High-level Modelling and Exploration of Coarse-grained Re-configurable Architectures

Anupam Chattopadhyay, Xiaolin Chen, Harold Ishebabi, Rainer Leupers, Gerd Ascheid, Heinrich Meyr
Integrated Signal Processing Systems, RWTH Aachen University 52056 Aachen, Germany
anupam@iss.rwth-aachen.de

Abstract

The increasing complexity of today's multimedia and wireless applications is motivating the system designers to innovate continuously. With the challenge to keep various performance metrics in a tight balance while designing a complex system, an entire range of components are now being offered as choices for system building blocks. Coarse-Grained Re-configurable Architecture (CGRA), a strongly emerging class, is currently receiving due attention for offering excellent performance as well as flexibility post fabrication. Compared to the programmable and flexible microprocessors these architectures are shown to yield stronger performance, especially in case of regular and data-driven applications. A variety of system designs are proposed of late, with CGRA as one of the key building blocks. Most of the research initiatives taken in this area have resorted to a template-based approach, where the structure of the reconfigurable architecture is partially fixed with several tunable parameters. In this paper, we present a language-driven modelling and exploration framework for CGRAs. In the domain of CGRAs, this framework attempts to bring modelling ease, genericity, early exploration and path to implementation together. The modelling formalism proposed in this paper as well as the exploration capabilities are demonstrated via experiments with several algorithmic kernels.

1. Introduction

Increasing complexity of wireless and multimedia applications are driving the system designers to innovate continuously over the last years. A fundamental trade-off between flexibility and the traditional performance metrics (energy-area-timing) is being deemed more important than ever. In such a scenario, the building blocks of modern System-on-Chip (SoC) - under critical review - are those with a tunable balance of performance and flexibility. Coarse-Grained Reconfigurable Architecture (CGRA), with strong performance advantage as well as ability to be flexed post fabrication, is one such key building block. A flurry of recent design proposals, both academic and commercial, include CGRAs as accelerators [1] [2] [3] or are completely based upon CGRAs [4] [5].

Since the traditional usage of FPGAs is mostly restricted to prototyping and emulation, design methodology of CGRAs is still a nascent field. As the defining roles of CGRAs in modern systems are getting clearer, more and more design flows are being proposed. Regardless of the approach followed by these design flows, the central challenges of design remain the same. In the following, an overview of the CGRA-based research efforts are outlined.

1.1 Modelling

Most of the research in the field of CGRA design are focussed on parameterized design space exploration [2] [6] [7] rather than on high-level modelling. The physical implementation, in most cases, is done beforehand due to the limitation of aggressive automatic physical optimization and/or lack of retargetable CGRA mapping-placement-routing tools. Commercial FPGA vendors have suggested RTL abstraction as a level of modelling, albeit only for simulation purposes [8]. As a result, the modelling platforms either provide limited flexibility of modelling or are completely devoid of implementation aspects [6]. The recently proposed ADRES [9] architecture from IMEC, for example, allows to change the parameters like - number and size of register file associated with each basic functional unit, operation set supported by the functional units and interconnect topology - of its re-configurable array. Similar architectural parameters are also found in [10] and [11]. The limitation of such a parameterized approach is that, the introduction of a new design parameter might lead to serious modification in the associated tool-chain.

1.2 Mapping, Placement and Routing

Mapping, placement and routing (referred together as *CGRA synthesis* henceforth) are three important phases of porting a datapath onto an CGRA. In several attempts to perform CGRA synthesis, the architecture is partially or completely fixed. However, that may not affect the genericity of synthesis algorithms.

A completely generic approach for exploring the functional units of a mesh-based CGRA with corresponding mapping algorithm is presented at [6]. For different grid configurations, interconnect topologies and functional units inside the processing elements - mapping is performed. The mapping algorithm is selected from a set of different topology traversal options, optimality of which is not studied. In a recent effort to generate mapping solutions for CGRAs [12], an approach based on Integer Linear Programming (ILP) is outlined for mesh-like topologies. The extension of this approach to different routing topologies and/or different basic functional units are not specified. A memory-aware mapping algorithm is proposed at [13]. Here, the CGRA is fed into the algorithm in form of an undirected graph. The input datapath is modelled as a data-dependence graph, where each node of the graph is assigned with its priority. The mapping algorithm works on the principle of list scheduling, where the nodes residing on the critical path are mapped first and so on. For each node, a list of mapping-cum-placement choices are first determined. On the basis of available routing resources, the best choice is selected. In a notable attempt, Mei et al. [2] proposed to merge the scheduling part of a C compiler to take the CGRA structural constraints into account via modulo scheduling algorithm. The common feature among all these CGRA synthesis algorithms are that those start from some form of data-dependence of the input graph and consequently finds the best mapping, placement and routing. Often, these phases are merged together.

