

Dokumentation der Scripts

Version 1.0

Betreuer:
Delvai Martin
Huber Wolfgang

Erb Bernhard - 0026163 - bernhard.erb@gmx.net
Granzer Wolfgang - 0026013 - woif@woif.org
Praus Fritz - 0025854 - fritz@praus.at
Wielander Gerd - 0026165 - gerd.wielander@gmx.net

hdl@ecs.tuwien.ac.at

Institut für Technische Informatik (E 182-2)
Technische Universität Wien
A-1040 Wien, Treitlstraße 3

27. Juni 2004

Inhaltsverzeichnis

1	Einleitung	3
2	Allgemein	3
2.1	Kurzbeschreibung	3
2.2	Anforderungen	4
2.3	Changelog	7
3	Makefile	8
3.1	Erzeugung eines neuen Workspaces	8
3.1.1	Target .newdir	8
3.1.2	Target .gendir	8
3.1.3	Target .updatedir	8
3.1.4	Target .mkbasiccfg	8
3.1.5	Target .mktopcfg	8
3.1.6	Target .mktopscr	8
3.1.7	Target .mkautomakefile	8
3.2	Erzeugen der Top Architecture und Entity und der Testbench	9
3.2.1	Target .mktopvhdl	9
3.2.2	Target .mktoptb	9
3.3	Synthese	9

3.3.1	Target .syn	9
3.3.2	Target .syn_func	9
3.3.3	Target .syn_pre	9
3.4	Simulation	10
3.4.1	Targets zu Simulation mittels graphischen Simulator	10
3.4.2	Targets zu Simulation mittels Commandline- Simulator	10
3.4.3	Targets zur Analyse der VHDL Dateien	10
3.5	Loeschen eines vorhandenen Workspaces	11
3.5.1	Target .rmdir	11
3.5.2	clean	11
3.5.3	Target all	11
3.6	Anzeigen der Hilfe	11
3.6.1	Target all	11
4	Scripts	11
4.1	checkError	11
4.2	copyHex	12
4.3	generic.awk	12
4.4	main.awk	12
4.5	makeAutoMake	12
4.6	makeBasic	12
4.7	makeConfig	12
4.8	makeTopcfg	12
4.9	makeTopscr	13
4.10	modulcfg.awk	13
4.11	parse.awk	13
4.12	replaceFunc	13
4.13	replacePre	13
4.14	testDir	14
4.15	testDir2	14
4.16	testDir3	14
4.17	testFile	14
4.18	testbench.awk	14
4.19	use.awk	14
4.20	trimNames.awk	14
5	Ablaufdiagramme	16

1 Einleitung

Diese Arbeit entstand im SS04 im Rahmen der Arbeitsgemeinschaft HDL. Ziel war es, das fehleranfällige und zeitkonsumierende Zusammenstellen eines Prozessors samt Erweiterungsmodulen zu automatisieren. Mithilfe der praktischen Erfahrung und den Erkenntnissen der Betreuer sowie der Arbeit der Studenten sollte ein System geschaffen werden, welches selbstständig den Prozessor und die Module konfigurieren kann, es aber dem Benutzer auch ermöglicht Veränderungen - zum Beispiel durch Konfigurationsfiles - einfach durchzuführen.

Entstanden sind eine Fülle von Scripts, die es dem Benutzer ermöglichen, den gewünschten Core und die gewünschten Module per einfachem Menü auszuwählen. Nach dieser Auswahl kann der Benutzer - aber muss nicht - im Konfigurationsfile Änderungen an z.B. der Baseadresse bzw. den Interrupts durchführen. Danach kann das Gesamtsystem die Synthese und die Simulation durchführen.

2 Allgemein

2.1 Kurzbeschreibung

Zum Anlegen eines neuen Prozessorkerns müssen folgende Schritte durchgeführt werden:

- 1) `make <proc_name>.newdir` im Workspace-Verzeichnis aufrufen
Legt die Direktories fuer die Instanz an
- 2) Gewünschte Module auswählen
- 3) Konfigurieren der Instanz im `top.cfg`
- 4) `make <proc_name>.mktopvhdl` im Workspace-Verzeichnis aufrufen
Legt top VHDL Files an
- 5) `make <proc_name>.mktoptb` im Workspace-Verzeichnis aufrufen
Legt Testbench und Configurationfiles an
- 6) Assembler Routinen erzeugen
- 7) Synthese starten mit:
`make <proc_name>.syn_func`
`make <proc_name>.syn_pre`
`make <proc_name>.syn`
- 8) Simulation ausführen mit:
`make <name>.sim_[beh][func][pre][post] |`
`make <name>.dbx_sim_[beh][func][pre][post]`
Diese Targets starten die Analyse und danach wird der Simulator aufgerufen.

