# TABLE OF CONTENTS

## CHAPETER

1. TAYLOR DECOMPOSITION SYSTEM MANUAL  ........................................ 1

    1.1 Taylor Decomposition System Data Structure  ............................... 3

        1.1.1 TDS interface with high level synthesis tools  .................... 6

    1.1.2 Transforming Internal Data Structures  ................................. 8

    1.2 TDS Environment .................................................................. 8

    1.3 ALIASES ......................................................................... 10

    1.4 Commands ........................................................................ 11

        1.4.1 Variable Ordering ....................................................... 11

        1.4.2 Optimization .............................................................. 12

        1.4.3 Gaut and Quartus .......................................................... 14

        1.4.4 Registers ................................................................... 17

        1.4.5 Precision ................................................................... 20

        1.4.6 Linearization ............................................................... 25

        1.4.7 Decomposition .............................................................. 26

        1.4.8 TED to DFG Transformations ........................................... 27

        1.4.9 Replacing constant multiplication with shifters ................... 29
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 partitions.
- 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.
- dfgtrash      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.
- lcsed         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 partitions 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 structure.

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 eXtended Markup Language (XML) format. The high level synthesis tool parses C/C++ designs, compiles them, and translate 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.

![Diagram of Taylor Decomposition System flow.](image)

**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 with 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).

![Example behavioral design in C](image1.png)  ![Initial TDS network](image2.png)

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 commands 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 [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:
- 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-cdfg]
disables register "DFF" discovering when reading the cdf
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]
output file
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.
- xml XML format for the DFG data structure.
- scf|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 written: cdfg->NTL, dfg->DFG, ted->TED
EXAMPLE
write poly2.cfg
... writes the NTL in a cdfg file format
write —ted poly1.cfg
... 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

TDS system manual, by Daniel Gomez-Prado 7
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: ---cluster
       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:
ted2ndl, 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/GAUT242/GautC/cdfgcompiler/bin/ |
| const_as_yars | false |
| const_cdfg_eval | false |
| const_prefix | const- |
| default_design_name | tds2ntl |
| delayADD | 1 |
| delayLSH | 1 |
| delayMYPY | 2 |
| delayREG | 1 |
| delaySUB | 1 |
| dot_bin | dot |
| fpga_device | AUTO |
| fpga_family | "Stratix II" |
| gaut_allocate_strategy | -distributedstructlib |
| gaut_binpath | /home/daniel/Gaut/GAUT242/GautC/bin/ |
| gaut_adency | 200 |
| gaut_cost_extension | .gcost |
| gaut_gantt_generation | false |
| gaut_libpath | /home/daniel/Gaut/GAUT242/GautC/lib/ |
| gaut_schedule_strategy | true |
| gaut_register_strategy | 0 |
| gaut_alloc_strategy | "--distributedstructlib" |
| gaut_toolbin | noteclib.16.lib |
| gaut_toolvhd | noteclib.vhd |
| gaut_toolvhdl | noteclib.vhd |
| negative_prefix | mins- |
| ps_bin | evince |
| quartus_binpath | "ADD", "EMP", "SUB" |
| reorder_type | proper |
| show_big_font | true |
| show_debug | false |
| show_level | true |
| show_versbose | false |

# Other possible values

# gaut_schedule_strategy | {"", "force_no_pipeline", "force_no_mobility", "no_more stage", ",
# gaut_allocate_strategy | {"--distributedstructlib", "--distributedreusestr", "--distributedreusecomp", 
# gaut_register_strategy | "0", "1", "2", "3"} 
# MIBA [default] 
# 1 MEA 
# 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.

