

Synthesis of Object Oriented Models on Reconfigurable Hardware

Giovanni Agosta, Francesco Bruschi, Marco Santambrogio, Donatella Sciuto
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32 – 20133 Milano, Italy
agosta, bruschi, santambr, sciuto@elet.polimi.it

Abstract—In this work the problem of modeling reconfigurable systems behavior with a precise, executable semantics is considered. The possibility of synthesising such models onto reconfigurable technologies is also faced. The modeling problem is tackled with the definition of a formalism based on the possibility of Java to vary its class pool at run-time. Feasibility of the synthesis approach is verified with the implementation of a reconfigurable component prototype on a Xilinx FPGA, and reconfiguration time and area figures are measured and discussed.

I. INTRODUCTION

Reconfigurable circuits such as FPGA are nowadays a widespread technology in embedded system implementations. Among the reasons for their adoption there are flexibility and reduced non-recurrent engineering costs; an interesting, but so far little exploited possibility would be the implementation of systems that change computational resources at run time. Among the problems in exploiting such a possibility there is the absence of a formalism that provides the possibility of modeling the reconfigurable behavior in executable system representations, early in the design process.

One of the aims of this work is to investigate object oriented language features in the development of digital system models with dynamically reconfigurable computational resources.

To reach this aim, we extended the approach to the high level modeling of reconfigurability presented in [1], that exploits the possibility, offered by some modern programming languages, to describe a system with the object oriented concepts that have found widespread application in the software design industry and large consensus in the community of software engineering researchers. To this end, we show how reconfigurability can be naturally introduced by the native class loading mechanisms of Java.

After having formally defined the reconfigurability problem, we apply the data oriented reconfigurability model (DORM) to the problem of modeling a reconfigurable differential data stream decoder. To obtain the DORM model we first define classes that represent the data packets forming the stream. The data packet types set is thus represented as a class hierarchy, and it can be expanded by simply adding new classes to the current pool.

The following step is the definition of a general model that captures the main features of a reconfigurable hardware

architecture. To this end, we adopted RADL, the abstraction defined in [2] at the transaction level of abstraction.

The problem of mapping DORM models onto the reconfigurable architecture description layer (RADL) is then considered with the definition of an ad-hoc methodology, that is tested upon an implementation of an internally reconfigurable cell.

II. ISSUES IN RECONFIGURABLE SYSTEMS SYNTHESIS

In [3], synthesis is considered for descriptions that contain class hierarchies with a root class R . Each object instanced from a class C in the hierarchy has its own state space S , that is composed by the Cartesian product of the state spaces of all the properties of the class. Moreover, two objects can have the same state space but belong to different classes; the complete notion of state of an object must thus include the information on the class type which it belongs to. We model the effect of the invocation of a method on an object of a particular class taking into account two effects:

- the production of a result that is function of both the object state and of the parameters passed;
- the alteration of the state of the object.

According to this model of method invocation, an instance of a given class can be mapped to a set of reconfigurable RADL cells in this way:

- the *memory elements* store the object's state;
- for each class method redefinition, a behavior is stored in the *behavior table*.

When a behavior is executed upon a unit, it can both modify the information content of the memory of the cell and perform the operations needed to produce the result.

For the sake of simplicity, we will consider a root DORM class R interface composed of a single method *action*. Stated this, in order to map the DORM representation to the RADL structure, the following steps must be performed:

- 1) for each class in the DORM hierarchy, a unique identifier *TypeId* is generated ;
- 2) for each class in the hierarchy, if *action* is redefined, generate a corresponding function and store it in the behavior table; as *OPID*, use the *TypeId* of the class;

- 3) for each static data member of the root class R , generate a memory element in the reconfigurable cell.

