

DSD EXERCISE

HALF-, FULL-, AND RIPPLE-CARRY ADDERS IN VHDL,
COMMAND LINE TOOLS



AARHUS UNIVERSITY
SCHOOL OF ENGINEERING
PHM, CEF

JUNE 2018

Document History

2015-06-01: PHM, initial version.
2018-01-26: CEF, converted to \LaTeX .
2018-06-19: CEF, added 'Command Line Tools' exercise.
2018-08-30: CEF, fixed comma in heading,
2018-09-10: CEF, updated CLI exercise, minor text additions, and make d) and e) optional.
2019-01-15: CEF, minor text fix in CLI exercise.
2019-02-21: CEF, updated title and added hints to Command Line Tools.
2019-09-17: CEF, minor spell correction in CLI exercise.

Goals

The goals for this exercise are:

- To analyze and implement a half-adder in behavioural, dataflow and structural style.
- Design a full-adder and a 4-bit adder by combining the different coding styles.
- Understand how the code maps to the resulting RTL design and how it is implemented in the FPGA.
- Learn to use basic functional- and timing simulation tools.
- Use command line interface (CLI) to the Quartus compile pipeline, and make automated compile and timing scripts-

Prerequisites


- Have Quartus II up and running.
- That you have read THE DSD EXERCISE GUIDELINES!

1 The Half-Adder

The half-adder allows you to add two bits and output the sum and carry. The carry output is high if the sum overflows. In this exercise you must implement a half-adder in different coding styles and investigate the implementation and outcome.

Before you commence, you should investigate the half-adder listings in the book, sections: 2.6.1, 2.6.2 and 2.6.3 within the group – make sure to understand both the functionality and syntax. Also see page p. 288-289 in *Digital Fundamentals* for further information about half/full adders.

With confidence in half-adders, you may start your work:

- a) Implement the half-adder as Dataflow-, Behavioural- and Structural style in Quartus II. Use three different files, but within the same project. Set the file that you wish to compile, by setting it as the top-level design. (Project Navigator->Files Tab->select file->right-click->set as top level). Check the syntax by clicking on the  'start analysis and synthesis' (fast compile) icon.
- b) Verify that your design is synthesized as expected. View the compilation and synthesis results in **Tools => Netlist Viewers => RTL Viewer** and **Tools => Netlist Viewers=>Technology Map Viewer(post mapping)**. How do the three designs compare? Discuss the differences between the three and compare the resource usage in the Compilation Report summary.
- c) Perform a Timing Simulation and set the appropriate stimuli to verify the three design styles. Let the adder inputs change at the same time. Is there anything weird about the Simulation waveform? Maybe some spikes? (Hint: Static Hazard!) Discuss the simulation results and relate them to your implementation.
- d) Perform a Functional Simulation. Discuss the results and variations from the timing simulations.

2 The Full-Adder

The full-adder is the basic building block for adding several bits. In addition to the two input bits, it also takes a carry input, allowing us to add the carry output of a lower order addition to the sum.

The sum and carry outputs of a full-adder are with reference to figure 1 described as:

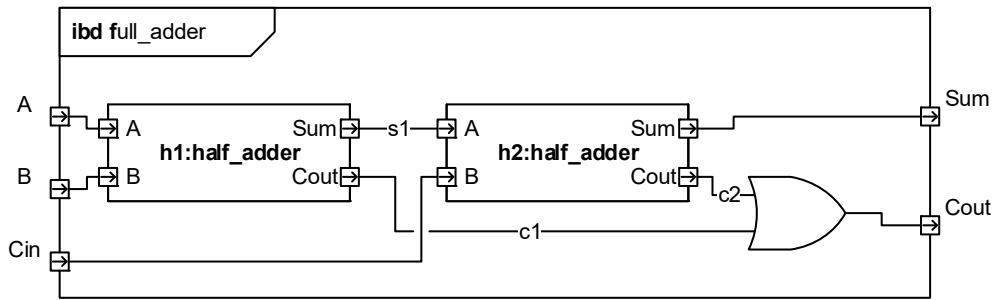


Fig. 1: Full-adder IBD.

