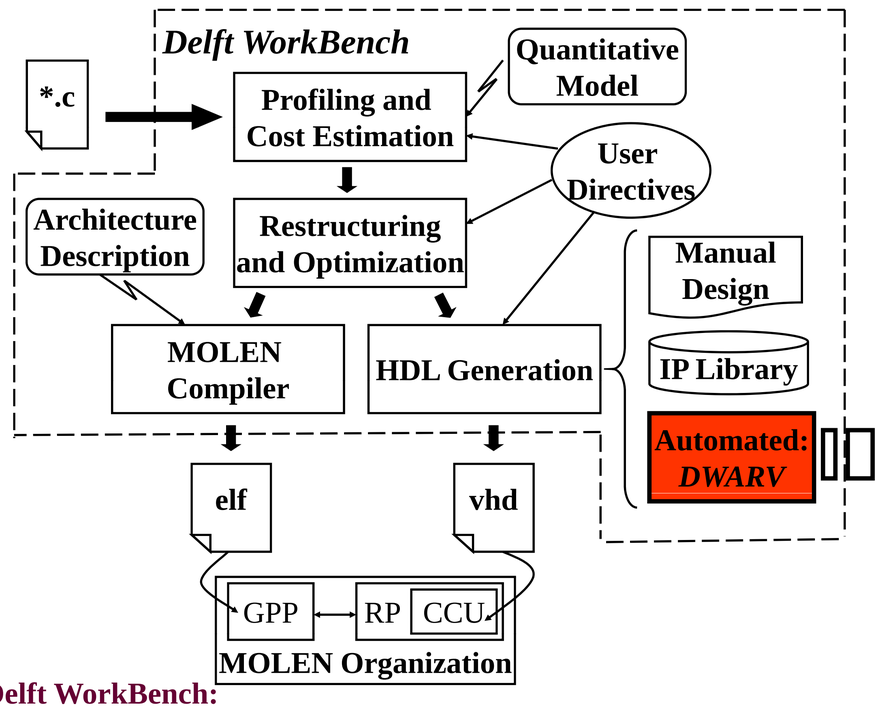
DWARV 2.0: A CoSy-based C-to-VHDL Hardware Compiler

Razvan Nane, Vlad-Mihai Sima, Bryan Olivier, Roel Meeuws, Yana Yankova, Koen Bertels

Context



Delft WorkBench:

- Semi-automatic tool platform for hardware/software co-design in the context of Custom Computing Machines (CCM).
- Targets the MOLEN polymorphic machine organization [1].
- Supports the MOLEN programming paradigm.

MOLEN Polymorphic Processor:

- Couples General Purpose Processor (GPP) with Reconfigurable Processor (RP) with one or more Custom Computing Units (CCU).

HDL Generation:

- Manual implementation or IP library instantiation for extremely critical kernels.
- **Automated**: for fast prototyping and fast performance estimation during design space exploration as well as for other kernels.

Design Goals

- **No limitation** on the application domain
- **Actual** execution on a **real hardware** prototype platform

Toolset Input

- *Pragma-annotated C*: identifies the functions to be translated into VHDL

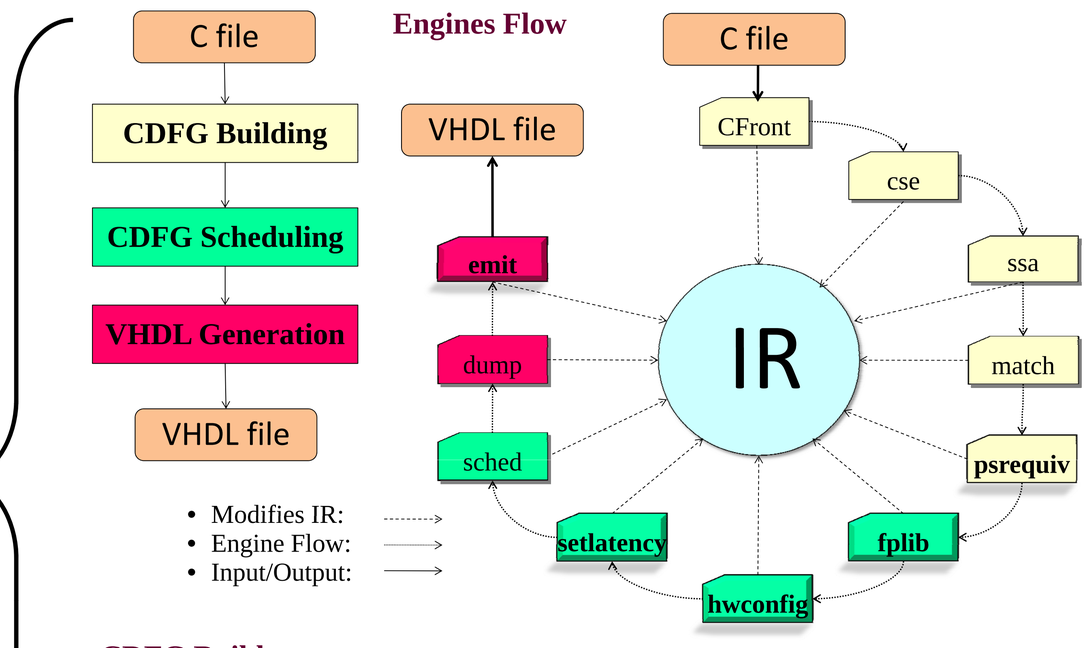
Configuration File

- C-data sizes
- Memory and XREG latency and bandwidth

Toolset Output

- FSM-based VHDL Design
- MOLEN-CCU Interface[5]