Die Analyse des Cores/Module sowie des User-parts kann folgendermassen aufgerufen werden:
`make <name>.[beh][func][pre][post]_core |`

```
make <name>.[beh][[func]][[pre]][[post]]_user
```

2.2 Anforderungen

Brom Bausteine:

Folgende Syntax wird vorausgesetzt:

Im `core_spear.cfg` wird die Art des ROM Baussteins und der Dateiname mithilfe folgendem Eintrag festgelegt .

```
INIT_FILE : <ROM Name (z.B. asyn_rom_128x16 ) > := <file name>
```

z.B.

```
INIT_FILE : asyn_rom_128x16 := "src/brom.hex"
```

Dabei sind die Zeichen ":" und ":=" zwingend notwendig (Whitespaces werden ignoriert).

Im VHDL File, welches bei der Synthese erzeugt wird, werden die notwendigen Aenderungen durchgefuehrt.

Dabei wird folgendes ersetzt :

```
component <ROM name>
wird ersetzt in
component <ROM name>
-- pragma translate off
generic (LPM_FILE :string):
-- pragma translate on
```

Ausserdem wird folgens ersetzt:

```
brom: <ROM name>
wird ersetzt durch
brom: <ROM name>
-- pragma translate off
generic map (LPM_FILE => "<filename>")
-- pragma translate on
```

Dabei wird jeweils nach den Schlüsselwoertern "component" und "brom" gesucht (in Verbindung mit dem Rom Namen z.b. `asyn_rom_128x16`).

Whitespaces werden ignoriert, der ":" bei "brom" ist zwingend.

Abschneiden der Komponentennamen:

Bei der Synthese vergibt das Synthesetool den verschiedenen Komponenten (Core und Extension Module) eigene Namen. Diese sind allerdings fuer das Simulationstool zu lang (das Tool gibt folgende Fehlermeldung aus "Name too long").

Daher werden die Namen mithilfe des AWK Scripts "trimNames.awk" abgeschnitten.

Dabei wird die vom Synthesetool erzeugte VHDL Datei nach folgenden Mustern durchsucht:

```
core_.*
```

```
ext_.*_.*
```

Diese Strings werden nun durch folgende Namen ersetzt:

core_<corename> (z.B. core_spear)

bzw. ext_<modulname>_<instanznummer> (z.B. ext_Dis7Seg_1 oder ext_Dis7Seg_2)

Es ist zubeachten, dass eine Entity Deklaration mit "end <entityname>;" beendet werden muss. Daher wird dieser Strichpunkt extra beruecksichtigt , da die Vergabe der Instanznummern auf den gesamten Namen beruht. Besitzen 2 Muster den selben Namen, wird die selbe Instanznummer vergeben.

Beispiel :

```
entity ext_Dis7Seg_ABC_DEF_GH_IJ_KLMN_MFIE is
```

```
    port( clk , reset , AccViol, ExtEna, ExtWr : in std_logic;  ExtAddr, Data2Ext
          : in std_logic_vector (15 downto 0); WrBData_Dis : out
          std_logic_vector (15 downto 0); ExtIntReq : out std_logic;  LED : out
          std_logic_vector (7 downto 0); SegA, SegB, SegC, SegD, SegE, SegF,
          SegG, SegDOT : out std_logic; PIN_select : out std_logic_vector (3
          downto 0));
```

```
end ext_Dis7Seg_ABC_DEF_GH_IJ_KLMN_MFIE;
```

```
architecture FUNC_behaviour of ext_Dis7Seg_ABC_DEF_GH_IJ_KLMN_MFIE is
```

```
    component SYNOP_BASIC_THREE_STATE
        port( function_port , three_state : in std_logic;  output : out std_logic
              );
    end component;
```

```
    component GTECH_AND2
        port( A, B : in std_logic;  Z : out std_logic );
    end component;
```

```
.
.
.
```

wird ersetzt durch:

```
entity ext_Dis7Seg_1 is
```

```
    port( clk , reset , AccViol, ExtEna, ExtWr : in std_logic;  ExtAddr, Data2Ext
          : in std_logic_vector (15 downto 0); WrBData_Dis : out
          std_logic_vector (15 downto 0); ExtIntReq : out std_logic;  LED : out
          std_logic_vector (7 downto 0); SegA, SegB, SegC, SegD, SegE, SegF,
          SegG, SegDOT : out std_logic; PIN_select : out std_logic_vector (3
          downto 0));
```

```
end ext_Dis7Seg_1;
```

architecture FUNC_behaviour of ext_Dis7Seg_1 is

```

component SYNOP_BASIC_THREE_STATE
  port( function_port, three_state : in std_logic ; output : out std_logic
        );
end component;

```

```

component GTECH_AND2
  port( A, B : in std_logic ; Z : out std_logic );
end component;

```

```

.
.
.

