

The CS-220 Development Environment

(No relevant sections in text)

Picking the right tool for the job



Integrated Development Environment

DVT - uvm-1.1d_ibus/sv/ibus_bus_monitor.sv - Eclipse Platform - /home/amiq/dvt_workspace

File Edit Navigate Search Project UVM Run Window Help

Navigator Code Test Type Help Trace Coverage

ibus_bus_monitor.sv ip_mac_clk_mng_g.v

```

950 class ubus_bus_monitor extends uvm_monitor;
96
97   // The virtual interface used to view HDL signals.
98   protected virtual ubus_if vif;
99
100  // Property indicating the number of transactions occurred
101  protected int unsigned num_transactions = 0;
102
103  // The following two bits are used to control whether
104  // done both in the bus monitor class and the interface
105  bit checks_enable = 1;
106  bit coverage_enable = 1;
107
108  // Analysis ports for the item collected and state no
109  uvm_analysis_port #(ibus_transfer) item_collected_port;
110  uvm_analysis_port #(ibus_status) state_port;
111
112  // The state of the ubus
113  protected ubus_status status;
114
115  // The following property is used to store slave address
116  protected slave_address_map#(string) slave_addr_map;
117
118  // The following property holds the transaction information
119  // being captured (by the collect_address_phase and
120  // protected ubus_transfer trans_collected;
121
122  // Events needed to trigger covergroups
123  protected event cov_transaction;
124  protected event cov_transaction_beat;

```

ip_mac_rx_hash_g.v

```

153 /*
154 ****
155
156 always @ (posedge pi_g_clock or negedge pi_reset) begin
157   if (~pi_reset) begin
158     po_rd_addr <= #`TP 4'h0;
159     po_err_match <= #`TP 1'b1;
160     po_match <= #`TP 1'b0;
161     po_fc_match <= #`TP 1'b0;
162     fsm_hash_st <= #`TP 'IDLE;
163   end
164   else begin
165     case (fsm_hash_st)
166       'IDLE: begin
167         // Control frame match
168         if (pi_abort == 1'b0 && new_match == 1'b1 &&
169           po_rd_addr <= #`TP 4'h0;
170           po_err_match <= #`TP 1'b0;
171           po_match <= #`TP 1'b1;
172           po_fc_match <= #`TP 1'b1;
173         end
174         // Promiscuous mode selected (always match)
175         else if (pi_abort == 1'b0 && new_match == 1'b1)
176           po_rd_addr <= #`TP 4'h0;
177           po_err_match <= #`TP 1'b0;
178           po_match <= #`TP 1'b1;
179           po_fc_match <= #`TP 1'b0;
180         end
181         // Pass all multicast mode selected
182       end

```

Out

Filter by: element_name

- ibus_bus_monitor
 - vif
 - num_transactions
 - checks_enable
 - coverage_enable
 - item_collected_port
 - state_port
 - status
 - slave_addr_map
 - trans_collected
 - cov_transaction
 - cov_transaction_beat
 - addr
 - data
 - wait_state
- cov_trans
 - trans_start_addr
 - trans_dir
 - trans_size
 - trans_addrXdir

Problems Tasks # Macros Coverage Checks Layers Console Terminal

0 errors, 17 warnings, 0 others

Description	Resource	Path	Location	ID	Type
Warnings (17 items)					
UNDECLARED_MODULE: Module 'ip_gate_clock' is not declared	ip_mac_host_if.v	/emac/rtl	line 2740	37	Verilog Semantic Problem
UNDECLARED_MODULE: Module 'ip_sync_reset' is not declared	ip_mac_host_if.v	/emac/rtl	line 2725	36	Verilog Semantic Problem
VERILOG_2001: NON_STANDARD: Parameter value not enclosed in parentheses	ip_mac_rx_top_g.v	/emac/rtl	line 381	22	Verilog Syntax Problem
VERILOG_2001: Redefinition of macro name: IDLE	ip_mac_cfg_hash_g.v	/emac/rtl	line 83	27	Verilog Syntax Problem
VERILOG_2001: Redefinition of macro name: IDLE	ip_mac_mdio_g.v	/emac/rtl	line 130	23	Verilog Syntax Problem
VERILOG_2001: Redefinition of macro name: RX_IDLE	ip_mac_hostif_rx.v	/emac/rtl	line 14	25	Verilog Syntax Problem