A datapath synthesis system for coarse-grained FPGA is presented at [14]. This allows the FPGA users to write the input datapath in a language called ALE-X (which is oriented to C). The FPGA structural details, though can be altered by the designer, is strongly embedded in the mapping, placement and routing algorithms. Post routing, a scheduling is performed to optimally sequence the I/O operations in view of limited bus resources. An interesting approach is adopted for mapping applications to the GARP architecture [15]. By recognizing that both synthesis and compilation are actually solving the same problem, in [15] the FPGA synthesis is performed using similar techniques as found in the domain of high-level compilers. The input data-flow graph is split into trees and then tree covering algorithm is applied for mapping. Furthermore, noting that the placement decisions seriously affect the mapping results, a dynamic programming-based placement is performed simultaneously with the mapping. A drawback of the synthesis algorithm in [15] is that, it requires split-

978-3-9810801-3-1/DATE08 © 2008 EDAA

ting of input graphs into trees, which comes with sub-optimal result in the global context. In [16], tiles of coarse-grained FPGA named Montium are used. For Montium, a set of template operators are first created. Thereafter, for each node, all possible template matches are outlined. From these matches a conflict graph is created and a heuristic to determine maximum non-overlapping match set is employed. This method, though good for area constraints, does not address the delay element.

1.3 Implementation

Compared to the volume of research in the field of CGRA synthesis, relatively fewer attempts are made to automatically derive an implementation from high-level CGRA specification. In one of the CGRA exploration tools [14], automatic HDL is generated from an intermediate architectural abstraction. The initial architectural parameters are estimated from the input datapath (to be mapped) and then can be fine-tuned via a graphical editor. For the parameterizable CGRA design flows the RTL generation is often used for simulation [17] [2] but, not for the final implementation. In keeping tune with the parameterized modelling approach, a parameterizable CGRA template using VHDL is described in [10].

In contrast, it is interesting to take a view in the realm of fine-grained FPGAs. The architectures of fine-grained commercial FPGAs are always tuned with physical optimizations. Therefore, high-level modelling is relevant only to the extent of allowing the FPGA synthesis tools to encompass a variety of FPGAs. On the FPGA modelling and automated implementation, fine-grained FPGA research have limited results. On the FPGA synthesis aspect, expectedly, significant research contribution is made over years. Optimal synthesis algorithms for LUT-based FPGA architectures are proposed [18] [19] and the interplay between synthesis phases are extensively studied [20].

Contribution : In this paper, the aforementioned challenges are confronted as following. Firstly, a **high-level description formalism** is conceived to model CGRAs in a completely generic way, thereby avoiding restricted parameterized design approach. This requires powerful tooling support, which will allow optimized synthesis/implementation for all the design points. For **CGRA synthesis**, a different course of path than outlined in existing CGRA literature is taken. The CGRA structure is viewed as a homogeneous one with clustering. A cluster is a collection of one or more Logic Elements (LEs). Each LE may consist of multiple varied functions. A CGRA structure is termed homogeneous if the same cluster (possibly with heterogeneous LEs inside) is repeated to build the overall CGRA. On that basis, best-in-class synthesis algorithms from the domain of fine-grained FPGAs are applied with necessary adaptation. Furthermore, it is shown that certain form of homogeneity can be established in the so-called non-clustered heterogeneous CGRAs. Therefore, the same algorithms can be applied with little bit of tuning. Thirdly, to obtain the **CGRA implementation**, RTL description (VHDL, Verilog) is automatically generated from the high-level language. The RTL description serves the important purpose of providing simulation environment as well as early estimation of the performance. The RTL description can be conveniently integrated with a commercial physical synthesis flow. Finally, the complete CGRA design flow is integrated to an existing **ADL-driven processor design framework** [21] to allow complete design space exploration of partially re-configurable processors.

The rest of the paper is organized as follows. The following section 2 points the scope of current CGRA design flow. The section 3 presents an overview of the CGRA modelling formalism. In section 4 and in section 5, the CGRA synthesis flow and the RTL implementation are described respectively. Section 6 reports our experiments with the tool-flow outlined in this paper. The paper is concluded and future directions are mentioned in section 7.

2. Integration with Processor Design Framework

The integration of a commercial ADL-driven processor design framework (CoWare/LISATek) [21] with the CGRA synthesis and implementation flow is captured graphically using the figure 1. The software tool suite as well as the processor RTL description are automatically generated from the extended ADL description. The proposed CGRA description is conceived as a part of the ADL LISA. For RTL implementation, recent advances in LISA-based tools also allow partitioning of the processor datapath to indicate if some part is going to be synthesized on to the CGRA. The partitioning results into a DFG description (of processor's partial datapath), which is fed directly as input to the CGRA synthesis tools. The CGRA synthesis flow generates configuration bitstream, which can be simulated on the HDL implementation of the CGRA. The simulation can be done stand-alone or together with the processor.

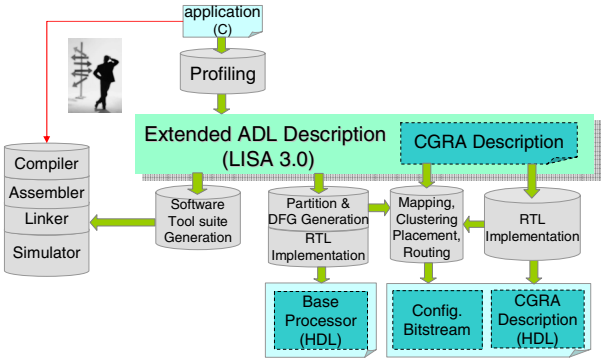

Figure 1. Integrated Processor Design Flow

3. CGRA Modelling

In this section, an overview of the proposed modelling formalism is provided. The CGRA description is organized in three interrelated parts. First, the *logic elements*, which defines the basic functional elements. The logic elements are arranged hierarchically in the *topology* part. Finally, the *interconnect* part organizes the connections.

