UCLA COMPUTER SCIENCE DEPARTMENT

Towards Layout-Friendly High-Level Synthesis

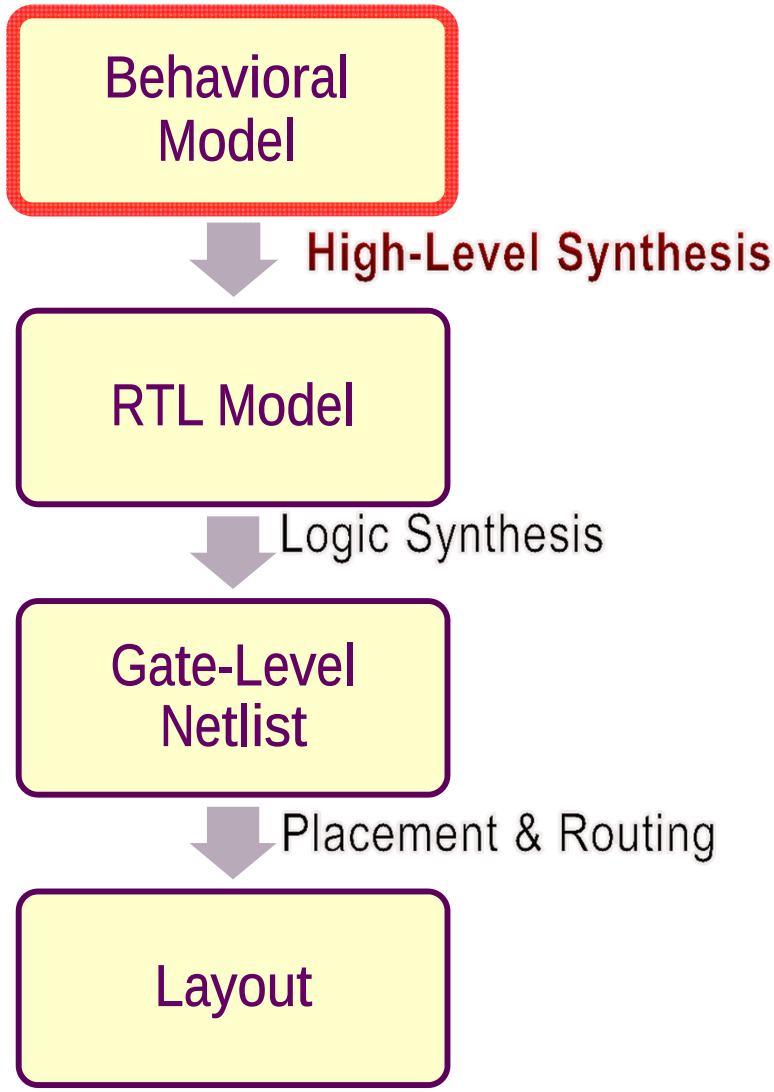
Jason Cong	UCLA
Bin Liu	UCLA
Guojie Luo	Peking University
Raghu Prabhakar	UCLA

Outline

- High-level synthesis and layout-friendly architecture
- Evaluation of the impact of high-level decisions
- Evaluation of metrics for scheduling/binding
- Conclusion

