Periodic System Generation in MMALPHA

Anne-Marie Chana Patrice Quinton

Version of February 20, 2009

Contents

To do list

- Separate the types in a separate component
- Assign values to the coefficients of the filter
- Infrer le type de ovsfaddress etc... partir de celui de la ROM

1 Introduction

This document presents the current state of development of an extension of MMALPHA to generate multi-dimensional periodic systems. The theory underlying this work is described in [**?**].

2 The `synPeriodic` function

This function is located in the `Synthesis.m` package. Ils goal is to analyze a data-flow periodic systems and to find out the periods of each one of its components. It also generates VHDL files for such a system.

Recall that such a system is a special case of ALPHA system where all variables are indexed with an infinite first dimension. This means that all variables of such a system have a domain of the form $\{i, j, k...|0 \leq i, ...\}$. The first index, i, is considered as a time index. It also uses subsystems, which have all the property that all their variables are also indexed with a first infinite dimension.

Now the time dimension of each variable may be *stretched* or *contracted* depending on a fixed period. This means that for some variables, the first index i is in fact an affine function $at + b$ of an underlying, commun time clock, t. For another variable, this affine function might be of the form $ct+d$, where c and d are different from a and b.

Some particular subsystems are oversamplers and undersamplers. They have the property of changing the rate of the time dimension, i.e., the values of the periods a or c of the variables.

All other subsystems have the property of being *monochronous*: all their variables have the same period. We expect these subsystems to represent hardware components that may be run at various periods, depending on the way they are instanciated (we shall return to this point later on.)

In [**?**], it is shown how one can compute the value of the periods of each individual signal or component of such a periodic system, and from there, if a solution to this problem exists, to schedule in detail the full system.

The `synPeriodic` function analyzes the graph of the MMALPHA data-flow system, and from its dependencies, it groups signals that have the same period (returned as list of list). Each one of the groups has un period, say λ_i, which is calculated by this function. It then schedules the full system, and generates the VHDL components of it. We describe hereafter how this is done.

3 Generation of VHDL Programs

3.1 Generalities

The generation of VHDL programs is implemented as the `synPeriodic` MMALPHA function of the `Synthesis.m` package.

The parameters of this function are not fully determined yet.

```
synPeriodic[]
```

generates a VHDL file in the directory `$MMALPHA/VHDL` and provides some other results.

This function makes use of the `genVhdl` function, included in the `Vhdl2.m` package, and that we describe in appendix A.

Its operations are currently as follows:

Step 1: Built the graph of periods, and solve the balance equations of this graph.

Step 2: Schedule the whole system, by injecting the value of the periods in the schedule patterns of each component.

Step 3: Generate the delayed enable signals.

Step 4: Generate the enable signals.

Step 5: Generate the FSM that controls the periodic system.

Step 6: Generate the VHDL code for the called systems.

Step 7: Generate the VHDL code for the calling system.

We consider successively each one of these steps.

3.2 How to Write a Periodic System

Recall that all signals that are manipulated by periodic systems have a domain whose first index is an infinite half-line. The other indexe can be anything (provided this matches the type of the subsystems).

A periodic system is made out of either:

- Connexions between periodic signals.

  ```
  A = B;
  ```

- Combinatorial elements.

  ```
  A = B + (C * D);
  ```

- Over-samplers.

  ```
  use overSampling[6] (A) returns (B);
  ```

  The number inside brackets is the over-sampling factor. It is a fixed integer. Variable `A` is the input, and variable `B` is the output.

- Under-samplers.

  ```
  use underSampling[6] (A) returns (B);
  ```

  Same remark as for the over-samplers.

- Calls to monochronous subsystems, among which, ROM and address generators.

  ```
  use subsystem[ pm ] (A, B, etc.) returns (X, Y, etc.)
  ```

3.3 Step 1: Solve the Balance Equations

Each variable and each component are assigned a period which is computed by solving the so-called *balance equations*. The rules that are applied are the following ones:

- Connected signals have the same period.
- Inputs and outputs of a subsystem have the same period (subsystems are assumed to be monochronous).
- The period of the output of an over-sampler is equal to `x` times that of its input, where `x` is the sampling factor.
- The opposite is true for an under-sampler.

