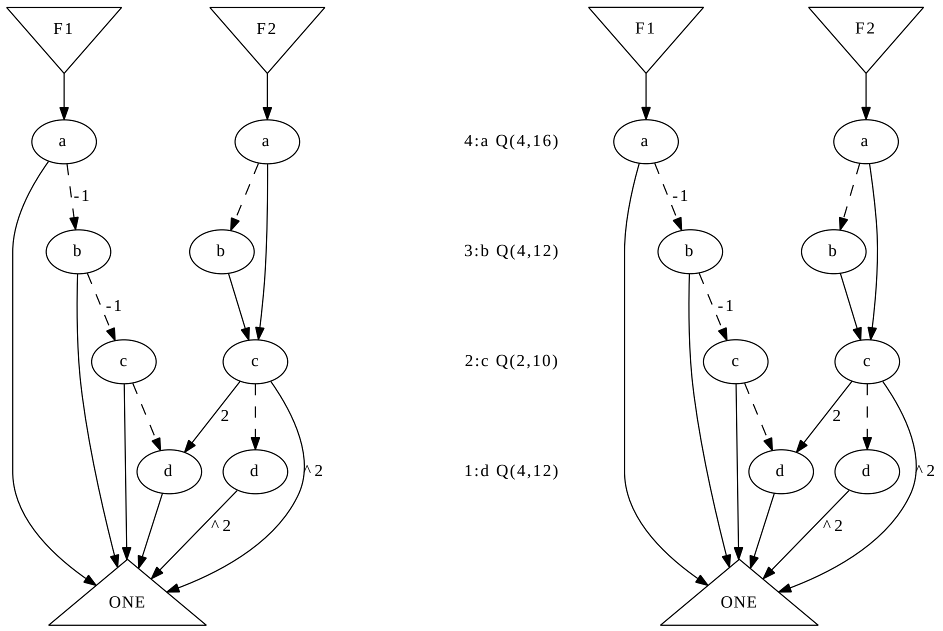


Figure 1.9. (a) Initial TED. (b) TED with bitwidth annotation in nodes.

```
Tds 06> compute -t -b
Tds 07> show precision_computed_ted.dot
```

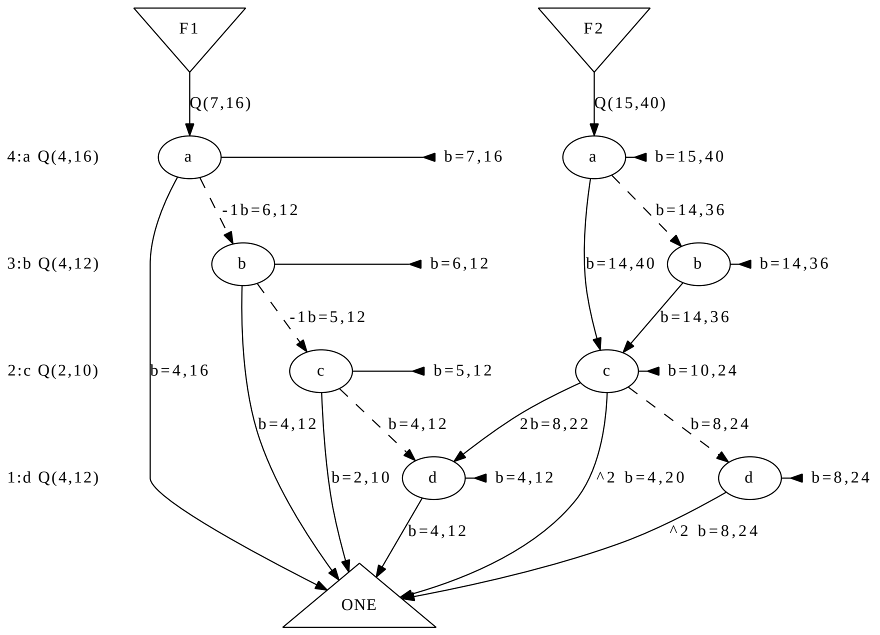


Figure 1.10. TED with bitwidth annotation for exact computation.

```
Tds 08> ted2dfg -f
Tds 09> compute -b -d
Tds 10> show --ids -d precision_computed_dfg.dot
```