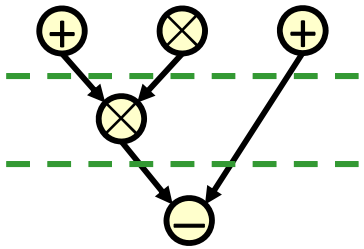
High-Level Synthesis

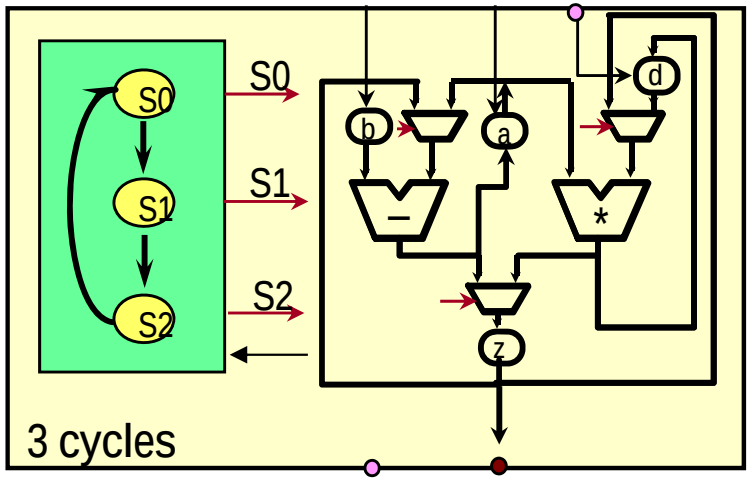
- Synthesis as a model refinement process
- Mature RTL-to-layout flow today
- Behavior model: one level above RTL
  - C/C++/SystemC/Matlab, etc.
- High-level synthesis
  - Untimed behavioral model to cycle-accurate RTL
  - Typically: C to Verilog



