Flow Engineering for Physical Implementation: Theory and Practice

Steve Golson
Trilobyte Systems

Pete Churchill
Magma Design Automation

June 10, 2008
45th Design Automation Conference

Define: flow

flow *n.*

1. the act or manner of flowing
2. the smooth motion characteristic of fluids
3. anything that flows; stream or current
4. a continuous production
5. the sequence in which operations are performed

from http://education.yahoo.com

Define: audience

- design engineer
- EDA vendor
- manager
- CAD

Define: flow

- Management perspective

  flow *n.*

  a four-letter word often associated with work that can't be scheduled and seems to never end

- CAD perspective

  flow *n.*

  a four-letter word representing an endless project whose result is underappreciated and underused by the design community

- Design engineer perspective

  flow *n.*

  a four-letter word representing a necessary tool which you are not given time to work on, even though what you have is not adequate or complete

- EDA vendor perspective

  flow *n.*

  a four-letter word representing all the stuff that customers do with other vendor's tools, often making us look bad

Flow (our definition)

flow *n.*

- all the files and infrastructure required to implement a methodology

  "The CAD team has created a new flow for us to use on our next design."

- a sequence of tool invocations

  "You need to run the flow for clock synthesis."

Define: physical implementation

- "IC design" or "chip design" or "IC implementation"
- Everything from RTL ➔ GDSII
- Everything *other than* functionality
- *Not* writing RTL, *not* functional verification
- Hold function constant, change the representation

Define: engineering

engineering *n.*

- The application of scientific and mathematical *principles* to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems.

from http://education.yahoo.com

The Principles of Flow Engineering

#1 Flows are hard

#2 Flows are iterative

#3 Flows are automated

#4 Flows fail more than they succeed

Principle #1: Flows are hard

Flows are expensive

- We spend as much on flow
  as we do on EDA tools themselves

> "The IDMs and large fabless companies are spending between $1 and $2 on CAD support activities for every dollar they spend on the tools themselves. In fact, there are nearly as many CAD engineers inside the semiconductor industry as there are in all of the EDA vendors… combined!"

Thomas Harms
Infineon Technologies

SCDsource
12 Dec 2007

Principle #1: Flows are hard

We've been designing chips for a long time

Why hasn't this been solved already?

- EDA perspective:
  - There's no money in it
  - Vendors can only work with their own tools
- CAD perspective:
  - Too many tools to support them all
  - Too many process nodes/libraries to support them all
  - Customization capabilities are difficult to include

Principle #1: Flows are hard

"We build chips, not flows"

- This is only true if you *reuse* a flow
- Flows are only reusable when they are *designed* to be reused
- Making changes and improvements to your flow for reuse will *not* make your current chip project finish sooner…

  however your *next* chip will finish sooner…
  - but only if you actually reuse your flow and don't start over from scratch!

Principle #1: Flows are hard

Flow must change/adapt as the design matures

- Prototype flows needed in the beginning
- Production flows needed once implementation matures
- Hopefully we are building the process while the RTL is still being written

Principle #1: Flows are hard

- EDA tools are complex and expensive
- Combining them into a flow is *also* complex and expensive
- Always lots of time pressure
  - it seems there is never any time to develop what we need
  - instead, we have to make do

Quote Quiz

"Can't we use the flow from the last project?"

1. Management [X]
2. CAD [ ]
3. EDA [ ]
4. Design [ ]

"Nope, we just hacked something together to get the chip out."

1. Management [ ]
2. CAD [ ]
3. EDA [ ]
4. Design [X]

"You should just use our recommended flow."

1. Management [ ]
2. CAD [ ]
3. EDA [X]
4. Design [ ]

Principle #2: Iteration

Iteration is an overloaded term

We have two competing definitions:

iteration *n.*

- hand-fixing a single netlist or placement or routed block in order to meet your design goals (e.g., timing, DRC)

  These are *micro iterations*

  Goal: single iteration convergence using automated flow

iteration *n.*

- running your automated flow when changes occur to RTL, floorplan, constraints, etc.

  These are *macro iterations*

  Reality: many many runs

Principle #2: Iteration

Physical implementation…

- is *not* like building a bridge

  because you only build a bridge once

Physical implementation…

- is like building a *machine* that builds a bridge

Imagine building a new bridge *every day!*

Our goal is to build a *flow* that builds a chip (RTL→GDSII)

Principle #2: Iteration

- Iterate early, iterate often
- Iterations will overlap
- How fast can you iterate?
- Each iteration must be
  - repeatable
  - reliable
  - robust

How can we accomplish this?

Quote Quiz

"The RTL freeze was three weeks ago."

1. Management [X]
2. CAD [ ]
3. EDA [ ]
4. Design [ ]

"What about the five changes since then?"

1. Management [ ]
2. CAD [ ]
3. EDA [ ]
4. Design [X]

"Those weren't releases, they were ECOs."

1. Management [X]
2. CAD [ ]
3. EDA [ ]
4. Design [ ]

Principle #3: Automation

Automation means your computer does the work

- Scripted, not GUI
  - No GUI implies no hand edits
- Batch, not interactive
- Self-checking, not user judgment
- Repeatable, reliable, robust
- Predictable results

Can I sleep while this is running?

Principle #3: Automation

"Success in physical design
will come from relentless automation."

Paul Rodman
DAC 2002

Quote Quiz

"We've got scripts
to automate your ECO flow.
The documentation is
on our website."

1. Management ☐
2. CAD ☒
3. EDA ☐
4. Design ☐

"146 pages of documentation.
How long will it take us
to learn this?"

1. Management ☒
2. CAD ☐
3. EDA ☐
4. Design ☐

"I don't need an ECO flow.
Get me a six-pack of Red Bull
and I'll have it ready
in the morning."

1. Management ☐
2. CAD ☐
3. EDA ☐
4. Design ☒

Principle #4: Failure

Failure is an overloaded term

We have two competing definitions:

failure *n.*

- when a particular flow (sequence of tool invocations) does not generate tapeout-quality results

  Reality: Most iterations *fail* for one reason or another

  Reality: Many many more failures than tapeouts

failure *n.*

- when a particular flow (all the scripts and infrastructure) is painfully awkward and difficult to use

  Reality: Many flows *fail* although we rarely admit it

Principle #4: Failure (of scripts and infrastructure)

Fallacy: Flows never fail

- You always get the chip out the door, so by definition your flow "works"
- The only flows that fail are the ones that are developed *without* a chip

Who develops flows without a chip project?

Principle #4: Failure (of scripts and infrastructure)

Dangerous fallacies:

- Our flow is a success
- The CAD department's flow is a failure
- The EDA vendor's flow is a failure

Outrageous truths:

- Failure of your flow infrastructure is *bad* failure
- Acknowledge what is broken
- Consider what else might work

Principle #4: Failure (of sequence of tool invocations)

- This is *good* failure
- Failure is inherent in the process
- Failure is your friend

Principle #4: Failure

Success

- How to define success?
  - Must have something to measure
  - Zero errors (DRC, timing, etc.)
  - Success metrics will change as the design matures
- We should not expect *first* pass success
  - rather, we want *last* pass success
  - eventually, *every* pass success

Quote Quiz

"Look at the cool wrapper we built around the tool. Now it's much easier for you to use."

1. Management
2. CAD **X**
3. EDA
4. Design

"I got a segmentation fault. What do I do now?"

1. Management
2. CAD
3. EDA
4. Design **X**

"Can you provide a test case? Without your wrapper?"

1. Management
2. CAD
3. EDA **X**
4. Design

Principle #4: Failure

- All tools suck

Goals

Building from the 4 principles:

- What are the *goals* for our flow?
- How should it look and work?
- Goals are used to drive decisions during flow development and ongoing maintenance

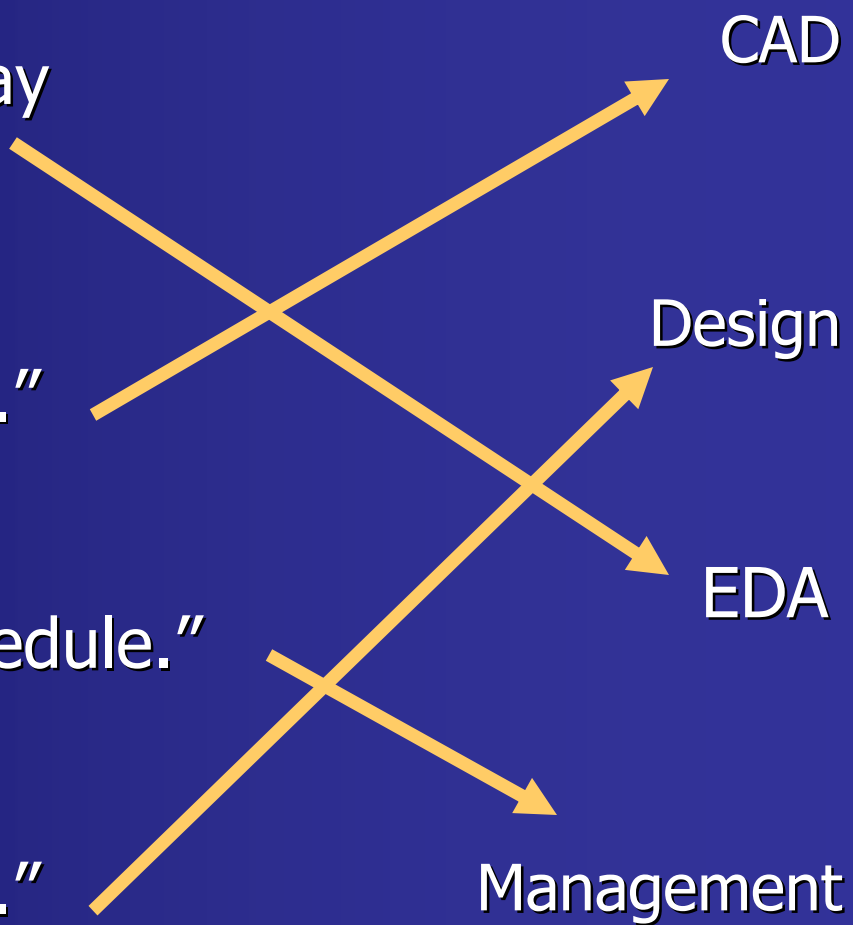
Quote Quiz

"Our goal
is to show you the best way
to use our tools."

"Our goal
is to build flows, not chips."

"Our goal
is to meet the original schedule."

"Our goal
is to build chips, not flows."



The Goals of Flow Development

#1 Design independent flow

#2 Tool independent flow

#3 Technology independent flow

#4 User independent flow

Principle: iteration
Goal: user independence

- implies isolated sandbox (aka workspace)
- all inputs, libs, etc.
- no external sources
- no common files (except: EDA tools, OS)
- no rug to get pulled out
- isolation for each iteration (multiple sandboxes)

Iteration

- can you exactly repeat what you did before?
  - if you can't repeat it, then you are not iterating, you are flailing
- if your result is a failure, you want to make sure you don't do it that way again
- if your result is a success you want to repeat it
- what files went into your successful run?
  this requires…

Configuration management

- Configuration management means which…
  - RTL?
  - scripts?
  - libraries?
  - EDA tool versions?
- …means any user should be able to repeat
  - same user rerun
  - different user rerun
  - rerun next year
  - EDA vendor rerun (e.g., test case)
- …implies revision control

Revision control

- all sorts of these have been used for IC design
  - copy directory structure
  - SCCS
  - RCS
  - CVS
  - Subversion
  - Synchronicity, Clearcase, AccuRev, ICManage, ClioSoft, etc.

Revision control

- What files need to be under revision control?
- Some that you might not immediately consider:
  - libraries
  - derived files: netlist, layout
  - anything that the next step in the flow might need
  - anything that you might modify for an ECO

Revision control needs to handle:

- large binary files
- directories as primary objects
- symbolic links as primary objects
- CVS doesn't cut it…

Revision Tags

- Configuration management uses *revision tags* or equivalent
- Must uniquely identify each iteration
- Bad names:
  - Latest
  - Bronze, silver, gold
- Good names:
  - Anything that will never run out
  - Integers
  - Dates
    - but be sure and include the year
    - and allow for multiple names per day

Iteration + user independence

Consider the user's *shell environment*:

- It should *not* affect your flow,
  because it isn't under revision control!
- Tool version selection should *not* be determined by your shell search path

Automation + tool independence

- The flow should work with different tools and any number of copies of the tools you have
- There is one precious resource that you *must* manage in an automated fashion…

Licenses!

