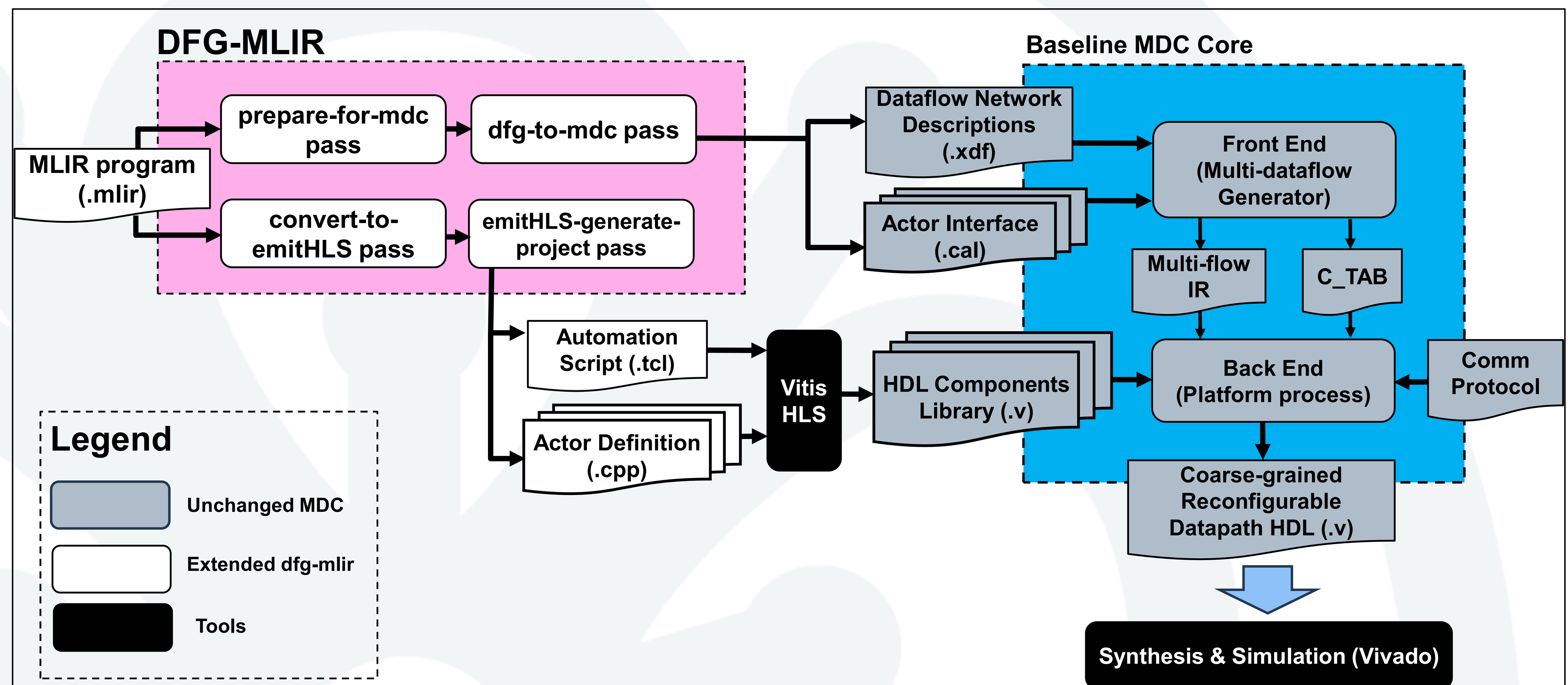
Dataflow-based FPGA Accelerators Generation via MLIR and Multi-dataflow Composer tool

[1]Mohammad Cheshfar, [2]Jiahong Bi, [1]Francesco Ratto
[1]Università degli Studi di Cagliari, [2]Technische Universität Dresden
mohammad.cheshfar@unica.it

*This work presents an automated compilation flow for generating synthesizable **FPGA**[2] accelerators from high-level dataflow specifications using the **MLIR**[1] framework and the **MDC**[3] **tool**. The flow leverages the **dfg-mlir** dialect to model static dataflow graphs and integrates a dedicated lowering pipeline to generate project files. Actor implementations are synthesized automatically using the **emitHLS** dialect which **MDC-compatible**can target **the Vitis HLS** tool, enabling compatibility with **FPGA** toolchains without manual **HDL**[4] coding.To validate the proposed automated approach, a comparison with manual HDL-based flow is carried out. **FPGA** accelerators are synthesized targeting the **AMD Kria KV260** platform. Resource usage results confirm functional equivalence and comparable resource utilization and computation time. The integration with the **MDC** tool opens up the possibility of generating coarse-grained reconfigurable accelerators, which will be explored in future work.*

Proposed work: dfg-mlir extension to target coarse grained reconfigurable accelerator through MDC tool.

