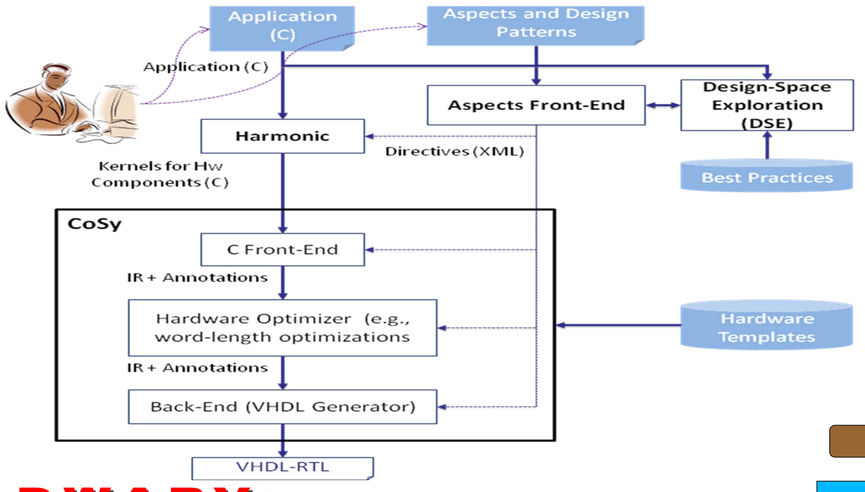
DWARV: A HDL Compiler with Support for Scheduling Custom IP Blocks

Razvan Nane, Vlad Mihai Sima, Hans van Someren and Koen Bertels

General Context

- This work is part of the FP7 REFLECT project [1], where Aspect Oriented Specifications (AOS) and Design Patterns (DP) are used to complement the HW/SW co-design tool-flow depicted in the figure below.
- We focus on the integration of IP blocks in applications using the DWARV hardware compiler [2].
- To achieve the required performance, efficient scheduling of shared resources is needed.



DWARV:

- DelftWorkbench Automated Reconfigurable VHDL Generator
- Input: Annotated C
- Output: VHDL
- Platform: Molen Machine Organization [3]

CoSy:

- Commercial, easy-targetable and highly flexible compiler development system [4]
- Provides standard transformations, e.g. SSA
- Provides engines to extract CFG and DFG, e.g. DDG
- Allows simple RULE based code generation

❑ **HW template IP information**

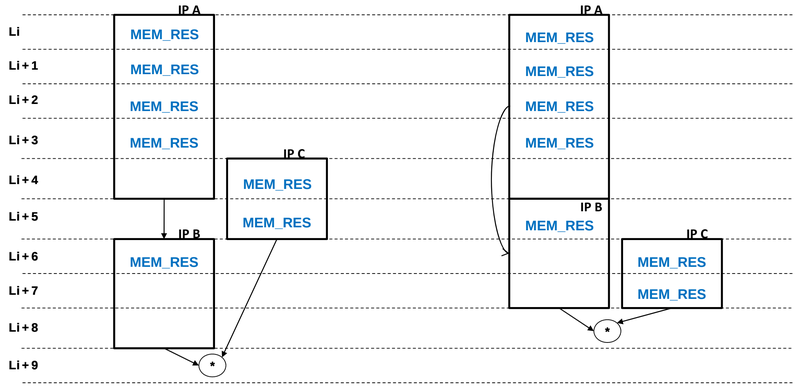
- name template
- input operand types
- input operand sizes
- input operand port names
- output operand type
- output operand size
- output operand port name
- component delay
- auxiliary port names
- <cycle, resource type>

example:

```
#start_func
 #start_operations
  muladd_32
 #end_operations
 #start_operand
 $role=@in_0
 #start_instance
 $operation=muladd
 $type=@realt
 $size=32
 $port=in_0
 #end_instance
 #end_operand
 #start_operand
 $role=@in_1
 #start_instance
 $operation=muladd
 $type=@realt
 $size=32
 $port=in_1
 #end_instance
 #end_operand
#end_operand
 #start_operand
 $role=@in_2
 #start_instance
$operation=muladd
 $type=@realt
 $size=32
 $port=in_2
 #end_instance
 #end_operand
 #start_operand
 $role=@result
 #start_instance
$operation=muladd
 $type=@realt
 $size=32
 $port=result
 #end_instance
 #end_operan
  $frequency=15
  $cycles=13
$pipeline_interval=1
 $library_name=work
 $component_name=
fp_sp_mul_add_top
 $reset=reset
 $clock=clk
 #start_aux_port
 $name=uflow
 $direction=@out
 $width=1
 #end_aux_port
 #start_aux_port
 $name=oflow
 $direction=@out
 $width=1
 #end_aux_port
 #end_func
```