3.1 Logic Element

A logic element can be written using the `ELEMENT` keyword. Within an element, the I/O ports are defined. For each I/O port, attributes can be specified. An attribute can be either `REGISTER` or `BYPASS` indicating that particular port can be held or bypassed while connecting. The behavior of the element is captured within `BEHAVIOR` section of element in form of a subset of C language (without e.g. pointers, structures, arrays). In order to specify a wide number of possible operators, configurable statically or dynamically, the keyword `OPERATOR_LIST` is used. An exemplary element definition and corresponding hardware representation can be observed in the figure 2. From the `OPERATOR_LIST` and the `ATTRIBUTE` definition, configuration bits are automatically inferred during RTL implementation. Note that, the logic elements can be used for the purpose of routing, too. This is exemplified with the outport `z` in the figure 2. The logic element may also include some control structure like multiplexing, which can be used to cover applications in form of Control Data Flow Graph (CDFG).

```
ELEMENT alu {
  PORT{
    IN unsigned<16> a,b;
    OUT unsigned<16> y,z;
  }
  ATTRIBUTES {
    REGISTER(y);
    BYPASS(y);
  }
  BEHAVIOR {
    OPERATOR_LIST op = {<<,>>,+};
    y = a op b;
    z = a;
  }
}
```

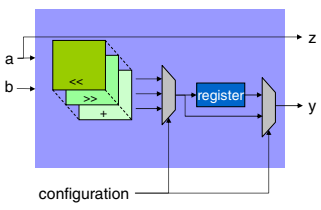

Figure 2. Logic Element Definition

3.2 Topology

The `TOPOLOGY` section contains several `CLUSTER`s. Similar to the logic element, I/O ports and corresponding attributes can be defined inside these clusters. Within the `LAYOUT` part of cluster, the previously defined elements can be put together in a `ROW`. Several

rows can be then defined consecutively, building a 2-dimensional structure. A cluster can be formed using elements and/or clusters. By this process, a hierarchical structure can be designed. An exemplary topology definition is shown in the figure 3. Using this hierarchical modelling style, grid-like structures can be easily described.

```
TOPOLOGY {
  CLUSTER cluster1 {
    PORT{ IN unsigned<16> inport;
          OUT unsigned<16> outport; }
    ATTRIBUTES { ... }
    LAYOUT{
      ROW row0 = {alu, alu};
      ROW row1 = {alu, clb};
    }
  }

  CLUSTER cgra {
    PORT{ IN unsigned<16> g_in;
          OUT unsigned<16> g_out; }
    ATTRIBUTES { ... }
    LAYOUT{
      ROW row0 = {cluster1, cluster1, cluster1, cluster1};
      ...
    }
  }
}
```

Figure 3. Topology Definition

3.3 Interconnect

The interconnects between the clusters and the elements can be specified in `CONNECTIVITY` section. For each cluster, a set of `BASIC` rules are described. Within one cluster's context several such rules connect the I/O ports of the cluster and its children entities. Furthermore, the rules are bound by a definite `STYLE` e.g. mesh, point-to-point, nearest neighbor. In case of several connectivity rules implying multiple inputs to a single port, a multiplexer is inferred, which is connected to the global configuration control bitstream. The connectivity style is associated with a parameter to indicate the hop-length of the connections. The default parameter is 1. The connectivity stride decides the number of hops the interconnect makes to establish a direct connection. By this way, a wide range of different connectivity styles at different hops can be established. As shown in the figure 4, multiple overlapping connectivity styles can be modelled. This enables the CGRA designer to decompose a complex routing architecture into several styles. For clarity, the effect of local and global connections are shown separately.

```
CONNECTIVITY{
 RULE cgra {
  STYLE(NEARESTNEIGHBOUR,1);
  STYLE(MESH, 2);

  BASIC (cluster1,cluster1) {inport TO outport;}
  BASIC (cgra, cluster1) {g_in TO inport;}
  BASIC (cluster1, cgra) {outport TO g_out;}
 }
```

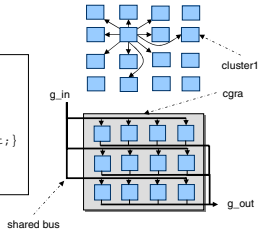

Figure 4. Connectivity Definition

Together, the three parts of the CGRA modelling is able to describe a wide range of architectures. All of those architectures can be processed with the CGRA implementation flow. The CGRA synthesis flow can also, in principle, handle each of those. However, for a large cluster size or for a complicated connectivity network - the synthesis runtime will be much too long with the current algorithm implementation. Even then, the proposed language provides more freedom in describing arbitrary logic element functionalities, interconnects and topologies, which cannot be expressed in a pre-defined parameterizable architectural template.

4. Synthesis of Datapath onto CGRA

For CGRA synthesis, the first input is the application's DFG-based representation. The second input is the CGRA description. The output is the configuration bitstream for simulating the application on CGRA's RTL implementation. The various phases of CGRA synthesis flow are discussed below.

