An Integrated toolchain for Overlay-centric System-on-chip

Jean-Christophe Le Lann and Théotime Bollengier and Mohamad Najem and Loïc Lagadec
ENSTA Bretagne and Lab-STICC UMR 6285
Email: firstname.lastname@ensta-bretagne.fr

***Abstract*—The overlay approach –FPGA over FPGA– has a number of expected benefits, including bitstream compatibility between different vendors and parts, fast reconfiguration and, more generally, ease of use. However the number of complex engineering tasks to design, explore and make use of such overlays severely restrains their diffusion. This paper presents a downloadable integrated tool flow named Argen. Argen supports defining reconfigurable architecture and generating the corresponding overlay along with its System-on-chip exploitation environment.**

***Index Terms*—FPGA, prototyping, software environment, overlay**

I. Introduction

When considering general purpose hardware acceleration, the potential users are mostly software programmers. Those, when addressing FPGA design, often find themselves battling with the much lower design productivity in developing hardware designs [1]. Hence, there is a need for abstracting the complexity away from the designer, enabling architectures to be exploited as programmable accelerators, alongside general purpose processors, within a software-centric runtime framework.

Existing overlay architectures offer a number of advantages for hardware acceleration, among which application portability and fast compilation. Yet, not only is the architecture flexible, supporting tailoring the architecture towards a given application field or a programming model, but also the bitstream's definition is made public. On one hand, this comes at the expense of area and performance overheads due to limited consideration for the underlying FPGA architecture. On the other hand, there are many potential optimizations and thus many research fields that overlays make possible to explore: impact of architecture against metrics, security mechanisms, etc.

Obviously, this requires a complete environment, so that researchers can serialize topics on which to focus one by one. This environment must support defining the architecture, synthesizing application, scoring metrics, offering advanced debug facilities, and potentially deploying the overlays to create a topology of reconfigurable nodes.

This paper introduces such an environmentt, which is currently downloadable as a virtual machine image through the provided link [2].

The rest of this paper is organized as follows: section II reviews existing overlays and their toolset, section III itemizes the requirements that guided the tools development, Sections IV and V respectively present the resulting tool flow for overlay modelling and generation, and for application synthesis. Section VI exposes some use cases, and section VI summarizes some learned lessons and draws perspectives.

II. Literature overview and Problem statement

As FPGA themselves are becoming mainstream, overlays have quickly become a field of research in their own rights. In the field of reconfigurable systems however, the notion of overlay has been around for several years (sometimes called *virtual FPGAs* [3]). Overlays' main intent is to bring more productivity to application developers by providing a new layer on top of physical FPGAs.

Technically, they can be viewed as programmable and mostly regular architectures, either fine or coarse grained. An immediate analogy can be drawn with the adoption of virtual machines like JVM in the software domain : instead of using the physical processor, programmers rely on a software-defined processor to develop and deploy their applications. By using VMs, programmers have made the explicit choice to trade strict performance for flexibility and binary compatibility. Overlays also offer a mean to apply a separation of concerns, by tuning the architecture to domain specific problems.

A. *Fine grained overlays*

Among some emblematic overlays found in the literature is ZUMA [4]. ZUMA can be considered as a virtual FPGA, as it offers the same granularity as classical FPGAs, for operations at the bit level. Brant & Lemieux have optimized ZUMA so that virtual LUTs can be directly mapped on physical ones. Accordingly, many more optimizations can be studied. As an example, authors in [5] have introduced the notion of Virtual Time Propagation Registers (VTPRs) that optimize the synthesis of the overlay itself, by conforming to a strictly synchronous RTL model of the overlay. Despite their evident drawbacks in terms of performance loss versus the host FPGA, fine-grained overlays offer a playground to study overlay architectures under a microscope. The template architecture used in Argen will be further described in section IV-A.

B. *Coarse grained overlays*

Coarse grained overlays have attracted even more attention, due to their higher computational performances. Design covers a wide range of architectures, starting from simple word-level operators to assembly of software cores. The literature also exposes many types of interconnexions, either point-to-point connexions, simply restricted to a close neighbourhood or bus-based. The pairing of general purpose processors and coarse-grained overlays makes them an ideal coprocessor, either tighly or loosely coupled [6], or part of their pipeline [7].

Jain [8] proposed an interesting categorization of coarse-grained FPGA overlays: the first distinction separates spatially-configured and timed-multiplexed architectures. Then in each category, we may further distinguish overlays presenting nearest-neighbor style routing mechanisms (e.g CGRAs) vs island style (e.g Intermediate Fabrics [9], Dyser [10], etc.) or even customized routing topologies.

Coarse grained overlays offer much more interesting performances than their fine grain counterparts. A typical reported overhead in the literature in terms of clock cycles and surface (with respect to a native synthesis on the host FPGA) amounts to respectively only 7% and 44% (case of Intermediate Fabrics [9]).

As in the case of fine-grained overlays, the source of optimizations naturally comes from a synergetic design of the overlay and a cautious use of host resources. For instance, Jain et al. [11] have designed the functional units of their overlay by reusing the host DSP blocks (Xilinx DSP48E1). A different strategy has been reported by Coole and Stitt, called "supernets" (netlist superset), which aims at minimizing the overhead in terms of surface. The strategy consists in finding shareable parts of various application netlists, in order to fuse them in the overlay. A secondary interconnect network is then added to the overlay in order to bring more flexibility to the whole architecture. Even more generally, Fahmy [12] addresses the typical overhead encountered in island-style interconnect by a close look at applications, where arithmetic expressions naturally lead to direct-acyclic graphs: cone-shaped cluster of FUs utilizing a simple linear interconnect between them suffices without substantial lost of programmability, but with huge gains in terms of surface of the overlay.

C. *Tooling associated to overlays*

While a great amount of research has been conducted so far in the exploration of such architectures, the EDA tooling associated to overlays seems to lag quite behind. Most of the reported experiments have concentrated on mapping dataflow graphs on overlays, or on runtime management. Instead, papers barely report the level of usability of the tools associated to these experiments.