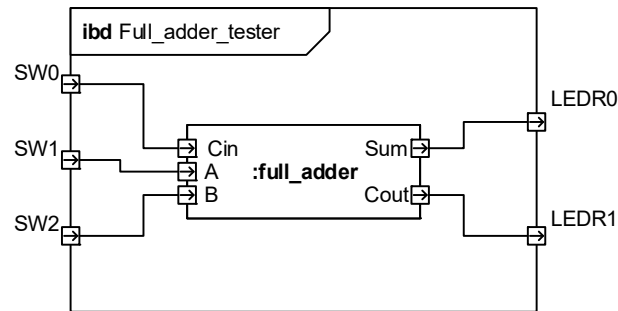


Fig. 2: Full-adder tester IBD.

$$\begin{aligned}
 \text{Sum} &= (A \text{ XOR } B) \text{ XOR } C_{\text{in}} \\
 &= (A + B) + C_{\text{in}} \\
 &= h1_{\text{Sum}} + C_{\text{in}}
 \end{aligned}$$

$$\begin{aligned}
 C_{\text{out}} &= (A \text{ AND } B) \text{ OR } (C_{\text{in}} \text{ AND } (A \text{ XOR } B)) \\
 &= (A \text{ AND } B) \text{ OR } (C_{\text{in}} \text{ AND } (A + B)) \\
 &= h1_{\text{Cout}} \text{ OR } (C_{\text{in}} \text{ AND } h1_{\text{Sum}}) \\
 &= h1_{\text{Cout}} \text{ OR } h2_{\text{Cout}}
 \end{aligned}$$

Your job now, is to build the full-adder from the half-adders previously created.

- Design and implement a full-adder by combining dataflow and structural style. You must choose yourself, which half-adder implementation to use. Do NOT duplicate existing sections of code, rather instantiate components based on existing code and include the source files in the Quartus project. Your hand-in will FAIL if you duplicate code in your work!
- Verify your design is synthesized as you expect it in the RTL Viewer/Technology map Viewer (Tools => Netlist Viewers). Is the structure similar to your design? What is contained within? (right-click on block->flatten netlist). What is the full-adder implemented as (TM Viewer)? How many Logic Elements has been used and why (Compilation Report)?
- Validate your design using a functional simulation. Can you validate the full-adder? How many combinations are needed?
- Test it on the DE2-Board. Create a new top-level design entity, `full_adder_tester`, instantiate the full adder in it and port map it to the DE2 board ports

