The Synthesis Program of MmAlpha
Version of July 26, 2009

Patrice Quinton

Version of July 2008*

Abstract

This documentation presents the `syn` command of MmAlpha. This command, which stands for *synthesize*, allows an Alpha program to be translated into VHDL, whenever possible, by running in a systematic way the commands necessary to transform the program into VHDL.

1 Introduction

The `syn` command allows a VHDL program to be synthesized directly from an Alpha file. In this document, we explain how this command can be used and how it is implemented. To better follow this documentation, you may lauch MmAlpha and type the command: `start[]`. A notebook will open, and in this notebook, go in the section *New: synthesis notebooks* where you will find the examples presented here.

2 Basic usage

```
syn[ "file.alpha" ]
```

in its simplest form.

However, there are several options that allow the basic method to be changed, if needed.

All functions are called with a special parameter `mute`: by default, they are silent.

*There have been modifications since this note was written.

During the execution of one of the transformation steps, if an error message is triggered, then `syn` aborts. It then writes the ALPHA program in the state that it had when the error occured.

A directory is created where various files produced during the synthesis are stored. A trace file, named `trace.txt`, contains the list of MMALPHA commands that have been executed until something wrong happened.

Finally, if everything goes well, a report file is written in this directory.

Example:

```
syn["fir.alpha"]
```

synthesizes the program contained in file `file.alpha`, and shown in Appendix **??**.

This basic command stops just before generating VHDL, as the values of the `K` and `M` parameters are not set. To obtain VHDL, run

```
syn["fir.alpha", parameterRules -> {"N" -> 20, "M" -> 100}]
```

The synthesis is almost silent, and takes a few seconds. If it succeeds, a congratulation message is issued[1].

You get more information by adding the `verbose` option:

```
syn["fir.alpha", parameterRules -> {"N" -> 20, "M" -> 100},
  verbose->True]
```

The `syn` program has a few options that can be displayed by evaluating `Options[ syn ]`.

2.1 The SYN directory

The execution creates a directory named `"firSYN"` which contains:

- `fir.report`: a file giving an analysis of the program after the `Alpha0` code generation (corresponding to the file `firAlpha0.alpha"`.
- `fir.scd`: the value of the schedule that was produced and used.
- `firAlpha0.alpha`: the program, after `Alpha0` generation.
- `firAlphard.alpha`: the program, after `Alphard` generation.
- `firParametersFixed.alpha`: the program, after parameters are fixed. This is the last transformation step before VHDLgeneration.

[1] You'll understand why when you start synthesizing you own program!

- `firPiped.alpha`: the program, after piping variables.
- `firScheduled.alpha`: the program, after scheduling and placement.
- `trace.txt`: the list of MMALPHAcommands executed.
- `VHDL`: a directory that contains the VHDL programs.

If the directory contains a file including `Wrong` in its name, then one of the steps went wrong (or maybe, this was produced during a previous execution of `syn`, as the directory is not cleaned.)

3 The `VHDL` Directory

The `VHDL` directory contains the following files, if the synthesis was successful:

- `ControlfirModule.component`: the component description of the controller part.
- `ControlfirModule.vhd`: the VHDL description of the controller part.
- `cellfirModule1.component`: the component description of the first cell.
- `cellfirModule1.vhd`: the VHDL description of the first cell.
- `cellfirModule2.component`: the component description of the second cell.
- `cellfirModule2.vhd`: the VHDL description of the first cell.
- `cellfirModule4.component`: the component description of cell number 4.
- `cellfirModule4.vhd`: the VHDL description of cell number 4.
- `definition.vhd`: a useless file.
- `fir.component`: the component description of the fir.
- `fir.vhd`: the VHDL description of the fir filter.
- `firModule.component`: the component description of the module part.
- `firModule.vhd`: the VHDL description of the module part.

The generation of VHDL follows the structure shown in Fig. 1. The main file, `fir.vhd`, may not be usable for reasons that are difficult to explain here... But VHDL file called `firModule.vhd` should be synthesizable. It calls a controller and instanciates three types of basic cells.