4.1 Mapping and Clustering

On the assumption of general delay model [22], delay-optimal Simultaneous Mapping And Clustering (SMAC) algorithm for Look-Up Table (LUT)-based FPGAs is proposed in [18]. In the current work, SMAC is extended to fit the coarse-grained scenario. Our extended algorithm, dubbed as CG-SMAC (Coarse-Grained SMAC), is distinguished by several features. Firstly, for each node of the input dataflow graph, SMAC generates a new possible cluster entry or checks if it can be fit into already existing cluster entries. CG-SMAC, in addition, checks all possible graph-matching options rooted at the current node. Accordingly, new or existing entries are created/modified. This guarantees a delay-optimal clustering decision even in presence of varied possible library patterns. Secondly, in CGRA several diverse logic blocks can be grouped together to form a cluster unlike more homogeneous structure of the fine-grained FPGAs. To account for such an heterogenous structure as well as the designer-defined connectivity constraints - in-cluster placement and routing are done during CG-SMAC. Finally, a first-fit heuristic for obtaining area-optimized mapping solutions is integrated in CG-SMAC, too. The synthesis steps are presented in the following.

The inputs to the CG-SMAC algorithm are (i) the input datapath (I_d), (ii) the CGRA connectivity constraints, (iii) the CGRA pattern library, which is automatically generated out of the *elements*' behavior description. The output of the CG-SMAC is the mapping and clustering decisions. The input datapath is represented in Data Flow Graph (DFG) format, which is generated automatically by parsing a C description (either from an application or from a processor's datapath description). The CG-SMAC algorithm can be decomposed into two subsequent phases. The *labelling* phase and the *cluster realization* phase. The labelling phase traverses the input datapath from the Primary Input (PI) nodes to the Primary Output (PO) nodes in topological order and computes the possible arrival times given that node's possible allocation in a cluster. In the second phase of CG-SMAC i.e. the cluster realization phase, the nodes are traversed from PO to PI and allocated into the cluster with minimum arrival time possible for that node. The overall algorithm is based on dynamic programming i.e. for each node of the input datapath, all the possible clustering solutions are outlined to avoid locally optimal results.

Labelling : During the labelling phase, it is first necessary to enumerate all possible mapping solutions rooted at a particular node n for the given library patterns. This step can be considered similar to the K-feasible cut generation of SMAC (for a K-LUT). The mapping solutions are generated via graph mapping against the pattern library. An exemplary graph mapping flow is illustrated with the figure 5. In this example, the node n_6 of the input graph is matched against the available patterns. A node is first matched with the root node of the pattern graph. A node-to-node matching is done by checking the operator (size, type) and the number of inputs. In the following iterations, the matched pattern graphs are traversed from root node upwards level-wise to check if those can completely cover a sub-graph of (I_d). Only in the case of complete cover, a pattern is considered to be a match for the current node. In the figure, several such matches (within rectangular boxes) are shown for pattern graphs P_1, P_2 and P_3 whereas no match is found for P_4.

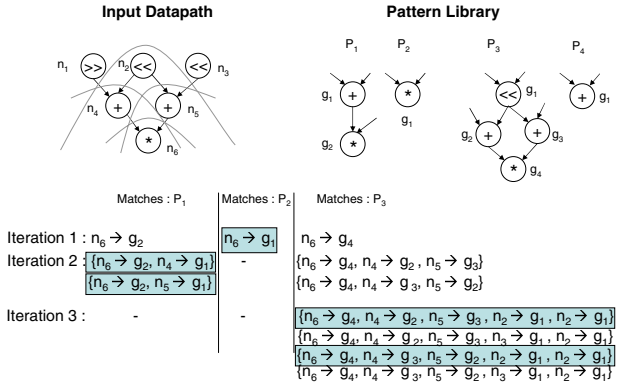

Figure 5. Graph Mapping during Labelling

For each of the graph-mapping results, at a node $node_{curr}$ an empty clustering solution $cluster_{new}$ is created (which does not require placement and routing checks) or the clustering solution $cluster_i$ from its predecessor nodes are checked if it can accommo-

date $mapping\_solution$ of $node_{curr}$. The predecessor node has to be considered on the basis of mapping solution currently being considered. The check with accommodation within predecessor nodes' clustering solutions actually resembles the clustering capacity check done in LUT-based FPGA synthesis. A marked difference is that, for CGRA, the clustering capacity can be defined in terms of the various elements available within the cluster as well as the connectivity constraints allowed. To address this problem, an in-cluster place-and-route is performed to check if the clustering capacity is met (algorithm 1). The algorithm is run over each mapping solution of the current node. The key part of the algorithm is to decide if a predecessor node's existing clustering solution do have capacity to take the current node's mapping solution. This is done first by checking if the cluster has an unfilled logic block corresponding to the mapping solution. In that case, the mapping solution is added to form a new cluster. However, this is not sufficient. Given the existing cluster's connectivity restrictions the newly added mapping solutions may not be routable. This is checked via generating all possible placement combinations within the scope of $cluster_{pred}$ and performing routing. Out of the various possible placements, the one with minimum routing cost is added to the current node's possible clustering solutions. For each clustering solution, an arrival time is calculated on the basis of a delay model and the parent nodes' arrival times.