```

Sourcdateien und Namensgebung:

Samtliche Sourcdateien muessen sich im Verzeichnis "src/" des jeweiligen Moduls befinden. Dabei ist zu beachten, dass das Modulverzeichnis als Modulname (MODULVERZEICHNIS=MODULNAME!!) verwendet wird (Gross- und Kleinschreibung beachten). Zur Erstellung der Top.cfg Datei, werden alle "generics" aus dem Entityfile des Modul herausgesucht (Achtung: das Entityfile muss den Namen "<modul_name>.ent.vhd" haben; Gross- und Kleinschreibung beachten).

Zusaetzlich wird die Konfigurationsdatei aus dem Verzeichnis "cfg/" des Moduls herangezogen. Das File muss folgenden Namen besitzen: "<modul_name>.cfg".

Auch hier muss der Modulname gleich dem Verzeichnis des Modul sein.

In der Datei "sources", welches sich im jeweiligen Modulverzeichnis befindet, sind alle fuer die Synthese notwendigen Sourcefiles des Moduls aufgelistet.

Dabei sind die Dateien relativ zum Modulverzeichnis anzugeben (z.B. "src/core_spear.vhd"). Da die Sourcdateien genau in der Reihenfolge synthetisiert werden, in der sie in dieser Datei aufgelistet sind, ist auf die richtige Reihenfolge achtzugeben.

Desweiteren wird auf Existenz der Datei "sources_sim" geprueft. In dieser Datei sind jene Sourcdateien aufgelistet, die nur fuer die Simulation gebraucht werden (d.h. z.B. RAM Bausteine). Existiert diese Datei, so werden die aufgelisteten Sourcdateien bei der Simulation vor den Dateien in "sources" behandelt.

Erstellen der top-Entity und -Architecture:

Im File top.cfg muessen die betrachteten Signale stets mit dem Schluesselwort "vhdl_generic" beginnen.

Wenn es sich um einen Interrupt handelt, muss "INT" in der restlichen Zeile vorkommen, analog bei der BaseAddress "BASE". (Achtung auf die Gross-Kleinschreibung!)

Es ist darauf zu achten, dass andere "generics" in den Modulen nicht dieses Muster beinhalten.

Die "vhdl_generic" Interrupts aus der top.cfg muessen im Bereich 0–12 und die Base-Adressen im Bereich 1–64 liegen, andernfalls terminiert das Script mit einem Fehler.

Alle Modulnamen im top.cfg muessen mit einem Buchstaben ([a–z][A–Z]) beginnen.

Die Signale "clock" und "reset" des Top-Konstrukts werden standardmaessig nach aussen gefuehrt.

Namenskonventionen in den Modulen:

Die Signale des generischen Interfaces muessen in allen Modulen wie folgt benannt werden:

```
"clk "
"reset "
"AccViol "
"ExtEna "
"ExtWr "
"ExtAddr "
"Data2Ext "
"ExtIntReq "
"WrBData "
```

Die restlichen Signale der einzelnen Modul-Entities werden durch das Muster ": OUT"|" : IN" bzw. deren Lowercases gesucht.

2.3 Changelog

```
27.05.2004: core_spear: brom.hex pfadname absolut gemacht
24.05.2004: ext_Dis7Seg: modulnamen in .vhd _ent.vhd und pkg auf ext_Dis7Seg geaendert
24.05.2004: mit for fn in $(ls); do cat ${fn} |
    sed -e 's/work.core_spear.all/work.pkg_core_spear.all/' >
    ${fn}; done package namen beim core_spear geaendert,
    ACHTUNG: irgendwelche komische sonderzeichen werden
    in files eingefuegt
19.05.2004: pkg_basic beim core_spear hinzugefuegt
19.05.2004: sources files angepasst (src/)
12.05.2004: requirements.txt: erforderliche Syntax fuer Skripts wird hier beschrieben
12.05.2004: pkg_basic: wieder ins src/ verzeichnis kopiert und angepasst
12.05.2004: core_spear: pfad src/ bei dateien in sources geloescht
12.05.2004: extDis7: source file angepasst (gross-kleinschreibung)
12.05.2004: file created
```

3 Makefile

Das Makefile im workspace Verzeichnis ruft alle noetigen Scripts zur Erzeugung der Synthese und Simulation Scripts auf. Es gibt folgende Targets:

3.1 Erzeugung eines neuen Workspaces

Die Erzeugung eines neuen Projekts erfolgt mit Hilfe des Targets **.newdir** (z.B. *make myproject.newdir*).

3.1.1 Target **.newdir**

Dieses Target testet zunaest, ob die Verzeichnisstruktur des neuen Projekts bereits existiert. Ist dies nicht der Fall, wird der neue Workspace angelegt (Target **.gendir** und Target **.updatedir**).