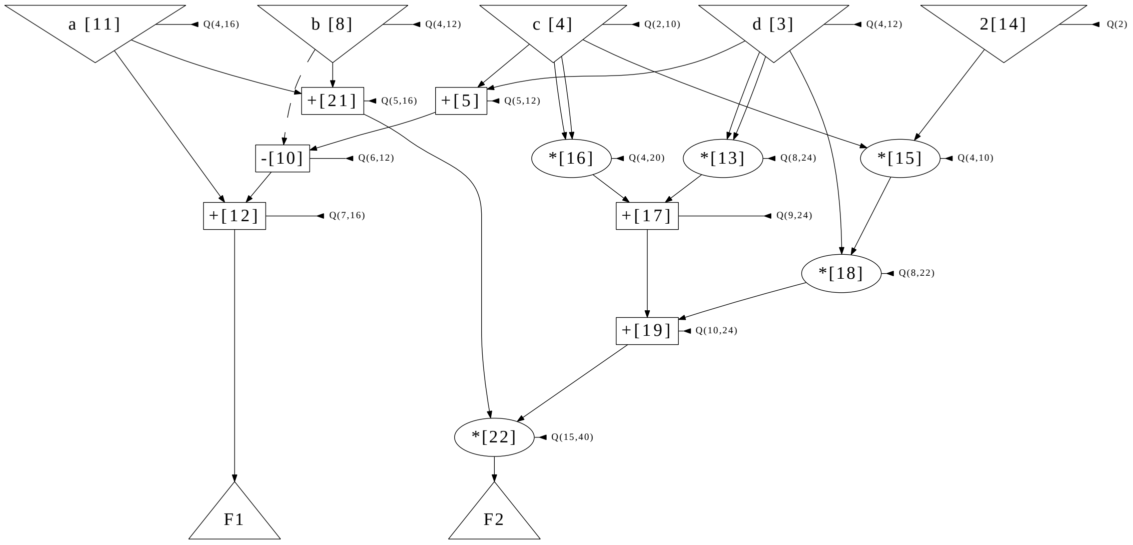


Figure 1.11. DFG with bitwidth annotation for exact computation.

```
Tds 11> set -r d:[0.128,1.123432] b:[-5.3223,321.32e-3] c:[0,1]
Tds 12> set -e F1:1.324 F2:0.983
Tds 13> compute -d -g
Tds 14> optimize
[====================================] 100\%
Done. Type "info -d" for more information
Tds 15> show --ids -d precision_optimized_dfg.dot
```