III. Requirements for overlay-centric integrated tooling

At least two branches of the underlying activities must be addressed : the overlay building (modelling and generation) and the synthesis of applications onto the overlay.

However, to offer a true one-stop-shopping point environment, ta number of requirements has to be satisfied:

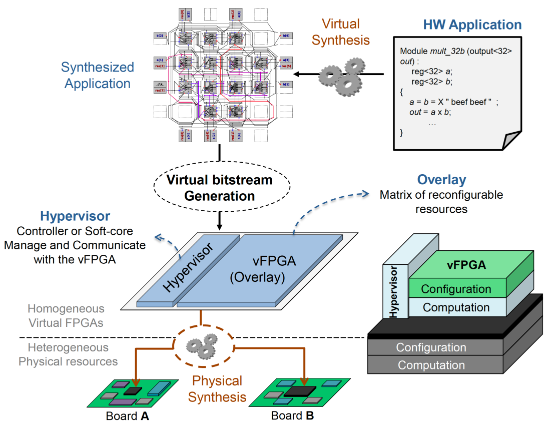
1) **Overlay Domain Space Exploration** must be supported. In the same way software virtual machines (VMs) can be tuned to specific requirements, the design of specific overlays should be proposed to the programmer.
2) The environment must support **Third-party tooling integration**, as, low level tooling may be overlay specific, and at a higher level, many opportunities can arise by the advent of overlays, seen as means to propose alternatives to classical HLD languages, proprietary tools and closed methodologies.
3) **Open bitstream structure** is mandatory for researchers working on overlay security.
4) **SoC integration** of overlays should be easy, meaning that at both hardware and system levels, facilities are provided to simplify the deployment of modeled overlays. At a hardware level, DMA access, pinout interconnect, control signals should be easily accessible.
5) **Metrics and simple feedback** must be provided to the user for him to evaluate his overlay, or the overlay-application adequation.
6) **Debug capabilities** must help the designer in setting up a fully functional solution. That requires a full controlability over the overlay (halting, resuming execution) and introspection capabilities (read/write over registers and memories).
7) **Environment availability** and **ease of installation** are two strong requirements for succeeding on offering a community animation support.

The Argen environment conforms to all of these requirements.

An overview of Argen toolchain is depicted on figure 1.

Argen is a complete EDA toolchain, which encompasses traditional design phases in a compact way : architectural and application design, application mapping and synthesis, as well as final binary code generation, for both the overlay and the application. It seeks to facilitate the chaining of these activities, from a user point of view, through a GUI that makes the methodology explicit and serves as guidelines for the user. The strict respect of these guidelines is recommended, but not mandatory, which opens up several possibilities of enrichment of the toolset. In the sequel (as well as in the tool), we will resort to 'architecture' or 'overlay' interchangeably.

This methodology amounts to the application of 10 passes as illustrated by figure 2. Some passes consist in a direct call to external tools, while some others are strictly handled by Argen. The interleaving of theses various calls remains transparent to the user.



```
Module mult_32b (output<32>
out) :
    reg<32> a;
    reg<32> b;
{
   a = b = X " beef beef "  ;
   out = a x b;
         ...
}
```

Fig. 1. Overlay: principles

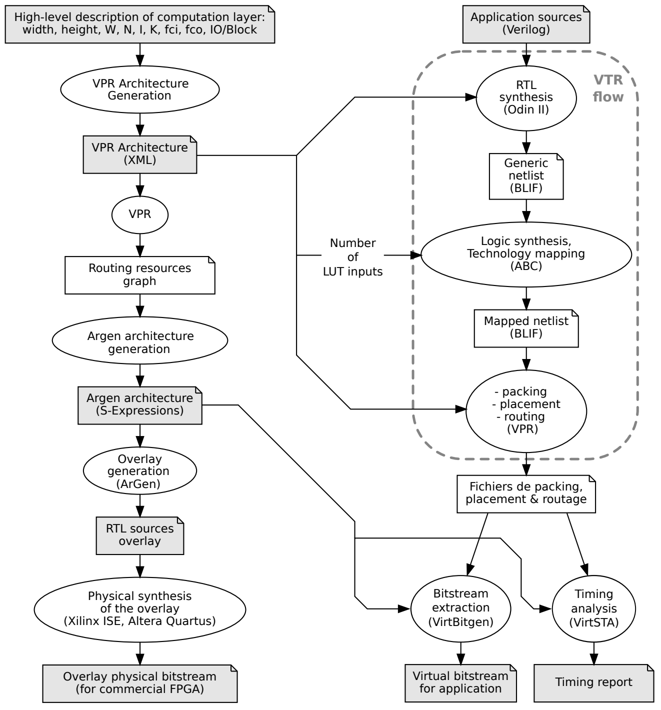

High-level description of computation layer: width, height, W, N, I, K, fci, fco, IO/Block
VPR Architecture Generation
VPR Architecture (XML)
VPR
Routing resources graph
Argen architecture generation
Argen architecture (S-Expressions)
Overlay generation (ArGen)
RTL sources overlay
Physical synthesis of the overlay (Xilinx ISE, Altera Quartus)
Overlay physical bitstream (for commercial FPGA)
Application sources (Verilog)
VTR flow
RTL synthesis (Odin II)
Generic netlist (BLIF)
Number of LUT inputs
Logic synthesis, Technology mapping (ABC)
Mapped netlist (BLIF)
- packing
- placement
- routing
(VPR)
Fichiers de packing, placement & routage
Bitstream extraction (VirtBitgen)
Timing analysis (VirtSTA)
Virtual bitstream for application
Timing report

Fig. 2. Overview of Argen tool flow

IV. Argen, hardware side

A. Overlay definition

The first activity is the description of the overlay itself as illustrated by fig. 3. The architecture conforms to a VPR [13] [14] template. This template describes a regular array of cells (CLBs), each of which embeds several logic elements (BLEs). A BLE contains a Look-Up Table and a register. The cells are surrounded by routing channels, interconnected through switch boxes. The switch boxes can expose several connection schemes, but the Wilton [15] connectivity is most commonly used.