- **Prepare-for-mdc** and **dfg-to-mdc**, integrated within the MLIR infrastructure, transforms **MLIR program** into a dataflow specification compatible with the MDC toolchain(**XDF and CAL**).
- **emitHLS-generate-project** pass produces synthesizable Verilog actor modules using **Vitis HLS** (standard C++ expressions or classes, compatibility with the **AMD design toolchains** such as **Vivado** and **Vitis HLS**).
- resulting hardware was packaged into a **Vivado project** for simulation and synthesis (verification of functionality and hardware compatibility) .

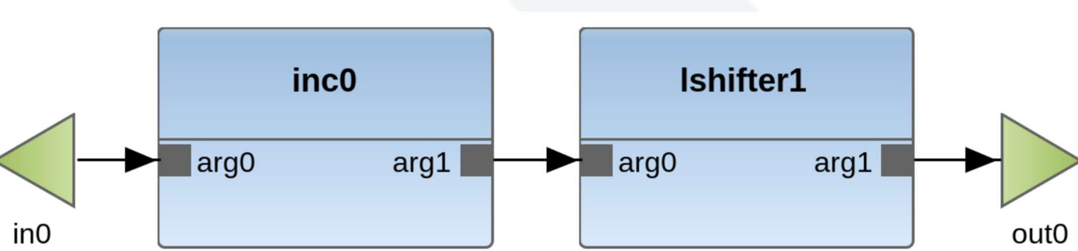


An MLIR program example

```
dfg.operator @inc inputs(%in: i16) outputs(%out: i16) {
%0 = arith. constant 1 : i16
%1 = arith.addi %in, %0 : i16
dfg.output %1: i16
}
dfg.operator @lshifter inputs(%in: i16) outputs(%out: i16) {
%0 = arith. constant 2 : i16
%1 = arith.muli %in, %0 : i16
dfg.output %1: i16
}
dfg.region @top inputs(%arg0: i16) outputs(%arg2: i16) {
%0:2 = dfg.channel(1) : i16
dfg.instantiate @inc inputs(%arg0) outputs(%0#0) : (i16) -> i16
dfg.instantiate @lshifter inputs(%0#1) outputs(%arg2) : (i16) -> i16
}
```

Representation of the example in MDC

XDF file for network topology representation



CAL files for actors interfaces

lshifter.cal

```
package custom;
actor lshifter()
  int(size=16) arg0
  ==>
  int(size=16) arg1:
end
```

inc.cal

```
package custom;
actor inc()
  int(size=16) arg0
  ==>
  int(size=16) arg1:
end
```

Result

To quantitatively assess the two approaches, post-synthesis resource utilization results on the Kria KV260 platform, comparing the automated toolchain with the manually written HDL in the following table. Reported metrics include the number of Look-Up Tables (LUTs), Flip-Flops (FFs), and LUT-based memories (LUTRAMs), along with execution time and maximum achievable frequency (fmax). Percentages in parentheses indicate the share of total available device resources consumed by each design. As shown, both the automated pipeline and the manually written HLS consume nearly identical hardware resources.

Synthesis Resource Utilization on Kria KV260

Method	LUTs	FFS	LUTRAMs	Execution time (Cycle)	F_{max}
Automated	115(0.10%)	63(0.03%)	38(0.07%)	65	190
Manual	116(0.10%)	63(0.03%)	38(0.07%)	65	165

Conclusion and Future Work

- **Integrated MDC toolchain with MLIR** to enable a unified and extensible compilation flow for coarse-grain reconfigurable hardware accelerators.
- **Implemented MDC as a target in dfg-mlir**, making it accessible from MLIR-based compiler pipelines.
- **Validated integration with a dataflow kernel**, showing no performance loss compared to manually optimized designs.
- **Future directions:** extend evaluation to diverse dataflow applications, exploit MDC's resource sharing, and **embed resource-sharing algorithms as MLIR optimization passes**.

1- Multi-Level Intermediate Representation, 2- Field Programmable Gate Arrays, 3- Multi-Dataflow Composer 4- Hardware Description Language

Funded by the European Union

www.myrtus-project.eu https://www.linkedin.com/company/myrtus-eu/ @MYRTUS_EU

Funded by the European Union, by grant No. 101135183. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.