License management

- Job should *never* fail due to lack of license
- Implies job queueing system (LSF, SGE)
  - even if you have only one license!
  - even if you have only one execution host!
  - even if you have only one user!
- Your job control system *must* be license-aware
- Each command within a tool
  may require a different set of licenses
- Far easier to be efficient with licenses
  if the tool invocations are *short* and *focused*

Define: target

target *n.*

An invocation of a tool

modular target *n.*

An invocation of a tool
to perform a single focused task

Why modular targets are a good idea

- simple to specify/understand/write/reuse
- more efficient license use (check in/out)
- more efficient host use
  - different steps have different resource requirements (e.g., memory, threading)
- enables parallelism: execute targets in parallel
- faster results (critical target first)
- ease of modification
- self-checking targets can detect a problem earlier in the flow
- noncritical target can fail and rest can complete

Example modular targets

- dc_read_sdc
  1. read ddc database
  2. read SDC
  3. write ddc database
- dc_compile
  1. read ddc database
  2. compile
  3. write ddc database
- dc_write_verilog
  1. read ddc database
  2. write out netlist
- pt_analysis_using_sdc
  1. read netlist
  2. read parasitics (optional)
  3. read SDC
  4. update timing
  5. save session
- talus_fix_time
  1. read constrained volcano
  2. fix time
  3. write optimized volcano
- talus_fix_cell
  1. read floorplan volcano
  2. fix cell
  3. write placed volcano

Sequence management

- GNU make
  - Understands and manages dependencies
  - Supports parallel execution
  - Free, and available on all the tool platforms

flowfile: a make-based target sequence

```
dc_load_design_i1:
    @$(start-target)
    @$(MAKE) --no-print-directory \
        MAKEFILES=../../flow/design_compiler/design_compiler_master.mk \
        TARGET_DIR=$(TARGET_DIR) \
        TARGET_INSTANCE=$(TARGET_INSTANCE) \
        DC_TOPO="1" \
        dc_load_design
    @$(end-target)

dc_load_constraints_i1:  dc_load_design_i1
    @$(start-target)
    @$(MAKE) --no-print-directory \
        MAKEFILES=../../flow/design_compiler/design_compiler_master.mk \
        TARGET_DIR=$(TARGET_DIR) \
        TARGET_INSTANCE=$(TARGET_INSTANCE) \
        input_ddc="$(DERIVED_DIR)/dc_load_design_i1/$(BLOCK_NAME).ddc" \
        constraints_file="flowfiles/cpu_constraints.tcl" \
        DC_TOPO="0" \
        dc_load_constraints
    @$(end-target)

dc_compile_i1:  dc_load_constraints_i1
    @$(start-target)
    @$(MAKE) --no-print-directory \
        MAKEFILES=../../flow/design_compiler/design_compiler_master.mk \
        TARGET_DIR=$(TARGET_DIR) \
        TARGET_INSTANCE=$(TARGET_INSTANCE) \
        input_ddc="$(DERIVED_DIR)/dc_load_constraints_i1/$(BLOCK_NAME).ddc" \
        DC_TOPO="0" \
        dc_compile
    @$(end-target)

dc_syn_reports_i2:  dc_compile_i1
    @$(start-target)
    @$(MAKE) --no-print-directory \
        MAKEFILES=../../flow/design_compiler/design_compiler_master.mk \
        TARGET_DIR=$(TARGET_DIR) \
        TARGET_INSTANCE=$(TARGET_INSTANCE) \
        input_ddc="" \
        DC_TOPO="0" \
        dc_syn_reports
    @$(end-target)
```

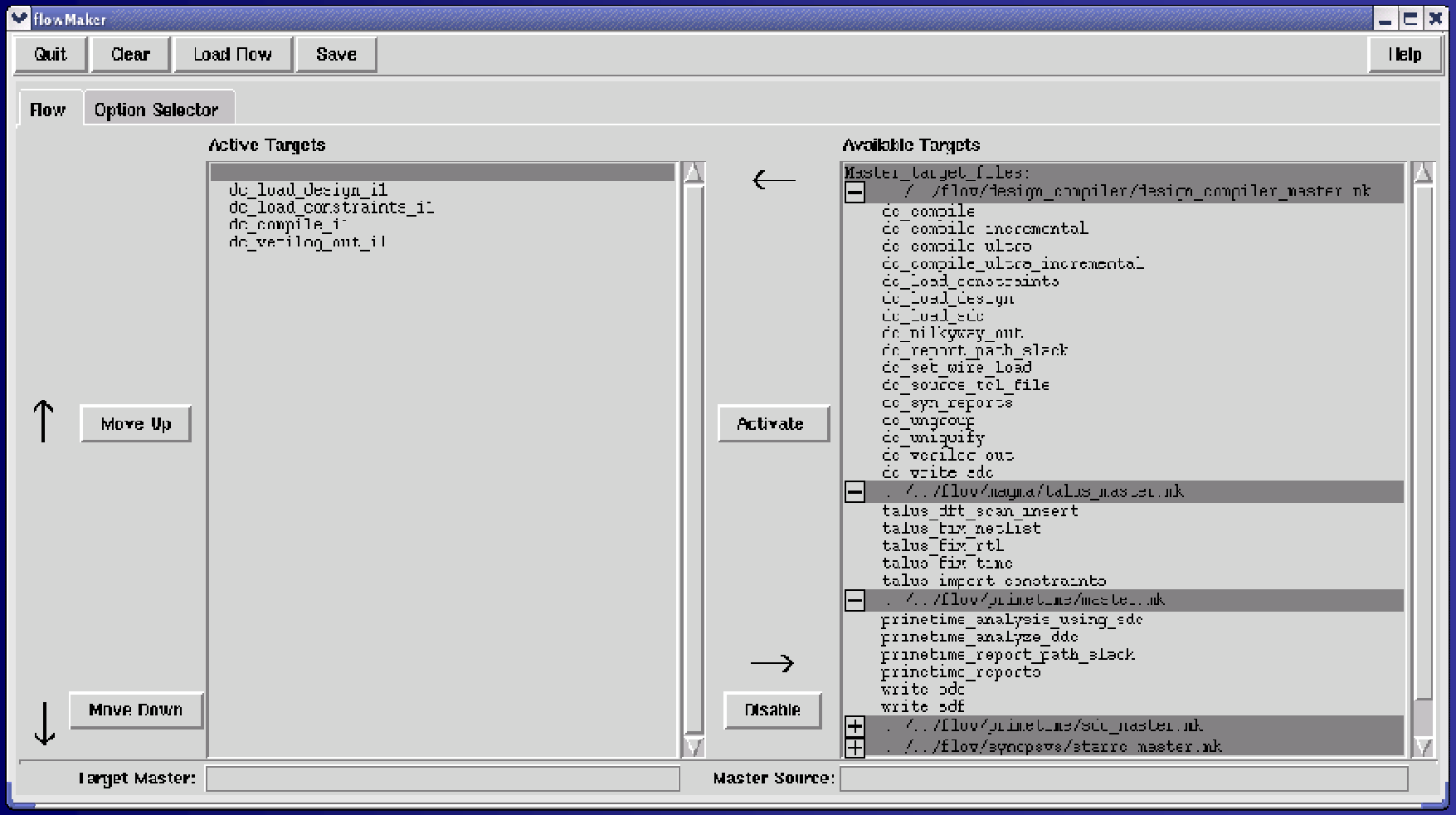
```
dc_compile_incremental_i1:  dc_syn_reports_i2
    @$(start-target)
    @$(MAKE) --no-print-directory \
        MAKEFILES=../../flow/design_compiler/design_compiler_master.mk \
        TARGET_DIR=$(TARGET_DIR) \
        TARGET_INSTANCE=$(TARGET_INSTANCE) \
        input_ddc="$(DERIVED_DIR)/dc_compile_i1/$(BLOCK_NAME).ddc" \
        DC_TOPO="0" \
        dc_compile_incremental
    @$(end-target)

dc_verilog_out_i1:  dc_compile_incremental_i1
    @$(start-target)
    @$(MAKE) --no-print-directory \
        MAKEFILES=../../flow/design_compiler/design_compiler_master.mk \
        TARGET_DIR=$(TARGET_DIR) \
        TARGET_INSTANCE=$(TARGET_INSTANCE) \
        input_ddc="$(DERIVED_DIR)/dc_compile_incremental_i1/$(BLOCK_NAME).ddc" \
        TOOL_VERSION="B-2008.06-SP2" \
        DC_TOPO="0" \
        dc_verilog_out
    @$(end-target)

post_compile_reports:  dc_compile_incremental_i1
    @$(start-target)
    @$(MAKE) --no-print-directory \
        MAKEFILES=../../flow/design_compiler/design_compiler_master.mk \
        TARGET_DIR=$(TARGET_DIR) \
        TARGET_INSTANCE=$(TARGET_INSTANCE) \
        input_ddc="$(DERIVED_DIR)/dc_compile_incremental_i1/$(BLOCK_NAME).ddc" \
        DC_TOPO="0" \
        dc_syn_reports
    @$(end-target)

complete: parallel_2
```

How would you create and maintain these sequences?

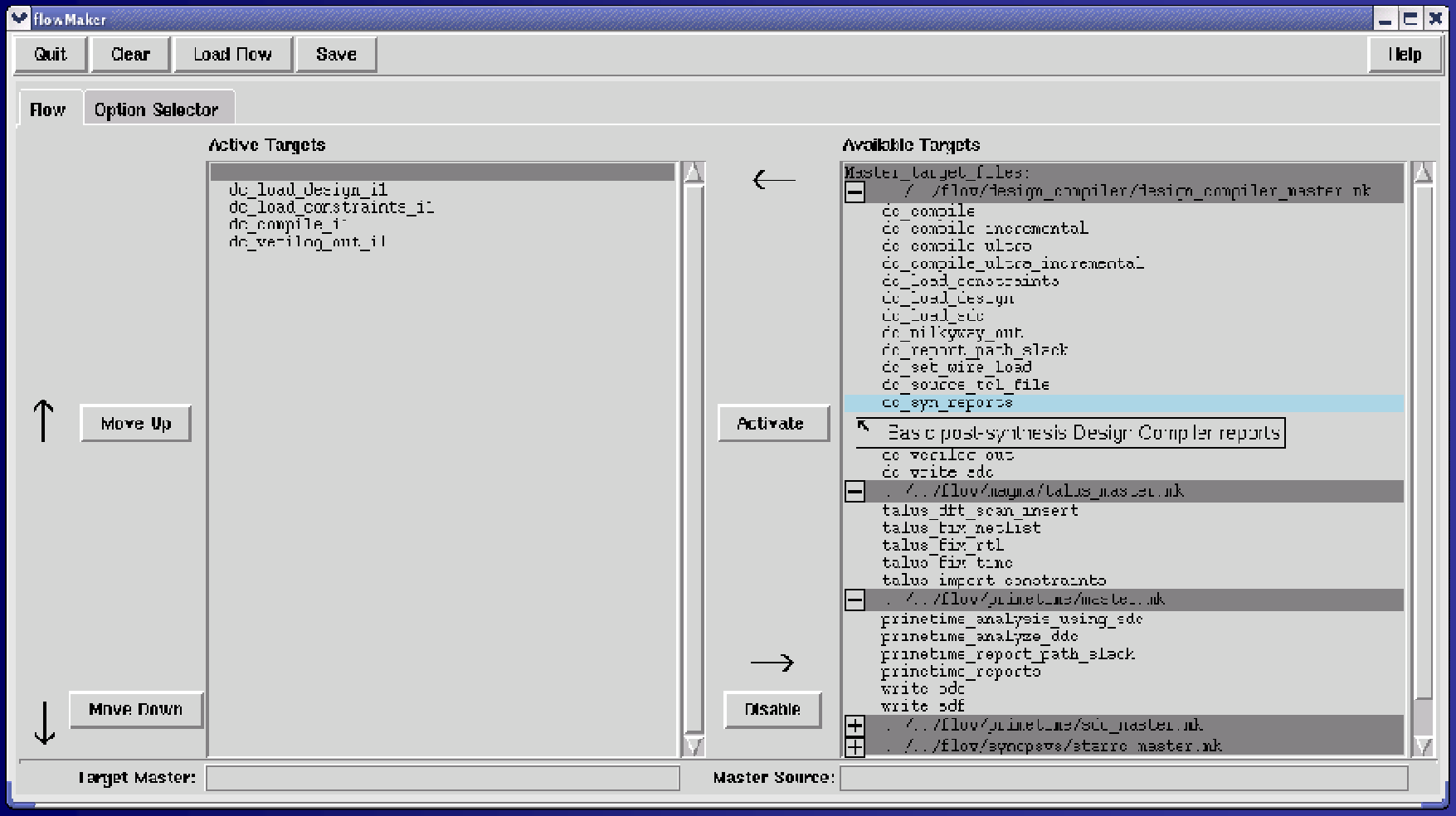
- A *flowfile* is a makefile that describes your flow (a sequence of tool invocations, i.e., targets)

- flowMaker
  - Tcl/Tk script written to build and maintain *flowfiles*
  - Only thing we couldn't get off the shelf
  - Runs on all tool platforms (e.g., RHEL)
  - Download a copy at www.flowMaker.info