The CLBs can drive a portion of their close routing tracks, while some of the tracks can drive the CLB's inputs. The number of inputs for a CLB is usually less than the sum of inputs of its BLEs. As a consequence, efficient resources sharing through a smart packing is a key issue.

The left side of figure 9 provides a schematic view of such an architecture. Specializing this template requires to fill up several fiels. In the Argen GUI, these fields correspond to the number of rows and columns of the overlay, the number of routing channels per track, the logical grain, the number of BLEs per CLB and the CLB connectivity. Also, the pinout is specified through a number of IOs per OIPin.

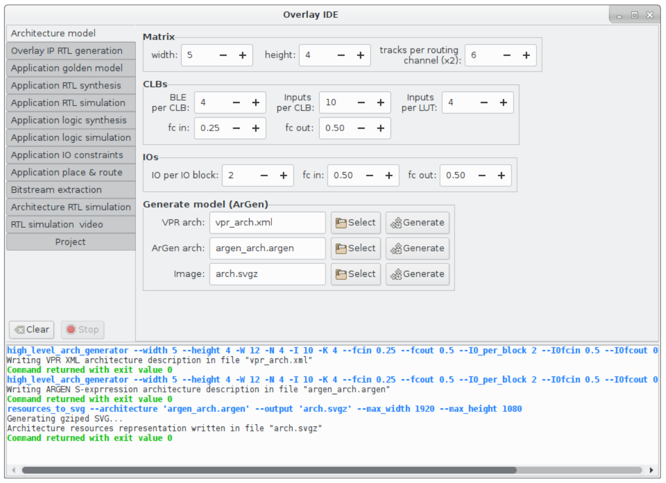
This leads to the generation of an XML file, that is readable by VPR for architecture synthesis, but also by Odin-II [16] for application synthesis (number of inputs by LUTs needs to be given).

B. Overlay generation

Starting from the architecture description, VPR generates a resources routing file, that, in turn, is read back by Argen. As a result, Argen can generate the RTL sources for the overlay, that can be synthesized using commercial FPGA tools.

Yet, as an overlay exhibits many potential combinational paths, the generation relies on extra registers, called VTPRs (Virtual Time Propagation Registers) [5]. The VTPRs latch the output of every configurable multiplexer that connects routing wire tracks. The VTPRs break down physical logic chains into short segments, and prevent any combinatorial loop from appearing on the overlay implementation, whichever its configuration. As VTPRs are transparent for circuits mapped on the overlay, VTPRs are not considered during the modelling phase but are rather silently injected during the generation step.

VTPRs exhibit two decisive advantages. First, using VTPRs in a overlay alleviates the task of the physical synthesizer, as VTPRs reduce timing paths and prevent combinatorial loops hence promotes architectures' scalability. There is no more need to limit size and complexity of synthesized architectures,


Overlay IDE
Architecture model
Overlay IP RTL generation
Application golden model
Application RTL synthesis
Application RTL simulation
Application logic synthesis
Application logic simulation
Application IO constraints
Application place & route
Bitstream extraction
Architecture RTL simulation
RTL simulation video
Project
Clear
Stop
Select
Generate

Matrix
width: 5
height: 4
tracks per routing channel (x2): 6

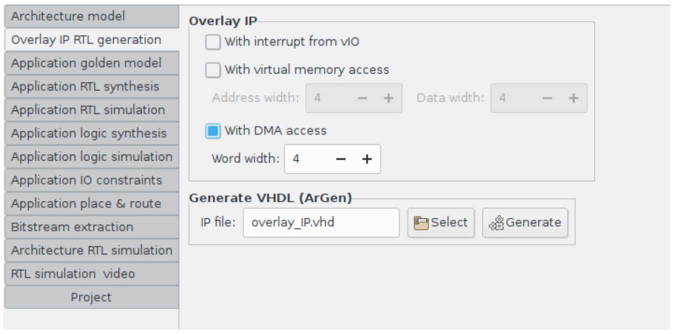
CLBs
BLE per CLB: 4
Inputs per CLB: 10
Inputs per LUT: 4
fc in: 0.25
fc out: 0.50

IOs
IO per IO block: 2
fc in: 0.50
fc out: 0.50

Generate model (ArGen)
VPR arch: vpr_arch.xml
ArGen arch: argen_arch.argen
Image: arch.svgz

```
high_level_arch_generator --width 5 --height 4 -W 12 -N 4 -I 10 -K 4 --fcin 0.25 --fcout 0.5 --IO_per_block 2 --IOfcin 0.5 --IOfcout 0
Writing VPR XML architecture description in file "vpr_arch.xml"
Command returned with exit value 0
high_level_arch_generator --width 5 --height 4 -W 12 -N 4 -I 10 -K 4 --fcin 0.25 --fcout 0.5 --IO_per_block 2 --IOfcin 0.5 --IOfcout 0
Writing ARGEN S-exprression architecture description in file "argen_arch.argen"
Command returned with exit value 0
resources_to_svg --architecture 'argen_arch.argen' --output 'arch.svgz' --max_width 1920 --max_height 1080
Generating gziped SVG...
Architecture resources representation written in file "arch.svgz"
Command returned with exit value 0
```

Fig. 3. Argen overlay modelling


Architecture model
Overlay IP RTL generation
Application golden model
Application RTL synthesis
Application RTL simulation
Application logic synthesis
Application logic simulation
Application IO constraints
Application place & route
Bitstream extraction
Architecture RTL simulation
RTL simulation video
Project
Select
Generate

Overlay IP

With interrupt from vIO [ ]

With virtual memory access [ ]

Address width: 4

Data width: 4

[✓ With DMA access]

Word width: 4

Generate VHDL (ArGen)

IP file: overlay_IP.vhd

Fig. 4. Surrounding DMAs can feed the overlay with data

nor to restrict the signal flow in one direction. This, however, raises the need for an extra and faster clock (Clk_{VTPR}), to allow signal propagation through VTPRs within one applicative clock cycle. Second, VTPRs favor timing closure and make the Place&Route step both simple ans predictable. This is due to the uniform cost of routing segments.