ibus_bus_monitor

Writable Insert 114 : 1 223M of 778M

Command mode: -----

Command Line Mentality



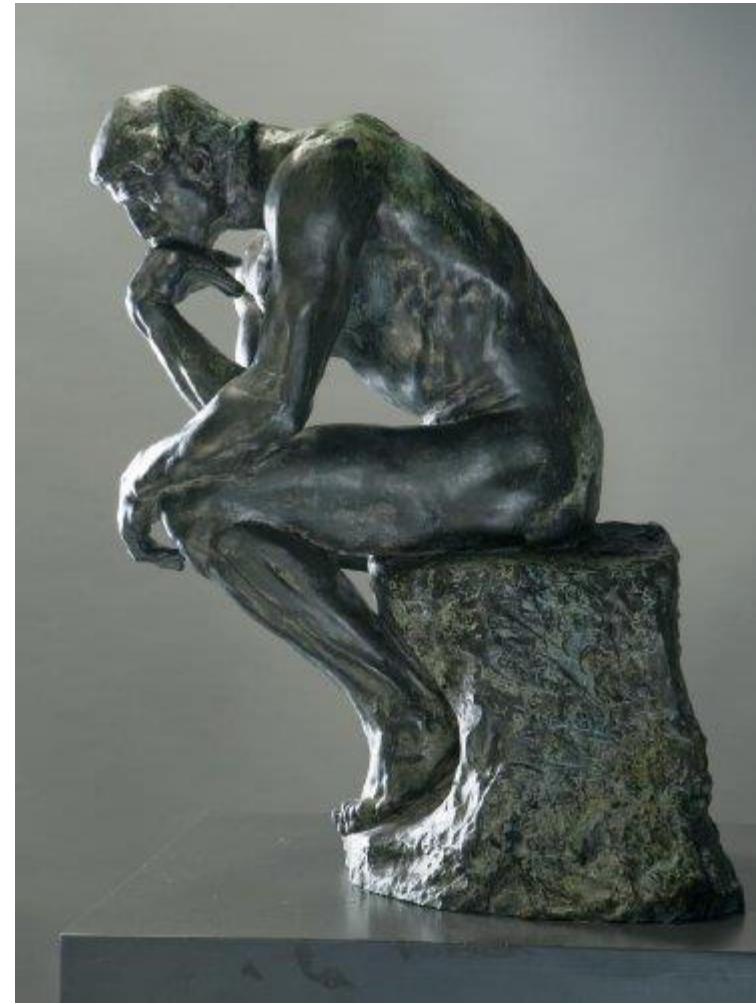
Old fashioned but...
...surprisingly efficient...
... (except for editing).

Command Line Mentality



- Artisan vs. Factory Worker
- Learn to use lots of tools
- May be slower
- Take more knowledge
- Require more effort
- Tools you can take with you
- Tools universally available
- Applicable to wide variety of projects

Picking the Right Tool for the Job



Operating System: UNIX

- Oldest “modern” operating system
- Base for Linux, HP/UX, Solaris, AIX, Android, etc.
- Most widely used
- Available on Lab accounts
- Free version available on Windows : CYGWIN
(<https://www.cygwin.com/>)
- Large library of free software - GNU compilers, editors, debuggers, etc.
(<http://en.wikipedia.org/wiki/GNU>)
- Assume basic knowledge of UNIX
(<http://www.ee.surrey.ac.uk/Teaching/Unix>)

WARNING: Not All UNIX is the same!