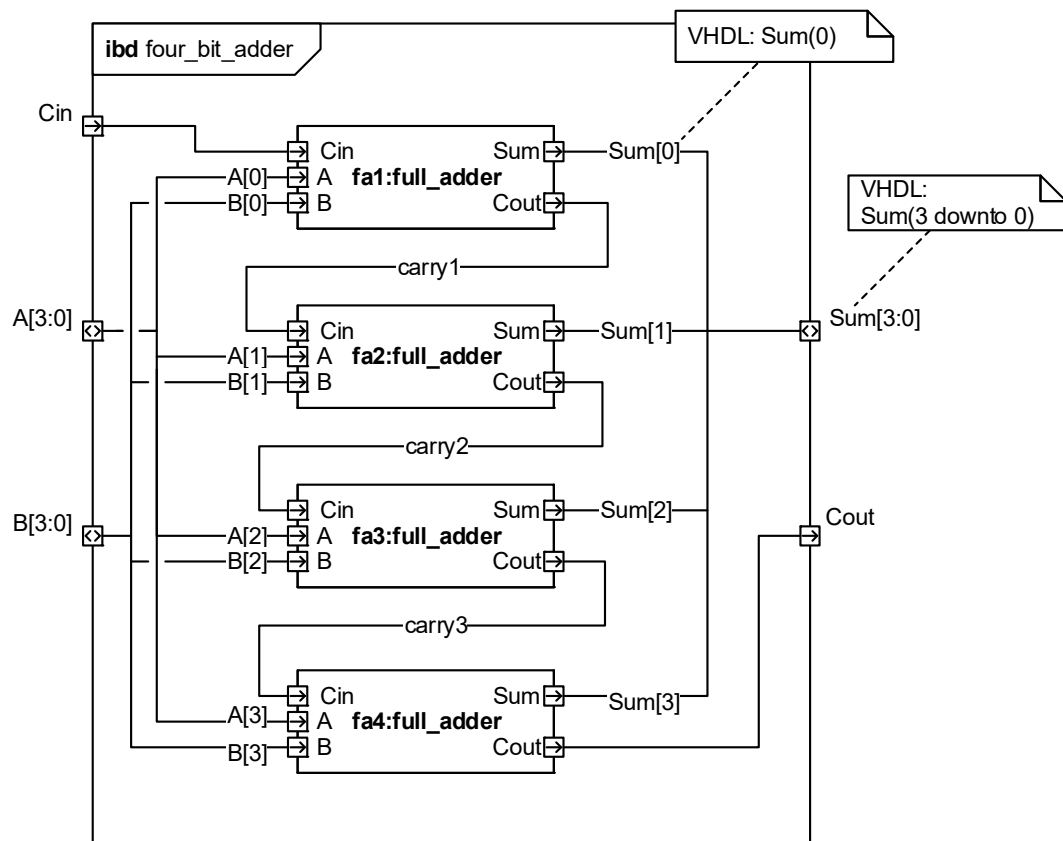


Fig. 3: Structural style 4-bit adder IBD.

as described in figure 2. Use the Pin Planner to assign pins to the DE2 board ports. See the DE2 User's Manual for pin connections. Remember to set the `full_adder_tester` as your top-level design before compiling. Add a photo of the result in the report.

3 The 4-Bit Adder

To add several bits, we produce a chain of full-adders. In this chain, we pass the carry bit from one full-adder to the next, to let the carry propagate up through the sum. This is also known as a ripple-carry adder. Your job is to create a 4-bit adder based on four of the full-adders you have created in the previously.

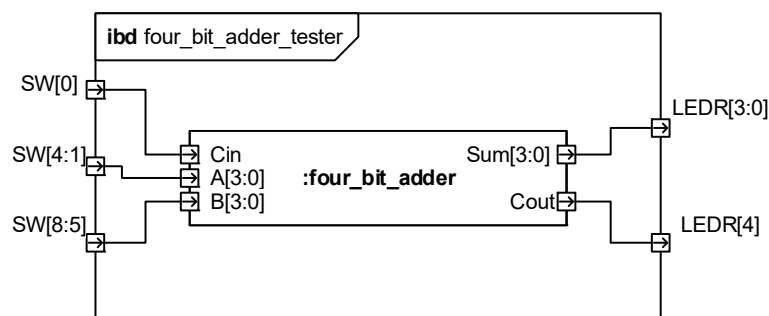


Fig. 4: 4-bit adder DE2 board tester IBD.

- a) Similar to the full-adder exercise, create the 4-bit adder using structural style. See the internals of the 4-bit adder in figure 3. Note the three carry signals and the ports, which are now `std_logic_vectors` rather than just `std_logic` bits. Remember to set the `four_bit_adder` as your new top-level design prior to compilation.
- b) Perform a timing simulation. Let the inputs switch simultaneously and note the outputs. Do you note any hazards? What determines how long it takes for the output to stabilize?—And what is thereby the limiting factor when it comes to processing speed?
- c) Create a new tester for the 4-bit adder as described in figure 4, build and test the design on your DE2-board

4 Command Line Tools

Using Quartus' IDE (integrated development environment) is easy and intuitively. But it also introduces some general GUI problems: how to automate the build and test process of software projects.

A typical setup, for a larger software project, will include ways of automatically building the entire system, running some regression tests and perhaps do a deployment of the newly build software package.

Building DE2 projects for VHDL source files consists of several distinct phases that can be controlled and called directly from a CLI (command line interface, or prompt/console). The phases for a Quartus compilation is shown in figure 5.

Note that Quartus projects consist of the following two main files

.qpf: Main project file. Just directs to the project settings revision.

.qsf: Project settings file. Keeps track of all project settings, files, and the place, where all your assignments goes into.

These files can be viewed and edited via a texteditor.

Hints

Correct directory: you must run the script from the directory *containing* the Quartus project (the .qpf and .qsf) files, otherwise it will fail.

Shell commands: use the shell commands `cd` (change directory), and `dir` (show directory), to navigate to your projectfolder, like (> denotes a CLI prompt)

```
> cd C:/ASE/Work/DSD/Exe2
```

and then

```
> dir.
```

Quartus projectfile splitup: sometimes your project files (.qpf and .qsf) are not named the same, say `halfadder.qpf` and `halfaddersub.qsf`. This will break the script! Be sure that you have say,

```
halfadder.qpf
halfadder.qsf
```

in you project folder.

- a) The full build pipeline for a VHDL project consist of several distinct steps. Describe the following steps found in figure 5: *synthesis*, *fitter*, *timing analysis*, *simulator* and *programmer*. In broad terms what does each processing step do, and what is its inputs and ouputs?