A Typical Synthesis Flow from Behavior Level

$t_1 = a + b;$
$t_2 = c * d;$
$t_3 = e + f;$
$t_4 = t_1 * t_2;$
$z = t_4 - t_3;$





Compiler transformation
- Program -> CDFG

Scheduling
- CDFG -> FSMD

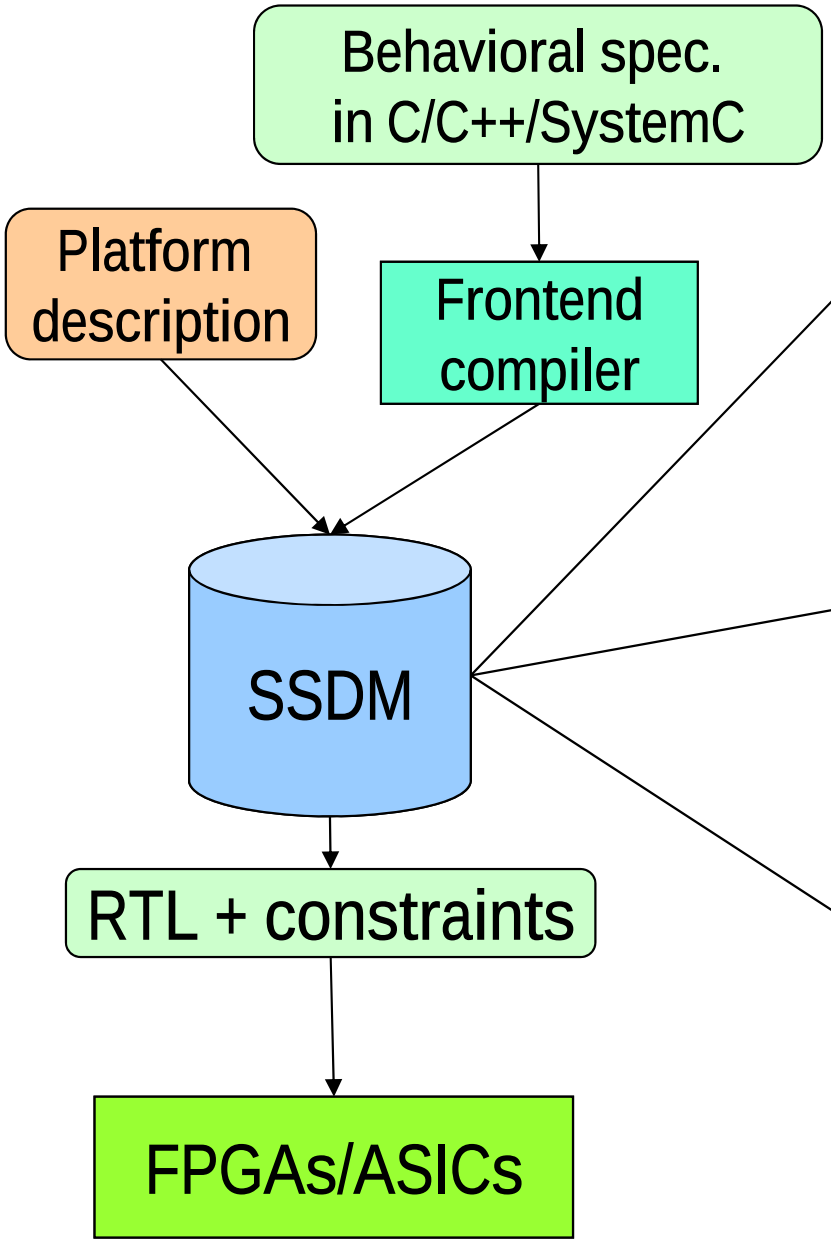
Binding
- FSMD -> RTL

RTL Synthesis, P&R …

A Short History of High-Level Synthesis

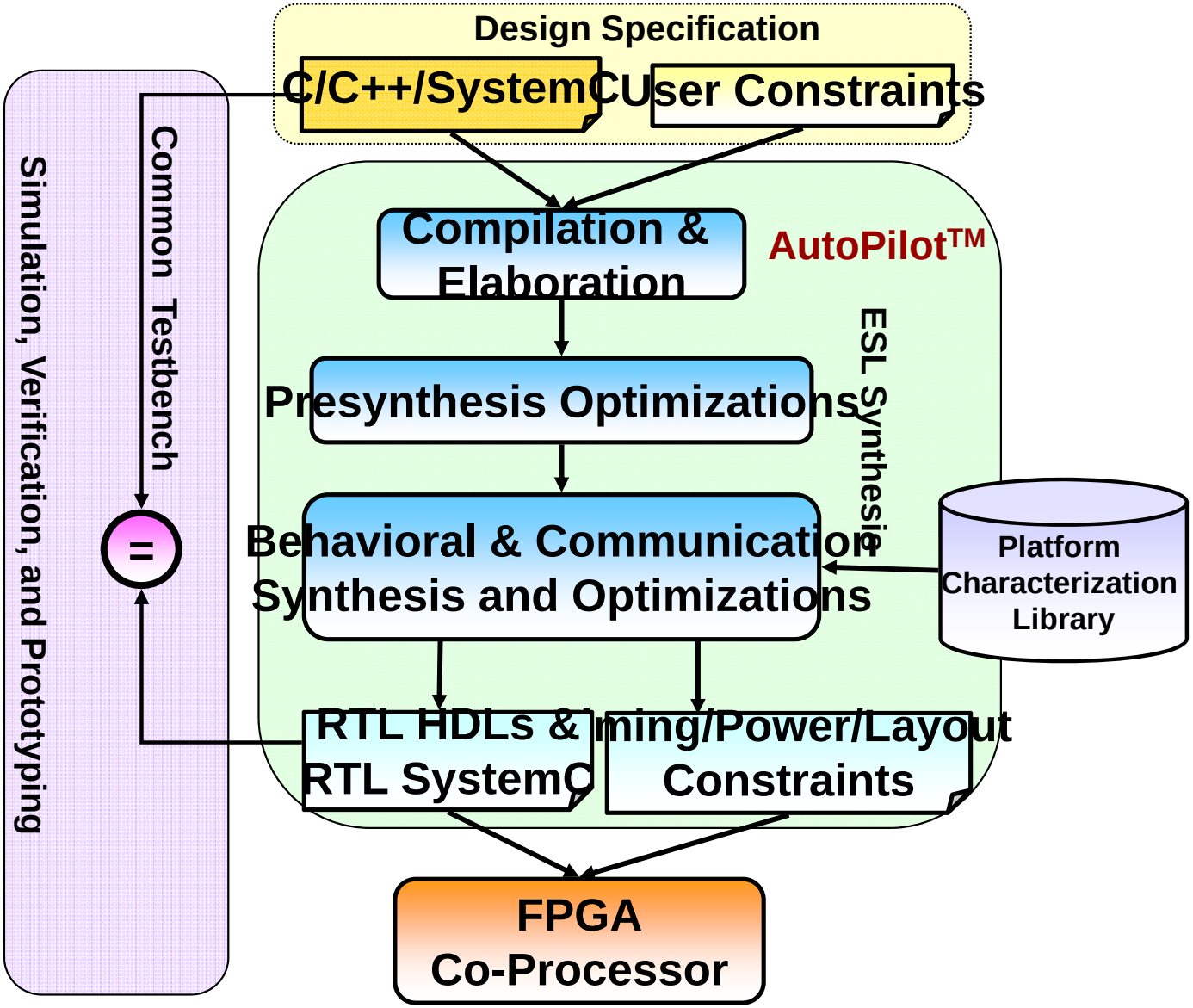
- 1980s—early 1990s: research and prototype
- Late 1990s: early commercialization
  - Synopsys Behavioral Compiler, etc.
  - Mostly from behavioral VHDL/Verilog
- 2000—present: another wave of commercialization
  - C-based languages (C/C++/SystemC) as input
  - AutoESL (Xilinx), Cadence, Forte, Mentor (Calypto), NEC, Synfora (Synopsys), Synopsys
  - Growing interest driven by design complexity and time-to-market pressue