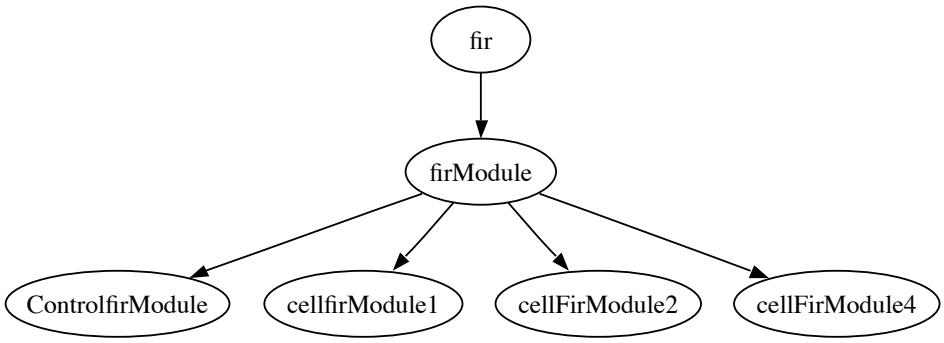

Figure 1: Structure of VHDL generated for the FIR filter

4 What `syn` does

1. Loads the specified file. If this step is successful, the *synthesis directory* is created. Its name is `xxxSyn` where `xxx` is the name of the local directory.

2. Inline program, if needed. In other words, if the loaded program contains several systems, then the last one that was loaded is inlined. If this step is successful, then only this last system is kept in the library, and it is saved under the name `yyyInlined.alpha` in the synthesis directory, where `yyy` is the name of this system.

3. Checks the program, by doing a static analysis.

4. Schedule the program. Without options, `schedule[]` is applied. Options can be provided (see Section 5) to either call `scd[]`and give hints to the scheduler. Found schedule is saved in the synthesis directory and if no schedule is found, the file is saved under the name `yyyWrongScheduled.alpha`.

5. Applies the schedule. Result is saved in file `yyyScheduled.alpha`.

6. Pipes variables using `pipeVars` command. Result is saved in file `yyyPiped.alpha` (or `yyyWrongPiped.alpha` if an error occured).

7. Generates

5 Options

`optionsOfScheduler:` contains options that are passed to the scheduler

`debug:` debug option

`verbose:`

`schedMethod:` allows the scheduler type to be specified. Values are `farkas` (by default) or otherwise (any other value), the vertex method.

`parameterRules:` specifies an association list of parameter values.

6 Report

1. Writes `Equation`

2. Gives the declaration of the lhs

3. Prints out the equation

4. Prints out the type of the lhs

5. Signals if equation is an output equation (the lhs is an output variable), or a local equation (the lhs is local)

6. Signals if we have an input equation, i.e., an equation of the form `lhs = input`[2].

7. Prints out the indexes.

8. Signals if equation is scheduled. It could be a scalar, a variable with one dimension of time, possibly with several dimensions of space.

[2]This should be extended to the case where there is an domain, and possibly an dependency.

```
system fir :   {K,M | 3<=K<=M-1}
               (x : {i | 0<=i<=M} of integer;
                w : {k | 1<=k<=K} of integer)
       returns (y : {i | K<=i<=M} of integer);
var
  Y: {i,k | K<=i<=M; 0<=k<=K} of integer;
let
  Y[i,k] =
    case
      {| k = 0 }: 0[];
      {| k > 0 }: Y[i,k-1] + w[k]*x[i-k];
    esac;
  y[i] = Y[i,K];
tel;
```