You might want to compare this pipeline with the description found in 'VHDL/PLD Design Flow' (figure 1.1.1 in 'VHDL for Engineers').

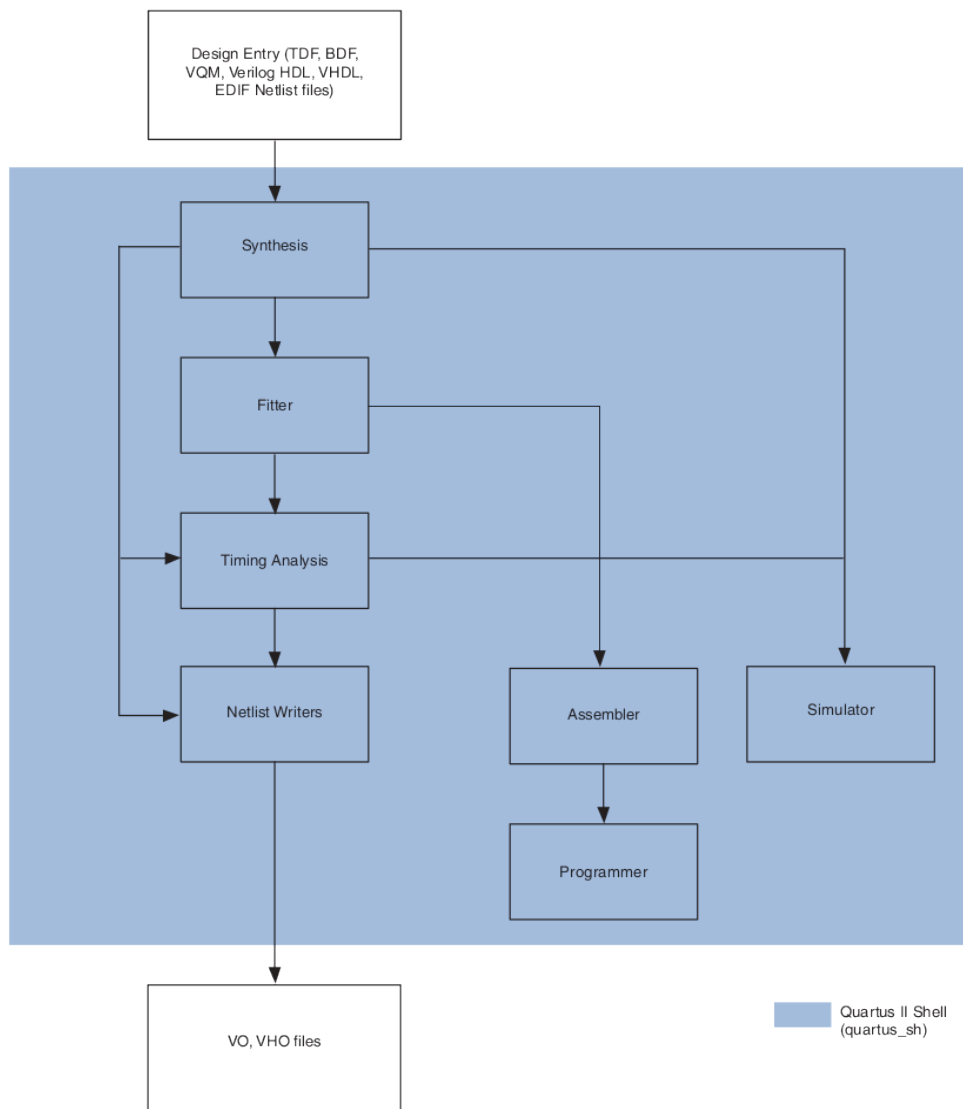


Fig. 5: The pipeline for sythetization of hardware outputfile from VHDL source files. (from 'Command-Line Scripting in the Quartus II Software', June 2003, Application Note 309 [command_line_tools_an309.pdf])

- b) Run a full build cyclus via the CLI-build script below. You need to set the parameter PROJ to your Quartus project name (the .qpf file). All settings (top-level-entities etc.) are read from the project files (.qpf, .qsf)**