3.1.2 Target **.gendir**

Target **.gendir** legt die Workspace- Verzeichnisstruktur an.

3.1.3 Target **.updatedir**

Dieses Target legt die komplette Verzeichnisstruktur an. Zusaetzlich werden die Konfigurations- und Tracedateien fuer die Simulation und Synthese in das jeweilige Verzeichnis kopiert. Als naechstes wird das Target **.mkbasiccfg** aufgerufen.

3.1.4 Target **.mkbasiccfg**

testet, ob die Datei *basic.cfg* bereits existiert. Ist dies nicht der Fall, wird das Script **makeBasic** aufgerufen. Der Benutzer hat nun die Moeglichkeit die verschiedenen Module auszuwaehlen (siehe 4.6)

Hat der User die Auswahl abgeschlossen, wird das Target **.mktopcfg** aufgerufen.

3.1.5 Target **.mktopcfg**

Dieses Target testet, ob die Datei *top.cfg* bereits existiert und ruft gegebenenfalls das Script **makeTopcfg** auf. Dieses Script erstellt nun die Konfigurationsdatei *top.cfg*, in der der Benutzer die Moeglichkeit hat, verschiedene Einstellungen (z.B. Veraendern der Base Adressen der Module) durchfuehren zu koennen.

Zum Abschluß werden noch die beiden Targets **.mktopscr** und **.mkautomakefile** aufgerufen.

3.1.6 Target **.mktopscr**

Hier werden die Scripts, die fuer die Synthese benoetigt werden, angelegt.

Das Generieren dieser Scripts erfolgt mithilfe des Bash- Scripts **makeTopscr** (siehe 4.9 makeTopscr).

3.1.7 Target **.mkautomakefile**

Dieses Target generiert mithilfe des Scripts **makeAutoMake** (siehe 4.5 makeAutoMake) ein Makefile, welches fuer die Simulation benoetigt wird.

3.2 Erzeugen der Top Architecture und Entity und der Testbench

Nachdem der Benutzer die Konfigurationsdatei *top.cfg* angepasst hat, kann er mittels den beiden Targets **.mktopvhdl** und **.mktoptb** die Top Architecture bzw. die Testbench erzeugen.

3.2.1 Target **.mktopvhdl**

Dieses Target erzeugt mithilfe des AWK- Script **main.awk** (siehe 4.4 main.awk) die Top Entity (Datei *top_ent.vhdl*) und die Top Architecture (Datei *top.vhdl*).

3.2.2 Target **.mktoptb**

Hier wird mithilfe des Scripts **testbench.awk** (siehe 4.18 testbench.awk) die Testbench erzeugt. Weiters wird das Bash- Script **makeConfig** (siehe 4.7 makeConfig) aufgerufen, welches die drei Configuration Files fuer die Testbench erzeugt.

3.3 Synthese

Nachdem die Top Architecture erstellt wurde (siehe 3.2.1 Target **.mktopvhdl**), kann mit verschiedenen Targets die Synthese durchgeführt werden. Mithilfe des Targets **.syn** (siehe 3.3.1 Target **.syn**) kann die Synthese bis zum Place and Route durchgeführt werden. Mit den Targets **.syn_func** (siehe 3.3.2 Target **.syn_func**) und **.syn_pre** (siehe 3.3.3 Target **.syn_pre**) kann die funktionale bzw. die prelayout Synthese gestartet werden.

3.3.1 Target **.syn**

startet die Synthese bis zum Place and Route.

Dabei werden folgende Scripts aufgerufen:

checkError (siehe 4.1 checkError)

copyHex (siehe 4.2 copyHex)

replaceFunc (siehe 4.12 replaceFunc)

replacePre (siehe 4.13 replacePre)

3.3.2 Target **.syn_func**

Dieses Target startet die funktionale Synthese.

Dabei werden folgende Scripts aufgerufen:

checkError (siehe 4.1 checkError)

replaceFunc (siehe 4.12 replaceFunc)

3.3.3 Target **.syn_pre**

startet die prelayout Synthese.

Dabei werden folgende Scripts aufgerufen:

checkError (siehe 4.1 checkError)

replacePre (siehe 4.13 replacePre)

3.4 Simulation

Nach der Erzeugung der Top Architecture (siehe 3.2.1 Target `.mktopvhdl`) und der Testbench (siehe 3.2.2 Target `.mktoptb`), können die verschiedenen Simulationen gestartet werden.

Um dies durchzuführen zu können, gibt es eine Reihe von Targets. Alle Targets, wo der Name mit `.sim` beginnt, starten zur Simulation den graphischen Simulator. Hingegen Targets, wo der Name mit `.dbx_sim` beginnt, starten den Commandline Simulator.