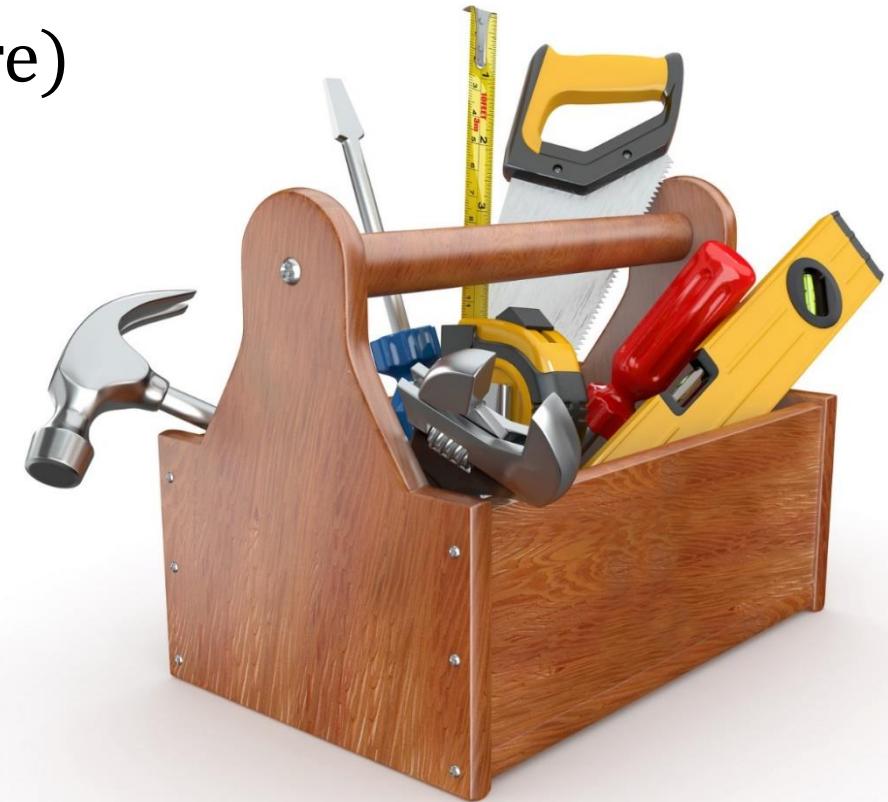
You may use:

Cygwin, virtual box, linux laptops, PODS linux, etc.
for labs, homework, projects, study, etc.

- Be aware – each UNIX installation is slightly different!
- If you have a technical problem, you are on your own!
- Grading will be performed on CS “LDAP” machines!
 - “It worked on my laptop” is not a valid excuse
- Test on LDAP machines!
- Use the lab(s) when no labs are running
- Access remotely with PUTTY