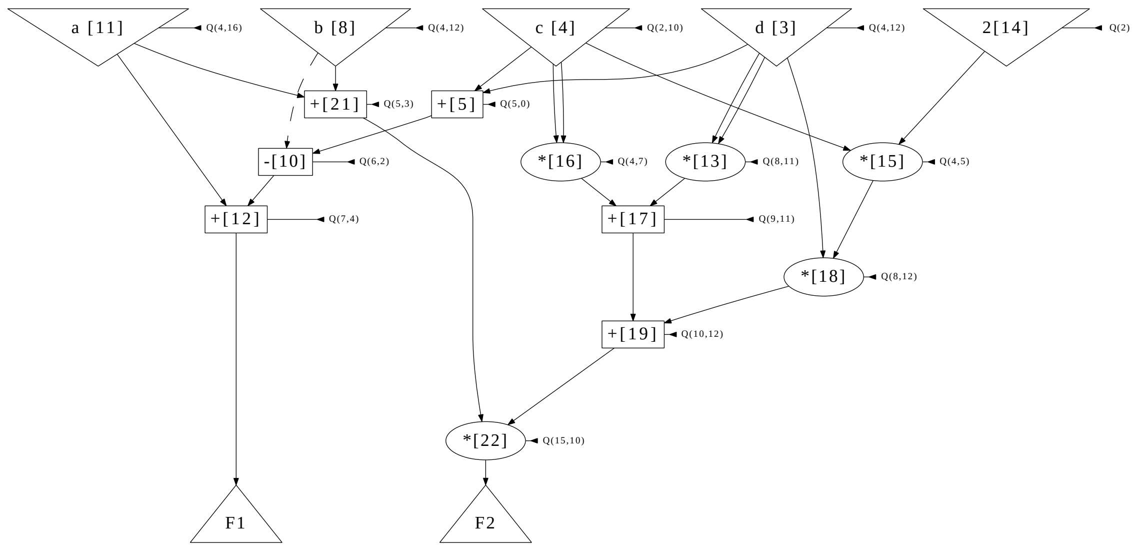


Figure 1.12. DFG with bitwidth optimization for target error.

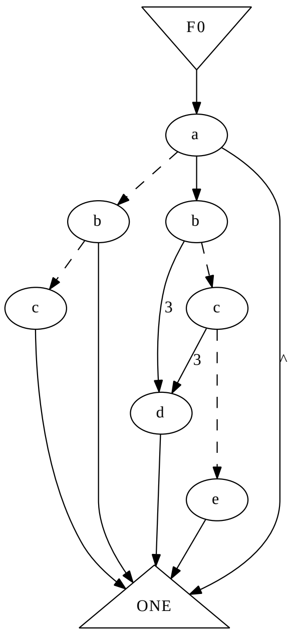
```
Tds 16> info -d
|------------------------------------------------------------------------|
| Level |  OP  |  ID  |  bitwidth   | Left ID | Right ID | Ref |
|-------|------|------|-------------|---------|----------|-----|
|     0 |    2 |   14 | Q(       2) |    NONE |     NONE |   1 |
|     0 |    a |   11 | Q(    4,16) |    NONE |     NONE |   2 |
|     0 |    b |    8 | Q(    4,12) |    NONE |     NONE |   2 |
|     0 |    c |    4 | Q(    2,10) |    NONE |     NONE |   4 |
|     0 |    d |    3 | Q(    4,12) |    NONE |     NONE |   4 |
|     1 |    * |   16 | Q(     4,7) |       4 |        4 |   1 |
|     1 |    * |   15 | Q(     4,5) |       4 |       14 |   1 |
|     1 |    * |   13 | Q(    8,11) |       3 |        3 |   1 |
|     1 |    + |    5 | Q(     5,0) |       4 |        3 |   1 |
|     1 |    + |   21 | Q(     5,3) |      11 |        8 |   1 |
|     2 |    + |   17 | Q(    9,11) |      16 |       13 |   1 |
|     2 |    - |   10 | Q(     6,2) |       5 |        8 |   1 |
|     2 |    * |   18 | Q(    8,12) |      15 |        3 |   1 |
|     3 |    + |   12 | Q(     7,4) |      11 |       10 |   0 | PO
|     3 |    + |   19 | Q(   10,12) |      17 |       18 |   1 |
|     4 |    * |   22 | Q(   15,10) |      21 |       19 |   0 | PO
|------------------------------------------------------------------------|
```

1.4.6 Linearization

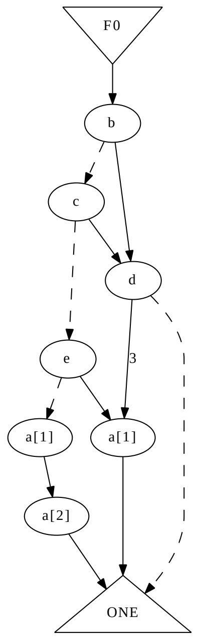
The TED data structure can have multiple edges as shown in the TED of Figure 1.13(a), nonetheless all internal nodes can be forced to have only additive-edges and multiplicative-edges through linearization as shown in Figure 1.13(b).

```
Tds 01> poly a^2+b+c+(3*(b+c)*d+e)*a
Tds 021> show
Tds 03> linearize
Tds 04> bottom a
Tds 04> show
```

The node with variable a in Figure 1.13(a) has an edge connected to node ONE with power $\hat{2}$, which represents a term a^2. After linearizing the TED, it can be observed that the edge has been replaced by two nodes $a[1]$ and $a[2]$ both of type a.