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

Listing 1.5. Alias Commands
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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reorder</td>
<td>The proper algorithm to reorder the TED as described in the paper. (DEFAULT BEHAVIOR)</td>
</tr>
<tr>
<td>reloc</td>
<td>Reconstructs the TED with the desired order of variables. This is slow, but serves as golden reference. (TO BE DEPRECATED IN A FUTURE RELEASE)</td>
</tr>
<tr>
<td>swap</td>
<td>A Hybrid implementation, that re-structures the TED with the desired order, but internally uses reconstructs and operations on the bottom graph. (TO BE DEPRECATED IN A FUTURE RELEASE)</td>
</tr>
</tbody>
</table>

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...
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, bitwidth, 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

```plaintext
[ iteration strategy ]
-a, --annealing [-n=anneal] [-s=anneal] [-s=anneal] [--p=anneal] [--l=anneal]
-: Prevents the annealing algorithm.
Sub option: -n=anneal
-: Specifies the number of annealing iterations.
Sub option: -s=anneal
-: Specifies the step size used for the temperature reduction.
Sub option: -p=anneal
-: Enables backtracking after annealing each single iteration.
Sub option: -l=anneal
-: Enables the probability that one variable will jump above another.
Sub option: --p=anneal
-: Defines the probability that one variable will jump below another.
Sub option: --l=anneal
-: Defines a scale factor for the initial temperature of the annealing algorithm.
Sub option: --ratio=anneal
-: Defines a scale factor for the new temperature of the annealing algorithm.
Sub option: --adjust=anneal
-: Specifies a scale factor for the number of iterations.
Sub option: --adjust=anneal
-: This option only affects the SIFT algorithm.
Sub option: --group
-: Enables backtracking after annealing each single iteration.
Sub option: --group
-: Prevents the abortion of the permutation when 'ESC' is pressed.
Sub option: --group
-: Prevents clustering multiple output TEDs.
Sub option: --group
-: Enables backtracking after annealing each single iteration.
Sub option: --group
-: Enables the probability that one variable will jump above another.
Sub option: --group
-: Enables the probability that one variable will jump below another.
Sub option: --group
-: Defines a scale factor for the initial temperature of the annealing algorithm.
Sub option: --group
-: Defines a scale factor for the new temperature of the annealing algorithm.
Sub option: --group
-: Specifies a scale factor for the number of iterations.
Sub option: --group
-: This option only affects the SIFT algorithm.
Sub option: --group
```

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`

```plaintext
{ cost functions }

Definitions:
- `ted_subexpr_candiates` - Number of product terms in the TED with 2 parents connecting to ONE
- `ted_nodes` - Number of nodes in the TED graph
- `ted_latency` - Latency computed from the TED graph
- `nMUL` - Number of multiplications in the DFG graph
- `nADD` - Number of additions in the DFG graph
- `nSUB` - Number of subtractions in the DFG graph
- `nMPY` - Number of multipliers after scheduling the DFG
- `nADJ` - Number ofadders and subtractors after scheduling the DFG
- `gLatency` - Latency of the Gau implementation
- `dLatency` - Latency of the DFG implementation

Environmental 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:
- `nMUL` [Default value = 4294967295]
- `ADD` [Default value = 4294967295]
- `SUB` [Default value = 4294967295]

--node
- Minimizes the function “10*ted_nodes - ted_subexpr_candiates”

--tl, --tLatency
- Minimizes the TED latency, its critical path, subject to the resources specified in the environment variables

--nm, --nMUL [legacy --nm], --nMUL
- Minimizes the function “10*nMUL - ted_subexpr_candiates” [DEFAULT BEHAVIOR]

--op
- Minimizes the function “10*nMUL + nADD + nSUB - ted_subexpr_candiates”

--opscheduled
- Minimizes nMUL, followed by (nADD + nSUB), dag latency, iMPY, iADD

--dl, --dLatency [legacy --dl]
- Minimizes the DFG latency subject to the resources specified in the environment variables

--bw, --bitwidth
- Minimizes the bit width of the HW implementation subject to unlimited latency [resources]

--gm, --gMUX [legacy --gmux]
- Minimizes the Gau mux count in the Gau implementation (each mux is considered 2 to 1)

--gl, --gLatency [legacy --glatency]
- Minimizes the Gau latency in the Gau implementation

--gt, --gREG [legacy --garv]
- Minimizes the Gau register count in the Gau implementation

--gap
- Minimizes the upper tighter bound found through Gappas
```