Bash (Linux) script:

```
#!/bin/bash

# cli_compile.sh
# Version: 0.3
# 2018-09-10: CEF

QBIN=/opt/altera/13.0sp1/quartus/bin
PROJ=mytestproj

function CheckRet
{
    if [ $1 != 0 ]; then
        echo "ERROR: got return code=$1 for state $2, sorry can't cont'"
        return $1
    fi
}

if [ ! -d $BIN ]; then
    echo "ERROR: no such dir '$QBIN', you need to se QBIN in the CLI script"
    exit -1
fi

if [ ! -f $QBIN/quartus_map ]; then
    echo "ERROR: missing quartus binaries in dir '$QBIN', sorry I can't cont'"
    exit -2
fi

if [ ! -f $PROJ.qpf ]; then
    echo "ERROR: cannot locate Quartus '$PROJ.qpf' project file"
    exit -3
fi

if [ ! -f $PROJ.qsf ]; then
    echo "ERROR: cannot locate Quartus '$PROJ.qsf' project settings file"
    exit -4
fi

$QBIN/quartus_map --read_settings_files=on --write_settings_files=off $PROJ
CheckRet $? map
$QBIN/quartus_fit --read_settings_files=off --write_settings_files=off $PROJ
CheckRet $? fit
$QBIN/quartus_asm --read_settings_files=off --write_settings_files=off $PROJ
CheckRet $? asm
$QBIN/quartus_sta $PROJ
CheckRet $? sta
$QBIN/quartus_eda --read_settings_files=off --write_settings_files=off $PROJ
```

DOS CMD (Windows) script:

```
@ECHO OFF

REM cli_compile.bat
REM Version: 0.3
REM 2018-06-19: CEF
REM 2018-10-11: CEF, added checks for QBIN, Quartus project file and write
permissions to folder.

SET QBIN=c:/altera/13.0sp1/quartus/bin
SET PROJ=testproj

REM PRECONDITIONS CHECKS...
IF NOT EXIST %QBIN%/quartus_map.exe (
    ECHO "ERROR: '%QBIN%/quartus_map.exe' does not exist, please set QBIN variable
in CLI script to the correct Quartus install dir!"
    EXIT /B -1
)

IF NOT EXIST %PROJ%.qpf (
    ECHO "ERROR: file '%PROJ%.qpf' does not exist, please set PROJ variable in CLI
script to a correct Quartus project name!"
```

```

    EXIT /B -1
)

IF EXIST dummy.txt (
    DEL dummy.txt
    IF %ERRORLEVEL% NEQ 0 (
        ECHO ERROR in map stage!
        EXIT /B -1
    )
)
ECHO "test file writable" > dummy.txt
IF NOT EXIST dummy.txt (
    ECHO "ERROR: it seems like you current directory is write protected; I'll try to
        continue but no guaranties mate!"
)
DEL dummy.txt
REM PRECONDITIONS CHECKS...END

%QBIN%/quartus_map --read_settings_files=on --write_settings_files=off %PROJ%
IF %ERRORLEVEL% NEQ 0 (
    ECHO ERROR in map stage!
    EXIT /B -1
)
%QBIN%/quartus_fit --read_settings_files=off --write_settings_files=off %PROJ%
IF %ERRORLEVEL% NEQ 0 (
    ECHO ERROR in fit stage!
    EXIT /B -1
)
%QBIN%/quartus_asm --read_settings_files=off --write_settings_files=off %PROJ%
IF %ERRORLEVEL% NEQ 0 (
    ECHO ERROR in asm stage!
    EXIT /B -1
)
%QBIN%/quartus_sta %PROJ%
IF %ERRORLEVEL% NEQ 0 (
    ECHO ERROR in sta stage!
    EXIT /B -1
)
%QBIN%/quartus_eda --read_settings_files=off --write_settings_files=off %PROJ%
IF %ERRORLEVEL% NEQ 0 (
    ECHO ERROR in eda stage!
    EXIT /B -1
)

```

- c) Perform a timing simulation via the CLI-script below, reuse the timing-simulation from the Half-adder exercise.

To run the scripts you first need to run at least one timing simulation inside the Quartus GUI. This will put the simulation wavefile to use into the project files settings(.qsf), then you can run the script as many time you want!