Weiters besteht die Möglichkeit, die VHL Dateien der Sourcen (core und Module) und die sogenannten Userpart VHDL Dateien (jene VHDL Dateien, die während des Projekts automatisch generiert worden sind z.B. `top.vhdl`, `top_func.vhdl`,...) getrennt zu analysieren.

3.4.1 Targets zu Simulation mittels graphischen Simulator

Folgende Targets starten nach der Analyse der VHDL Dateien ,den graphischen Simulator:

- **Target `.sim_beh`**
ruft `beh_core` und `beh_user` auf und startet den graphischen Simulator.
- **Target `.sim_func`**
ruft `func_core` und `func_user` auf und startet den graphischen Simulator.
- **Target `.sim_pre`**
ruft `pre_core` und `pre_user` auf und startet den graphischen Simulator.
- **Target `.sim_post`**
ruft `post_core` und `post_user` auf und startet den graphischen Simulator.

3.4.2 Targets zu Simulation mittels Commandline- Simulator

Folgende Targets starten nach der Analyse der VHDL Dateien ,den Commandline- Simulator:

- **`dbx_sim_beh`**
ruft `beh_core` und `beh_user` auf und startet den Commandline- Simulator.
- **`dbx_sim_func`**
ruft `func_core` und `func_user` auf und startet den Commandline- Simulator.
- **`dbx_sim_pre`**
ruft `pre_core` und `pre_user` auf und startet den Commandline- Simulator.
- **`dbx_sim_post`**
ruft `post_core` und `post_user` auf und startet den Commandline- Simulator.

3.4.3 Targets zur Analyse der VHDL Dateien

Mithilfe folgender Targets können sowohl die Source VHDL Dateien (Core und Module) als auch die VHDL Dateien der sogenannten Userparts (Dateien die automatisch generiert wurden z.B. `top.vhdl`, `top_func.vhdl`) analysiert werden:

- **beh_core**
behavioural Simulation des Core und der Module.
- **beh_user**
behavioural Simulation der Userparts.
- **func_core**
functional Simulation des Core und der Module.
- **func_user**
functional Simulation der Userparts.
- **pre_core**
prelayout Simulation des Core und der Module.
- **pre_user**
prelayout Simulation der Userparts.
- **post_core**
postlayout Simulation des Core und der Module.
- **post_user**
postlayout Simulation der Userparts.

3.5 **Loeschen eines vorhandenen Workspaces**

Das Loeschen des gesamten Workspaces mitsamt Verzeichnisstruktur erfolgt mithilfe des Targets **.rmdir**.

3.5.1 **Target .rmdir**

loescht Projekt.

3.5.2 **clean**

loescht work files der Simulation und Synthese.

3.5.3 **Target all**

Dieses Target zeigt an, welche Targets dieses Makefile enthaelt.

3.6 **Anzeigen der Hilfe**

3.6.1 **Target all**

Dieses Target zeigt an, welche Targets dieses Makefile enthaelt.

4 **Scripts**

4.1 **checkError**

Dieses Script prueft das dc Logfile auf Fehler und gibt diese mit dem Befehl less aus
Usage: checkError <filename>

4.2 **copyHex**

kopiert die Hex Datei des Cores in das Zielverzeichnis

Usage: copyHex <rootDirectory> <projectname>

4.3 **generic.awk**

AKW Script, welches die generics der Entityfiles extrahiert. Dabei werden die Generic Variablen mit dem Praefix 'vhdl-generic:' ausgegeben. Das Script erhaelt als Parameter die naechste freie Interruptnummer und die naechste freie Baseadresse. Diese werden als Standardwerte fuer die einzelnen Interrupts und Baseadressen vergeben. Die Interruptnummern und Base adressen, die der Cores verwendet, werden nicht mit Standardwerten versehen, da diese fix sind.

Parameter freeBase - Erste freie Baseadresse

Parameter freeInt - Erster freier Interrupt

4.4 **main.awk**

Das main.awk-Script parst die top.cfg und holt die dort konfigurierten Module, Generics und Baseadressen heraus. Weiters erstellt es die beiden Files (top.vhd, top_ent.vhd) und deren Header und nimmt verschiedene statische Eintraege vor. Es steuert weiters den Programmfluss, indem es die Unterscripts 'ise.awk' (fuer die Libraries) und parse.awk" (fuer das Mapping) aufruft.

Parameter argv[2]- rootDirectory

4.5 **makeAutoMake**

Generiert ein Makefile im Workspace des Projekts, welche Targets fuer das Ausfuehren der verschiedenen Simulationschritte enthaelt.