In addition to VTPRs' injection, the generation step also hosts several options that make the designer's life easier, as shown in figure 4. First, the generated architecture can be granted virtual memory access. Second, IOs can support an interruption mode. And last, in order to further simplify the integration of the overlay within a SoC, a DMA feature can be added, in which case, setting the word width is on the user.

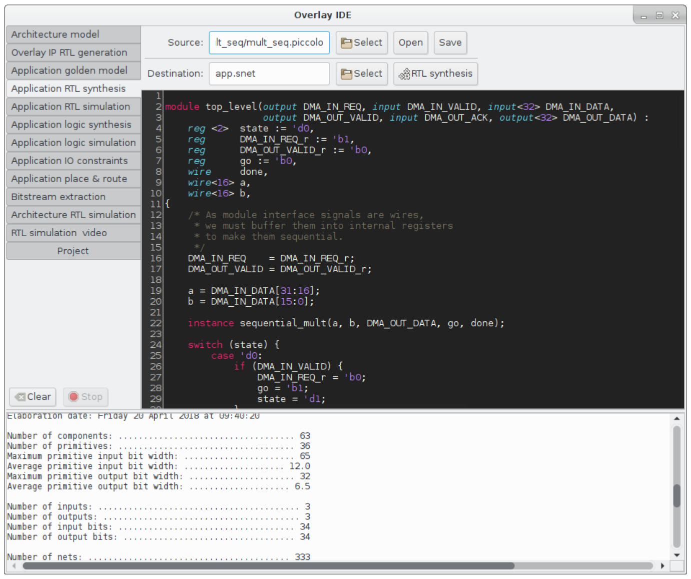
C. Overlay integration within a SoC

The overlay architecture has three main interfaces that must be handled by additional controllers: i) the configuration shift register (configuration layer), ii) the snapshot shift register (state layer), and iii) the virtual clock signal Fvirt (connected to virtual application registers of the computation layer). In order to make the overlay usable, a wrapper module instantiates the overlay architecture and includes the three controllers as shown in Fig. 5. The wrapper has a Wishbone bus interface, which maps configuration registers from these three controllers. A DMA controller can feed the overlay virtual inputs and save the virtual outputs from/to configurable memory areas accessible on the bus. The DMA controller uses some specific overlay virtual IOs to implement a handshake mechanism with virtual applications (data request, data ready). Alternatively, virtual IOs are also mapped as registers through the bus interface and can be accessed when the DMA controller is not used.

The *wrapper* module can be included to any SoC, allowing the control of the overlay through a software layer; it can be instantiated with an on-chip soft-core or used with an on-chip hard-core SoC (e.g. Zynq), or can be accessed by an off-chip system through a communication interface (such as PCIe with an appropriate Wishbone to PCIe converter).

D. SoC host

In this paper, we use the ZeFF SoC [17] as an hypervisor, completely synthesized on FPGA. ZeFF offers monitoring


Overlay IDE
Architecture model
Overlay IP RTL generation
Application golden model
Application RTL synthesis
Application RTL simulation
Application logic synthesis
Application logic simulation
Application IO constraints
Application place & route
Bitstream extraction
Architecture RTL simulation
RTL simulation video
Project
Select
Open
Save
Select
RTL synthesis
Clear
Stop

Source: lt_seq/mult_seq.piccolo

Destination: app.snet

```

module top_level(output DMA_IN_REQ, input DMA_IN_VALID, input<32> DMA_IN_DATA,
                 output DMA_OUT_VALID, input DMA_OUT_ACK, output<32> DMA_OUT_DATA) :
    reg <2>  state := 'd0,
    reg      DMA_IN_REQ_r := 'b1,
    reg      DMA_OUT_VALID_r := 'b0,
    reg      go := 'b0,
    wire     done,
    wire<16> a,
    wire<16> b,
{
    /* As module interface signals are wires,
     * we must buffer them into internal registers
     * to make them sequential.
     */
    DMA_IN_REQ    = DMA_IN_REQ_r;
    DMA_OUT_VALID = DMA_OUT_VALID_r;

    a = DMA_IN_DATA[31:16];
    b = DMA_IN_DATA[15:0];

    instance sequential_mult(a, b, DMA_OUT_DATA, go, done);

    switch (state) {
        case 'd0:
            if (DMA_IN_VALID) {
                DMA_IN_REQ_r = 'b0;
                go = 'b1;
                state = 'd1;
```

```
Elaboration date: Friday 20 April 2018 at 09:40:20

Number of components: .................................. 63
Number of primitives: .................................. 36
Maximum primitive input bit width: ..................... 65
Average primitive input bit width: ................... 12.0
Maximum primitive output bit width: .................... 32
Average primitive output bit width: ................... 6.5

Number of inputs: ....................................... 3
Number of outputs: ...................................... 3
Number of input bits: .................................. 34
Number of output bits: ................................. 34

Number of nets: ....................................... 333
```

Fig. 5. Argen and user friendly DSLs for RTL descriptions: example of Picollo experimental language.

and management facilities, such as guest configurations and data streams, and remote access through a standard Ethernet interface and TCP/IP protocol. The SoC platform embeds (Fig. 9), among others: a processor, memory controllers (external RAMs, flash memories), and some communication peripherals (*e.g.* Ethernet, UART). It is organized around a Wishbone shared bus, associated to a dedicated generator, easing the addition and removal of peripherals, thus making the platform more flexible. When porting the whole platform from one FPGA board to another, some parts of the SoC might change depending on the on-board IO connections and devices, such as memories or transceivers, which can have different IO interfaces between boards. In this architecture, the SoC sees the overlay as a peripheral device (Overlay IP), attached to the system bus.