The algorithm consists in finding out the connected components of the dependence graph (whose elements have necessarily the same period), and solve the balance equations. This returns one or several solutions. Currently, only the case of one single solution is handled properly.

3.4 Step 2: Schedule the System

We assume that each subsystem has a parameterized schedule, which is put in the schedule library. Building a parameterized schedule is explained in Appendix **??**.

Then, step 2 consists in scheduling the whole system.

A The `genVhdl` Function

The `genVhdl` functions allows one to generate component parts and architecture parts of several elements needed for the generation of periodic systems. Currently, this function allows the generation of:

- read-only memories,
- finite-state machines,
- periodic enable signals.

The principle of the generation is in all cases the same: it is made from `vhdl` pattern files available in `$MMALPHA/VHDL`. Place-holder are filled by the parameters given to `genVhdl`, depending on the type of VHDL program to be generated. Place-holders are character strings of the form `$string$`.

B Read-only Memories

Although this VHDL element is not specific to periodic systems, we describe here the use of `genVhdl` to generate a read-only memory.

```
genVhdl["ROM" , "$wordLength$" -> "5",
            "$name$" -> "kasami", "$size$" -> "7",
            "$comment$" -> "Memory for Kasami coefficients",
            "$values$" -> "{1,1,0,0,1,8,2}"];
```

The first parameter, `"ROM"`, identifies the type of block generated. Other parameters are given as string replacement rules. The second parameter, gives the number of bits of the words of the memory (it is mandatory). The third (mandatory) parameter is the name given to the element. The fourth (mandatory) parameter is the number of words of the memory. A comment place-holder allows one to add a comment to the produced program. Finally, the last parameter is the values assigned to each word of the memory. Its number should be equal to the size. There exist another place-holder in the

ROM, namely, to define the address size, but it is computed automatically by the generator.

String rewriting rules can be given in any order.

This command, when executed, returns a list of 2 strings: the first one is the component corresponding to the ROM, and the second one is its architecture.

```
COMPONENT kasami IS
  PORT
  (
    address : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    data : OUT STD_LOGIC_VECTOR(5 DOWNTO 0)
  );
END COMPONENT;
```

Figure 1: Component of a ROM, as generated by the `genVhdl` command of B.

Figure 1 gives an example of memory generated.

C Periodic enable signals

In periodic systems, each component is driven by a `clock-enable` signal that controls its execution. The `genVhdl` function allows one to produce a VHDLcomponent that generates a periodic signal that can be used to control such components.

```
genVhdl[ "PeriodicEnable" ,  "$name$" -> "genEnable10",
     "$period$" -> "10",
     "$comment$" -> "Periodic enable generator with period 7"];
```

The first component identifies the type of block produced. The second one, gives the name of the generated component. The third one gives the period of the corresponding clock. Finally, the last one gives a comment.

Fig. 2 shows an example of component for a 10 period enable. Notice that this component has 3 inputs: the basic clock of the system, `clk`, a reset signal `rst`, and a *global enable* signal, `ceGen`. This signal, when false, has the effect of *freezing* the generation of the periodic enable during at least one `clk` clock cycle.

```
--
-- Component for a periodic enable of period 10
--

COMPONENT genEnable10 IS
  PORT(
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    ceGen : IN STD_LOGIC;
    periodicGe : OUT STD_LOGIC
  );
END COMPONENT;
```

Figure 2: Component of a periodic enable signal generator, as produced by the `genVhdl` command in C.

D Finite-state machine

The third element that the `genVhdl` function allows one to generate is a finite-state machine. Fig. 3 displays the form of the `genVhdl` command to produce a finite-state machine. The first three parameters are position parameters: the first one identifies the type of block generated, the second one gives the list (of strings) of the outputs of the fsm. The type of these outputs are assumed to be `STD_LOGIC`. The third parameter describes the fsm. It is a list, the element of which are made of an integer and a string. The integer gives the time when a given action happens, and the string presents this action, in `vhdl` code. Usually, this action involves setting values to the outputs.

The first element of this list should always be { `0, "action"` }, as the initial state of the fsm is always named `state0`.