A VHDL Model Generated for the FIR Module

This is the file `firModule.vhd`.

```
-- VHDL Model Created for "system firModule"
-- 1/8/2008 10:29:33.954955
-- Alpha2Vhdl Version 0.9
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

PACKAGE TYPES IS
  TYPE HXMirr1InTYPE IS ARRAY (2 TO 2) OF  SIGNED (15 DOWNTO 0);
  TYPE xXMirr1InTYPE IS ARRAY (2 TO 2) OF  SIGNED (15 DOWNTO 0);
  TYPE ser1OutTYPE IS ARRAY (20 TO 100) OF  SIGNED (15 DOWNTO 0);
  TYPE HXMirr1TYPE IS ARRAY (2 TO 2) OF  SIGNED (15 DOWNTO 0);
  TYPE xXMirr1TYPE IS ARRAY (2 TO 2) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCx31TYPE IS ARRAY (2 TO 2) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCx3Reg2XlocTYPE IS ARRAY (3 TO 101) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCH31TYPE IS ARRAY (2 TO 2) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCH3Reg1TYPE IS ARRAY (3 TO 100) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCx32TYPE IS ARRAY (20 TO 100) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCH32TYPE IS ARRAY (20 TO 100) OF  SIGNED (15 DOWNTO 0);
  TYPE ser1Xctl1XInTYPE IS ARRAY (20 TO 100) OF  STD_LOGIC;
```

```
  TYPE ser12TYPE IS ARRAY (20 TO 100) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCx34TYPE IS ARRAY (3 TO 19) OF  SIGNED (15 DOWNTO 0);
  TYPE pipeCH34TYPE IS ARRAY (3 TO 19) OF  SIGNED (15 DOWNTO 0);
END TYPES;
USE work.types.all;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

library work;
use work.definition.all;


ENTITY firModule IS
PORT(
  clk: IN STD_LOGIC;
  CE : IN STD_LOGIC;
  Rst : IN STD_LOGIC;
  HXMirr1In : IN HXMirr1InTYPE;
  xXMirr1In : IN xXMirr1InTYPE;
  ser1Out : OUT ser1OutTYPE
);
END firModule;

ARCHITECTURE behavioural OF firModule IS
    SIGNAL ser1Xctl1 :  STD_LOGIC;
    SIGNAL HXMirr1 : HXMirr1TYPE;
    SIGNAL xXMirr1 : xXMirr1TYPE;
    SIGNAL pipeCx31 : pipeCx31TYPE;
    SIGNAL pipeCx3Reg2Xloc : pipeCx3Reg2XlocTYPE;
    SIGNAL pipeCH31 : pipeCH31TYPE;
    SIGNAL pipeCH3Reg1 : pipeCH3Reg1TYPE;
    SIGNAL pipeCx32 : pipeCx32TYPE;
    SIGNAL pipeCH32 : pipeCH32TYPE;
    SIGNAL ser1Xctl1XIn : ser1Xctl1XInTYPE;
    SIGNAL ser12 : ser12TYPE;
    SIGNAL pipeCx34 : pipeCx34TYPE;
    SIGNAL pipeCH34 : pipeCH34TYPE;
```

```
  -- Insert missing components here!
COMPONENT ControlfirModule
PORT(
  clk: IN STD_LOGIC;
  CE : IN STD_LOGIC;
  Rst : IN STD_LOGIC;
  ser1Xctl1 : OUT STD_LOGIC
);
END COMPONENT;


COMPONENT cellfirModule1
PORT(
  clk: IN STD_LOGIC;
  CE : IN STD_LOGIC;
  Rst : IN STD_LOGIC;
  HXMirr1 : IN  SIGNED (15 DOWNTO 0);
  xXMirr1 : IN  SIGNED (15 DOWNTO 0);
  pipeCx3 : OUT  SIGNED (15 DOWNTO 0);
  pipeCH3 : OUT  SIGNED (15 DOWNTO 0)
);
END COMPONENT;


COMPONENT cellfirModule2
PORT(
  clk: IN STD_LOGIC;
  CE : IN STD_LOGIC;
  Rst : IN STD_LOGIC;
  pipeCx3Reg2Xloc : IN  SIGNED (15 DOWNTO 0);
  pipeCH3Reg1 : IN  SIGNED (15 DOWNTO 0);
  ser1Xctl1XIn : IN  STD_LOGIC;
  pipeCx3 : OUT  SIGNED (15 DOWNTO 0);
  pipeCH3 : OUT  SIGNED (15 DOWNTO 0);
  ser1 : OUT  SIGNED (15 DOWNTO 0)
);
END COMPONENT;
```

```
COMPONENT cellfirModule4
PORT(
  clk: IN STD_LOGIC;
  CE : IN STD_LOGIC;
  Rst : IN STD_LOGIC;
  pipeCx3Reg2Xloc : IN  SIGNED (15 DOWNTO 0);
  pipeCH3Reg1 : IN  SIGNED (15 DOWNTO 0);
  pipeCx3 : OUT  SIGNED (15 DOWNTO 0);
  pipeCH3 : OUT  SIGNED (15 DOWNTO 0)
);
END COMPONENT;

BEGIN

  HXMirr1(2) <= HXMirr1In(2);

  pipeCH3Reg1(3) <= pipeCH31(2);

  G1 : FOR p IN 21 TO 100 GENERATE
    pipeCH3Reg1(p) <= pipeCH32(-1 + p);
  END GENERATE;

  G2 : FOR p IN 4 TO 20 GENERATE
    pipeCH3Reg1(p) <= pipeCH34(-1 + p);
  END GENERATE;

  pipeCx3Reg2Xloc(3) <= pipeCx31(2);

  G3 : FOR p IN 21 TO 101 GENERATE
    pipeCx3Reg2Xloc(p) <= pipeCx32(-1 + p);
  END GENERATE;

  G4 : FOR p IN 4 TO 20 GENERATE
    pipeCx3Reg2Xloc(p) <= pipeCx34(-1 + p);
  END GENERATE;

  G5 : FOR p IN 20 TO 100 GENERATE
    ser1Out(p) <= ser12(p);
  END GENERATE;
```

```
  G6 : FOR p IN 20 TO 100 GENERATE
    ser1Xctl1XIn(p) <= ser1Xctl1;
  END GENERATE;

  xXMirr1(2) <= xXMirr1In(2);

  G7 : ControlfirModule PORT MAP (clk, CE, Rst, ser1Xctl1);

    G8 : cellfirModule1 PORT MAP (clk, CE, Rst, HXMirr1(2), xXMirr1(2), pipeCx31(2),

  G9 : FOR p IN 20 TO 100 GENERATE
    G10 : cellfirModule2 PORT MAP (clk, CE, Rst, pipeCx3Reg2Xloc(p), pipeCH3Reg1(p),
  END GENERATE;

  G11 : FOR p IN 3 TO 19 GENERATE
    G12 : cellfirModule4 PORT MAP (clk, CE, Rst, pipeCx3Reg2Xloc(p), pipeCH3Reg1(p),
  END GENERATE;
END BEHAVIOURAL;
```