TDS system manual, by Daniel Gomez-Prado
1.4.3 Gaut and Quartus

For example, the estimated latency and area of implementing polynomial \( F = f_{bh} + a + cb + g_{fb} + edb \) 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 \( F = f_{bh} + a + cb + g_{fb} + edb \) without optimization

```
Tds 01> poly F=fh*bh*a*c*b*gfbf=be+ed+hb
Tds 02> cost
cadency is 200
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295

<table>
<thead>
<tr>
<th>node</th>
<th>edge0</th>
<th>edgeN</th>
<th>factor</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>node</th>
<th>edge0</th>
<th>edgeN</th>
<th>factor</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>nMUL</th>
<th>nADD</th>
<th>nSUB</th>
<th>Latency</th>
<th>rMPY</th>
<th>rADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Gaut

design name: ted
Tds 05> ted2dfg
Tds 06> dfg2gaut
cadency is 100
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295

<table>
<thead>
<tr>
<th>node</th>
<th>edge0</th>
<th>edgeN</th>
<th>factor</th>
<th>width</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Tds 07> cost

cadency is 100
OP delays: ADD=1 SUB=1 MPY=2
resources: ADD=4294967295 MPY=4294967295

<table>
<thead>
<tr>
<th>nMUL</th>
<th>nADD</th>
<th>nSUB</th>
<th>Latency</th>
<th>rMPY</th>
<th>rADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Gaut

design name: ted
Tds 08> quartus
Tds 09> quartus

```
INFO: Report file saved in "dfg_2intl.quartus.report"
INFO: Loading report file "dfg_2intl.quartus.report"
clock name | target freq | design freq

```
<table>
<thead>
<tr>
<th>clk</th>
<th>1000.0 MHz</th>
<th>167.76 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td>Synthesis</td>
<td>Fitter</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Combinalion ALUTS</th>
<th>263</th>
<th>287 / 12,480 ( 2 % )</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 input functions</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6 input functions</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>5 input functions</td>
<td>115</td>
<td>114</td>
</tr>
<tr>
<td>4 input functions</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>&lt;3 input functions</td>
<td>52</td>
<td>76</td>
</tr>
<tr>
<td>Dedicated logic registers</td>
<td>90</td>
<td>90 / 12,480 ( 1 % )</td>
</tr>
</tbody>
</table>
```

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

TDS system manual, by Daniel Gomez-Prado 14
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`

<table>
<thead>
<tr>
<th>Tds 01&gt; poly F=f+bhase+bsegf=bseg+d+b</th>
<th>Tds 02&gt; setenv gaut cadency=100</th>
<th>Tds 03&gt; reorder --annealing -<code>gl</code></th>
<th>strade=11111110</th>
<th>[ ================== ] 100% Cost=80.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tds 04&gt; setenv gaut cadency=80</td>
<td>Tds 05&gt; reorder --annealing -<code>gl</code></td>
<td>strade=11111110</td>
<td>[ =================== ] 100% Cost=70.00</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=39.90 Window=49 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=33.91 Window=47 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=33.91 Window=47 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=45.39 Window=45 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=28.83 Window=45 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=24.50 Window=43 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=20.83 Window=41 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=6.68 Window=27 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>% T=2.14 Window=13 Cost=70.00</td>
<td>too strong constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>Tds 06&gt; tds2d2fg</td>
<td>Tds 09&gt; balance --d</td>
<td>Tds 10&gt; dfg2ntl</td>
<td>Tds 11&gt; cost --n</td>
<td>Cadency=70</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>GP delays: ADD=1 SUB=1 MPY=2</td>
<td>resources: ADD=4294967295 MPY=4294967295</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tds 12&gt; quartus --p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tds 13&gt; quartus --c</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info: Report file saved in &quot;dfg2ntl .quartus .report&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Info: Loading report file &quot;dfg2ntl .quartus .report&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clock name</td>
<td>target freq</td>
<td>design freq</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>1000.0 MHz</td>
<td>182.58 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td>Synthesis</td>
<td>Fitter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synthetic ALUTs</td>
<td>696</td>
<td>720 / 12,480 (6 %)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 7 input functions</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 6 input functions</td>
<td>67</td>
<td>67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 5 input functions</td>
<td>339</td>
<td>339</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 4 input functions</td>
<td>100</td>
<td>102</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- &lt;=3 input functions</td>
<td>190</td>
<td>212</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dedicated logic registers</td>
<td>135</td>
<td>135 / 12,480 (1 %)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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*