V. Argen, Application side

A. Application synthesis: front end

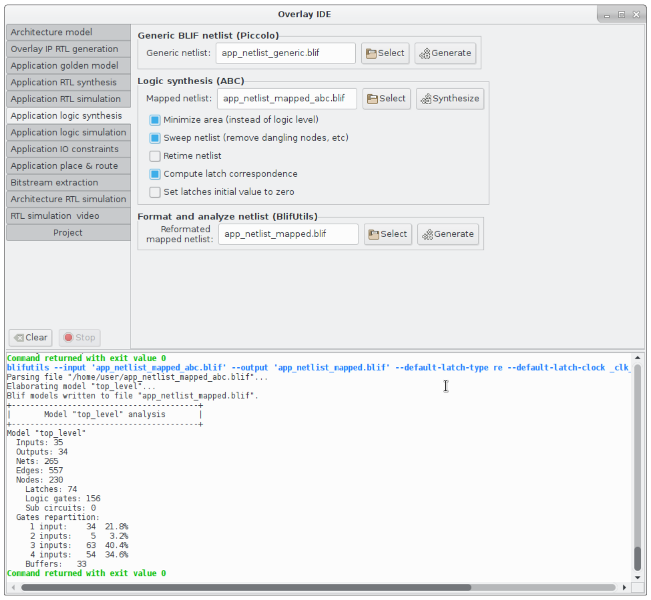
Once done, the overlay is generated and can be integrated on demand.

On the application side, several options are offered to the user: in our experiments, we have used Verilog as well as our own DSL (picollo) for RTL description (figure 5). Argen then calls ABC [18] to generate a blif netlist. This step is shown in figure 6. A reduced set of options is offered to the designer, including optimization and retiming.

Still, other alternatives, either commercial or open-source, can be envisioned, as long as a generic netlist in BLIF format can be generated.

B. Application synthesis: back end

VPR [14] is responsible for the packing, placement and routing tasks. The standard VPR options are made available through the GUI: clustering options, placement policy, routing


Overlay IDE
Architecture model
Overlay IP RTL generation
Application golden model
Application RTL synthesis
Application RTL simulation
Application logic synthesis
Application logic simulation
Application IO constraints
Application place & route
Bitstream extraction
Architecture RTL simulation
RTL simulation video
Project
Select
Generate
Synthesize
Clear
Stop

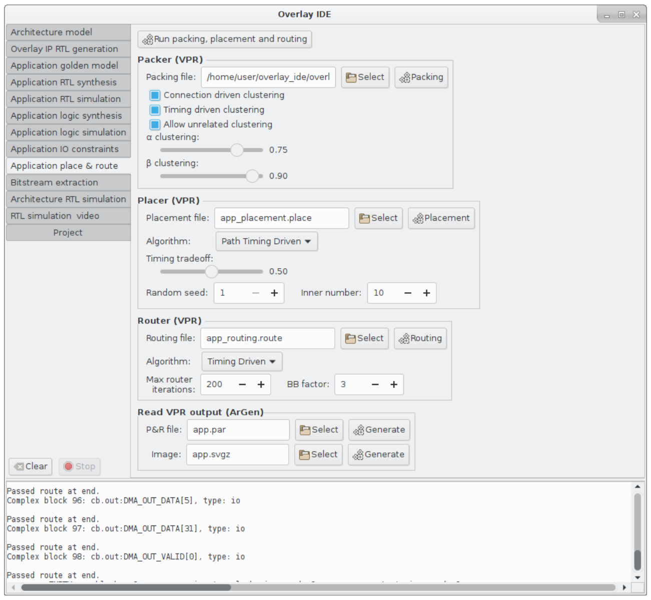
Generic BLIF netlist (Piccolo)
Generic netlist: app_netlist_generic.blif

Logic synthesis (ABC)
Mapped netlist: app_netlist_mapped_abc.blif
[✓] Minimize area (instead of logic level)
[✓] Sweep netlist (remove dangling nodes, etc)
[ ] Retime netlist
[✓] Compute latch correspondence
[ ] Set latches initial value to zero

Format and analyze netlist (BlifUtils)
Reformatted mapped netlist: app_netlist_mapped.blif

```
Command returned with exit value 0
blifutils --input 'app_netlist_mapped_abc.blif' --output 'app_netlist_mapped.blif' --default-latch-type re --default-latch-clock _clk_
Parsing file "/home/user/app_netlist_mapped_abc.blif"...
Elaborating model "top_level"...
Blif models written to file "app_netlist_mapped.blif".
+----------------------------------------+
|       Model "top_level" analysis       |
+----------------------------------------+
Model "top_level"
  Inputs: 35
  Outputs: 34
  Nets: 265
  Edges: 557
  Nodes: 230
    Latches: 74
    Logic gates: 156
    Sub circuits: 0
  Gates repartition:
     1 input:    34  21.8%
     2 inputs:    5   3.2%
     3 inputs:   63  40.4%
     4 inputs:   54  34.6%
    Buffers:   33
Command returned with exit value 0
```

Fig. 6. Argen takes advantage of established open-source tools for logic synthesis


Overlay IDE
Architecture model
Overlay IP RTL generation
Application golden model
Application RTL synthesis
Application RTL simulation
Application logic synthesis
Application logic simulation
Application IO constraints
Application place & route
Bitstream extraction
Architecture RTL simulation
RTL simulation video
Project
Run packing, placement and routing
Select
Packing
Placement
Routing
Generate
Clear
Stop