Basic Commands

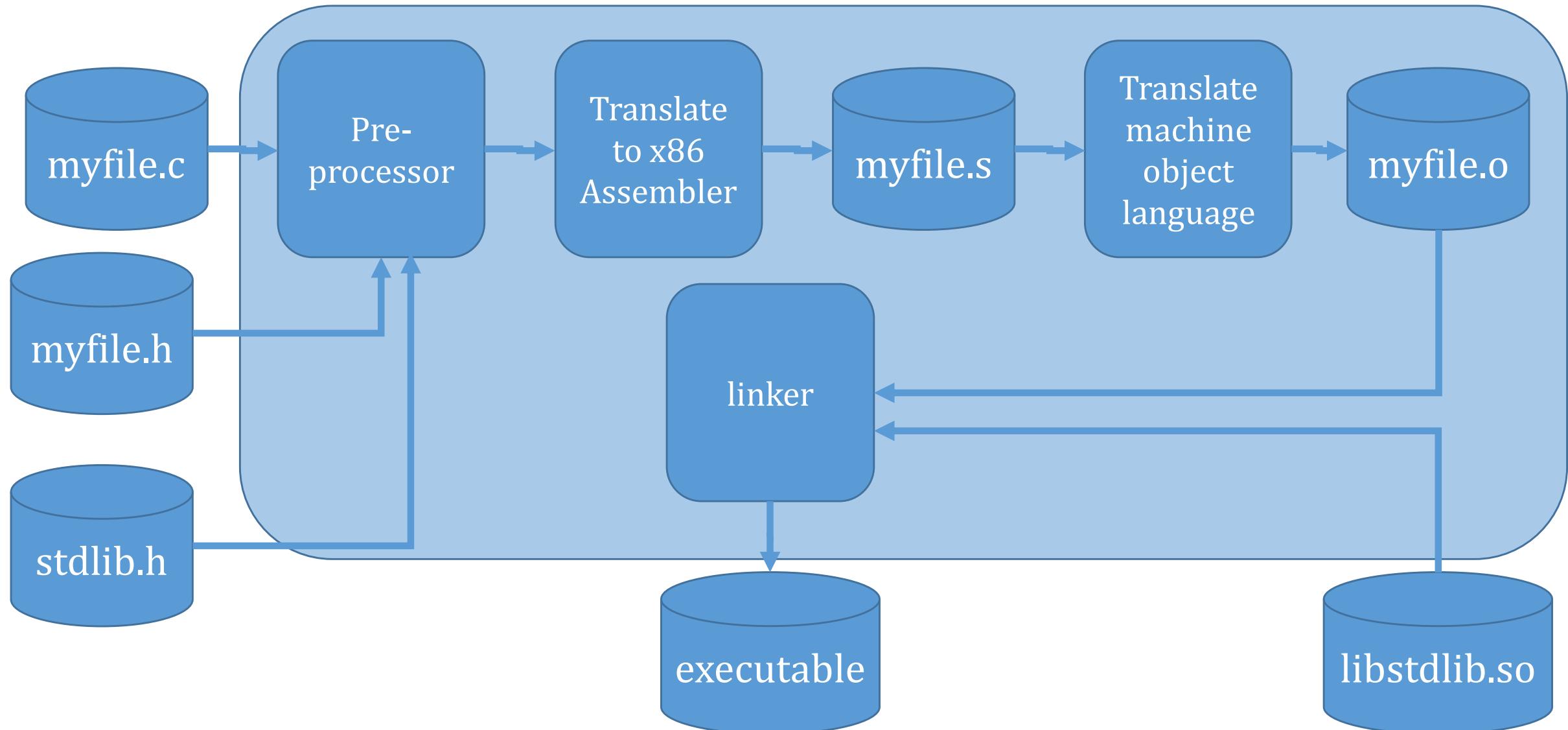
- Editor - gedit (not available everywhere)
- Compiler / Linker - gcc
- Build manager - make
- Debugger - gdb



Editor - gedit

- Hybrid command line / Graphical User Interface (GUI)
- First invocation from command line starts GUI
 - Suggest “gedit myfile.c &” first time – otherwise, hangs up terminal window until gedit is closed
- Once gedit is running, successive “gedit” commands send messages to existing GUI
- Recommend Edit/Preferences/
 - View - Display line numbers
 - View - Highlight matching brackets
 - Editor - Create a backup copy of files before saving

The compile / link process



GCC (Gnu C Compiler)

- `gcc <options> <source>`
- Basic options:
 - `-o <output file name / command name>`
 - `-g` [include debug information in output file]
 - `-Wall` [turn on all warning messages]
- For example: `gcc -g -Wall -o executable myfile.c`
- For more detail, `gcc -- help` or `man gcc`
- Complete documentation: <https://gcc.gnu.org/onlinedocs/>

Makefile

- File that tells “make” what to do and how to do it
- Composed of a list of “make rules”
- A rule has three parts:
 - Target – the file that this rule produces
 - Pre-requisite files – A list of file used to make the target
 - Recipe – Unix command(s) to produce target from the pre-requisite files
- For example:

```
mymain : mymain.c mymain.h comp.c comp.h
```

```
        gcc -g -Wall -o mymain mymain.c comp.c
```

Example Makefile

```
mycmd : mymain.c mymain.h comp.c comp.h
        gcc -g -Wall -o mycmd mymain.c comp.c
```

```
test : mycmd
        mycmd "test string"
        mycmd "test string 2"
```

```
clean:
        -rm mycmd
```