<table>
<thead>
<tr>
<th>list of cost functions</th>
<th>LICF . . . MICF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where LICF and MICF stands for the Least/Most Important Cost Function to optimize</td>
<td></td>
</tr>
<tr>
<td>node</td>
<td>nnode</td>
</tr>
<tr>
<td>Minimizes the number of TED nodes</td>
<td></td>
</tr>
<tr>
<td>latency</td>
<td>nlatency</td>
</tr>
<tr>
<td>Minimizes the critical path computed in the TED graph</td>
<td></td>
</tr>
<tr>
<td>edge</td>
<td>nedge</td>
</tr>
<tr>
<td>Minimizes the total number of TED edges</td>
<td></td>
</tr>
<tr>
<td>edgeN</td>
<td>nedgeN</td>
</tr>
<tr>
<td>Minimizes the number of multiplicative TED edges</td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td>nmUL</td>
</tr>
<tr>
<td>Minimizes the number of multiplications in the DFG</td>
<td></td>
</tr>
<tr>
<td>nADD</td>
<td>nADD</td>
</tr>
<tr>
<td>Minimizes the number of additions (and subtractions) in the DFG</td>
<td></td>
</tr>
<tr>
<td>nMPY</td>
<td>nMPY</td>
</tr>
<tr>
<td>Minimizes the number of multipliers erOfCandidates&quot; [DEFAULT BEHAVIOR]</td>
<td></td>
</tr>
<tr>
<td>dlat</td>
<td>dlatency</td>
</tr>
<tr>
<td>Minimizes the latency in the DFG</td>
<td></td>
</tr>
<tr>
<td>bitwidth</td>
<td>bitwidth</td>
</tr>
<tr>
<td>Minimizes the bitwidth of the HW implementation subject to unlimited latency</td>
<td>resources</td>
</tr>
<tr>
<td>gapp</td>
<td>gapp</td>
</tr>
<tr>
<td>Minimizes the upper tighter bound found trough Gappa</td>
<td></td>
</tr>
<tr>
<td>gMUX</td>
<td>gMUX</td>
</tr>
<tr>
<td>Minimizes the number of muxes in the GAUT implementation</td>
<td></td>
</tr>
<tr>
<td>dlat</td>
<td>dlatency</td>
</tr>
<tr>
<td>Minimizes the latency in the GAUT implementation</td>
<td></td>
</tr>
<tr>
<td>dgr</td>
<td>dgr</td>
</tr>
<tr>
<td>Minimizes the number of registers in the GAUT implementation</td>
<td></td>
</tr>
<tr>
<td>dadd</td>
<td>dadd</td>
</tr>
<tr>
<td>Minimizes the total area of the operators in the GAUT implementation</td>
<td></td>
</tr>
</tbody>
</table>

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.


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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.
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 used 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.
Figure 1.6. DFG corresponding to the unretimed TED.
Suggestion to restore ordering visualization:

\texttt{jumpAbove -p a a}

\texttt{Tds 07> show retimed_ted.dot}

\texttt{Tds 08> ted2dfg -n}

\texttt{Tds 09> show -d retimed_dfg.dot}

\textbf{Figure 1.7.} Retimed TED.
1.4.5 Precision

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.