Packer (VPR)
Packing file: /home/user/overlay_ide/overl
[✓] Connection driven clustering
[✓] Timing driven clustering
[✓] Allow unrelated clustering
α clustering: 0.75
β clustering: 0.90

Placer (VPR)
Placement file: app_placement.place
Algorithm: Path Timing Driven
Timing tradeoff: 0.50
Random seed: 1
Inner number: 10

Router (VPR)
Routing file: app_routing.route
Algorithm: Timing Driven
Max router iterations: 200
BB factor: 3

Read VPR output (ArGen)
P&R file: app.par
Image: app.svgz

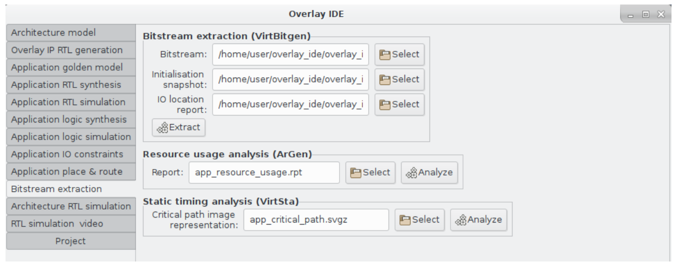
```
Passed route at end.
Complex block 96: cb.out:DMA_OUT_DATA[5], type: io

Passed route at end.
Complex block 97: cb.out:DMA_OUT_DATA[31], type: io

Passed route at end.
Complex block 98: cb.out:DMA_OUT_VALID[0], type: io

Passed route at end.
```

Fig. 7. Place and Route activities and virtual static timing analysis

algorithm, etc. Figure 7 illustrates this GUI panel. Again, Argen retrieves various result files for further extracting a virtual application bitstream.

C. Bitstream generation

At this time, the bitstream structure has no flexibility and strictly conforms to a serpentine based visit of the architecture. Then, the bitstream is outputted by serializing the local configuration in the proper order (figure 8). Note that the bitstream does not include the BLEs' register value; this information is collected through an extra wiring, dedicated to snapshot, (state capture and restoration).


Overlay IDE
Architecture model
Overlay IP RTL generation
Application golden model
Application RTL synthesis
Application RTL simulation
Application logic synthesis
Application logic simulation
Application IO constraints
Application place & route
Bitstream extraction
Architecture RTL simulation
RTL simulation video
Project
Select
Extract
Analyze

Bitstream extraction (VirtBitgen)
Bitstream: /home/user/overlay_ide/overlay_i
Initialisation snapshot: /home/user/overlay_ide/overlay_i
IO location report: /home/user/overlay_ide/overlay_i

Resource usage analysis (ArGen)
Report: app_resource_usage.rpt

Static timing analysis (VirtSta)
Critical path image representation: app_critical_path.svgz

Fig. 8. Bitstream generation in Argen

Argen also applies a static timing analysis to the synthesized netlist to identify the longest path in term of VTPRs.

D. SoC OS and dedicated functions

The embedded software is based on FreeRTOS [19], the LwIP TCP/IP stack and FatFs generic FAT file system module. The system has been extended to support overlay management functions, performing the following actions:

- *push_bitstream*: This function transfers a user vBitstream to the Overlay IP to configure the overlay resources.
- *set_vclk_div*: It specifies the clock divisor to generate the virtual clock frequency at which the application operates.
- *launch_vclk_for_N_cycles* and *stop_vclk*: They manage the virtual clock controller, and are used to run and stop the execution of the application whenever a live migration or scheduling is required.
- *save_snapshot* and *restore_snapshot*: These functions manipulate the interface between the snapshot register and the virtual application registers (detailed in the previous section), whether to take or restore a state snapshot.
- *pull_snapshot*, *push_snapshot*: The first function extracts the value of the snapshot register and saves it to an external file, the second loads it back from a given file.

Three levels of management are considered. First, a task manager is responsible for scheduling applications on the overlay. This manager relies on the above functions. Then a server waits for commands from the network, and allows remote control. Last, ZeFF acts as a local hypervisor and as an interface between its attached overlay and a higher-level supervisor, handling migration decisions.

VI. Argen toolchain : typical experimental usages

Overlays have recently emerged as a meaningful alternative to direct FPGA designs. There are several motivations for that. First, overlays can be easily customized to address specific applicative domains. This promotes a smart application-to-architecture adequation hence simplifies the programming process. Second, overlays usually exhibit a coarser grain than their implementation platforms. As a consequence, using overlays reduces the bitstreams size, hence speeds up the reconfiguration process. In addition, overlays can be decorated with some on-demand functionalities, such as dynamic partial reconfiguration (DPR) for any underlying platform.

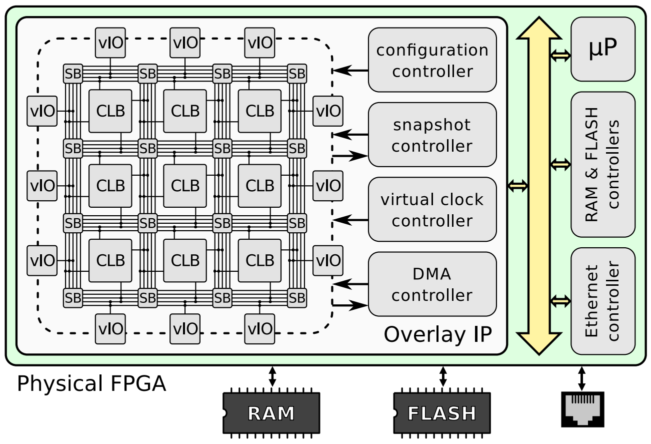


Fig. 9. Integrating an overlay in a SoC

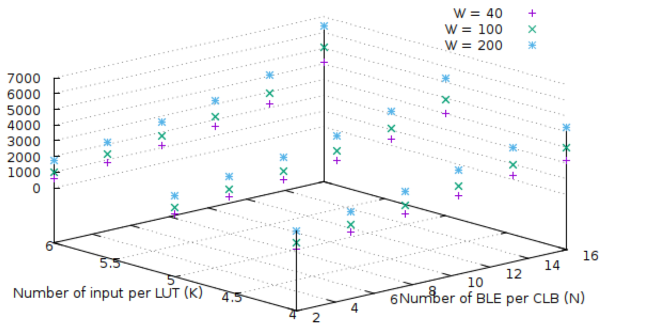


Fig. 10. Gate estimation for various architectures generated by Argen on STM 28nm FDSOI technology

silicon design often relies on human intervention. Hence, this feature still requires extra work.