Usage: makeAutoMake <rootDirectory> <project>

4.6 **makeBasic**

Dieses Skript ermoeoglicht dem User die Auswahl der einzelnen Module. Dabei wird das 'src' Verzeichnis durchsucht und alle Unterverzeichnisse als Modul herangezogen (pkg_basic wird ignoriert). Nach der Auswahl der Module wird jedem ausgewaehlten Modul ein Instanzname vergeben (ermoeoglicht die Unterscheidung mehrfacher Auswahl ein und desselben Moduls). Dieser Instanzname wird zusammen mit dem Modulnamen in die Datei 'basic.cfg' (befindet sich im Workspace des Projekts) geschrieben.

Usage: makeBasic <rootDirectory> <project>

4.7 **makeConfig**

Erstellt die ConfigFiles fuer die verschiedenen Konfigurationsmodule mit Instanzname top_a und modulname top

Keine Parameter

4.8 **makeTopcfg**

Dieses Bash Script erstellt das Konfigurationsfile "Top.cfg" im Workspace des Projekts. Dabei werden der Reihe nach alle Module aus der Datei "Basic.cfg" gewaehlt. Zu jedem Modul werden als naechstes alle Generics der Entitydatei extrahiert und in "Top.cfg" eingetragen (mithilfe

des Awk Scripts *generic.awk*). Danach werden die Configs der einzelnen Module (in der Datei *modulname_i.cfg* im Verzeichnis “*cfg*” im *src*- Verzeichnis des Moduls) in “*Top.cfg*” geschrieben (mithilfe des Scripts *modulconfig.awk*). Die beiden Variablen “*freeBase*” und “*freeInt*” dienen zur Vergabe der Defaultinterrupts und Defaultbaseadressen. Interrupts werden absteigend von 11 begunned vergeben (die oberen sind durch den Core belegt). Baseadressen hingegen absteigend von 60 beginnend. Die Interrupts und Baseadressen des Core werden nicht veraendert da diese fix sind (siehe *generic.awk*)

Usage: *makeTopcfg* <projectRoot> <projectName>

4.9 **makeTopscr**

generates synthesis scripts

Usage: *makeTopscr* <projectRoot> <projectName>

4.10 **modulcfg.awk**

AWK Script, welches die einzelnen Configs der Modulkonfigurationsdateien parst. Die einzelnen Konfigurationsdaten werden mit dem Praefix “*modul_config*” ausgegeben.

Keine Parameter

4.11 **parse.awk**

Das *parse.awk* wird von *main.awk* aufgerufen und parst aus den *entity*-Files der einzelnen Module die relevanten Informationen. Diese werden je nach Art in der *top*-Architecture eingetragen (*port_map*, *generic_map*) Von *main.awk* werden folgende Parameter uebernommen:

Parameter *argv[2]*- Modulnamen

Parameter *argv[3]*- Instanznamen

Parameter *argv[4]*- Generics des jeweiligen Moduls (durch ‘-’ getrennter String) - Werte kommen aus *top.cfg*

Parameter *argv[5]*- den zu belegenden Interrupt (aus *top.cfg* uebernommen)

4.12 **replaceFunc**

Dieses Bash Skript fuehret die notwendigen Aenderungen nach der Synthese bzw. vor der Funktionalen Simulation durch. Dabei wird das VHDL File vor jeder Entiy mit den entsprechenden Zeilen fuer die Verwendung der GTECH und IEEE Library ergaenzt. Ausserdem wird bei den Rom Bausteinen das Initialisierungsfile explizit angegeben, da es bei der Synthese verloren geht. Da die Komponentennamen nur eine bestimmte Laenge haben duerfen, werden diese mithilfe des AWK Script “*trimNames.awk*“ abgeschnitten.

Weiters wird der Name der Architecture von *SYN_behaviour* auf *FUNC_behaviour* geaendert.

Usage: *replaceFunc* <root-dir> <project-name> <input file name> <output file name>

4.13 **replacePre**

Dieses Bash Skript fuehrt die notwendigen Aenderungen vor der Prelayout Simulation durch. Dabei werden im VHDL File vor jeder Entity entsprechende Zeile, welche eine Verwendung der FLEX Library ermoeöglichen, hinzugefuegt.

Ausserdem wird bei den Rom Bausteinen das Initialisierungsfile explizit angegeben, da es bei der Synthese verloren geht.

Da die Komponentennamen nur eine bestimmte Laenge haben duerfen, werden diese mithilfe des AWK Script "trimNames.awk" abgeschnitten.

Weiters wird der Name der Architecture von SYN_behaviour auf PRE_behaviour geaendert.

Usage: replacePre <root-dir> <project-name> <input file name> <output file name>

4.14 testDir

Testet, ob ein Verzeichnis vorhanden ist.