❑ **Illustration of the two possible schedules for the 3 templates**

- left schedule takes 10 cycles to complete
- right schedule takes 9 cycles to complete

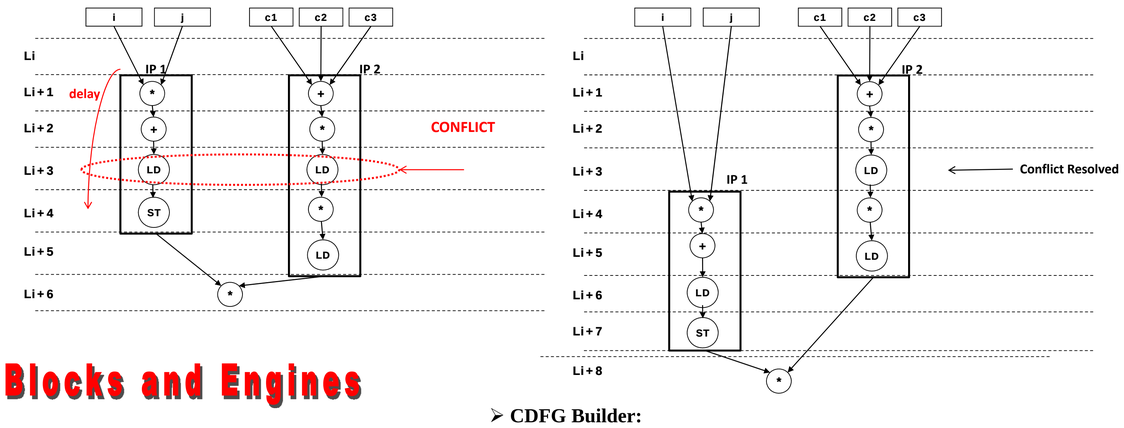


Applications available:

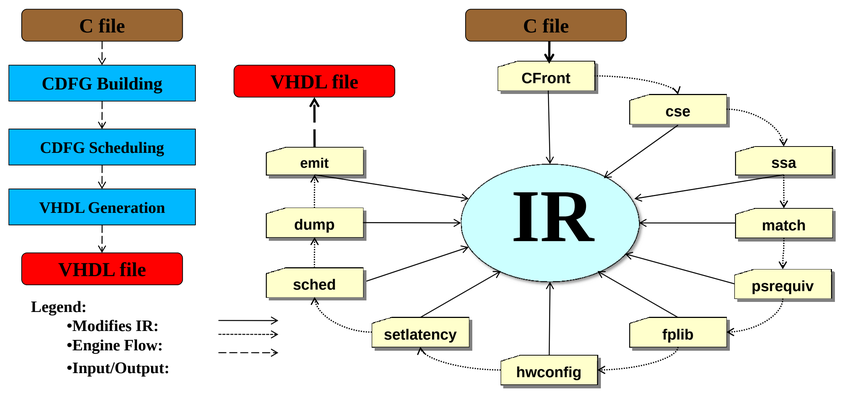
- MPEG Encoder
- Stereo Navigation
- 3D Path Planning

Problem Definition

The library approach can be used for coarse grained, previously developed or third party IP blocks. However, integrating such library elements, i.e. scheduling them, is not a straightforward task. The fact that some shared hardware resources require exclusive accesses, accurate timing information of each shared HW resource usage is needed for efficient scheduling. If such timing information is not available, the corresponding IP block will need to be scheduled in stand-alone mode, thereby reducing the overall performance.