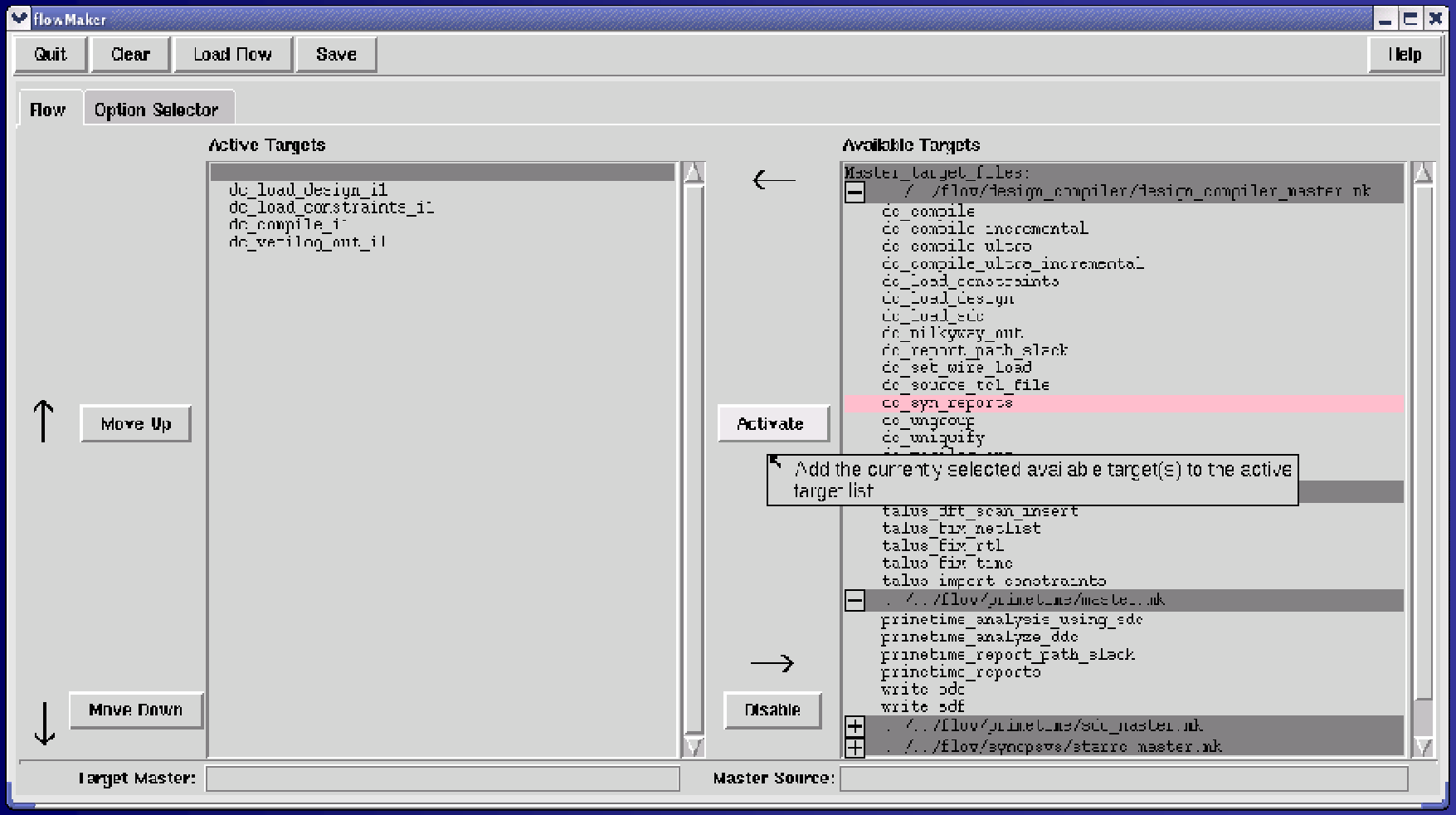
flowMaker window



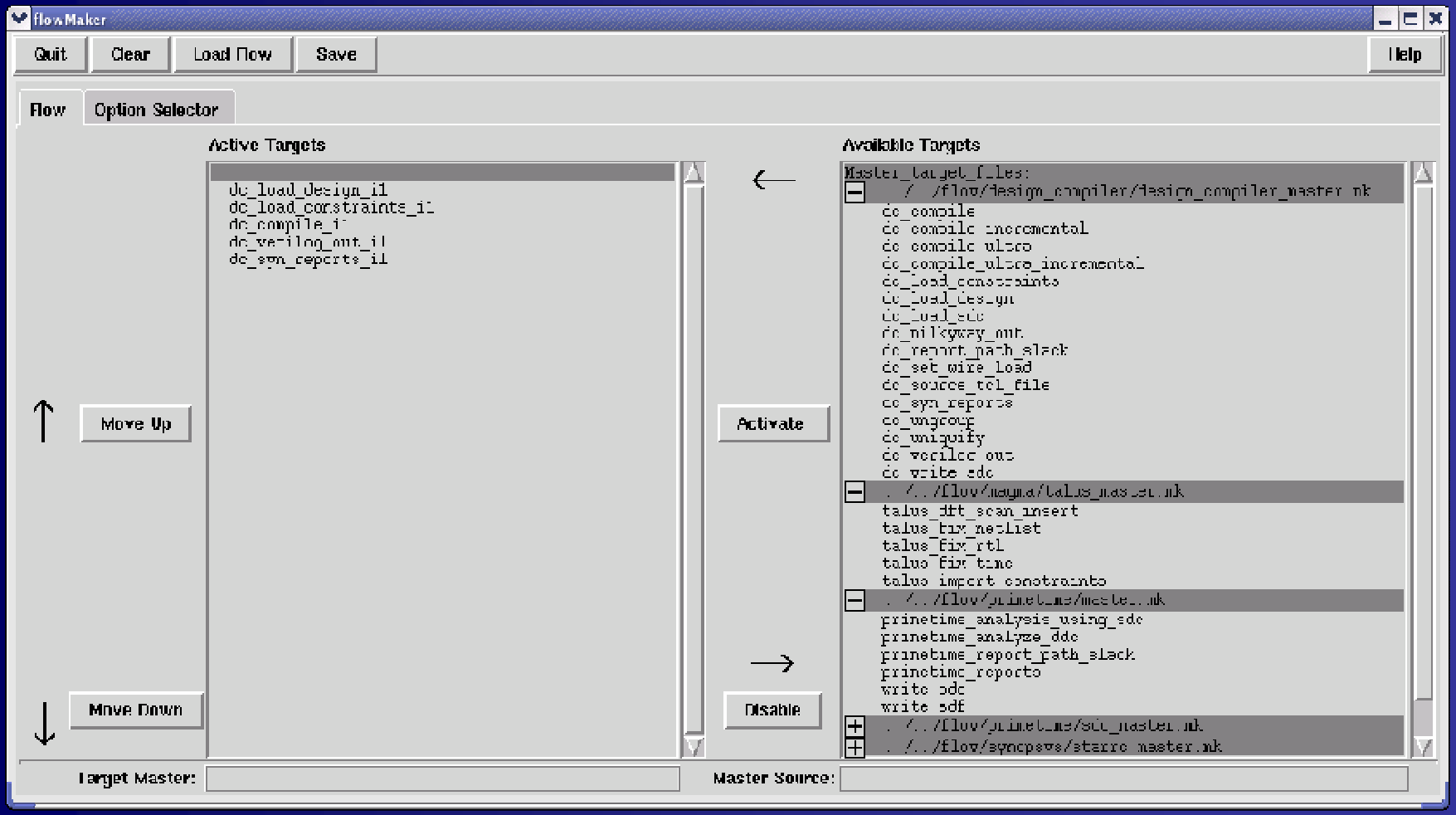
Select a master target



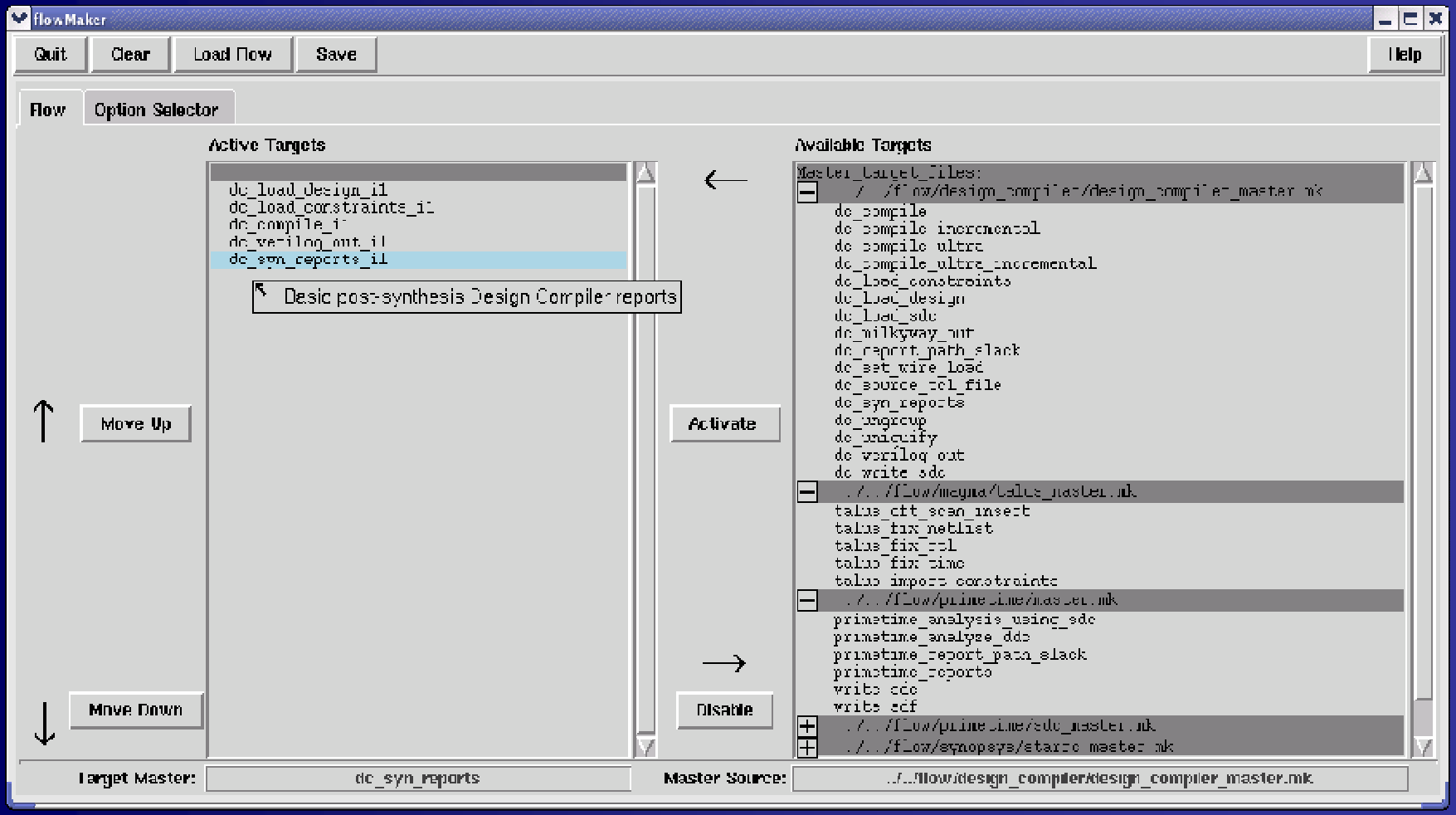
Activate the master target



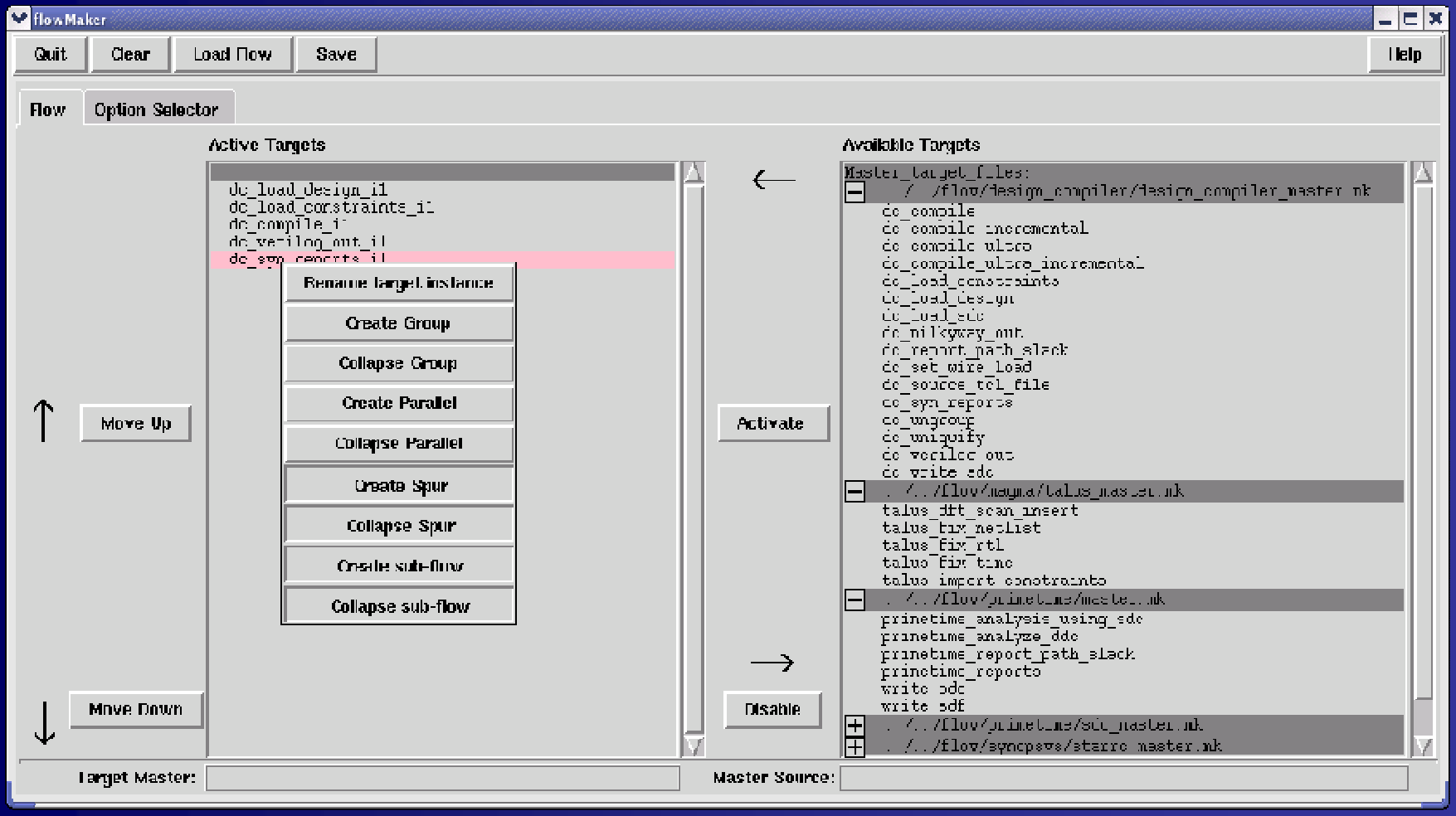
Added new target instance



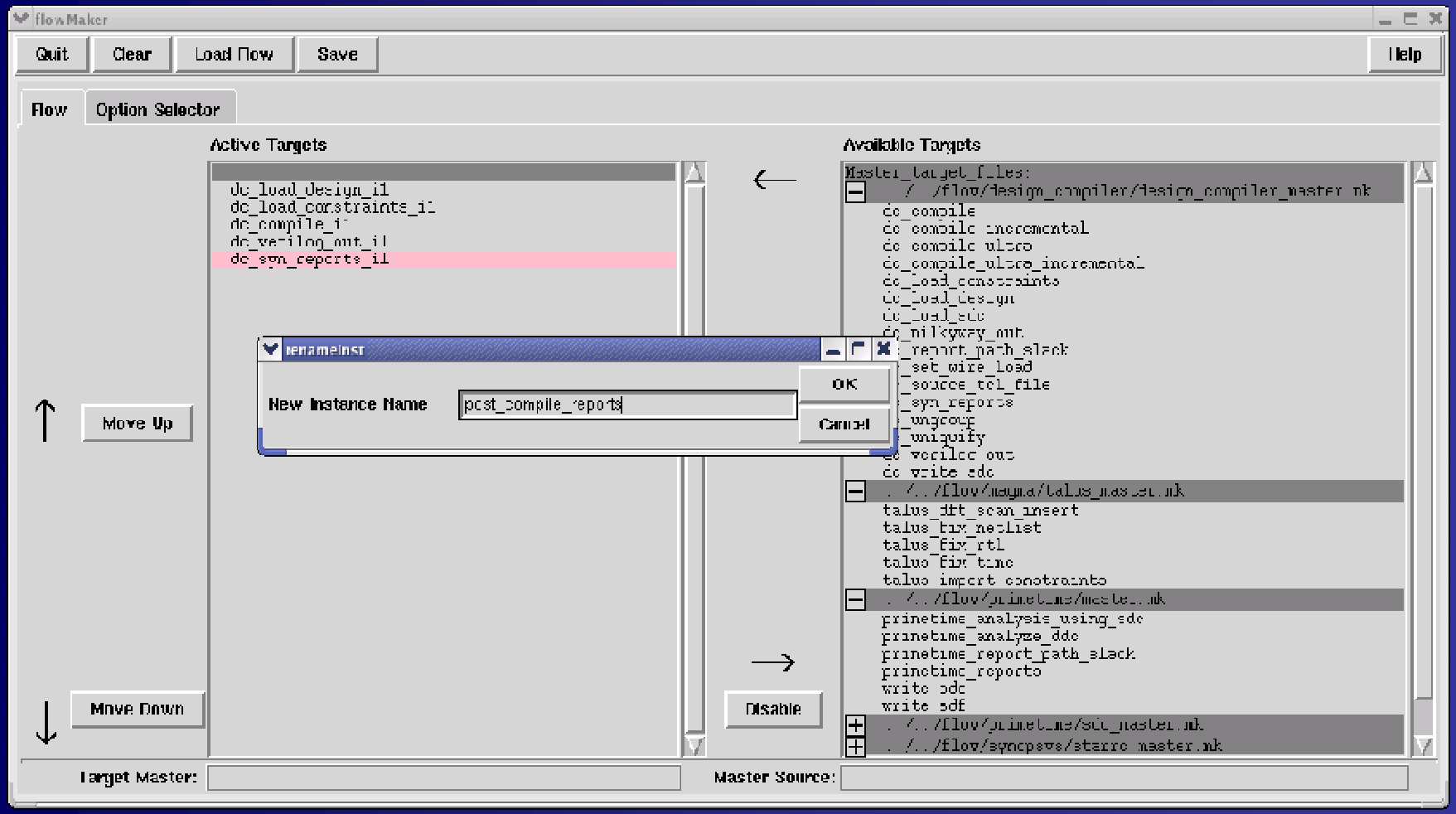
New target instance



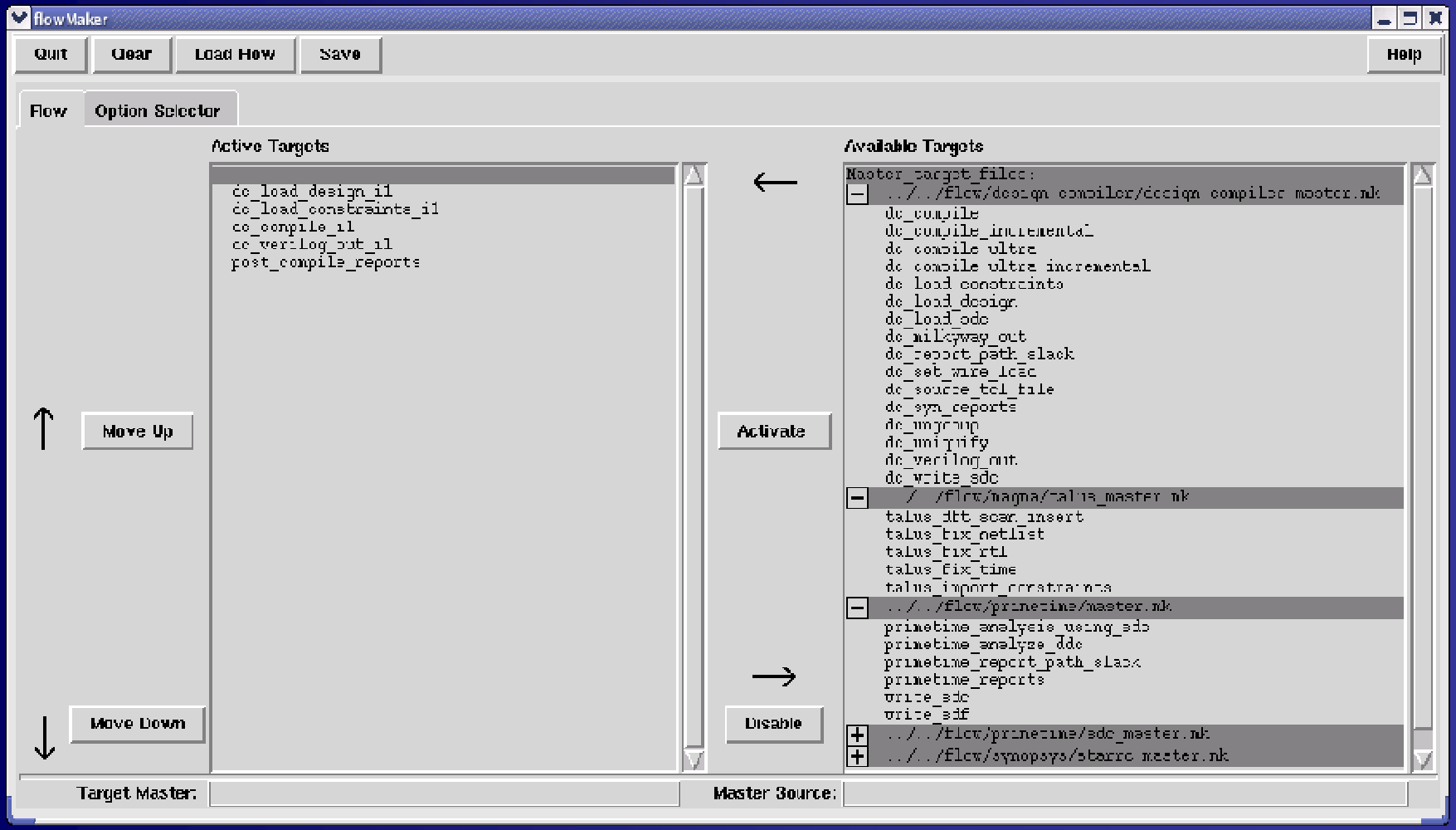
Target menu


flowMaker
Quit
Clear
Load flow
Save
Help
Flow
Option Selector
Active Targets
Available Targets
dc_load_design_il
dc_load_constraints_il
dc_compile_il
dc_verilog_out_il
dc_syn_reports_il
Rename target instance
Create Group
Collapse Group
Create Parallel
Collapse Parallel
Create Spur
Collapse Spur
Create sub-flow
Collapse sub-flow
Move Up
Move Down
Activate
Disable
Master target files:
/flow/design_compiler/design_compiler_master.mk
dc_compile
dc_compile_incremental
dc_compile_ultra
dc_compile_ultra_incremental
dc_load_constraints
dc_load_design
dc_load_edc
dc_milkyway_out
dc_report_path_slack
dc_set_wire_load
dc_source_tcl_file
dc_syn_reports
dc_ungroup
dc_uniquify
dc_verilog_out
dc_write_sdc
/flow/magma/talus_master.mk
talus_dft_scan_insert
talus_fix_netlist
talus_fix_rtl
talus_fix_time
talus_import_constraints
/flow/primetime/master.mk
primetime_analysis_using_sdc
primetime_analyze_sdc
primetime_report_path_slack
primetime_reports
write_sdc
write_sdf
/flow/primetime/sdc_master.mk
/flow/synopsys/starrc_master.mk
Target Master:
Master Source:

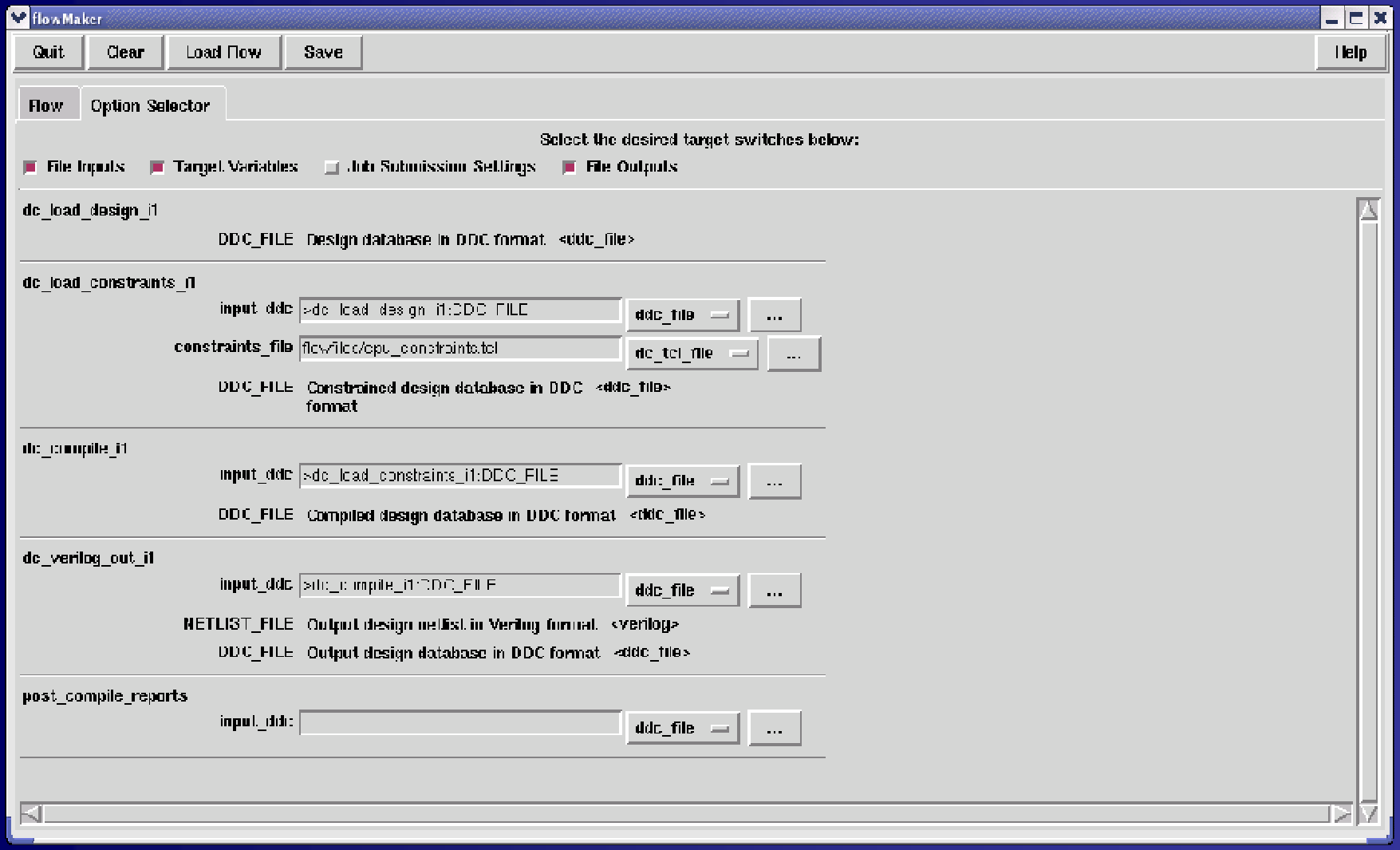
Rename the target instance



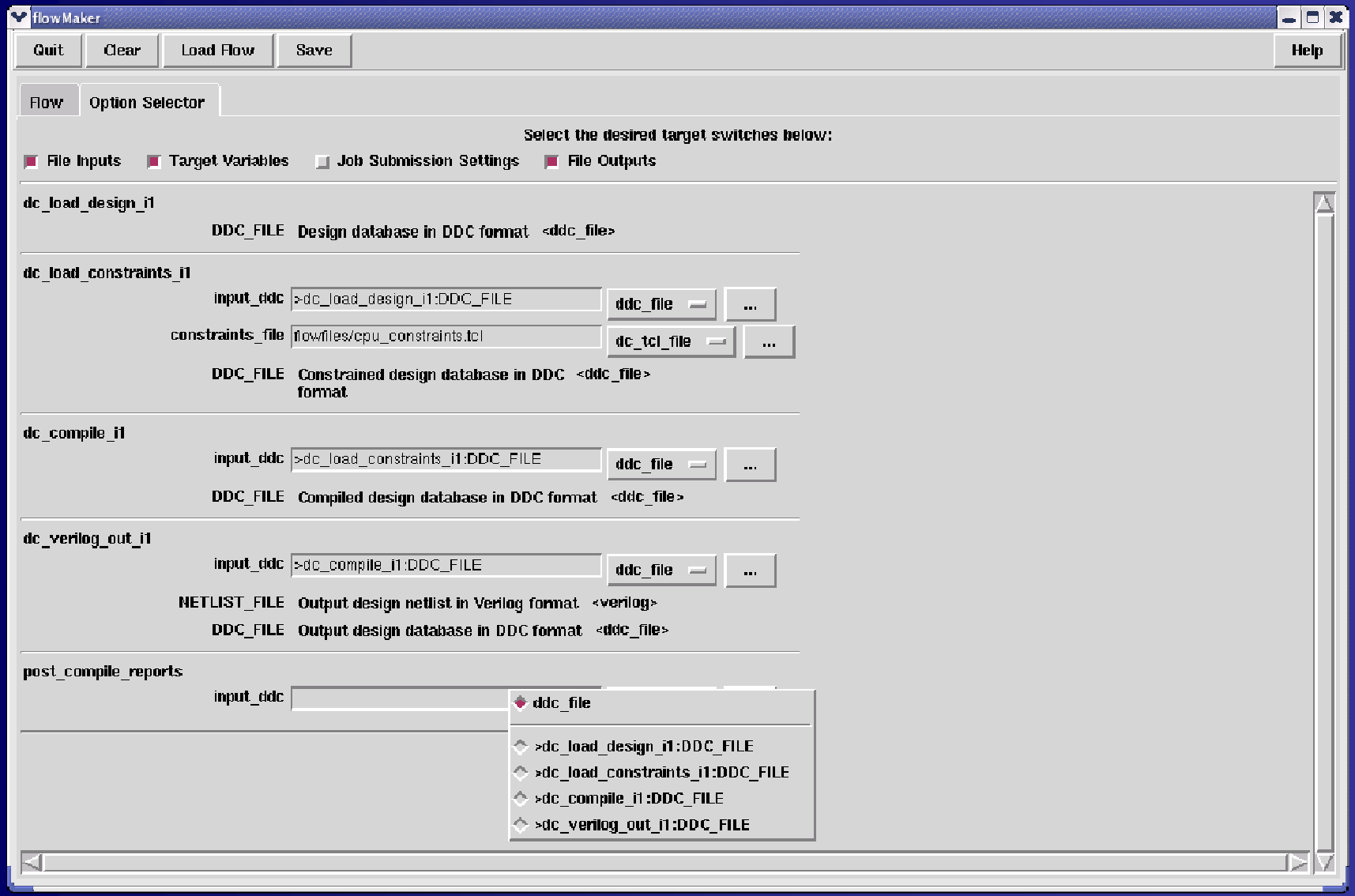
Target instance renamed



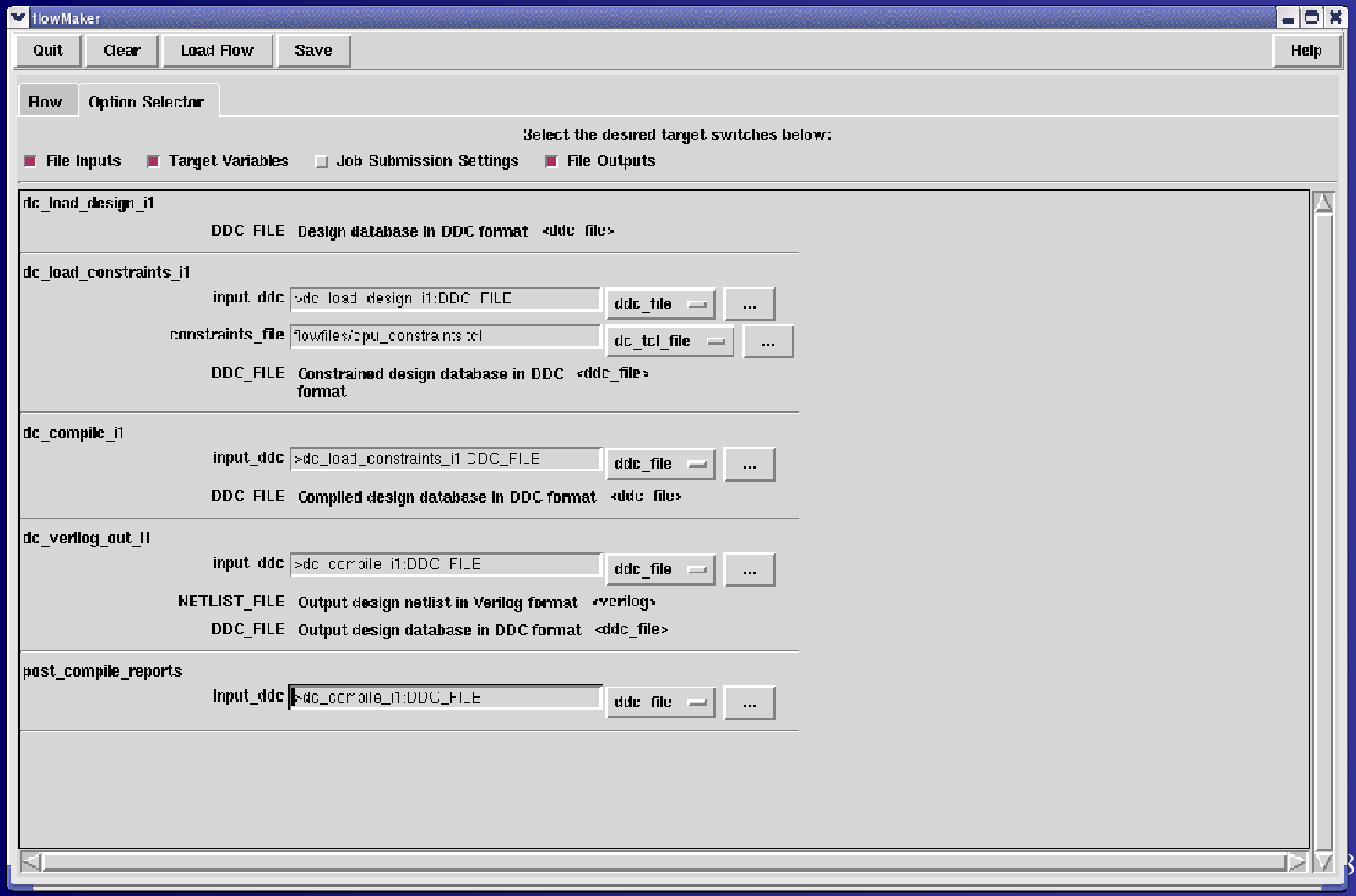
Option Selector



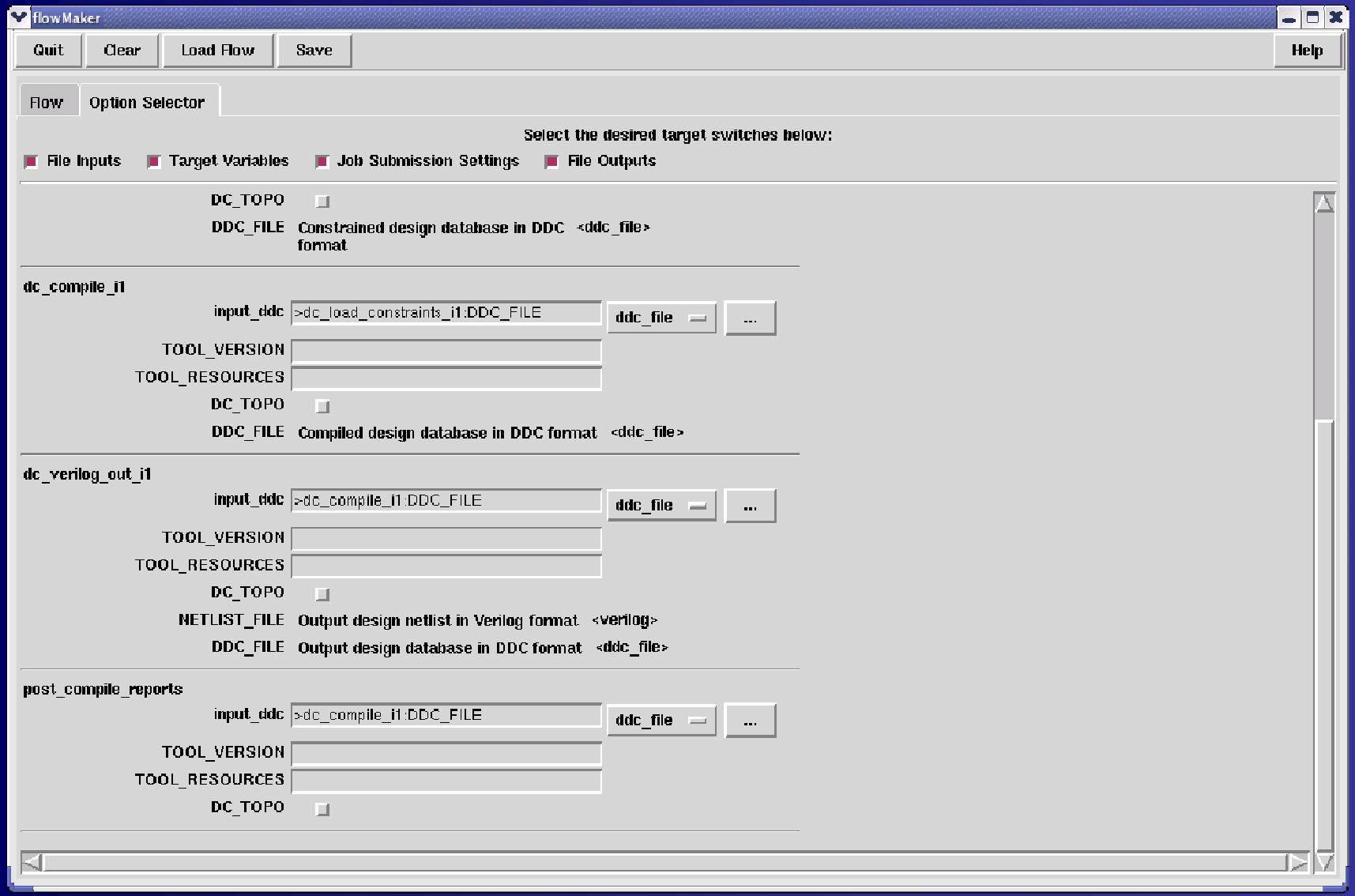
Derived file forwarding



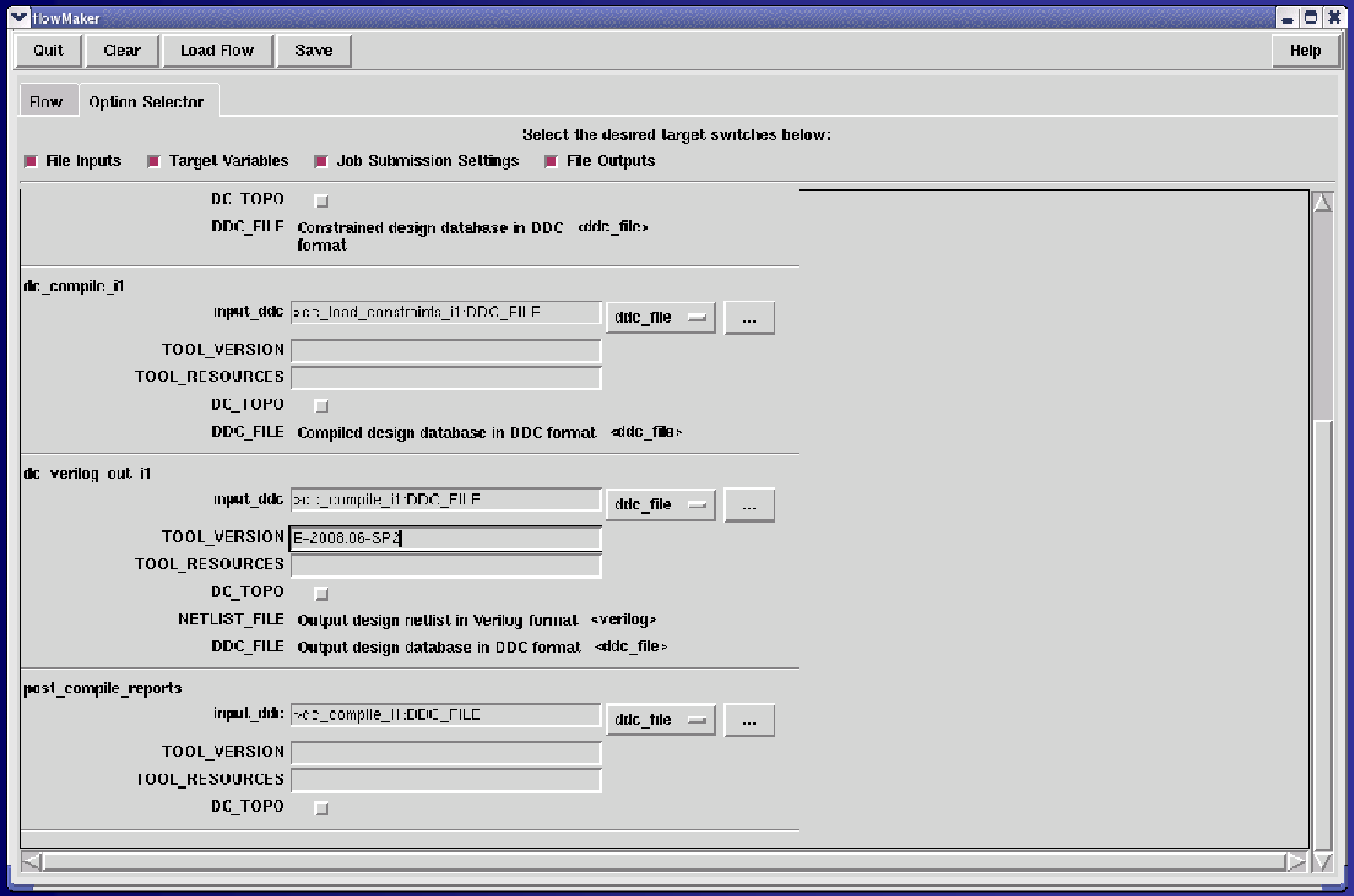
Derived file selected



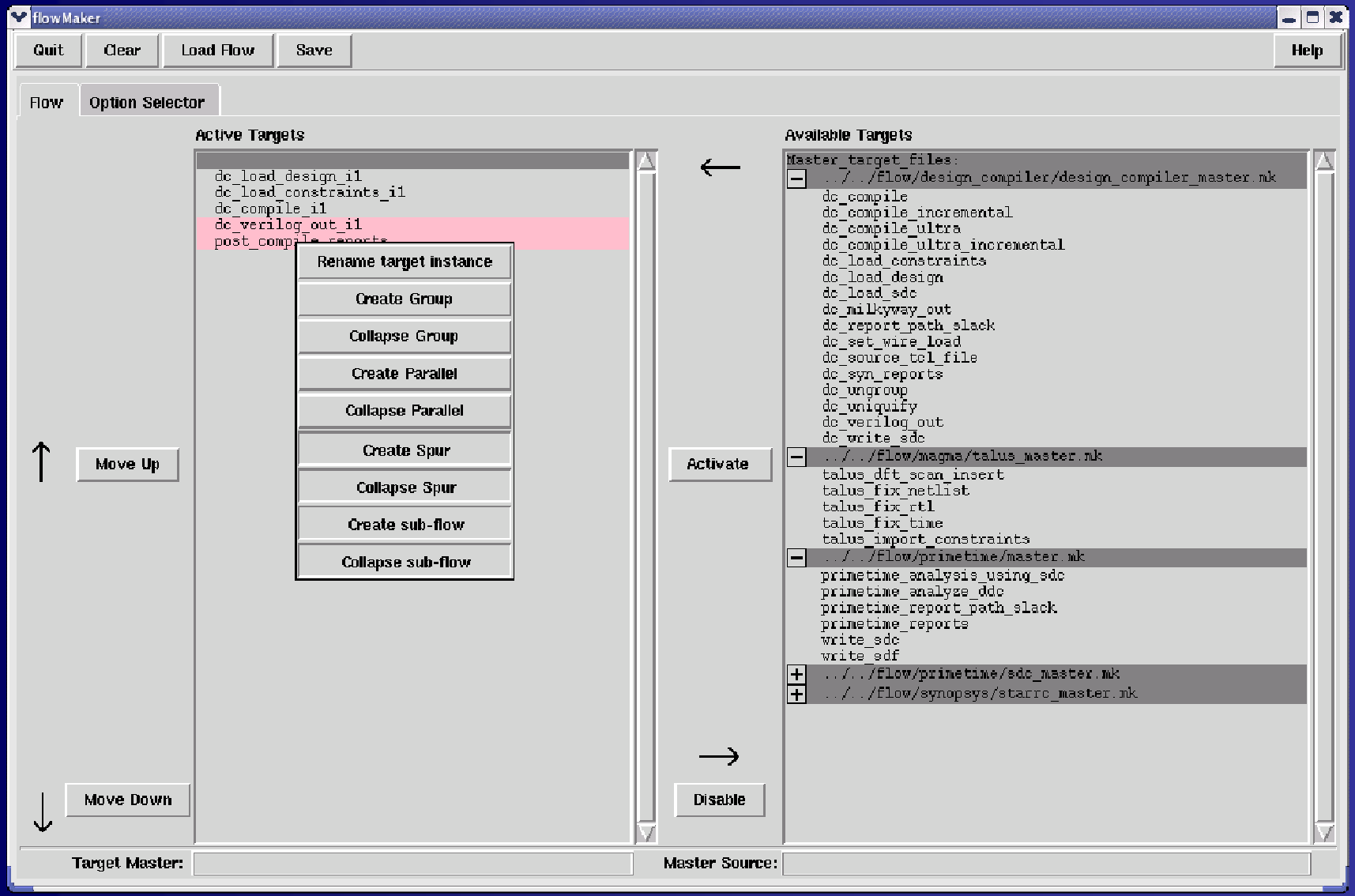
Job Submission Settings



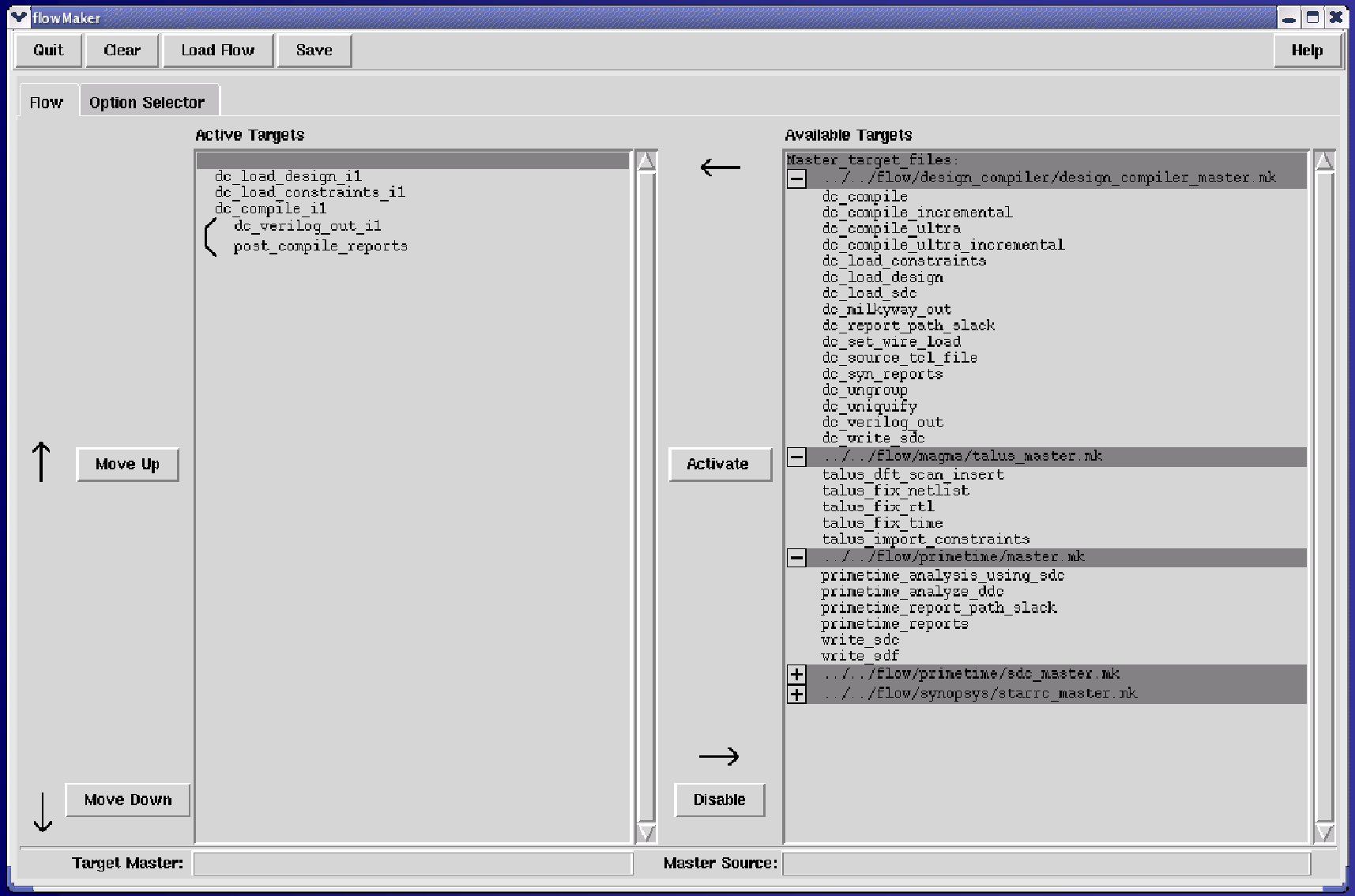
Change tool version for a target



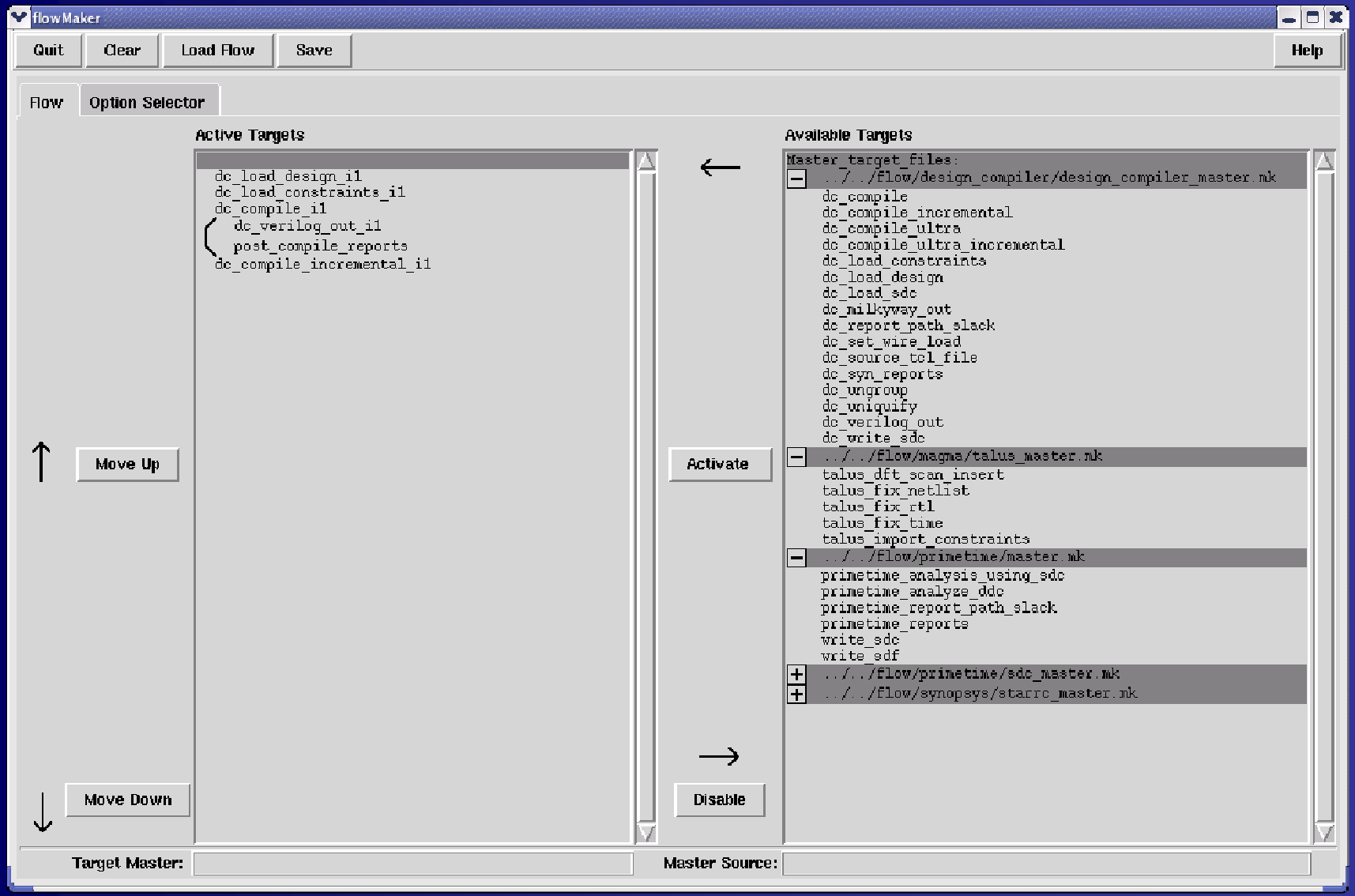
Target menu


FlowMaker
Quit
Clear
Load Flow
Save
Help
Flow
Option Selector
Active Targets
dc_load_design_i1
dc_load_constraints_i1
dc_compile_i1
dc_verilog_out_i1
post_compile_reports
Rename target instance
Create Group
Collapse Group
Create Parallel
Collapse Parallel
Create Spur
Collapse Spur
Create sub-flow
Collapse sub-flow
Move Up
Move Down
Activate
Disable
Available Targets
Master_target_files:
../../flow/design_compiler/design_compiler_master.mk
dc_compile
dc_compile_incremental
dc_compile_ultra
dc_compile_ultra_incremental
dc_load_constraints
dc_load_design
dc_load_sdc
dc_milkyway_out
dc_report_path_slack
dc_set_wire_load
dc_source_tcl_file
dc_syn_reports
dc_ungroup
dc_uniquify
dc_verilog_out
dc_write_sdc
../../flow/magma/talus_master.mk
talus_dft_scan_insert
talus_fix_netlist
talus_fix_rtl
talus_fix_time
talus_import_constraints
../../flow/primetime/master.mk
primetime_analysis_using_sdc
primetime_analyze_ddc
primetime_report_path_slack
primetime_reports
write_sdc
write_sdf
../../flow/primetime/sdc_master.mk
../../flow/synopsys/starrc_master.mk
Target Master:
Master Source:

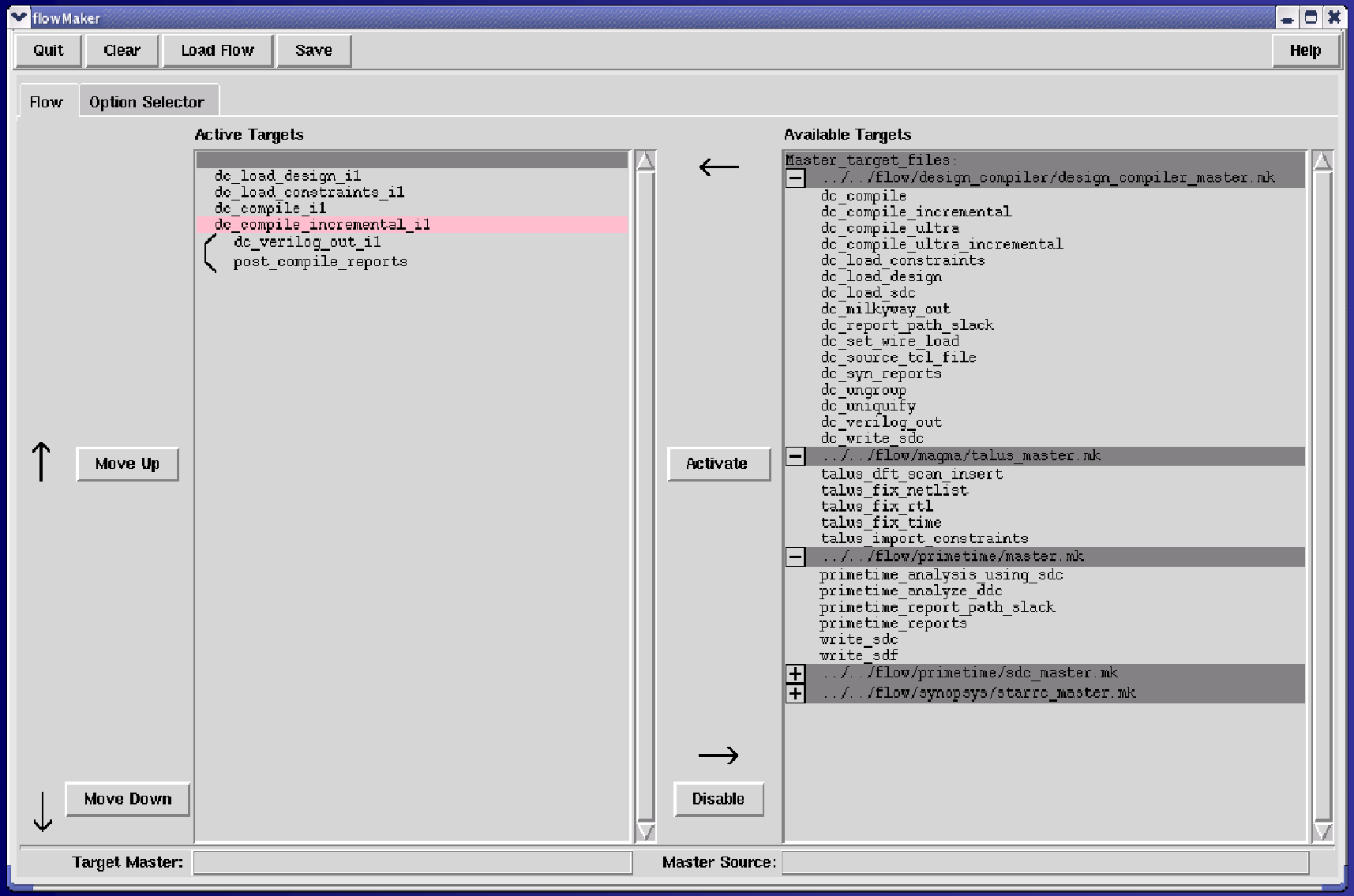
Parallel targets



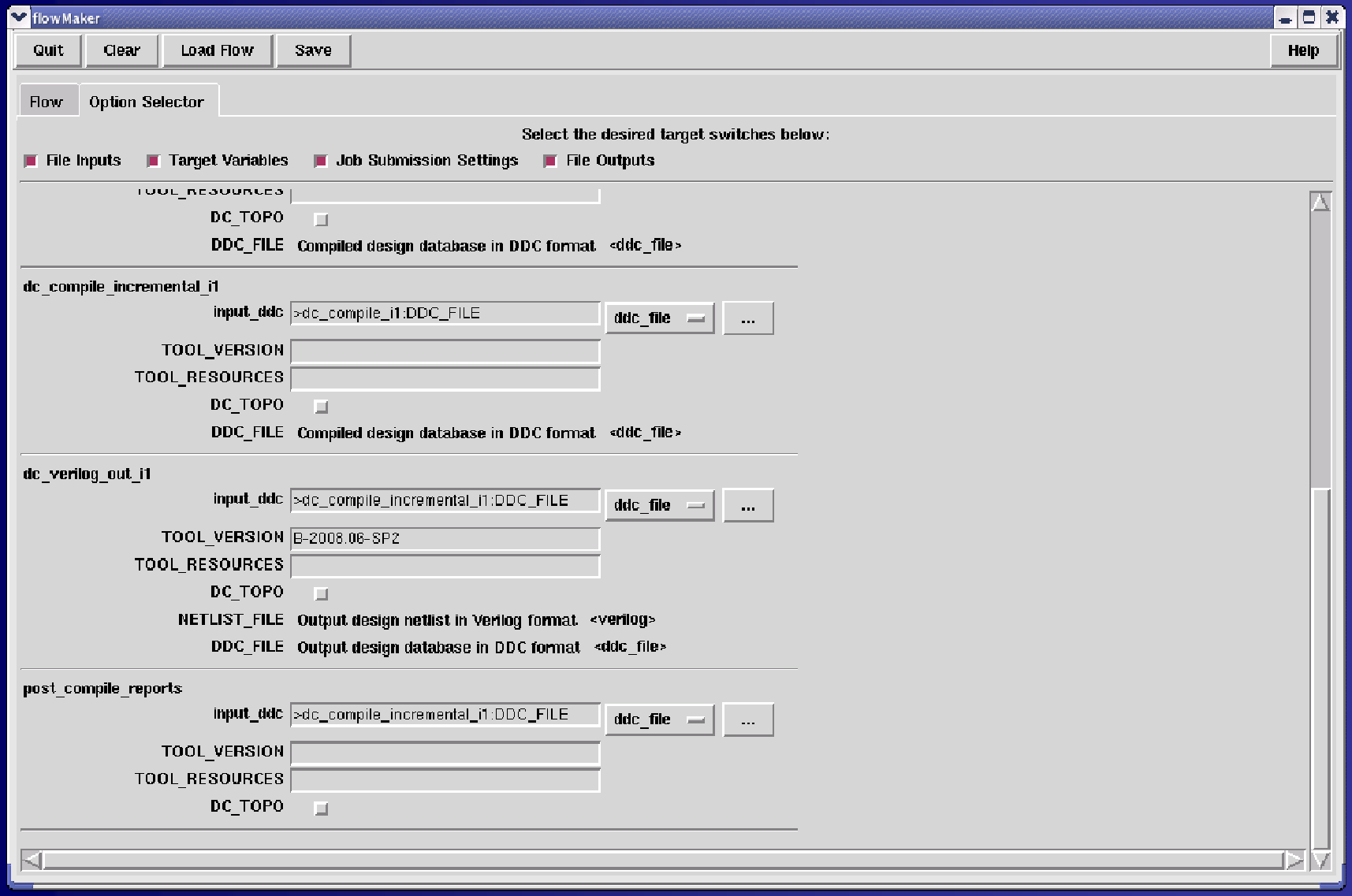
Add another target



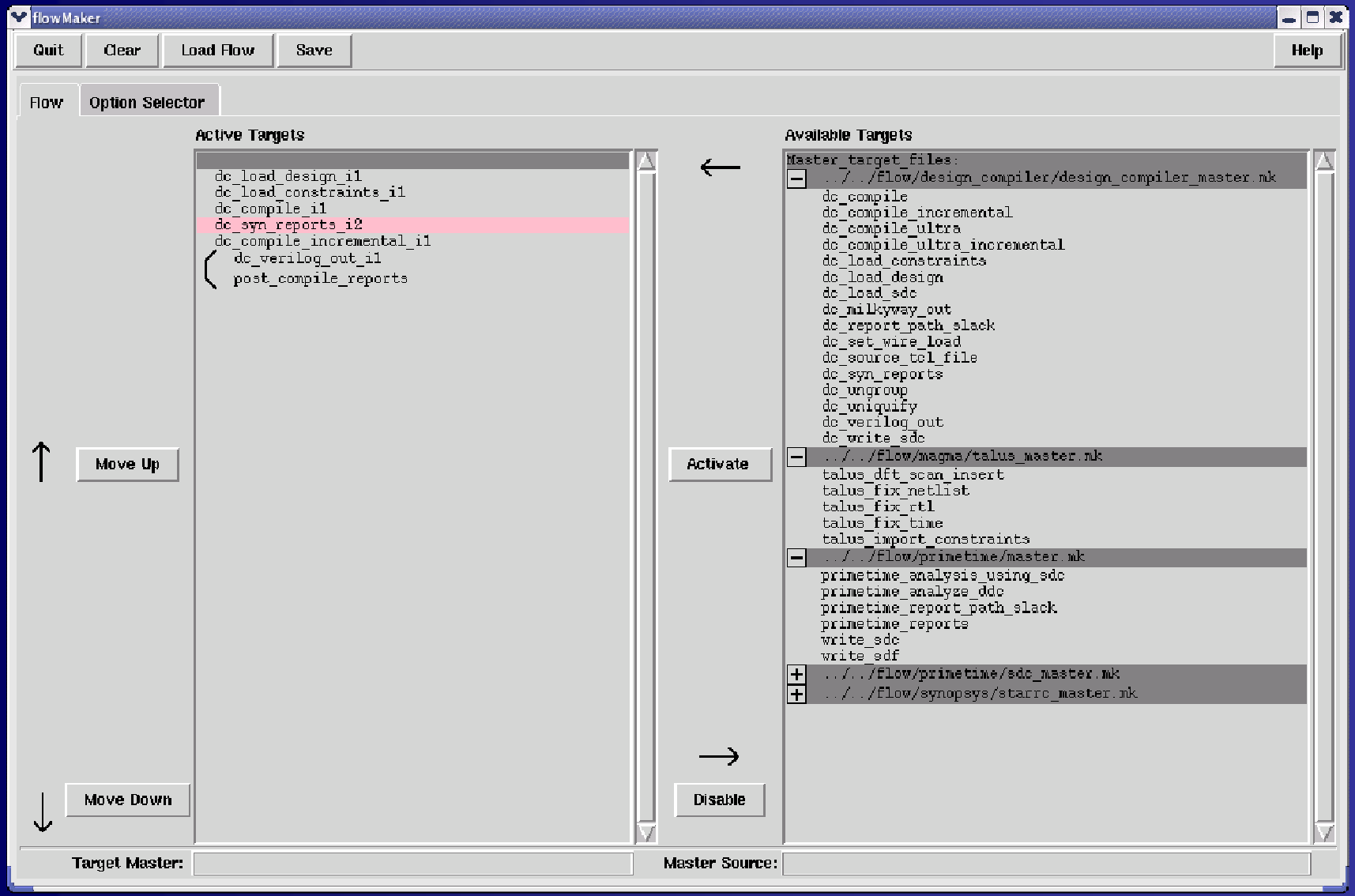
Rearrange target order



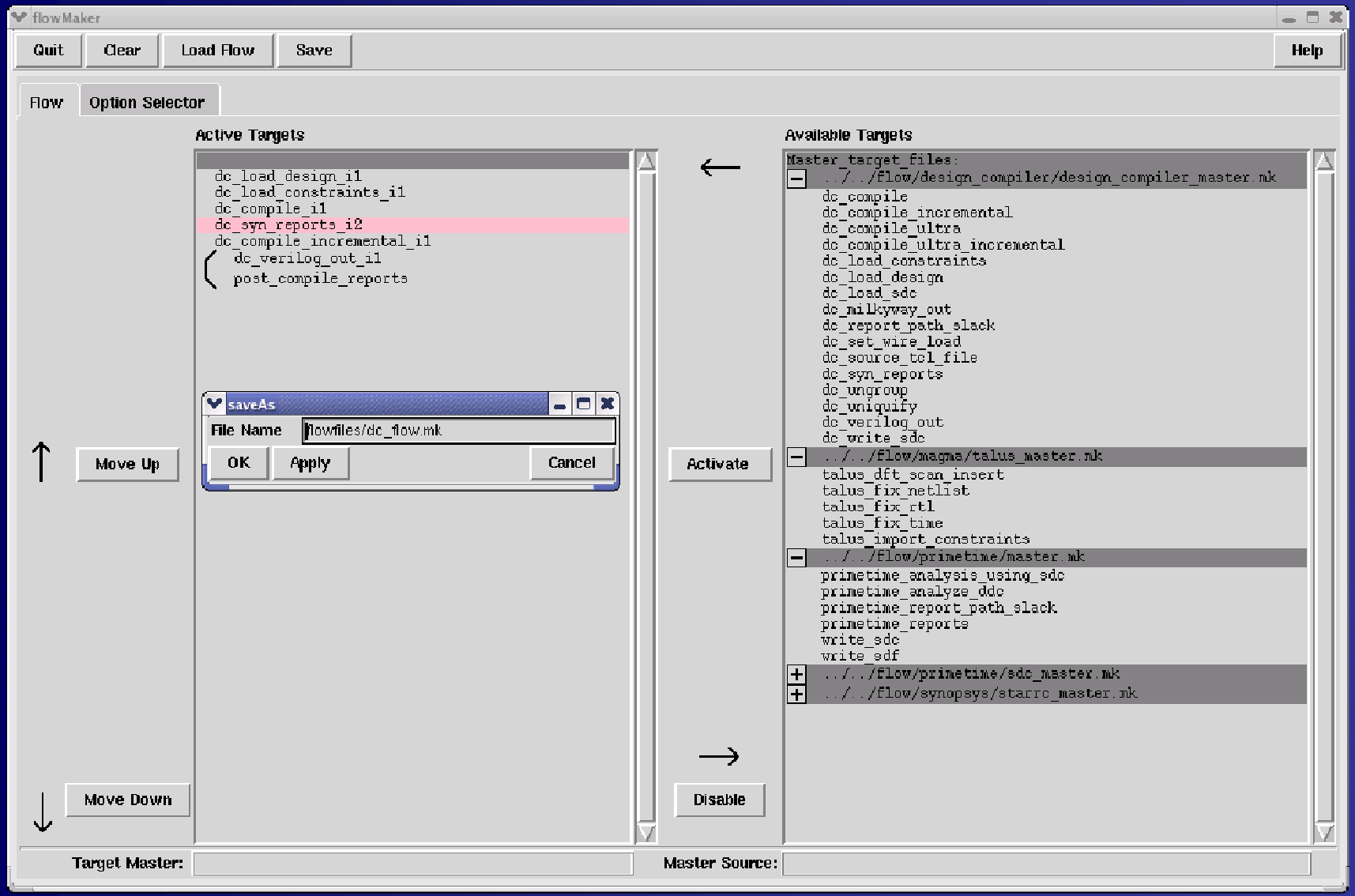
Update target options



Add duplicate target



Save flowfile



How to run a flow

```
% make FLOW=dc complete

% make FLOW=dc complete -j -k >& make.log &
```

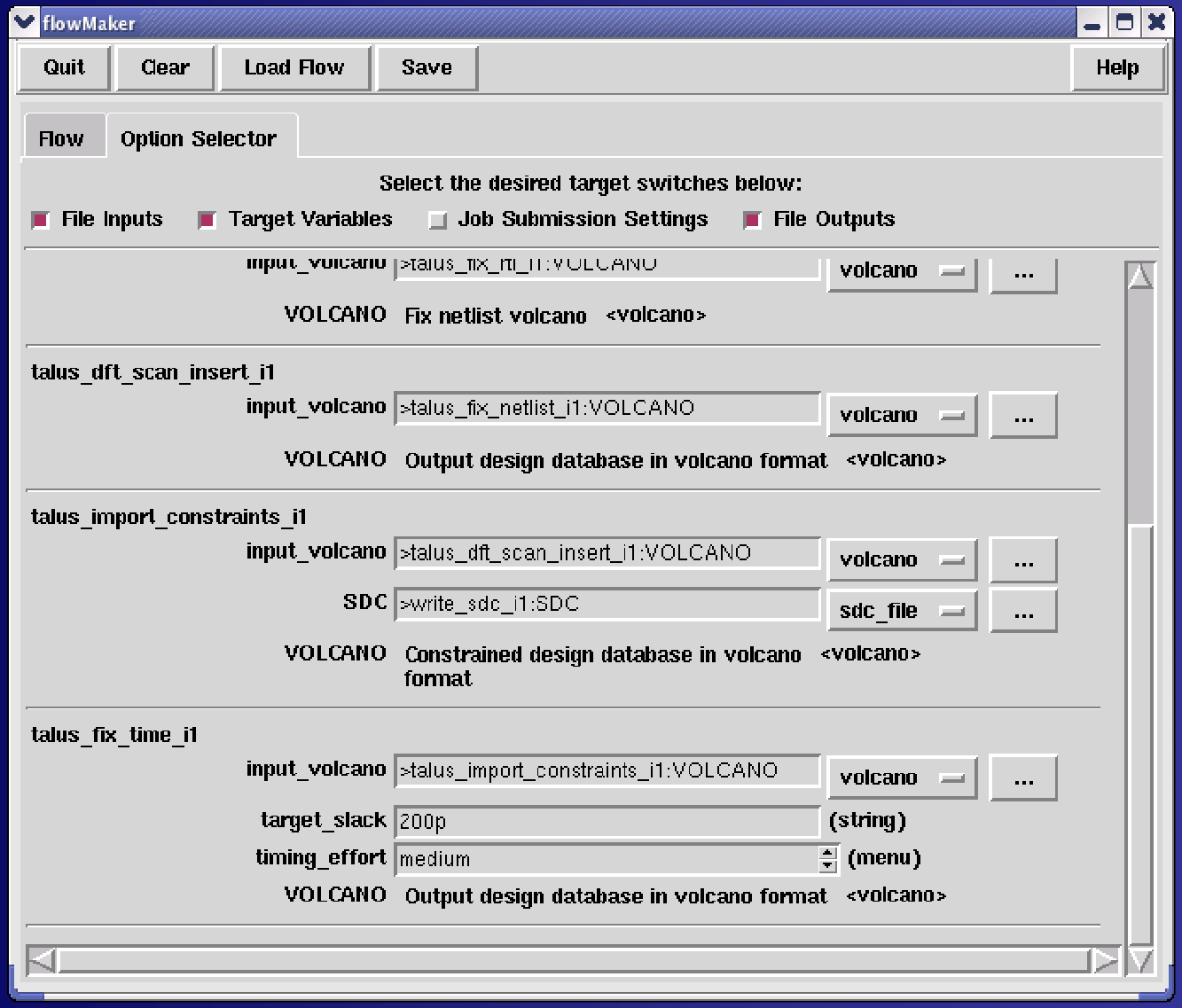
Anatomy of a flowMaker master target

```
dc_syn_reports:
## Basic post-synthesis Design Compiler reports
#
# in input_ddc
#   input_ddc(type) = ddc_file
#   input_ddc(comment) = Input design database in DDC format
#
# job DC_TOPO
#   DC_TOPO(type) = boolean
#   DC_TOPO(value) = 0
#   DC_TOPO(comment) = Enable DC topographical mode
#
   $(call dispatch, dc_shell \
        $(if $(findstring $(DC_TOPO),1),-topographical) \
        -f $(FLOW_ROOT)/design_compiler/dc_syn_reports.tcl \
        ,$(TARGET_DIR)/$(TARGET_INSTANCE).log)
```

Anatomy of a flowMaker master target

```
talus_fix_time:
## Reads in volcano, runs fix time
#
# in input_volcano
#    input_volcano(type) = volcano
#    input_volcano(comment) = Input design database in volcano format
# var target_slack
#    target_slack(type) = string
#    target_slack(value) = 0p
#    target_slack(comment) = Target slack with time unit
# var timing_effort
#    timing_effort(type) = menu
#    timing_effort(value) = medium
#    timing_effort(allowedvalues) = low medium high
#    timing_effort(comment) = timing effort option to fix time
# out VOLCANO
#    VOLCANO(type) = volcano
#    VOLCANO(filename) = $(BLOCK)%fix-time.volcano
#    VOLCANO(comment) = Output design database in volcano format
#
   $(call dispatch, \
        talus -f $(FLOW_ROOT)/magma/$@.tcl -logdir $(TARGET_DIR)  \
        , $(TARGET_DIR)/$(TARGET_INSTANCE).log )
```

Master target options



Different flowMaker users

- Just runs an existing flowfile.
  Invokes "make" from the command line.
- Uses flowMaker to edit the target options,
  then runs the modified flowfile.
- Uses flowMaker to edit the target options,
  edit the target sequence (add, delete, rearrange targets),
  then runs the modified flowfile.
- Writes new master targets
  to support new EDA tools and new features.

Things to consider

- What sort of directory structure do you have?
- What sort of revision control?
- What sort of configuration management and tags?
- How do you manage job control?
- How do you manage licenses?
- How do you manage tool versions?
- How do you manage your project?

Notes on project management

What you do *not* need: MS Project

- IC design is an iterative process, particularly if your flow is not stable
  - Flows will never be stable when used on an immature design
  - PERT charts are linear tools
  - OK to have major milestones but minor milestones are too detailed
- Don't be afraid of another iteration
  - with automation, rerun is easy

A suggested management method

- Make a list of everything you haven't done yet for your design (past successes don't count)
- Make a second list of everything not automated in your flow yet
- Schedule each item from both lists to occur concurrent with maturation of the RTL
- How long will the implementation take?
  - You mean to run through the tools the final time, or to get all the items in the list achieved?
  - You won't know the run time until both lists are empty
  - Hopefully both lists will be empty before the RTL stops changing
- Increase frequency of RTL releases as project matures
  - You need time between releases early on to work on the two lists
  - Towards the end, you need to iterate to make sure you are capable of running it all through