Algorithm 1: In_Cluster_PlaceAndRoute

```
begin
    cluster_new = node_curr.newCluster (mapping_solution);
    node_curr.appendClusterSolution(cluster_new);
    pred_node_list = getParentNodes(mapping_solution);
    foreach node_p ∈ pred_node_list do
        cluster_list = getClusterSolutions(node_p);
        foreach cluster_i ∈ cluster_list do
            if cluster_i can take mapping_solution then
                cluster_pred =
                addMapping(cluster_i,mapping_solution);
                min_cost = infinity;
                possible_placements =
                getAllPlacements(cluster_pred);
                foreach placed_i ∈ possible_placements do
                    routed_i = route(placed_i, routable);
                    routing_cost = computeCost();
                    if routable == true and routing_cost <
                    min_cost then
                        min_cost = routing_cost;
                        routed_best = routed_i;
                node_curr.appendClusterSolution(routed_best);
end
```

Cluster Realization : Using the graph mapping and in-cluster placement and routing, the complete labelling (arrival time calculation) of input DFG is done. This is followed by the cluster realization phase, where the algorithm traverses from the PO to the PI nodes and at each node selects the optimum clustering decision. In this work, delay-optimal clustering decision is chosen as in original SMAC. During cluster realization conflicting clustering requirements may exist at nodes with multiple fan-outs [18]. In such cases, nodes are duplicated.

Area Optimization : To obtain an area-optimized version of CG-SMAC, we resorted to a variant of first-fit heuristic. During the labelling phase, each node is annotated with the number of clusters it currently consumes along the path from PI. During cluster realization, the clustering decisions with least cluster count are chosen.

4.2 Placement and Routing

To solve the non-trivial placement problem (NP-hard), several effective heuristics are proposed in literature. As the routing strategy is strongly influenced by the FPGA placement, it is customary to include a routing validation [6] or even the complete routing [23] inside each iteration of placement algorithm. For this work, a simulated annealing based placement heuristic is applied, with each iteration performing a state-of-the-art negotiation-based routing algorithm [19] internally. At the end of placement and routing, clusters are placed in the CGRA with definite routing paths. The routing is done either via direct connections between neighbouring elements/clusters or via an empty element/cluster. During the placement and routing implementation, the routing resources are modelled as per the connectivity specifications. Exemplarily, for every dedicated connection between two clusters, an intermediate routing resource is assumed, which can be shared among the cluster's internal elements for porting data. This is a notable extension in our implementation since, it follows a diverse set of topology and connectivity constraints directly from the CGRA description. After the placement and routing, the configuration information from each port is loaded from the CGRA description and configuration bitstream is generated automatically.

4.3 Synthesis on Non-clustered Heterogeneous CGRA

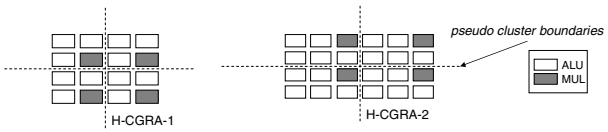

Figure 6. Heterogeneous CGRAs

The proposed description style permits modelling of a heterogeneous CGRA by allowing various elements to be grouped in a cluster and various different clusters forming the complete CGRA. However, adapting CG-SMAC and the placement-and-routing algorithm for heterogeneous CGRA is a major challenge as those are designed with clustered LUT-based FPGAs in view. One possible alternative is to model the entire heterogeneous CGRA as one single large cluster. This will pose difficulty to the in-cluster placement and routing phase, which permutes over all possible placement positions. The chosen alternative is to determine some form of homogeneity within the heterogeneous CGRAs and thereby, impose *pseudo cluster boundaries*. In this case, the inter-cluster routing cost is considered equal to the intra-cluster one. This enables the entire CGRA synthesis flow presented in this section to be applied to heterogeneous CGRAs. Two CGRAs, with pseudo cluster boundaries are shown in the figure 6.

It must be noted that the CGRA synthesis flow presented here is not dependent on the proposed modelling formalism or vice-versa. A different compilation approach (e.g. [6]) can be applied here, too. The reason for selecting the SMAC [18] as basis for CGRA synthesis is that, it proposed delay-optimal mapping and clustering unless any of the known CGRA synthesis solutions. Furthermore, dealing the CGRA synthesis independently from the processor's C Compiler (unlike [2]) allows the designer the freedom to map the application directly or via the intermediate ADL structure.

5. Implementation Flow

Algorithm 2: CGRA_RTL_Implementation

```
begin
    E_top = parse(CGRA_Description);
    connectPaths(E_top);
    connectConfiguration(E_top);
    HDL_Description = generateHDL(E_top);
end
```

The CGRA description is subjected to a tool-flow developed for this work in order to generate synthesizable RTL description. The major phases of the RTL implementation is presented in form of pseudo code in algorithm 2. At the beginning, the CGRA description is parsed. Followed by parsing, the module hierarchy is established, returning the top-level module. Then, the connectivity section is accessed to establish the link between the ports of different modules via `connectPaths` function. This function also infers the necessary multiplexer modules locally. The function `connectConfiguration` traverses the entire module hierarchy to determine the multiplexing points, thereby creating configuration bits for each of those. The configuration bits are hierarchically accumulated and connected to a global configuration port. The entire data-structure, built so far, is then subjected to HDL back-end in order to generate an RTL description.

6. Case Study

The algorithm kernels selected for our experiments are FFT Butterfly (BFLY), FIR (8-tap), DES and IDCT. FFT and FIR are well-known algorithm kernels widely used in communication and digital signal processing. DES is a block cipher algorithm with 64-bit block size. DES consists of 16 identical processing stages, referred as *rounds*. The block targeted for CGRA exploration is one such DES-round. IDCT is a fourier-related transformation, often used for signal and image processing applications, especially for lossy data compression. It has two components reflecting similar traits, namely IDCT-row operations and IDCT-column operations. For the experimentation described in the following, IDCT-row function is taken.