xPilot: Behavioral-to-RTL Synthesis Flow [SOCC'2006]



- Advanced transformtion/optimizations
  - Loop unrolling/shifting/pipelining
  - Strength reduction / Tree height reduction
  - Bitwidth analysis
  - Memory analysis …
- Core behvior synthesis optimizations
  - Scheduling
  - Resource binding, e.g., functional unit binding register/port binding
- μArch-generation & RTL/constraints generation
  - Verilog/VHDL/SystemC
  - FPGAs: Altera, Xilinx
  - ASICs: Magma, Synopsys, …

AutoPilot Compilation Tool (based UCLA xPilot system)

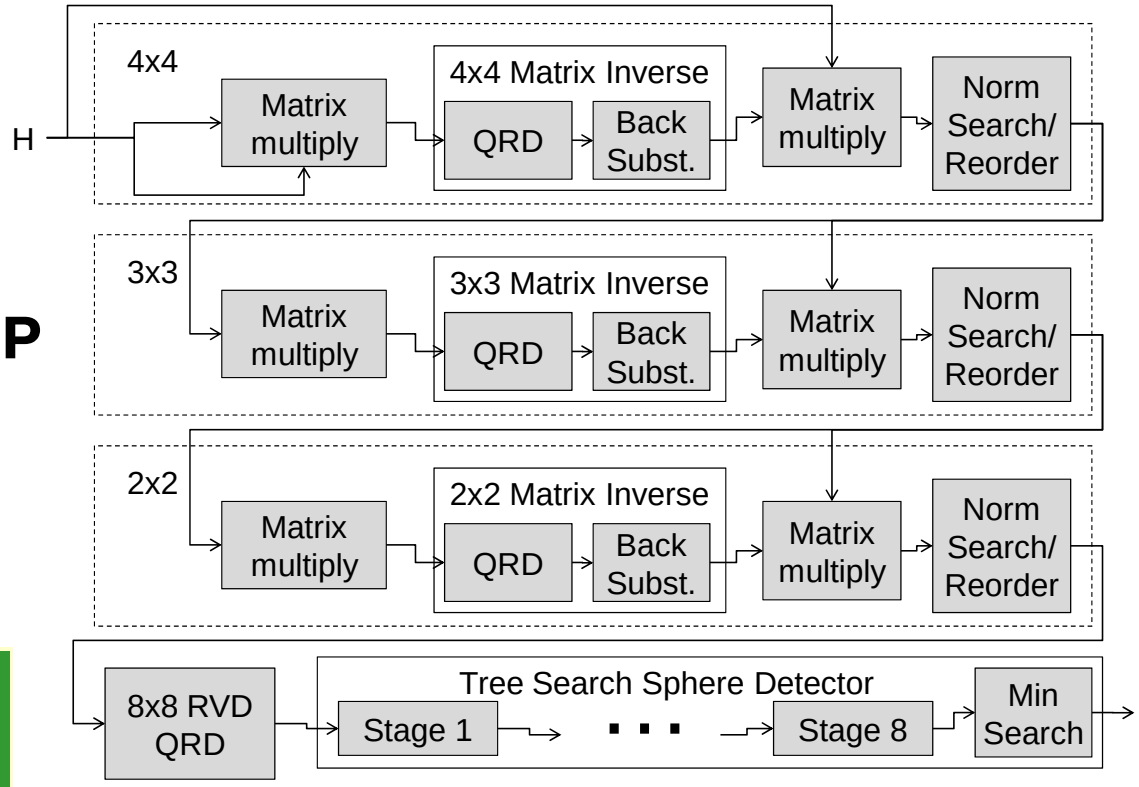


- Platform-based C to FPGA synthesis
- Synthesize pure ANSI-C and C++, GCC-compatible compilation flow
- Full support of IEEE-754 floating point data types & operations
- Efficiently handle bit-accurate fixed-point arithmetic
- More than 10X design productivity gain
- High quality-of-results

Developed by AutoESL, acquired by Xilinx in Jan. 2011

AutoPilot Results: Sphere Decoder (from Xilinx)

- **Wireless MIMO Sphere Decoder**
  - **~4000 lines of C code**
  - **Xilinx Virtex-5 at 225MHz**
- **Compared to optimized IP**
  - **11-31% better resource usage**



Metric	RTL Expert	AutoPilot Expert	Diff (%)
LUTs	32,708	29,060	-11%
Registers	44,885	31,000	-31%
DSP48s	225	201	-11%
BRAMs	128	99	-26%

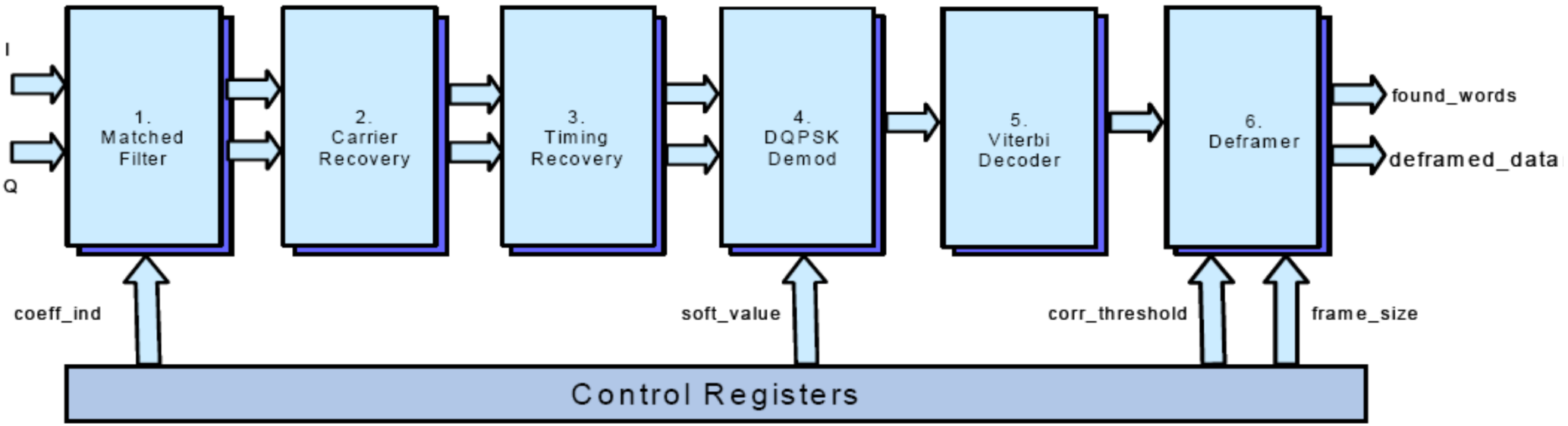
TCAD April 2011 (keynote paper)
"High-Level Synthesis for FPGAs: From Prototyping to Deployment"