Figure 1 shows how the DORM to RADL translation can be formalized by means of a BNF grammar with semantic actions.

```

data_token_class :
  CLASS class_id ':' IMPLEMENTS data_token_interface '{' dt_class_definition
  '}',
  {
    $$ = new radl_token();
    $$token_type = hash($2);
    $$token_data = $7.data;
    global_behavior_list.append($2,$7.behavior)
  };

dt_class_definition :
  dt_attribute ':' dt_class_definition
  {$$ = $3.data.append($1);} |
  dt_method ':' dt_class_definition
  {$$ = $3.behavior.append($1);} |
  /* empty */
  {$$ = new radl_data_info(); };

dt_attribute :
  type_id attribute_id
  {
    $$ = new radl_data_field();
    $$type = $1;
    $$id = $2;
  };

dt_method :
  type_id method_id '(' parameter_list ')' '{' dt_method_implementation '}'
  {
    $$ = new radl_behavior();
    $$RADL_implementation = java2RADL($1, $2, $4, $7);
  };

```

Fig. 1. DORM to RADL translation formalized by means of a BNF grammar with semantic actions

III. RADL CELL SYNTHESIS EXPERIMENTS

One of the main issues we need to address in order to prove the validity of our modeling approach is the feasibility of the implementation of a RADL cell on an existing physical architecture. To address this concern we implemented and tested a prototype of a single cell reconfigurable system implementing the RADL model.

The target architecture chosen for the prototype implementation was a Xilinx FPGA XC2VP7. The *frame* is the smallest reconfigurable unit of the device: a single frame is 424 bytes (or 3,392 bits) long, and a total of 1320 frames is available.

In our prototype, the dynamic reconfiguration relies on an internal reprogramming schema: the reconfigurator is implemented on an embedded processor that receives the information on the current packet type and, if reconfiguration is needed, reconfigures one of the functional units of the array.

The functionalities of the pool were then synthesized and formatted according to the difference bitstream format. The prototype cell has four reconfigurable functional units. We used nine different functionalities that range from adder array up to a FIR filter.

Table I shows the results of the experimental evaluation in terms of reconfiguration times and throughput, calculated on bitstream size. It also provides information on the number of

TABLE I
EXPERIMENTAL DATA: RECONFIGURATION TIMES FOR THE RADL
IMPLEMENTATION

Test Func.	Reconf. frames	Bitstream size (Byte)	Configuration time (msec)	Throughput (MByte/sec)
TF0	33	24k	15.509	1.487
TF1	46	40k	25.674	1.492
TF2	50	44k	28.628	1.496
TF3	60	51k	32.700	1.497
TF4	68	53k	34.203	1.495
TF5	88	67k	42.814	1.495
TF6	104	60k	38.598	1.459
TF7	132	100k	64.140	1.496
TF8	148	94k	60.045	1.494
TF9	182	119k	76.049	1.496

FPGA frames used by each Reconfigurable Module, since the process requires that complete frames are reconfigured.

Times for the internal configuration case have been obtained directly from the ICAP module and include only the real copy and configuration time, not the additional overheads for kernel system calls management.

The throughput of the reconfiguration operations, as it can be seen, is slightly less than 1.5 MByte/sec. Similar tests, performed on the Evaluation Board with the processor running at 100 MHz, have shown a throughput of more than 3 MByte/sec. The difference in these performance figures is due to the overhead introduced by the operating system: the reconfiguration process does not involve interrupt or DMA mechanisms for transfer as the hardware ICAP component does not support it and it is thus bound to the execution speed of the software code.

The overhead should be anyway considered acceptable as the kernel module provides a convenient way to access the reconfiguration controller, trading some of the system performance with the possibility to make it available to the processes running in the OS.

IV. FUTURE WORKS

The work will continue in different directions: application of the design methodology to other potentially interesting scenarios, with special regard to mobile code applications for embedded systems; implementation of a tool for the mapping of the dynamic class loading upon reconfigurable devices; analysis of the application of other advanced programming language features, among which aspect oriented programming [4], to the refinement of the high level models we defined.

REFERENCES

- [1] G. Agosta, F. Bruschi, M. Santambrogio, and D. Sciuto. A Data Oriented Approach to the Design of Reconfigurable Stream Decoders. In *IEEE 2005 3rd Workshop on Embedded Systems for Real-Time Multimedia*, Sep 2005.
- [2] *System C Version 2.0 Beta-1 User's Guide*, 2001. <http://www.systemc.org>.
- [3] Martin Radetzki. *Synthesis of Digital Circuits from Object-Oriented Specifications*. PhD thesis, 2000.
- [4] G. Agosta, F. Bruschi, and D. Sciuto. Aspect Orientation in System Level Design. In *Forum on specification and Design Languages*, Sep 2005.