(a)



(b)

Figure 1.13. (a) TED for function $F0 = a^2 + b + c + (3(b + c)d + e)a$. (b) linearized TED.

1.4.7 Decomposition

A TED can be further decomposed in terms of chain of adders or chain of multipliers by using the command **decompose**.

Listing 1.15. Annotating bitwidth in TED

```
Tds 01> decompose -h
NAME
   decompose - Decompose the TED in its Normal Factor Form.
SYNOPSIS
   decompose [-a|--pt|--st] [--force]
OPTIONS
  -h, --help
     Print this message.
  --force
     perform aggressive extraction by treating support as primary outputs
  --st
     Decompose all sum terms available in the current TED
  --pt
     Decompose all product terms available in the current TED
  -a, --all
     Decompose changing the given order if necessary until
     the entire TED is reduced to a single node
DETAILS
   The TED must be linearized first
SEE ALSO
   show
Tds 01>
```

The command **decompose –all** applied to the TED shown in Figure 1.13(b) results in the TED shown in Figure 1.14. The resulting TED contains pseudo outputs labeled PT and ST corresponding to product terms and sum terms respectively.

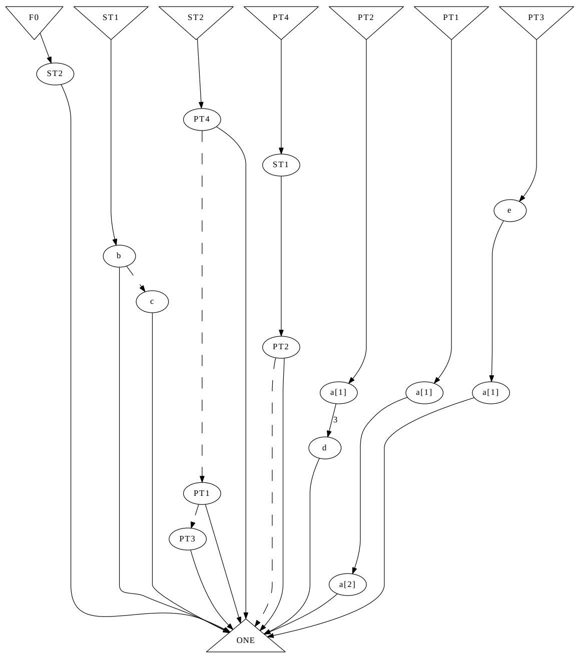


Figure 1.14. Decomposed TED.

1.4.8 TED to DFG Transformations

Although the TED data structure is canonical, that is given a fixed variable ordering the representation of its data structure is unique, the DFG generated from the TED is not unique.

Listing 1.16. Annotating bitwidth in TED

```
Tds 01> ted2dfg -h
NAME
  ted2dfg - Generate a DFG from the TED.
SYNOPSIS
  ted2dfg [method]
OPTIONS
  -h, --help
     Print this message.
  [method]
  -l, --latency [--cluster, --clusterA, --clusterD] [--level]
     Generate a DFG by balancing all operations on it
     Sub option: --clusterA
     Generate the DFG using only TEDs in clusterA.
     Sub option: --clusterD
     Generate the DFG using only TEDs in clusterD.
     Sub option: --cluster
     Generate the DFG using TEDs in both clusterA and clusterD.
     Sub option: --level
     Maintain the delay level of extracted boxes in the Netlist.
  -n, --normal [--cluster, --clusterA, --clusterD]
     Generate a one to one translation of the DFG through a TED NFF traversal [DEFAULT BEHAVIOR].
  -f, --factor, --flatten [--show]
     Flatten the DFG by factorizing common terms in the TED 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 exist this command will
  overwrite it.
EXAMPLE
  poly X = a-b+c
  poly Y = a+b-c
  poly F = X+Y
  dfg2ted --normal
  show --dfg
  echo produces a DFG with outputs X,Y and F
  purge --dfg
  dfg2ted --factor
  show --dfg
  echo polynomials X and Y disappear in the DFG as evaluation of F,
  echo the resulting polynomial F = 2*a, has no record of X or Y
SEE ALSO
  ted2ntl, ntl2ted, dfg2ted, dfg2ntl
Tds 01>
```