Bash (Linux) script:

```

#!/bin/bash

# cli_timing.sh
# Version: 0.2
# 2018-06-19: CEF

QBIN=/opt/altera/13.0sp1/quartus/bin
PROJ=mytestproj

function CheckRet
{
    if [ $1 != 0 ]; then
        echo "ERROR: got return code=$1 for state $2, sorry can't cont'"
        return $1
    fi
}

if [ ! -d $QBIN ]; then
    echo "ERROR: no such dir '$QBIN', you need to se QBIN in the CLI script"

```



```

    exit -1
fi
if [ ! -f $QBIN/quartus_map ]; then
    echo "ERROR: missing quartus binaries in dir '$QBIN', sorry I can't cont'"
    exit -2
fi
if [ ! -f $PROJ.qpf ]; then
    echo "ERROR: cannot locate Quartus '$PROJ.qpf' project file"
    exit -3
fi
if [ ! -f $PROJ.qsf ]; then
    echo "ERROR: cannot locate Quartus '$PROJ.qsf' project settings file"
    exit -4
fi

$QBIN/quartus_map --generate_functional_sim_netlist \
    --read_settings_files=on --write_settings_files=off $PROJ
CheckRet $? map
$QBIN/quartus_sim --simulation_results_form=VWF \
    --read_settings_files=on --write_settings_files=off $PROJ
CheckRet $? sim

```

DOS CMD (Windows) script:

```

@ECHO OFF

REM cli_timing.bat
REM Version: 0.3
REM 2018-06-19: CEF
REM 2018-10-11: CEF, added checks for QBIN, Quartus project file and write
REM permissions to folder.

SET QBIN=c:/altera/13.0sp1/quartus/bin
SET PROJ=testproj

REM PRECONDITIONS CHECKS...
IF NOT EXIST %QBIN%/quartus_map.exe (
    ECHO "ERROR: '%QBIN%/quartus_map.exe' does not exist, please set QBIN variable
    in CLI script to the correct Quartus install dir!"
    EXIT /B -1
)

IF NOT EXIST %PROJ%.qpf (
    ECHO "ERROR: file '%PROJ%.qpf' does not exist, please set PROJ variable in CLI
    script to a correct Quartus project name!"
    EXIT /B -1
)

IF EXIST dummy.txt (
    DEL dummy.txt
    IF %ERRORLEVEL% NEQ 0 (
        ECHO ERROR in map stage!
        EXIT /B -1
    )
)
ECHO "test file writable" > dummy.txt
IF NOT EXIST dummy.txt (
    ECHO "ERROR: it seems like you current directory is write protected; I'll try to
    continue but no guaranties mate!"
)
DEL dummy.txt
REM PRECONDITIONS CHECKS...END

%QBIN%/quartus_map --generate_functional_sim_netlist ^
    --read_settings_files=on ^
    --write_settings_files=off %PROJ%
IF %ERRORLEVEL% NEQ 0 (
    ECHO ERROR in map stage!
    EXIT /B -1
)
%QBIN%/quartus_sim --simulation_results_form=VWF ^
    --read_settings_files=on ^
    --write_settings_files=off %PROJ%
IF %ERRORLEVEL% NEQ 0 (

```

```

        ECHO ERROR in sim stage!
        EXIT /B -1
    )

```

The output wavefile from the script will put into `db/mytestproj.sim.vwf` (if the project is named `mytestproj`), and can be opened via the Quartus GUI (or directly via the Quartus command `qwedt`).

You could modify the input `.vwf` file directly by editing the `.qsf` settings file looking for the line

```

set_global_assignment
    -name INCREMENTAL_VECTOR_INPUT_SOURCE <path>/mysim.vwf

```

with `<path>` and `.vwf` being set to your simulation file.

- d) (OPTIONAL) How do you detect errors (if any), when the timing simulation script is running? How do you detect the same errors, when running the same simulation in the GUI? (HINT: running the simulation in Quartus brings up a temporary output window, but this windows is auto-closed, even in the case where errors (red text) occurs in the simulation).
- e) (OPTIONAL) How could you modify the `cli_timing` script to create a timing simulation instead of a pure functional simulation? Checkup on the help for the simulator command via (`>` denotes a CLI prompt)

```
> quartus_sim -h
```

(that with full path (on Windows PC's) probably would look like `> c:/altera/13.0sp1/quartus/bin/quartus_sim -h`) and

```
> quartus_sim -help=mode
```

- f) (OPTIONAL) Run the full synthesization and timing via the Makefile as given below. You may need a Linux system with a Makefile package or you could install Cygwin on Windows (see <https://www.cygwin.com>) and use its makefile cababilities—remember to install `make` for Cygwin, as seen in figure 6.