Fig. 4 displays the component generated for a finite-state machine. Notice that this fsm is controlled by a reset signal `rst` and a general clock-enable signal `CE_gen`.

E The FIR Filter

The filter was generated using MMALPHA. It consists of a controller, a module that instanciates three types of cells.

```
genVhdl[ "Fsm" ,
     {"fifo1_Out","fifo2_Out"},
     {
      {0,"BEGIN fifo1_Out <= '0'; fifo2_Out <= '1' END"},
      {4,"BEGIN fifo1_Out <= '1'; fifo2_Out <= '0' END"},
      {7,"BEGIN fifo1_Out <= '0'; fifo2_Out <= '1' END"},
      {22,"BEGIN fifo1_Out <= '1'; fifo2_Out <= '0' END"}
     },
     "$name$" -> "myFsm",
     "$comment$" -> "This is a Fsm"];
```

Figure 3: Command for generating a finite-state machine.

E.1 The Module

It has 6 parameters: the clock (`clk`), the clock-enable signal (`CE`), the reset signal (`Rst`), the coefficients (`wXMirr1In`), the x input (`xXMirr1In`) and the outputs (`Yout`).

Note: separate the package from the remaining of the filter. Insert a return before the components...

The Module calls the controller, `ControlfirModule`. This controller has the same signals (clock, clock-enable and reset) as the Module, and it returns a control signal called `pipeCw7ctl1PXInit`.

It calls three types of cells. The simplest one is `cellfirModule4`, which does almost nothing: initialize the value of `Y` – bounded to `Y4(0)` in the call – to 0, and transmits the value of X (in the output signal `pipeCx5`, bounded to `pipeCx54(0)` in the call.

The second cell is `cellfirModule1`. Besides the control signals (clock, clock-enable and reset), it has four input signals:

- the w coefficient (`wXMirr1`, binded to `wXMirr1(1)`),
- the X input (`pipeCx5Reg3Xloc`), binded to `pipeCx5Reg3Xloc(1)`),
- the Y input (`YReg5Xloc`) binded to `YReg5Xloc(1)`),
- and the pipe signal (`pipeCw7Xctl1PXInitXIn` binded to `pipeCw7Xctl1PXInitXIn(1)`).

It outputs similar signals:

- the w coefficient (`pipeCw7Xctl1P`, binded to `pipeCw7Xctl1P1(1)`),

```
--
-- This is a Fsm
--
COMPONENT myFsm IS
  PORT(
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    CE_gen : IN STD_LOGIC;
    fifo1_Out : OUT STD_LOGIC;
    fifo2_Out : OUT STD_LOGIC
  );
END myFsm;
```

Figure 4: Component of a periodic enable signal generator, as produced by the `genVhdl` command in Fig. 3.

- the X output (`pipeCx5`), binded to `pipeCx51(1)`),
- the pipe output signal (`pipeIOCw9` binded to `pipeIOCw91(1)`).
- and the Y output (`Y`) binded to `Y1(1)`),

The second cell is `cellfirModule3`. Besides the control signals (clock, clock-enable and reset), it has four input signals:

- the w coefficient (`pipeCw7Xctl1PReg2Xloc`, binded to `pipeCw7Xctl1PReg2Xloc(p)`),
- the X input (`pipeCx5Reg3Xloc`), binded to `pipeCx5Reg3Xloc(1)`),
- the Y input (`YReg5Xloc`) binded to `YReg5Xloc(p)`),
- and the pipe signal (`pipeIOCw9Reg4Xloc` binded to `pipeIOCw9Reg4Xloc(p)`).

It outputs similar signals:

- the w coefficient (`pipeCw7Xctl1P`, binded to `pipeCw7Xctl1P1(1)`),
- the X output (`pipeCx5`), binded to `pipeCx53(p)`),
- and the pipe output signal (`pipeIOCw9` binded to `pipeIOCw93(p)`), and
- the Y output (`Y`) binded to `Y3(p)`).

E.2 The Controller

At reset time (reset is active at value 0), the controller initializes a counter to value -1, and enters state `initState`. It remains in this state until the counter reaches the value 34. During the first 35 cycles, the output of the pipe signal is 0. It gets the value 1 during cycle 36 (when the counter reaches the value 34).