Ist es nicht vorhanden, wird eine entsprechende Fehlermeldung ausgegeben und mit dem Exitcode 1 beendet

Usage: testDir <dirName>

4.15 testDir2

Testet, ob ein Verzeichnis nicht vorhanden ist.

Ist es vorhanden, wird eine entsprechende Fehlermeldung ausgegeben und mit dem Exitcode 1 beendet

Usage: testDir2 <dirName>

4.16 testDir3

Testet, ob ein Verzeichnis nicht vorhanden ist.

Ist es nicht vorhanden, wird es erstellt

Usage: testDir3 <dirName>

4.17 testFile

Testet, ob eine Datei vorhanden ist.

Ist sie nicht vorhanden, wird eine entsprechende Fehlermeldung ausgegeben und mit dem Exitcode 1 beendet

Usage: testFile <fileName>

4.18 testbench.awk

Erstellt leere Testbench für die Entity im Entityfile

Parameter output - Destinationfile

4.19 use.awk

Das use.awk-Script wird von main.awk aufgerufen und parst aus den entity und architecture -Files der einzelnen Module alle auftretenden Libraries.

Diese werden zu einem String-Array (usevector) zusammengefasst und zum Schluss in beide top-Files geschrieben

Keine Parameter

4.20 trimNames.awk

Dieses AWK Script schneidet die Namen der Components ab. Dies ist notwendig, da bei der Synthese die Namen der Components erweitert werden. Der Simulator kann allerdings nur mit