Continuing the example shown in Figure 1.13(a), the command **ted2dfg** can be used to transform the TED data structure into a DFG data structure. Three different DFGs are shown in Figure 1.15.

```
Tds 01> poly a^2+b+c+(3*(b+c)*d+e)*a
Tds 02> linearize
Tds 03> bottom a
Tds 04> ted2dfg --normal
Tds 05> purge -d
Tds 06> ted2dfg --factor
Tds 07> purge -d
Tds 08> ted2dfg --level
```

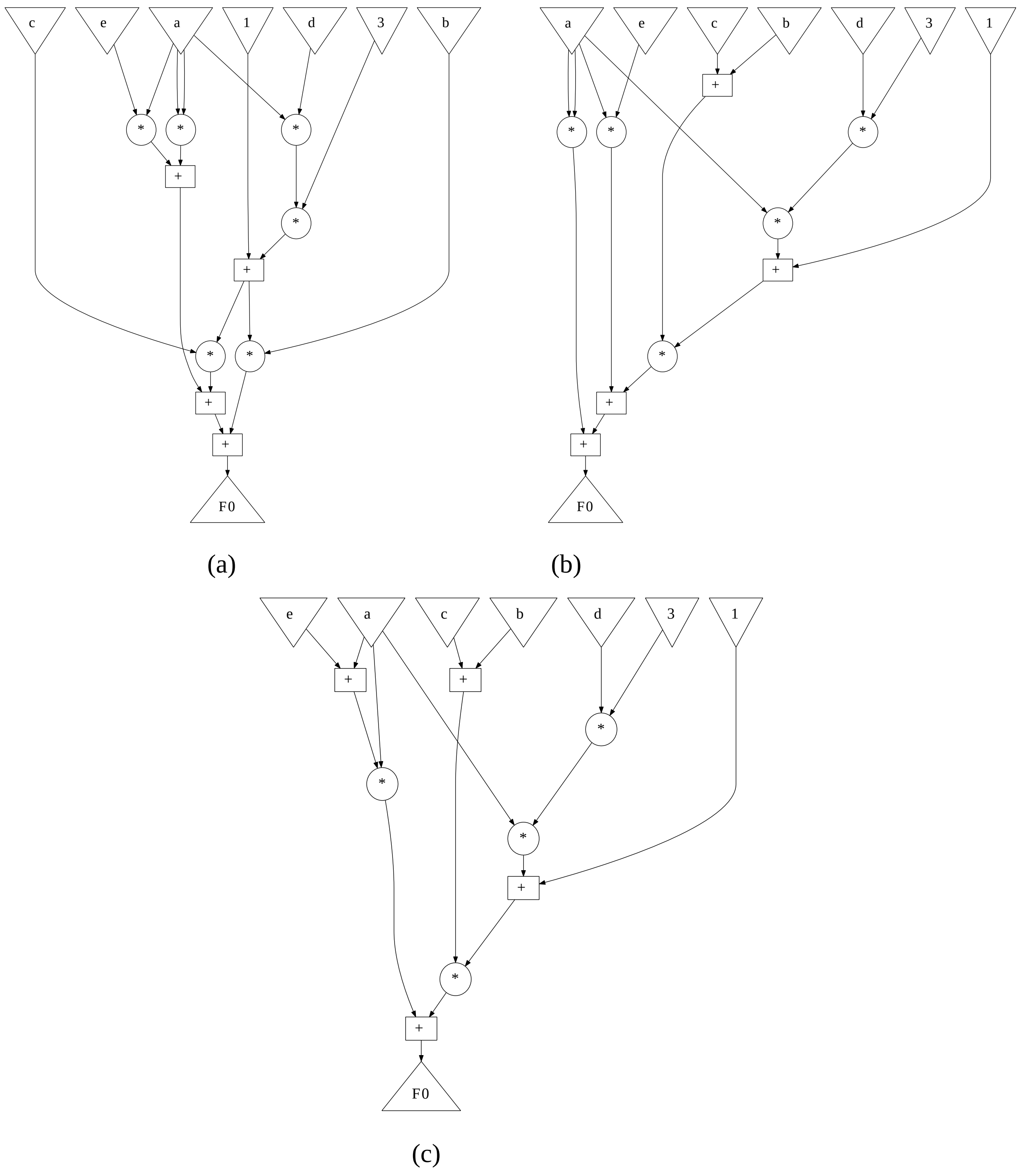


Figure 1.15. DFG generated through: (a) a normal factor form transformation. (b) factorization transformation. (c) levelized and balanced transformation.

1.4.9 Replacing constant multiplication with shifters