B VHDL Model Generated for the FIR cell 2

This is the file `cellfirModule2.vhd`.

```
-- VHDL Model Created for "system cellfirModule2"
-- 1/8/2008 10:29:33.067807
-- Alpha2Vhdl Version 0.9

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.numeric_std.all;

library work;
use work.definition.all;


ENTITY cellfirModule2 IS
PORT(
```

```
  clk: IN STD_LOGIC;
  CE : IN STD_LOGIC;
  Rst : IN STD_LOGIC;
  pipeCx3Reg2Xloc : IN  SIGNED (15 DOWNTO 0);
  pipeCH3Reg1 : IN  SIGNED (15 DOWNTO 0);
  ser1Xctl1XIn : IN  STD_LOGIC;
  pipeCx3 : OUT  SIGNED (15 DOWNTO 0);
  pipeCH3 : OUT  SIGNED (15 DOWNTO 0);
  ser1 : OUT  SIGNED (15 DOWNTO 0)
);
END cellfirModule2;

ARCHITECTURE behavioural OF cellfirModule2 IS
    SIGNAL ser1loc3 :  SIGNED (15 DOWNTO 0) := "0000000000000000";
    SIGNAL pipeCH3loc2 :  SIGNED (15 DOWNTO 0) := "0000000000000000";
    SIGNAL pipeCx3loc1 :  SIGNED (15 DOWNTO 0) := "0000000000000000";
    SIGNAL pipeCx3Reg2 :  SIGNED (15 DOWNTO 0) := "0000000000000000";
    SIGNAL ser1Reg3 :  SIGNED (15 DOWNTO 0) := "0000000000000000";
    SIGNAL TSep1 :  SIGNED (15 DOWNTO 0) := "0000000000000000";
    SIGNAL TSep2 :  SIGNED (15 DOWNTO 0) := "0000000000000000";


  -- Insert missing components here!---------
BEGIN

    ser1 <= ser1loc3;

    pipeCH3 <= pipeCH3loc2;

    pipeCx3 <= pipeCx3loc1;

    PROCESS(clk) BEGIN IF (clk = '1' AND clk'EVENT) THEN
      IF CE='1' THEN pipeCx3Reg2 <= pipeCx3Reg2Xloc; END IF;
                END IF;
    END PROCESS;

    PROCESS(clk) BEGIN IF (clk = '1' AND clk'EVENT) THEN
      IF CE='1' THEN ser1Reg3 <= ser1loc3; END IF;
                END IF;
    END PROCESS;
```

```
    pipeCx3loc1 <= pipeCx3Reg2;

    pipeCH3loc2 <= pipeCH3Reg1;

    TSep1 <= (pipeCH3loc2 * pipeCx3loc1);

    TSep2 <= (ser1Reg3 + TSep1);

    ser1loc3 <= "0000000000000000" WHEN ser1Xctl1XIn = '1' ELSE TSep2;

END behavioural;
```