Delay Model : In the entire case study section, a cycle-based cost model with inter-cluster routing delay set to 2 cycles and intra-cluster routing delay set to 1 cycle is used (DM1 cost model in [6]). Only in the case of heterogeneous CGRAs, where *pseudo cluster boundaries* are set up, the inter-cluster routing delay is set to be same as the intra-cluster routing delay, both being 1 cycle.

In order to compare between different architectural styles, we first modelled, in a limited way, the well-known coarse-grained reconfigurable architectures. The architectural features are summarized in the table 1. For CGRA-1, the cluster-level connectivity of MESH-1 is used, whereas for the rest the cluster-level connectivity is not relevant. The basic element used in all these architectures is a 32-bit ALU with arithmetic and logical operators inside those. The input and output ports of the basic elements can be registered or bypassed. The connectivity style and connectivity strides are also indicated in the table. For example, the MATRIX architecture supports a connectivity style of nearest neighbour (NN) with a stride of 1, mesh with a stride of 2, row-wise (ROW) and column-wise (COL) both with a stride of 4. Actually, the row-wise and column-wise 4-hop connection in the MATRIX architecture is present in alternative fashion, which is simplified for this study.

Architecture	Cluster Size	CGRA Size	CGRA-level Connectivity	Reflecting Topology of
CGRA-1	2x2	8x8	MESH-1	DReAM
CGRA-2	1x1	8x8	NN-1, MESH-2, ROW-4, COL-4	MATRIX
CGRA-3	1x1	8x8	MESH-1, ROW-1, COL-1	MorphoSys

Table 1. Instances reflecting known CGRAs

Two of the algorithm kernels are synthesized with the aforementioned CGRAs. From the results given in table 2, it is not hard to find out the following notes. Firstly, in the architecture which is close to DReAM, for both applications, the number of clusters used after placement and routing is much more than the number of clusters used before that. The reason for this is, since MESH connectivity is used in CGRA level, a lot of extra clusters are used for routing purpose. How many extra clusters are used for routing depends on the kind of connectivity style and the routing capacity of clusters. Therefore, further exploration of connectivity style in CGRA level is necessary to achieve better results. Secondly, in both the architectures CGRA-2 and CGRA-3, a cluster size of 1x1 is used. In such a case, the in-cluster configuration does not call for any exploration. The performance after placement and routing depends on the CGRA-level connectivity. A better performance is achieved for the architecture with MATRIX-like connectivity due to more availability of routing resources.

Architecture	Application	No. of Clusters		Critical Path (cycles)
		before P&R	after P&R	
CGRA-1	IDCT-row	26	42	53
	DES-round	12	23	23
CGRA-2	IDCT-row	58	62	24
	DES-round	28	31	24
CGRA-3	IDCT-row	58	67	32
	DES-round	28	35	30

Table 2. CGRA Synthesis Results

For the CGRA to have a right balance of performance and flexibility, it is imperative to select the basic elements, routing architecture and structural topology prudently. With different applications, it turned out that the performance varies with different architectures. To understand this variation, we begin the exploration with a simple architecture with a cluster size of 2x2. In each cluster, three basic elements are used namely, ALU (for arithmetic operations), CLB (for logic operations) and MULT (for multiplication). The cluster is optionally equipped with a multiplexer block to enable control flow mapping. The arrangement of the elements inside cluster is as shown in the figure 7 (CGRA-1). On that basis, the CGRA-level connectivity is varied to obtain the results as presented in table 3. Clearly a rich interconnect structure reduces the critical path but, not for all kernels. Interestingly, for DES-round, a connectivity style of NN-1 achieves better critical path as well as cluster count than a connectivity style of {MESH-1, NN-2}. This reflects the application data-flow organization, which is much denser than can be supported by MESH with 2-stride NN.

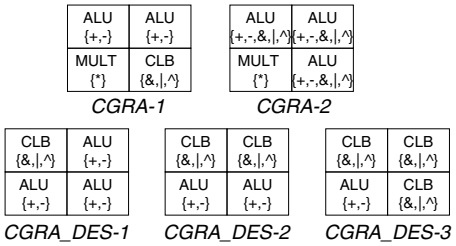

Figure 7. CGRAs for Exploration

Application	No. of Clusters		Critical Path (cycles)	Connectivity Style
	before P&R	after P&R		
BFLY	10	16	11	MESH-1
	10	11	7	NN-1
	10	13	7	MESH-1, NN-2
	10	10	5	NN-1, MESH-2
FIR	8	10	17	MESH-1
	8	8	12	NN-1
	8	8	12	MESH-1, NN-2
	8	8	12	NN-1, MESH-2
IDCT-row	31	44	39	MESH-1
	31	40	33	NN-1
	31	38	29	MESH-1, NN-2
	31	37	21	NN-1, MESH-2
DES-round	20	30	45	MESH-1
	20	23	23	NN-1
	20	25	25	MESH-1, NN-2
	20	21	21	NN-1, MESH-2

Table 3. CGRA Synthesis : Effect of Connectivity

In this experiment the functionality of the elements are altered. This can be performed easily by modifying the `OPERATOR_LIST` of an element. Here, the operators defined in CLB are moved into ALU and the CLB is replaced with another ALU. Now, the architecture becomes the CGRA-2 of figure 7. The results are shown in table 4.