Namen bis zu einer gewissen Laenge arbeiten.
Keine Parameter

5 Ablaufdiagramme

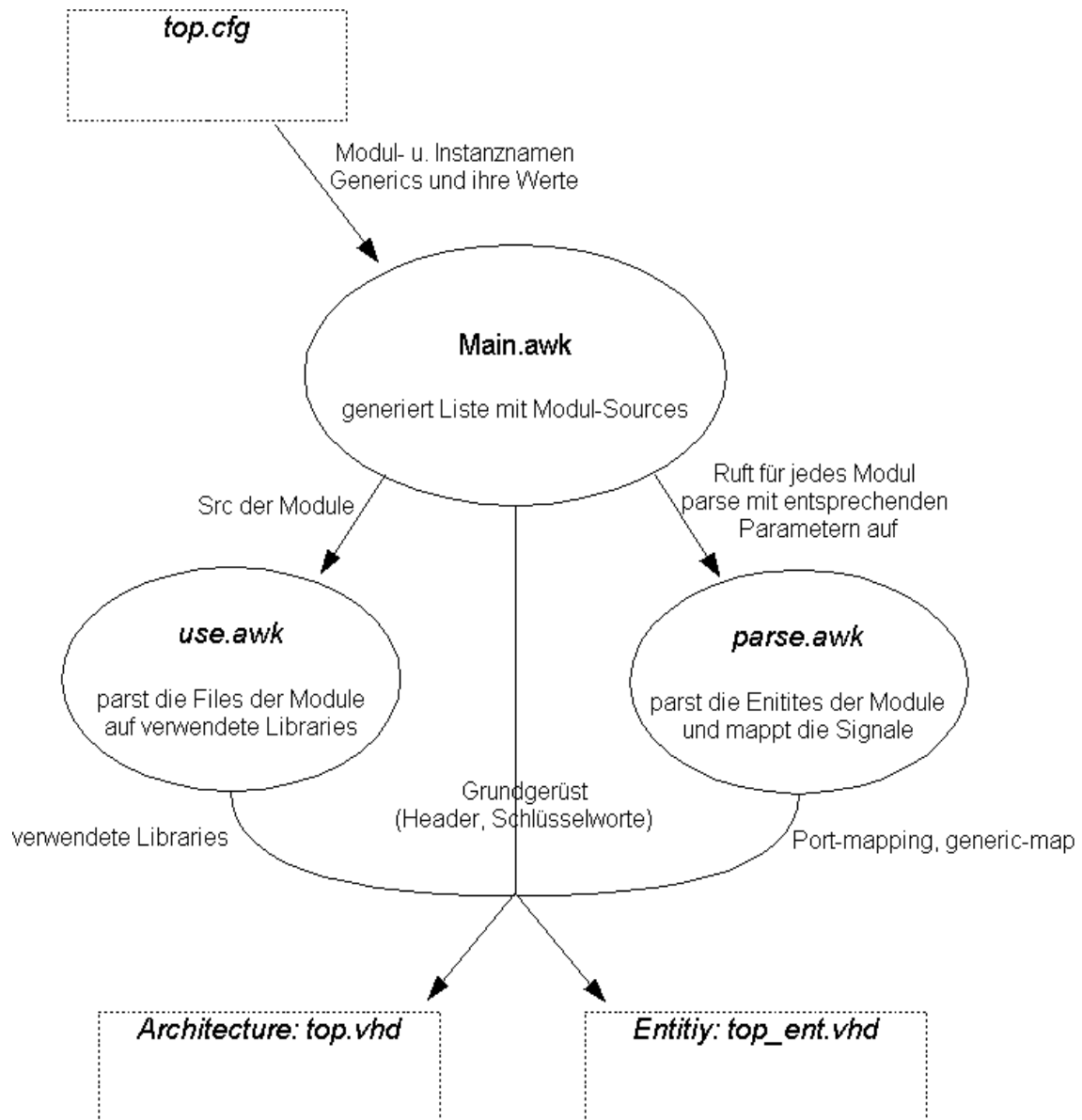


Abbildung 1: Ablauf Erstellung der Entity und Architecture

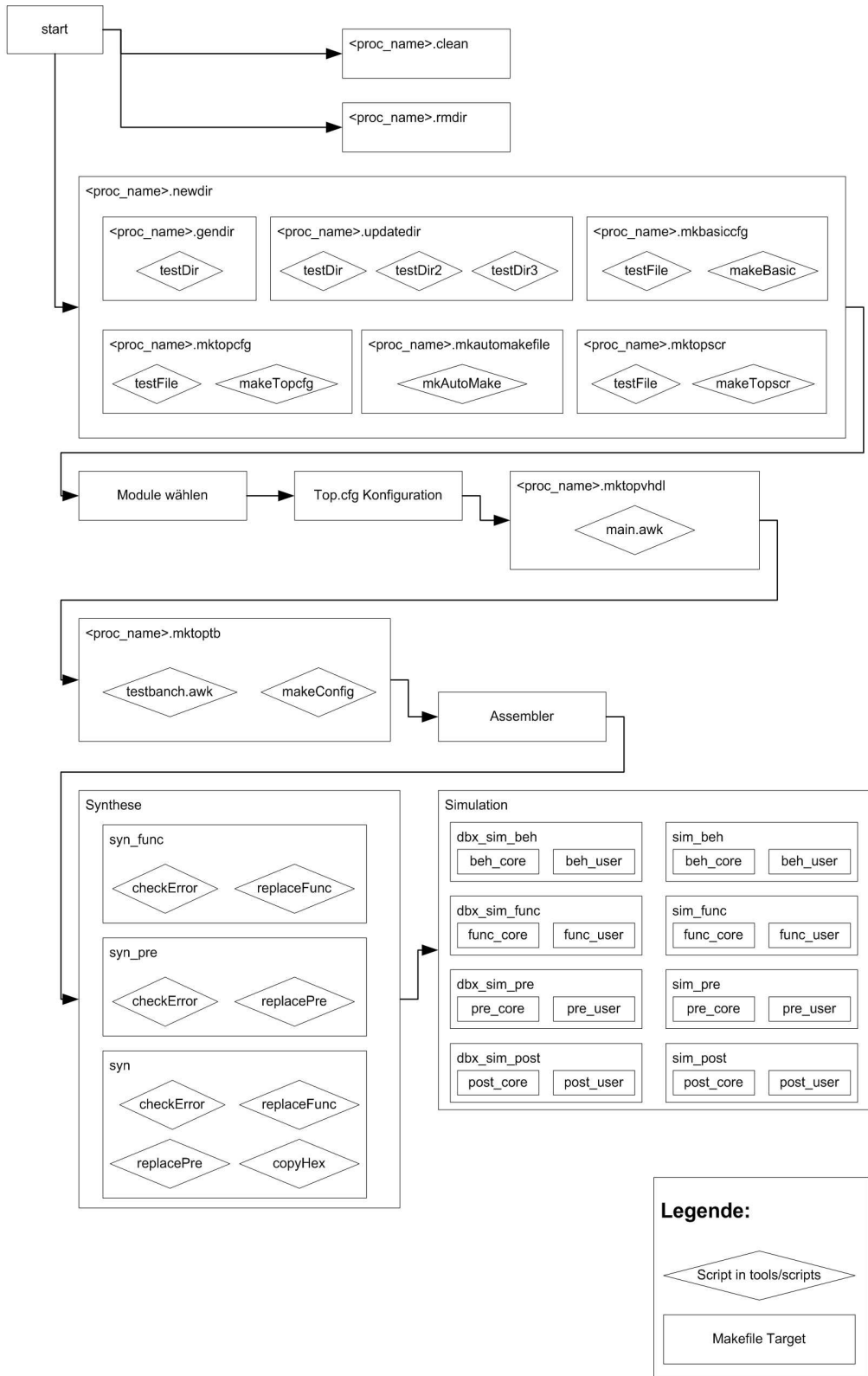


Abbildung 2: Ablauf Scriptaufrufe / Makefile