DWARV



CDFG Builder:

- *Cfront* creates and initializes the IR from the source code.
- *cse* performs common sub-expression elimination.
- *ssa* performs static single assignment.
- *psrequiv* performs custom FPGA register allocation.

CDFG Scheduler:

- *fplib* loads FP and IP core information from an external library file.
- *hwconfig* loads platform-dependent parameters, e.g. memory latency.
- *setlatency* adjusts the length of DFGraph's edges with true latencies.
- *sched* schedules the graph and cycle annotates IR nodes.

VHDL Generation:

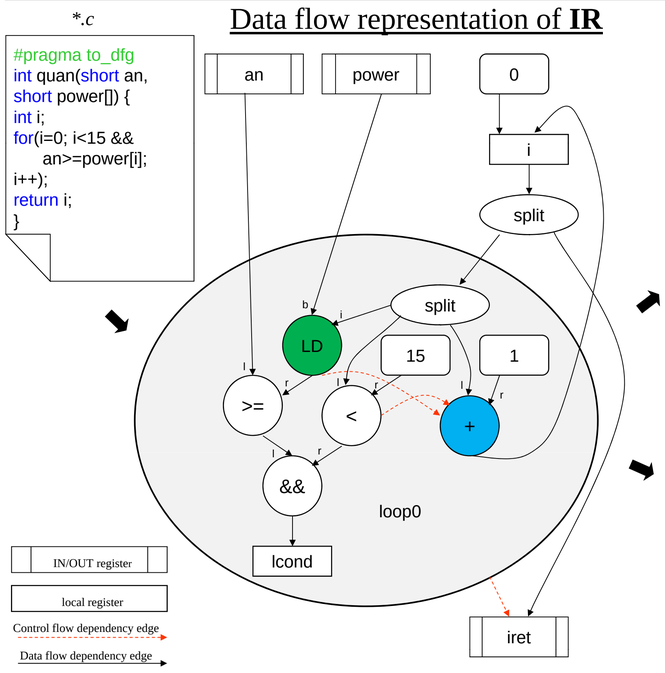
- *emit* prints vhdl code of the RULE annotated IR nodes.

C-to-VHDL Example

**.c*

```
#pragma to_dfg
int quan(short an,
short power[]) {
int i;
for(i=0; i<15 &&
     an>=power[i];
i++);
return i;
}
```

Data flow representation of **IR**



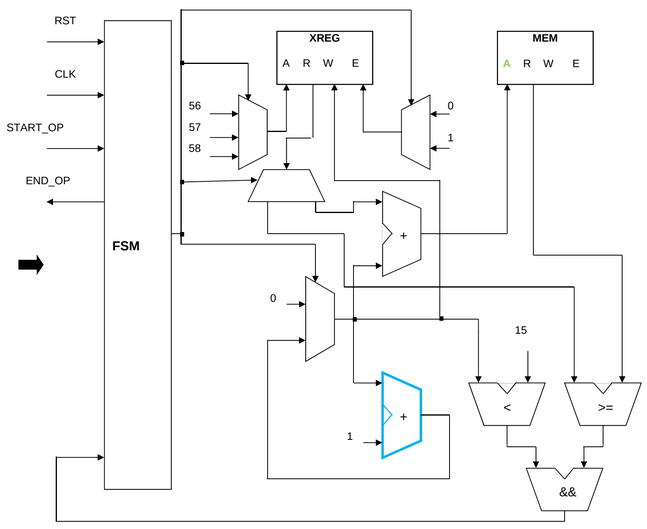

```
RULE [add_int_s] o:mirPlus(rs1:reg, rs2:reg) -> rd:reg;
CONDITION { IS_SINT(o.Type) };
EXPAND emit {
      fprintf (OUTFILE,"\t%s <= std_logic_vector(signed(%s) + signed(%s));\n",
                        REGNAME(rd), REGNAME(rs1), REGNAME(rs2));
};
END
```

```
RULE [ld_32] o:mirContent (rs:reg) -> rd:reg;
CONDITION { …. /* TRUE only for int values on 32 bits */ … };
WRITE REGISTER <DATA_ADDR>;
EXPAND {
        gcg_create_data_addr( … , rs, …);
(-> fprintf(OUTFILE, "\tDATA_ADDR <= %s;\n", REGNAME(rs));)
        gcg_create_read_data(…, rd );
(-> fprintf(OUTFILE, "\t %s  <= READ_DATA;\n", REGNAME(rd));)
};
END
```