All constant multiplications, that is, multiplications represented by weight on edges within the TED data structure can be replaced by a series of shift operations. The first step to replace constant multiplications by shifters is to force the TED data structure to consider all weights on edges as a factor of constant node 2.

```
Tds 01> poly F1=a∗91+(b∗a)∗77−b∗7
TDS 02> show
Tds 03> shifter
Tds 04> show
```

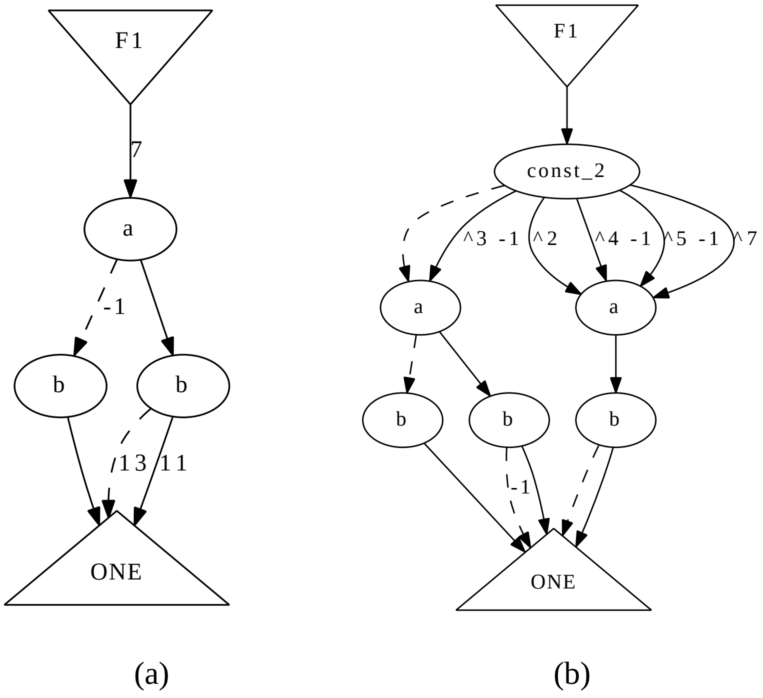


Figure 1.16. (a) TED with implicit constant multiplication on edges. (b) TED with constant multiplications explecitly represented by variable const_2

The DFG generated for the TED shown in Fgure 1.17(b) is shown in Figure **??**(a).

```
Tds 05> show −d
TDS 06> remapshift
Tds 07> show −d
Tds 08> balance −d
Tds 08> show −d
```