Argen, as previously explained, helps in exploiting such benefits. Still, other usages make sense. These can be dispatched into four classes. First, one may want to prototype a reconfigurable architecture. Second, software experts may want to develop and score a new algorithm or data representation for the toolset. Third, system designers can use Argen as a prototyping platform for a new usage. Last, Argen offers an innovative teaching support.

The second item has been indirectly addressed in section V-A, through mentioning some interchange issues (e.g. BLIF format as a requirement for plugging in any synthesizer). Hence, this section focuses on the other three points left unaddressed.

A. *Prototyping new embedded FPGAs*

Argen supports fast modeling of overlays, and offers some characterization metrics. Assuming some model to model transformations are properly calibrated, Argen can also serve as a prototyping tool to design embedded FPGAs (eFPGAs). Such eFPGAs are offered by some companies (like FlexLogic, Menta or Achronix), to add programmable parts in traditional complex SoCs. eFPGAs are generally provided as a fixed-transistor-layout building blocks, which require careful and in-depth custom technology manipulations and characterizations. In our case, our experiment involved estimating the number of ASIC gates corresponding to our overlays generated at the RTL level, using a standard Synopsys synthesis flow (a similar experiment was presented in [20]). Our target technology was STM 28nm FDSOI. The estimated gate counts are reported in figure 10.

The design parameters under investigation are the number of BLE per CLB (N), the number of inputs by LUT (k), as well as the width (W) of routing channels. The figure shows that a typical overlay ($N = 8, k = 5$) generated by Argen ranges from 3 to 4 Millions gates, depending on W. This work is still preliminary, but Argen appears as a promising tool to quickly get some early results for such eFPGA prototypes. However, getting accurate metric is highly challenging as optimizing a

B. *Innovative usage prototyping*

By being target independent, overlays are intended to last longer than the underlying FPGA technology that is used for implementation. Then, overlays offer a durable baseline when porting applications, regardless of the homogeneity of the underlying platform. This enables a partial renewal of FPGA boards in accordance with the policy of modernising a distributed infrastructure (FPGAs based platform for IoT as an example). Not only does Argen support ”blind” deployment of applications over heterogeneous physical support thanks to the overlay layer, but tasks can also be relocated on the fly.

In general, applications running on reconfigurable architectures can be represented by the resource configuration and the state of the application. Authors in [21] report two ways for accessing the state of a task that executes on a FPGA:

1) By using the Internal Configuration Access Port (ICAP), which is mostly used for DPR. This solution remains technology and vendor dependent. Additionally the state is read back along with configuration bits, which leads to a slow extraction process. However the mechanism is transparent to the application.
2) By decorating applications with some access facilities to state bits. This solution is portable, and state extraction is possible. However, every application has to be reworked, and both area and frequency are impacted.

In an overlay context, the configuration of the vBitstream on the vFPGA is done as presented in the previous sub-section, while the state of the application is held on memory elements in the computation layer. These memories correspond in this architecture to the virtual application registers integrated in each reconfigurable element.

pull_snapshot generates a file that can be used on another target node. As a consequence, tasks can be migrated between nodes.

Figure 11 illustrates such a task migration between Xilinx and Altera based FPGA boards.

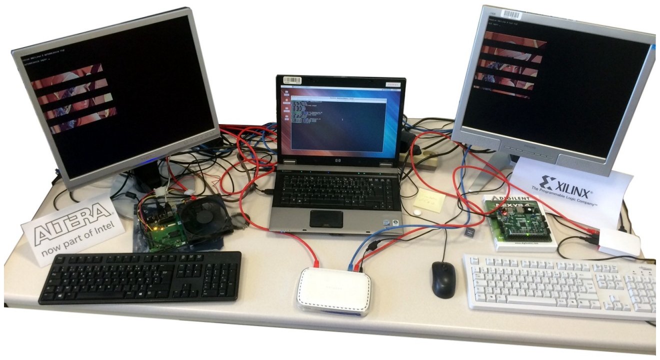


Fig. 11. Binary task migration between Altera and Xilinx platforms

C. Education

Overlays also offer a smart playground to students. Argen alleviates the need for a full mastery of complex traditional design flows. The architecture can also be tailored on demand, which promotes a just fit approach. The tool flow can incorporate any third-party tooling stage. The result is easy to visualize, as some simulation videos can be outputted as illustrated in figure 12. The videos [22] show how signals propagate through the routing topology, how IOs react, as well as the internal state of BLEs.

Also, Argen comes along with a video tutorial [23].

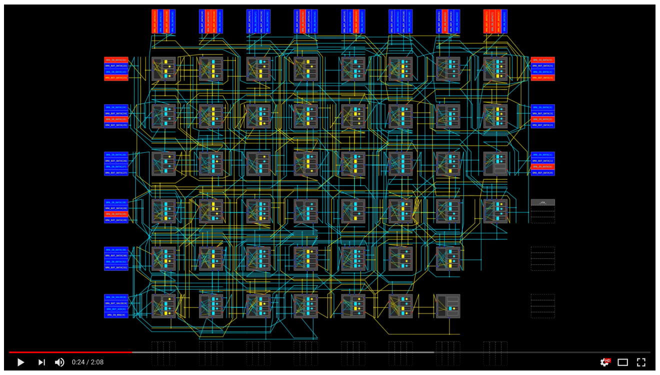


Fig. 12. Videos can be generated during the simulation phase

VII. Conclusion and Future Work

In this paper, the Argen toolchain for overlay-centric SoC design has been presented. It gathers a set of tools to design and explore fine-grained overlays, but also enables a traditionnal designer to directly program these overlays. Argen conforms to all of the requirements that have been summarized in section III: the environment supports domain space exploration, both in terms of hardware and associated tooling, it offers debug capabilities, and comes along with a sound documentation. As a result, Argen targets software designers as well as hardware experts, and is accessible enough to be a good candidate as teaching support.