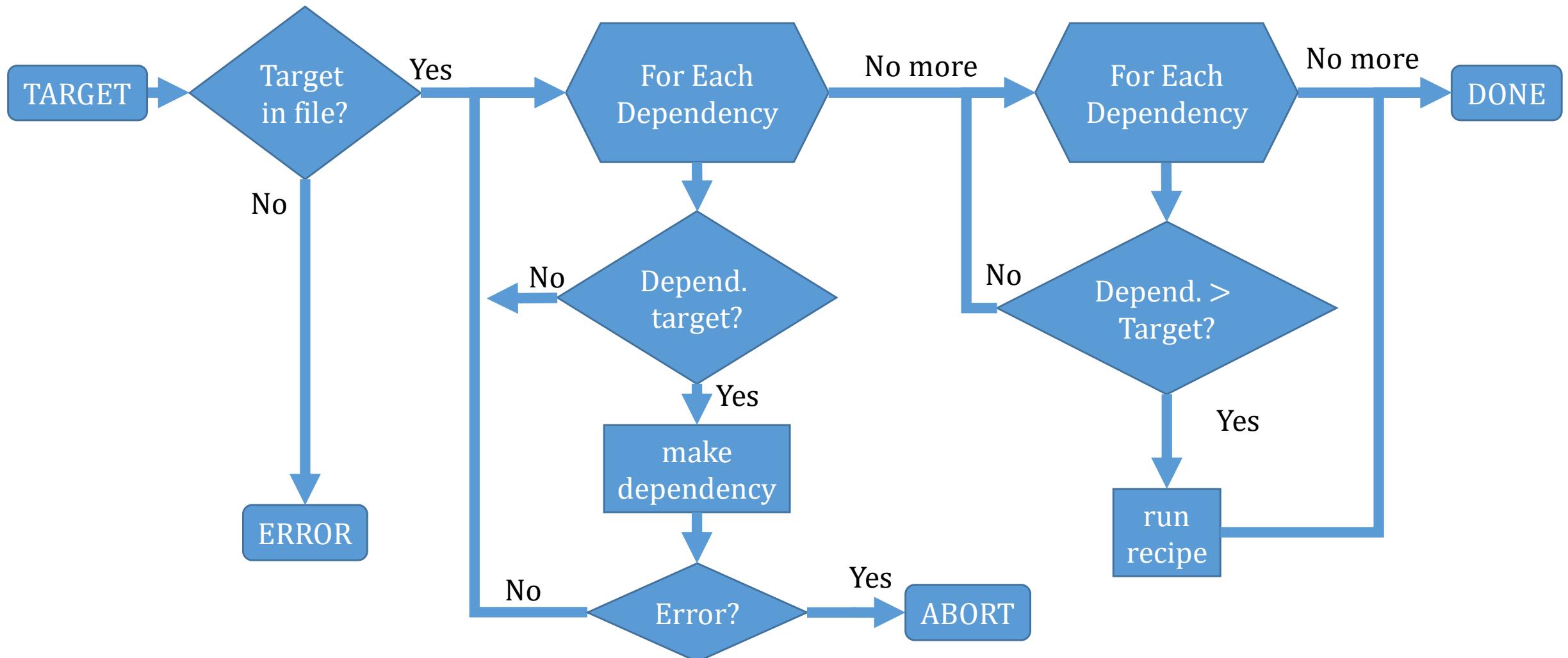
Invoking make

- Command : `make <target>`
- If `<target>` not specified, look for target “all”
- If `<target>` not specified and no “all” target, make first target in the make file

Make processing (simplified)

- Find the rule for the target specified
- Make all dependencies for that rule
 - At least those that show up as targets
- If any dependencies are newer than target file, invoke recipe

Make process (flowchart)