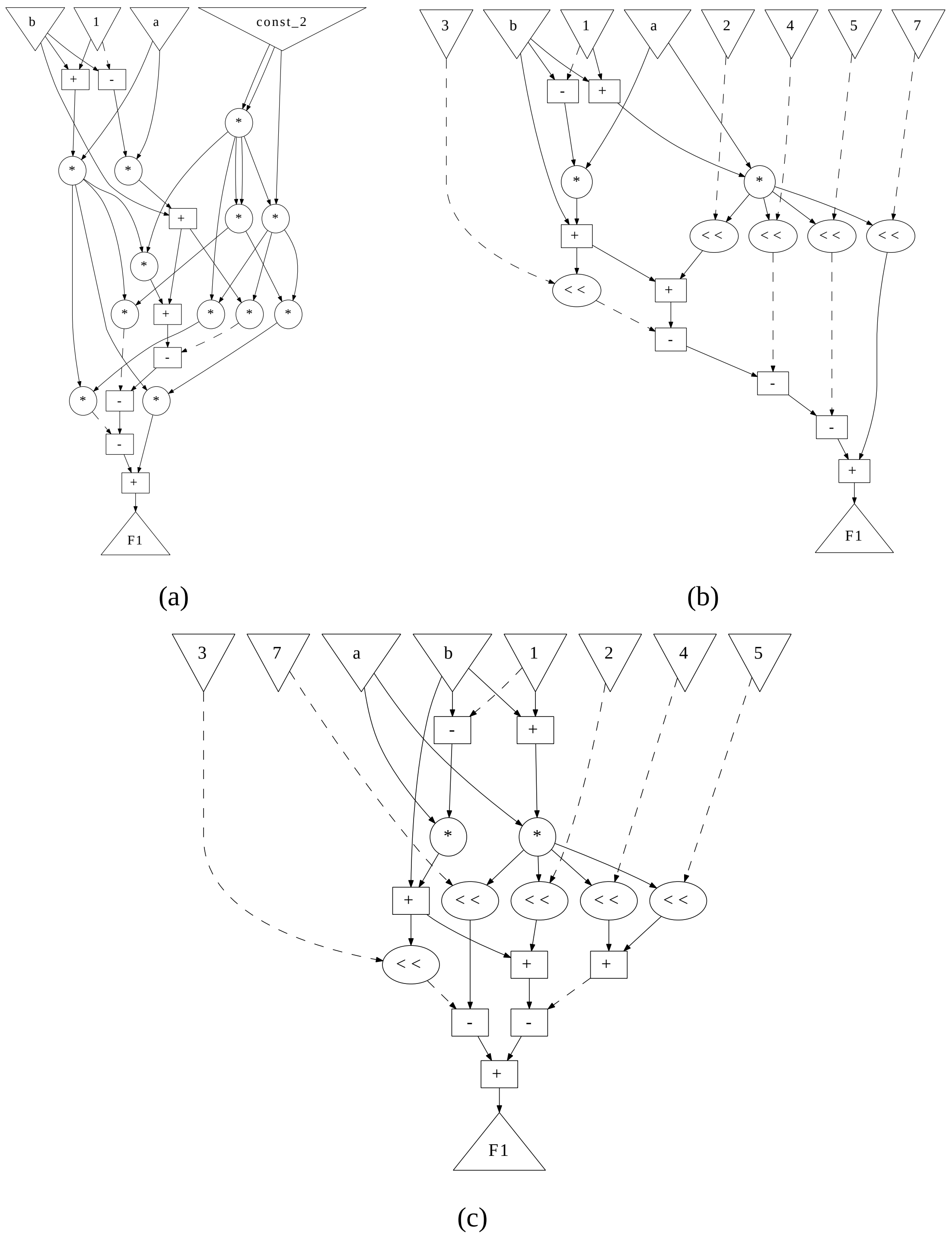


Figure 1.17. (a) DFG corresponding to TED in Figure 1.17(b). (b) DFG with replaced shifters. (c) Balanced DFG.