I short a Makefile is a collection of small recipies, telling how to compile and link a given software project. It can also contains script-snippes on how to automatically test and clean your project.

The makefile 'scripts' or recipies is build from *declarative*¹ steps like

```

target: dependencies # prerequisite for making target
      system command(s) # recipe for making target

```

Makefile:

¹The Makefile can be viewed as a declarative language—one that tell how to build but without explicit giving the specific order of the steps. This is in contrast to *imperative* languages, like C or C++, where all programming steps are specified in details through a series of statements.

```

# Makefile
# Version: 0.1
# 2018-06-19: CEF

# NOTE: you need to modify these settings:
#QBIN=C:/altera/13.0sp1/quartus/bin
#QBIN=/opt/altera/13.0sp1/quartus/bin
PROJECT=mytestproj

#DEPS = $(PROJECT).qpf $(PROJECT).qsf Makefile
#FASTMAP=--effort=fast --incremental_compilation=full_incremental_compilation
#FASTFIT=--effort=fast

.PHONY=all
all:
    $(QBIN)/quartus_map $(PROJECT) --read_settings_files=on \
        --write_settings_files=off $(FASTMAP)
    $(QBIN)/quartus_fit $(PROJECT) --read_settings_files=off \
        --write_settings_files=off $(FASTFIT)
    $(QBIN)/quartus_asm $(PROJECT) --read_settings_files=off \
        --write_settings_files=off

    @# NOTE: sta stage skipped
    @$(QBIN)/quartus_sta $(PROJECT)
    $(QBIN)/quartus_eda $(PROJECT) --read_settings_files=off \
        --write_settings_files=off

srcok:
    $(QBIN)/quartus_map $(PROJECT) --csf=filtref \
        --analyze_file=code_lock_advanced.vhd

.PHONY=timing
timing:
    $(QBIN)/quartus_map $(PROJECT) --generate_functional_sim_netlist
    $(QBIN)/quartus_sim $(PROJECT) --simulation_results_form=VWF
    @# NOTE: experimental..
    @#quartus_eda --gen_testbench --check_outputs=on \
    @# --tool=modelsim_oem \
    @# --format=verilog $(PROJECT) -c $(PROJECT)
    @#quartus_eda --functional=on --simulation=on
    @# --tool=modelsim_oem
    @# --format=verilog $(PROJECT) -c $(PROJECT)

.PHONY=program
program:
    # NOTE: not testet yet
    $(QBIN)/quartus_pgm --no_banner --mode=jtag -o "P;$(PROJECT).sof"

.PHONY=vwfview
vwfview:
    # NOTE: not working 100% yet
    $(QBIN)/qwedt --filename ./db/mytestproj.sim.vwf

.PHONY=clean
clean:
    @ rm -rf db incremental_db simulation output_files

```

Makefile CLI commands :

- > make
 - Build the entire project. A full compile, just like a calling 'Processing | Start compiling' in the GUI.
- > make clean
 - Cleanup the project, that is remove all the temporary files in the directory. This is not possible in the GUI!.
- > make timing
 - Build the timing output. The result can be viewed from with in Quartus by opening the output .vwf file.

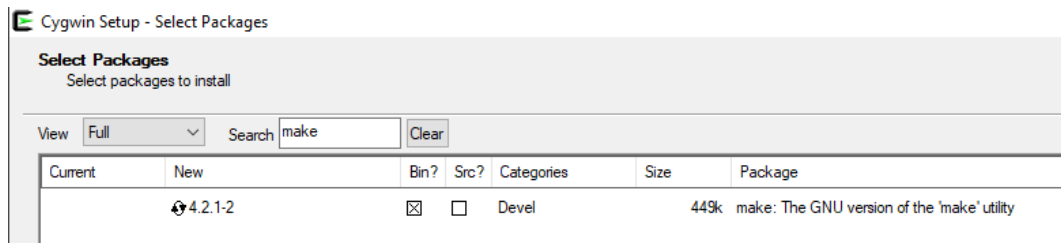


Fig. 6: When installing Cygwin for Windows you need to add the add the gnu 'make' package to be able to use makefiles.

- > `make srcok`
Checks a single file for syntax errors. File is hardcoded in the Makefile.
- > `make program`
Sends the output to the programmer. Functionality not tested.
- > `make vwfview`
Sends the waveform output to the Quartus waveform editor. Functionality not tested.