DWARV Building Blocks and Engines



TEMPLATE RESOURCES:
MEM_RES
TEMPLATES:
IP_A := MEM_RES & MEM_RES & MEM_RES & MEM_RES & ();
IP_B := MEM_RES & () & ();
IP_C := MEM_RES & MEM_RES;

RULE ASSOCIATION:

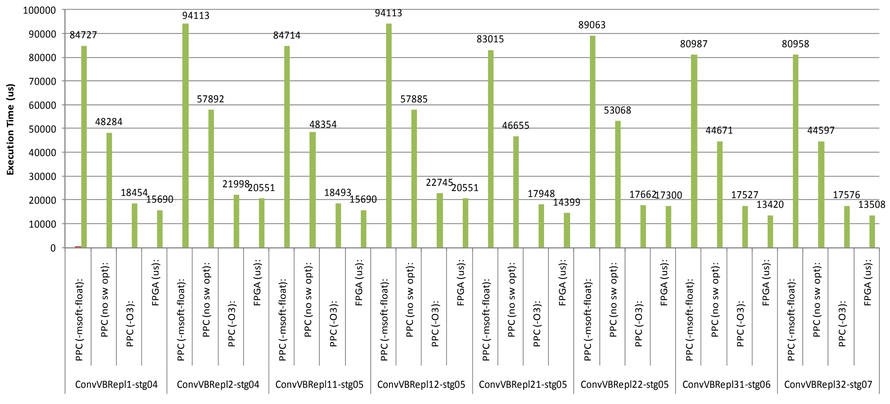
RULE [func_call_direct] call : mirFuncCall (…)
CONDITION {…};
TEMPLATE IP_A;
EXPAND emit {
……
fprintf(" %s_%s%d <= %s; \n", IPname, IPinputport, IPid, VHDLvariable);
……
};
END

- **CDFG Builder:**
  CFront creates and initializes the IR from the source code
  CSE does common sub-expression elimination
  SSA does static single assignment
  psrequiv is a custom written engine for custom FPGA register allocation
- **CDFG Scheduler:**
  fplib loads FP and IP cores from an external library file, e.g. FP_ADD
  hwconfig sets platform-dependent parameters, e.g. memory latency
  setlatency adjusts the length of DFGraph's edges with true latencies
  sched is CoSy standard engines that schedules the graph
- **VHDL Generation:**
  emit prints vhdl code of the encountered RULEs corresponding to scheduled graph's nodes.

```
void schedule(void) {
        if ( schedule finished ) {
                determine whether the found schedule is best so far and if so save it.
        }
        else for each candidate instruction C {
                // updates scheduler state based on defined UserPriority
                doschedule(c);
                schedule();
                unschedule(); // restores scheduler state
        }
}

int gcg_UserPriority(c) {
        if (no successor instruction nodes )        return 1;
        else for each successor instruction S {
                S_cost = gcg_UserPriority(s);
                if ( tmp_cost < S_cost )
                        tmp_cost = S_cost;
        }
        return C_cost + tmp_cost; //C_cost is the total latency of instr uctionC
}
```

Initial Results



Benefits:

- Allows user to fine-tune the application wrt. performance and area utilization.
- No domain restrictions.
- Reuse of IP blocks controlled from C-code.
- Enables fast DSE at a higher abstraction level.
- Can decrease code maintenance.

Future Work:

- Investigate consortium applications and extract useful IP blocks.
- Construct the resource templates dynamically.
- Implement mechanisms for template extension.
- Extend and test the scheduling algorithm.

References:

[1] REFLECT Project. Available: http://www.reflect-project.eu/
[2] Y.D. Yankova, G.K. Kuzmanov, K.L.M. Bertels, G.N. Gaydadjiev, J. Lu and S. Vassiliadis. *DWARV: Delft Workbench Automated Reconfigurable VHDL Generator*, FPL '07.
[3] S. Vassiliadis, S. Wong, G.N. Gaydadjiev, K. Bertels, G. Kuzmanov and E.M. Panainte *The Molen Polymorphic Processor.* In IEEE Transactions on Computers (November '04).
[4] Associated Compiler Experts ACE: *CoSy compiler platform*. Available: www.ace.nl

Acknowledgement:

This research is supported by FP7 REFLECT project (grant 248976)

Contact Person:
Razvan Nane
Email: r.nane@tudelft.nl

RENDERING FPGAS FOR MULTI-CORE EMBEDDED COMPUTING
REFLECT

ACE

TU Delft