Figure 1.8. DFG corresponding to the retimed TED.
Listing 1.13. Annotating bitwidth in TED

Tds 01> set -b
NAME
set — Set the variable bitwidth and other options.
SYNOPSIS
set [bitwidth] [range] [maximal error]
OPTIONS
-h, --help
Print this message.
[bitwidth]
-0, --bitwidth=integer[fixedpoint|fp
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(fp)> 4.28 [range]
-0, --range var1:intvar2 [var1:intvar2 ...]
Where interval has the syntax: [minval, maxval]
[ maximal error]
-0, --error pol:maxerror1 [pol:maxerror2 ...]
Where maxerror1 is the maximal error allowed at the primary output pol
EXAMPLE
poly F1 = a−b+c+d
poly F2 = (a+b)∗(c+d)^2
set -b fixedpoint a:4,16 c:2,10
set --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 bitwidths required for exact computation.
SYNOPSIS
compute [-t -b --snn] | [-d -b --g]
OPTIONS
-h, --help
Print this message.
-0, --bitwidth
Compute the bitwidth at each point in the graph [DEFAULT BEHAVIOR].
-g, --gappa
Compute a bound on the maximal error.
--snn
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

TDS system manual, by Daniel Gomez-Prado 21
Figure 1.9. (a) Initial TED. (b) TED with bitwidth annotation in nodes.

Figure 1.10. TED with bitwidth annotation for exact computation.

Tds 06> compute -t -b
Tds 07> show precision_computed_ted.dot
Figure 1.11. DFG with bitwidth annotation for exact computation.


Figure 1.12. DFG with bitwidth optimization for target error.

<table>
<thead>
<tr>
<th>Level</th>
<th>OP</th>
<th>ID</th>
<th>bitwidth</th>
<th>Left ID</th>
<th>Right ID</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>14</td>
<td>Q( 2</td>
<td>NONE</td>
<td>NONE</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>a</td>
<td>11</td>
<td>Q( 4,16</td>
<td>NONE</td>
<td>NONE</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>b</td>
<td>8</td>
<td>Q( 4,12</td>
<td>NONE</td>
<td>NONE</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>c</td>
<td>4</td>
<td>Q( 2,10</td>
<td>NONE</td>
<td>NONE</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>d</td>
<td>3</td>
<td>Q( 4,12</td>
<td>NONE</td>
<td>NONE</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>16</td>
<td>Q( 4,7</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>15</td>
<td>Q( 4,5</td>
<td>4</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>13</td>
<td>Q( 8,11</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
<td>5</td>
<td>Q( 5,0</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>21</td>
<td>Q( 5,3</td>
<td>11</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td>17</td>
<td>Q( 9,11</td>
<td>16</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>10</td>
<td>Q( 6,2</td>
<td>5</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>18</td>
<td>Q( 8,12</td>
<td>15</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td>12</td>
<td>Q( 7,4</td>
<td>11</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>19</td>
<td>Q(10,12</td>
<td>17</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>22</td>
<td>Q(15,10</td>
<td>21</td>
<td>19</td>
<td>0</td>
</tr>
</tbody>
</table>

TDS system manual, by Daniel Gomez-Prado
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).

The node with variable $a$ in Figure 1.13(a) has an edge connected to node ONE with power $2$, 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 $a$.

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 \texttt{decompose}.

\textbf{Listing 1.15.} Annotating bitwidth in TED

The command \texttt{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.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{decomposed_TED.png}
\caption{Decomposed TED.}
\end{figure}
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

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.
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
```

![Diagram](image)

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

The DFG generated for the TED shown in Figure 1.17(b) is shown in Figure ??(a).
Figure 1.17. (a) DFG corresponding to TED in Figure 1.17(b). (b) DFG with replaced shifters. (c) Balanced DFG.