The short term research directions are centered around adding security and supporting more complex overlay structures:

a) Bitstream specialization: Argen is intended to serve as a prototyping hub for overlay designers. Among others, the ability to define its own bitstream is a key issue. This is of prime importance when addressing security. In a close future, Argen will include a bitstream specification feature. Two options are under consideration. The first one is to simply add a cryptographic protection (e.g. AES block). The other one is a pure mixing system.

b) Heterogeneous architecture: Being based on VPR template, Argen focuses on regular homogeneous architectures. Still, other options exist, which support heterogeneous architectures and mixed grain in a smooth way such as [3]. A comparative study of these available environments is expected to drive development of new features.

References

[1] H. K. So and C. Liu, "FPGA overlays," in *FPGAs for Software Programmers*, D. Koch, F. Hannig, and D. Ziener, Eds. Springer, 2016, pp. 285–305. [Online]. Available: https://doi.org/10.1007/978-3-319-26408-0

[2] "Argen web page." [Online]. Available: http://www.ensta-bretagne.fr/lagadec/argen

[3] L. Lagadec, D. Lavenier, E. Fabiani, and B. Pottier, "Placing, routing, and editing virtual fpgas," in *Field-Programmable Logic and Applications, 11th International Conference, FPL 2001, Belfast, Northern Ireland, UK, August 27-29, 2001, Proceedings*, ser. Lecture Notes in Computer Science, G. J. Brebner and R. F. Woods, Eds., vol. 2147. Springer, 2001, pp. 357–366. [Online]. Available: https://doi.org/10.1007/3-540-44687-7

[4] A. Brant and G. G. Lemieux, "ZUMA: An open FPGA overlay architecture," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 2012, pp. 93–96.

[5] T. Bollengier, L. Lagadec, M. Najem, J. L. Lann, and P. Guilloux, "Soft timing closure for soft programmable logic cores: The argen approach," in *Applied Reconfigurable Computing - 13th International Symposium, ARC 2017, Delft, The Netherlands, April 3-7, 2017, Proceedings*, 2017, pp. 93–105.

[6] X. Li, A. K. Jain, D. L. Maskell, and S. A. Fahmy, "A time-multiplexed FPGA overlay with linear interconnect," in *DATE*. IEEE, 2018, pp. 1075–1080.

[7] D. Koch, C. Beckhoff, and G. G. F. Lemieux, "An efficient fpga overlay for portable custom instruction set extensions," in *2013 23rd International Conference on Field programmable Logic and Applications*, Sept 2013, pp. 1–8.

[8] J. Abhishek, Kumar, "Architecture centric coarse-grained fpga overlays," Ph.D. dissertation, Nanyang Technological University, 2017.

[9] J. Coole and G. Stitt, "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 13–22.

[10] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim, "Dyser: Unifying functionality and parallelism specialization for energy-efficient computing," *IEEE Micro*, vol. 32, no. 5, pp. 38–51, Sep. 2012. [Online]. Available: http://dx.doi.org/10.1109/MM.2012.51

[11] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient Overlay architecture based on DSP blocks," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 25–28.

[12] A. K. Jain, X. Li, P. Singhai, D. L. Maskell, and S. A. Fahmy, "Deco: A DSP block based FPGA accelerator overlay with low overhead interconnect," in *24th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2016,*

Washington, DC, USA, May 1-3, 2016, 2016, pp. 1–8. [Online]. Available: https://doi.org/10.1109/FCCM.2016.10

[13] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *International Workshop on Field Programmable Logic and Applications*. Springer, 1997, pp. 213–222.

[14] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, K. Kent, and J. Rose, "Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 32:1–32:23, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2068716.2068718

[15] M. I. Masud and S. J. E. Wilton, "A new switch block for segmented fpgas," in *Field-Programmable Logic and Applications, 9th International Workshop, FPL'99, Glasgow, UK, August 30 - September 1, 1999, Proceedings*, 1999, pp. 274–281.

[16] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, "Odin ii-an open-source verilog hdl synthesis tool for cad research," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*. IEEE, 2010, pp. 149–156.

[17] L. Lagadec, J. L. Lann, and T. Bollengier, "A prototyping platform for virtual reconfigurable units," in *9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip, ReCoSoC 2014, Montpellier, France, May 26-28, 2014*. IEEE, 2014, pp. 1–7. [Online]. Available: https://doi.org/10.1109/ReCoSoC.2014.6860689

[18] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.

[19] *The FreeRTOS™ Reference Manual*, Amazon Web Services, https://www.freertos.org/Documentation/.

[20] V. C. Aken'Ova, G. Lemieux, and R. Saleh, "An improved "soft" efpga design and implementation strategy," in *Proceedings of the IEEE 2005 Custom Integrated Circuits Conference, 2005.*, Sept 2005, pp. 179–182.

[21] K. Jozwik, H. Tomiyama, M. Edahiro, S. Honda, and H. Takada, "Comparison of preemption schemes for partially reconfigurable fpgas," *IEEE Embedded Systems Letters*, vol. 4, no. 2, pp. 45–48, June 2012.

[22] "Argen video on youtube." [Online]. Available: https://www.youtube.com/watch?v=WwIwGAp7BVs

[23] "Argen wtutorial on youtube." [Online]. Available: https://www.youtube.com/watch?v=tq2igF468pU