Application	No. of Clusters		Critical Path (cycles)	Connectivity Style
	before P&R	after P&R		
BFLY	9	15	10	MESH-1
	9	12	6	NN-1
	9	11	6	MESH-1, NN-2
	9	9	5	NN-1, MESH-2
FIR	8	10	14	MESH-1
	8	8	10	NN-1
	8	8	10	MESH-1, NN-2
	8	8	10	NN-1, MESH-2
IDCT-row	26	41	33	MESH-1
	26	37	28	NN-1
	26	34	23	MESH-1, NN-2
	26	31	15	NN-1, MESH-2
DES-round	14	24	28	MESH-1
	14	18	22	NN-1
	14	17	20	MESH-1, NN-2
	14	14	15	NN-1, MESH-2

Table 4. CGRA Synthesis : Effect of Functionality

Compared to the original structure, the functionality modification allowed the arithmetic operations and logic operations in the application to fit into the same element. Because one more ALU element is now available inside the cluster, chances of more arithmetic or logic operations to be put into one cluster is increased. Therefore, better mapping results are easily found in BFLY, IDCT-row and DES-round. However, there is not much difference for FIR. This is since there are no logical operations in FIR.

To show the effect of varying element numbers in a cluster, the DES-round kernel is chosen. In DES-round, there are only logic and arithmetic operations, which are distributed in a ratio of roughly 3 to 1. In this experiment, three architectures from figure 7, with all of those having {NN-1,MESH-2} connectivity at CGRA level. From the results (refer table 5), it can be observed that, a good architectural decision is based on the proper application characterization. The selection of element type and number of elements inside cluster should follow the basic characteristics of application e.g. the ratio of operators inside application. Here, when the ratio of elements of corresponding type is close to the ratio of operators in application, a better synthesis result is achieved.

Architecture	No. of Clusters		Critical Path
	before P&R	after P&R	(cycles)
CGRA_DES-1	20	22	23
CGRA_DES-2	14	14	16
CGRA_DES-3	11	11	13

Table 5. CGRA Synthesis : Effect of Diversity

By trading-off between the results of all the kernels and taking the effects which are analyzed above, CGRA-2 is found to be the best performing one. For area consideration, the MULT is kept as a separate element out of ALU. Considering the characteristics of BFLY and FIR application, only one MULT is arranged inside cluster. Since the logical and arithmetic (w/o multiplication) operations dominate in BFLY, IDCT-row and DES-round, three ALU which includes both logic and arithmetic operations are put inside cluster. An NN-1 connectivity in cluster level and an {NN-1, MESH-2} connectivity at CGRA level is used. For this CGRA, area-optimized version of CG-SMAC is applied to observe the effect. The results are recorded in table 6. The synthesis results without area optimization are indicated within square brackets. Better area results are obtained in all cases. Expectedly, a degradation of critical path is also observed. This area-optimized version of CG-SMAC can be employed suitably when the CGRA size is fixed beforehand and/or when the delay constraints are less strict. An interesting follow-up work can be to perform area-optimization in non-critical paths as in [15].

Application	No. of Clusters [w/o opt.]		Critical Path
	before P&R	after P&R	(cycles) [w/o opt.]
BFLY	8 [9]	8 [9]	7 [5]
FIR	7 [8]	7 [8]	11 [10]
IDCT-row	23 [26]	28 [31]	21 [15]
DES-round	12 [14]	12 [14]	23 [15]

Table 6. CGRA Synthesis : Area-optimization

For experimenting with heterogeneous CGRA structures, the application FIR is chosen with the architectures being same as presented in figure 6. An overall MESH-1 connectivity style is chosen with the inter-cluster and intra-cluster routing delay set as 1 cycle. For the H-CGRA-2 architecture, the MULT elements are placed more sparsely, which made the routing path longer. This resulted in higher number of clusters as well as longer critical path (table 7).

Architecture	No. of Clusters after P&R	Critical Path (cycles)
H-CGRA-1	20	8
H-CGRA-2	22	10

Table 7. Heterogeneous CGRA Synthesis

Though the algorithms used in the CGRA synthesis flow are computation-intensive, the relatively less complexity of interconnects in CGRA compared to fine-grained FPGAs allowed all the presented case studies to be synthesized in reasonable time. In a AMD Athlon Dual Core Processor (each running at 2.6 GHz), the case study applications finished within 1 (FIR, 15 operators) to 15 minutes (IDCT-ROW, 58 operators) for CGRA-2.

The architectures used in this case study are synthesized to obtain RTL description, followed by gate-level synthesis with Synopsys Design Compiler. For comparison's sake, we present the synthesis results for the architecture CGRA-2. CGRA-2 is synthesized with total 25 (5x5) clusters. For the designer-specified connectivity, total 2542 configuration bits are required to control CGRA-2. After gate-level synthesis, the entire architecture met a clock constraint of 5 ns (with register attributes at element's outports) for 130 nm process technology (1.2 V) and occupied an area of approximately 3.77 mm^2 of which 1.62 mm^2 area is consumed by the CGRA-level routing alone. It should be noted that this synthesis figures are bound to improve significantly after physical optimization and by using special library cells, evidently for, the routing architecture.