Managing iterations

- Iterate early, iterate often
- The last change in the RTL will be small
- A big change in the RTL won't be the last change
- How do you tell if you are done?
- How do you measure progress?
- Consider your process from RTL→GDSII:
  - Do you know the theoretical minimum time?
  - Do you know exactly how many steps are required?

Simple progress report

min_delay

		31Mar1019	30Mar0609	28Mar2229	27Mar1956	27Mar0000	25Mar1953
post_route	SLOW	2328	2471	1703	2146	1318	1376
	TYP					5412	5788
	FAST	4933	4923	81732	71500	62504	60014
DFT.scan_setup	SLOW	212	209	209	1573	1602	1646
	TYP					1652	1665
	FAST	538	535	535	1592	1666	1671
DFT.scan_shift	SLOW	14	14	27	15	15	15
	TYP					23	25
	FAST	54	44	49	21	42	43
DFT.stuck-at_scan_capture	SLOW	203	152	171	433	125	178
	TYP					166	241
	FAST	751	517		933	292	349
DFT.AC_scan_capture	SLOW	405	354		1730	1423	1476
	TYP					1458	1531
	FAST	1214	992		2219	1552	1609
DFT.boundary_scan	SLOW	44	51		1607	1651	1696
	TYP					1856	1881
	FAST	370	431		1843	1921	1926
DFT.dft_p1500	SLOW	200	3868		7711	7711	7711
	TYP					7711	7711
	FAST	526	3868		7711	7711	7711
DFT.sms_p1500	SLOW	200	659		6785	7067	7184
	TYP					7253	7271
	FAST	526	904		7224	7280	7281
	total	12518	19992	84426	115043	129411	127999

The Magic 8-Ball as a management tool

"Will there be problems with my new IP?"

"Is two weeks enough time to complete a floorplan?"

"Will we really see new timing problems if we turn on SI?"

"Will this chip go out on time?"

Checklist

Let's make up a checklist for physical implementation!

- Something like the Reuse Methodology Manual and Verification Methodology Manual
- All the things you need:
  - Software
  - Hardware
  - Other stuff

What you need: Other stuff

- Books: Tcl, perl, make, shell, UNIX, revision control
- Training
- Tool documentation
- Design documentation

What you need: Hardware and Infrastructure

- CPU -- compute farm (how many?)
- memory (how much?)
- disk (1Mbyte per placeable instance?)
- backups, printers, power, maintenance…
- desktop: usually PC so you can have office apps (email, Word)
- VNC or other connection to CPU
  how easily can you transfer files to/from desktop to CPU?
  how easily can you cut-and-paste?
- remote access: VPN
- bug tracking
- team communication (email, conferences, video links)

What you need: Software

- configuration management: revision control
- license management: job control
- flow management: flowMaker
- dependence management: make
- tool version management: job control
- compute farm management: job control
- error management: logscan + self-checking

- physical design tools
- libraries (standard cell, IO, memory, analog, 3rd party IP)

Quote Quiz

"I still don't see why we can't use the flow from the last chip."

1. Management X
2. CAD
3. EDA
4. Design

"How about a flow from the EDA vendor? Can't we start with what they offer?"

1. Management X
2. CAD
3. EDA
4. Design

"Didn't the guys in Bangalore just tape out? We can take their flow, right?"

1. Management X
2. CAD
3. EDA
4. Design

Thanks

- These companies loaned software keys to build up the example flow:
  - Magma
  - Synopsys
  - Cadence
  - CLK Design Automation
- Designs and libraries provided by:
  - Sun Microsystems
  - Artisan/ARM
  - Nangate
- Special thanks to
  Narendra Shenoy, Leon Stok and David Reda

Flow engineering: Things to remember

- **Principles**
  - #1 Flows are hard
  - #2 Iterate
  - #3 Automate
  - #4 Failure is your friend

- **Goals**
  - #1 Design independent flow
  - #2 Tool independent flow
  - #3 Technology independent flow
  - #4 User independent flow