Example Make internals

```
mycmd : mymain.c mymain.h comp.c comp.h
        gcc -o mycmd mymain.c comp.c
        ~> make test
        // dependency mycmd is a target
        // make mycmd
        // dependency mymain.c not a target
        // dependency mymain.h not a target
        // dependency comp.c not a target
        // dependency comp.h not a target
        // mymain.c older than mycmd
        // mymain.h older than mycmd
        // comp.c older than mycmd
        // comp.h newer than mycmd
        // run gcc -o mycmd mymain.c comp.c
        // mycmd is newer than "test"
        // run mycmd "test string"
        // run mycmd "test string 2"
```

test : mycmd

mycmd "test string"

mycmd "test string 2"

clean:

-rm mycmd

Phony Targets

- Targets for which there is no corresponding file
 - For example: clean & test
 - When recipe is invoked, it does not create the dependency file!
- Recipe invoked each time target is specified
 - If make cannot find the target file, it assumes its date is ancient
 - Therefore, the phony “target file” is ALWAYS “older” than dependencies
 - Note that dependencies are still made... “make test” will “make mycmd”
- make mycmd
 - will run gcc –o mycmd only if mycmd is older than its dependencies
- make test
 - will run mycmd “test string” and mycmd “test string 2” EVERY time

Make Errors

- What are errors in make?
 - Compile Errors
 - Test Failed
 - etc.
- How does make know there was a problem?
 - If recipe returns a non-zero return code, there was a problem
 - Unless recipe is preceded by “-” which indicates ignore non-zero rc
 - For example make clean often performs commands like “rm mycomp.o”
 - If mycomp.o does not exist, rm returns non-zero rc
 - therefore, typically we use “-rm mycomp.o” for make clean
 - Useful to stop make when there is a problem

Example make file with multiple tests

```
test : test1 test2 test3
```

```
test1 : mycmd  
        mycmd testfile1.txt
```

```
test2 : mycmd  
        mycmd testfile2.txt
```

```
test3 : mycmd  
        mycmd testfile3.txt
```

```
mycmd : mycmd.c mycmd.h comp.c comp.h  
        gcc -g -o mycmd mycmd.c comp.c
```

- Run “make” or “make test” to run all tests
 - make will build mycmd if you have changed source files
 - make will stop with an error if compile failed
 - make will run test1
 - make will stop with an error if test1 fails
 - make will run test2
 - make will stop with an error if test2 fails
 - make will run test3
 - make will stop with an error if test3 fails
- If test2 fails
 - change the code to fix the problem
 - run “make test2” to see if that problem is fixed
 - then run “make test” to run all tests

Typical Targets

- all - all executables required
- clean - remove all built files
- test – test all executables built
- install - any commands required to make executables generally available

Make Concepts

- Build a product from components
 - Standardize the build process
 - Make it repeatable - make builds the same way every time
- Rebuild only components that have changed
 - As long as dependencies are correct, make will build exactly what is needed – and no more
- Package complex commands using simple targets
 - clean, test, debug – all may be sophisticated commands
 - invoke as “make test”

Debugging – GDB (Gnu DeBugger)

- `gdb <executable>`
 - Starts debugger
 - Loads executable
 - Prompts for gdb commands
- Basic GDB commands:
 - `h[elp]` – help on gdb commands
 - `b[reak] <location>` – set a breakpoint at the specified location
 - Location can be either a line number in a file or a function name
 - `run <command arguments>` - invoke command and run to next breakpoint
 - `c[ontinue]` – continue to next breakpoint
 - `s[tep]` – run next program instruction, stepping into function invocations
 - `n[ext]` – run next program instruction, skipping over function invocations
 - `p[rint] <variable or expression>` - print the current value of variable
 - `x <options> <location>` - print (examine) memory at locations
 - `q[uit]` – exit out of gdb

GDB hints

- `gdb -x gdbcmds.txt mycommand`
 - Runs command from `gdbcmds.txt` before prompting
 - Useful when you have lots of breakpoints and/or complicated parameters
- Null gdb command same as “repeat previous”
 - E.g. “step” <enter> <enter> to step 3 instructions
- If you see a prompt like:
`Breakpoint 1, 0x00000001004010d8 in main ()`
You forgot to compile with `-g`
- Prompt shows the line of C code ABOUT to execute
- Documentation:
<http://www.gnu.org/software/gdb/documentation/>