7. Conclusion and Future Work

In this paper, a generic modelling language for describing CGRA is presented. To perform design space exploration for different architectures, a complete synthesis as well as RTL generation flow from this high-level language is developed. The synthesis flow is based on that applied in fine-grained FPGA synthesis domain. The entire CGRA design flow is integrated with a state-of-the-art processor design framework to enable partially re-configurable processor design. We plan to incorporate several features in the CGRA description formalism e.g. the possibility of buffering data inside a cluster (only one register port is supported now). The CG-SMAC algorithm can be improved, in particular, with solution space pruning techniques. Several design points of the CGRA are currently supported by the generic modelling formalism without any support from the synthesis and/or implementation flow. We will like to extend the tooling for that, too.

8. REFERENCES

[1] Stretch, *http://www.stretchinc.com*.

[2] B. Mei, A. Lambrechts, D. Verkest, J. Mignolet and R. Lauwereins, "Architecture Exploration for a Reconfigurable Architecture Template," *IEEE Design and Test*, vol. 22, no. 2, pp. 90–101, 2005.

[3] H. Singh, M. Lee, G. Lu, F. J. Kurdahi, N. Bagherzadeh and E. M. Chaves Filho, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 465–481, 2000.

[4] A. S. Y. Poon, "An Energy-Efficient Reconfigurable Baseband Processor for Flexible Radios," in *IEEE Workshop on Signal Processing Systems Design and Implementation*, 2006.

[5] MathStar, *http://www.mathstar.com/*.

[6] N. Bansal, S. Gupta, N. Dutt and A. Nicolau, "Analysis of the Performance of Coarse-Grain Reconfigurable Architectures with Different Processing Element Configurations," in *Workshop on Architecture Specific Processors (WASP)*, 2003.

[7] R. Hartenstein, M. Herz, T. Hoffmann and U. Nageldinger, "KressArray Xplorer: a new CAD environment to optimize Reconfigurable Datapath Array," in *Proceedings of the conference on Asia South Pacific Design Automation*, 2000.

[8] G. Dupenloup, T. Lemeunier and R. Mayr, "Transistor Abstraction for the Functional Verification of FPGAs," in *Proceedings of DAC*, 2006.

[9] B. Mei, S. Vernalde, D. Verkest and R. Lauwereins, "Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study," in *Proceedings of the conference on Design, Automation and Test in Europe*, 2004.

[10] G. Dimitroulakos, M. D. Galanis, N. Kostaras and C. E. Goutis, "A Unified Evaluation Framework for Coarse Grained Reconfigurable Array Architectures," in *Proceedings of theInternational Conference on Computing Frontiers*, 2007.

[11] T. von Sydow, M. Korb, B. Neumann, H. Blume and T. G. Noll, "Modelling and Quantitative Analysis of Coupling Mechanisms of Programmable Processor Cores and Arithmetic Oriented eFPGA Macros," in *Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGA*, 2006.

[12] M. Ahn, J. W. Yoon, Y. Paek, Y. Kim, M. Kiemb and K. Choi, "A Spatial Mapping Algorithm for Heterogeneous Coarse-grained Reconfigurable Architectures," in *Proceedings of DATE*, 2006.

[13] G. Dimitroulakos, M. D. Galanis and C. E. Goutis, "Design Space Exploration of an Optimized Compiler Approach for a Generic Reconfigurable Array Architecture," *Journal of Supercomputing*, vol. 40, no. 2, pp. 127–157, 2007.

[14] R. W. Hartenstein and R. Kress, "A Datapath Synthesis System for the Reconfigurable Datapath Architecture," in *Proceedings of ASP-DAC*, 1995.

[15] T. J. Callahan, P. Chong, A. DeHon and J. Wawrzynek, "Fast Module Mapping and Placement for Datapaths in FPGAs," in *FPGA '98*.

[16] Y. Guo, G. J.M. Smit, H. Broersma and P. M. Heysters, "A Graph Covering Algorithm for a Coarse Grain Reconfigurable System," in *Proceedings of the Conference on Language, Compiler and Tool for Embedded Systems*, 2003.

[17] T. von Sydow, B. Neumann, H. Blume and T. G. Noll, "Quantitative Analysis of Embedded FPGA-Architectures for Arithmetic," in *Proceedings of the 17th International Conference on Application-specific Systems, Architectures and Processors*, 2006.

[18] J. Y. Lin, D. Chen and J. Cong, "Optimal Simultaneous Mapping and Clustering for FPGA Delay Optimization," in *Proceedings of DAC*, 2006.

[19] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-based Performance-driven Router for FPGAs," in *Proceedings of the International Symposium on Field-programmable Gate Arrays*, 1995.

[20] G. Chen and J. Cong, "Simultaneous Placement with Clustering and Duplication," in *Proceedings of DAC*, 2004.

[21] CoWare/LISATek, *http://www.coware.com*.

[22] R. Murgai, R.K. Brayton and A. Sangiovanni-Vincentelli, "On Clustering for Minimum Delay/Area," in *Proceedings of ICCAD*, 1991.

[23] A. Sharma, C. Ebeling and S. Hauck, "Architecture Adaptive Routability-Driven Placement for FPGAs," in *Proceedings of the International Symposium on Field-programmable Gate Arrays*, 2005.