**.cgd*

```
ENTITY CCU IS
    PORT(
                – Declare Molen interface ports
                –  e.g. data_addr, read_data, start, done
        );
END ENTITY CCU;

ARCHITECTURE ARCH_example OF CCU IS
    -- declare the FP components
    –  e.g. input ports, output ports, name, op selection
BEGIN
    -- FP component instantiation & signal declarations

    STATES: PROCESS (sl  STATE, START  OP)
        -- Sequential logic = FSM
        -- Example: when "00101" =>
        --                  sl_state <= "00110";
    END PROCESS;

    EXECUTION: PROCESS (CLK, RST)
        -- Combinational logic = data path
        – Example: when "00101" =>
        --                  DATA_ADDR <= var32;
        --                  var33 <= var30 + var31;
    END PROCESS;

END ARCHITECTURE ARCH_example;
```

**.vhd*

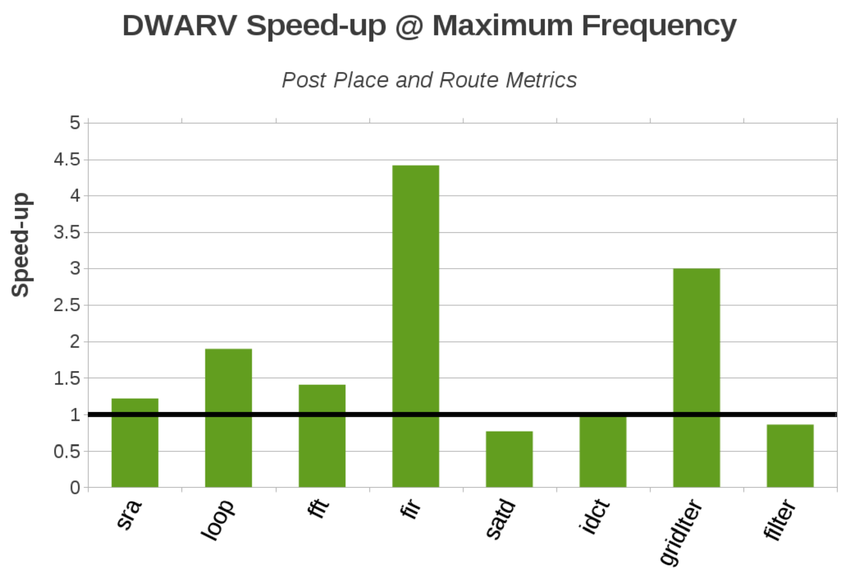


**.ngc*

Experimental Results

Methodology:

- Target Platform: Xilinx Virtex5 ML510 board.
- LegUp [2] cycle information obtained by running the LegUp simulation scripts.
- DWARV cycle information obtained by running the generated CCU in Modelsim.
- To integrate LegUp in our design, we built a simple wrapper around the generated verilog module.
- Both DWARV generated CCU and the LegUp module were synthesized decreasingly from 350 MHz to obtain Max. Freq. (the highest frequency for which the design was successfully routed).
- Cycle information and Max. Freq. obtained used to compute the Speed-ups.



DWARV 2.0 Speed-ups vs. LegUp times.

DWARV 2.0 New Features

Data Types	Statements	IP Library Fields
Integer 64 bit	div, mod	IP name
Floating Point	case, label, switch	Input port names
Multi-dim arrays	function calls	Output port names
Struct	while, do-while	Operation type & size
Union	return, break	Latency & frequency

Conclusion

- Up to 4.41x kernel speedup vs. LegUp HW compiler [2].
- Highly extensible due to the CoSy framework [3].
- Support for a wide range of C-language constructs.
- Support for FP operations and custom IP block calls.

Future Work

- Study and identify hardware compiler optimisations.
- Hardware reuse based on scheduling templates.
- Tool-chain integration and support for AOP.

References

[1] **The MOLEN Polymorphic Processor**, S.Vassiliadis, S.Wong, G.N.Gaydadjiev, K.Bertels, G.Kuzmanov, E.M.Panainte, *IEEE Transactions on Computers, 2004, pp1363-1375*

[2] **LegUp: high-level synthesis for FPGA-based processor/accelerator systems.** A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. Anderson, S. Brown and T. Czajkowski. *Proceedings of the 19th ACM/SIGDA international symposium on FPGA '11*: 33–36.

[3] **CoSy compiler platform.** Associated Compiler Experts ACE**.** *[Online]. Available: www.ace.nl*

Acknowledgement

This research is partially supported by:

- Artemisia iFEST project (grant 100203)
- Artemisia SMECY project (grant 100230)
- FP7 Reflect project (grant 248976)

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Computer Engineering
Mekelweg 4 (16th floor)
2628 CD Delft, The Netherlands
http://ce.et.tudelft.nl

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