AutoPilot Results: DQPSK Receiver (from BDTI)

- Application
  - DQPSK receiver
  - 18.75Msamples @75MHz clock speed
- Area better than hand-coded

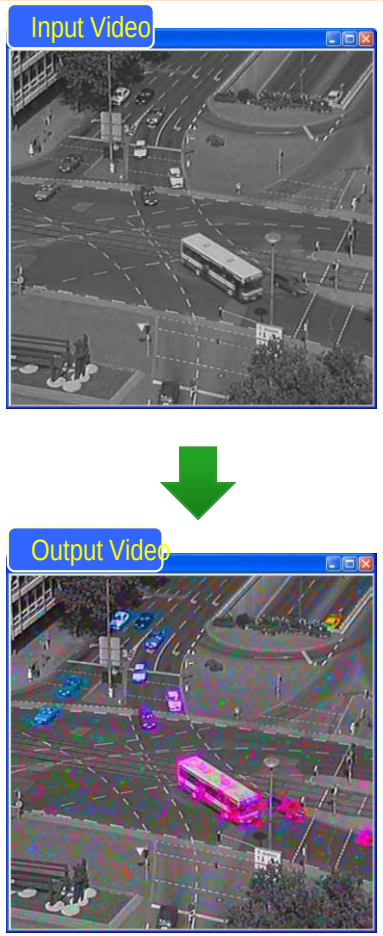
	Hand-coded RTL	AutoPilot
Xilinx XC3SD3400A chip utilization ratio (lower the better)	5.9%	5.6%

BDTi evaluation of AutoPilot
http://www.bdti.com/articles/AutoPilot.pd



AutoPilot Results: Optical Flow (from BDTI)

- Application
  - Optical flow, 1280x720 progress scan
  - Design too complex for an RTL team
- Compared to high-end DSP:
  - 30X higher throughput, 40X better cost/fps



	Chip Unit Cost	Highest Frame Rate @ 720p (fps)	Cost / performance ($ / frame / second)
Xilinx Spartan3ADSP XC3SD3400A chip	$27	183	$0.14
Texas Instruments TMS320DM6437 DSP processor	$21	5.1	$4.20

BDTi evaluation of AutoPilot
http://www.bdti.com/articles/AutoPilot.pdf

Impact on Quality of Result

- Big impact on QoR due to drastically different architectures
  - Parallel/sequential/pipelined
  - Different ways to map operations to control states
  - Different ways to share functional units/registers/interconnects
- Opportunity to select from multiple possible implementations
  - Instead of struggling with a sub-optimal RTL
  - Need metrics/models to decide which implementation is superior
- Performance/throughput/area can be estimated reasonably well in HLS
  - Frequency/congestion is quite hard
  - Some RTL structures lead to long interconnect delay after layout

Interconnect Estimation: the Challenge

- Estimation of interconnect timing and congestion is hard at a high level
  - Long wires/congestion occur during layout
- Incorporate layout in synthesis?
  - Reasonable, but time consuming.
  - May not be necessary if we just want to estimate if one solution is better than the other
  - Try to get the more layout-friendly solution
- In this work
  - Experimentally evaluate the impact of HLS decisions on congestion
  - Evaluate some possible metrics without doing layout

Outline

- High-level synthesis and layout-friendly architecture
- Evaluation of the impact of high-level decisions
- Evaluation of metrics for scheduling/binding
- Conclusion

Experiment Setup

- Varying strategies in HLS
- Impacts of compiler transformation (Loop unrolling, memory partitioning, etc.) and synthesis engine (scheduling & binding) evaluated separately
- 5 DSP benchmarks (lots of multiplication/addition, simple or no control flow) for synthesis engine

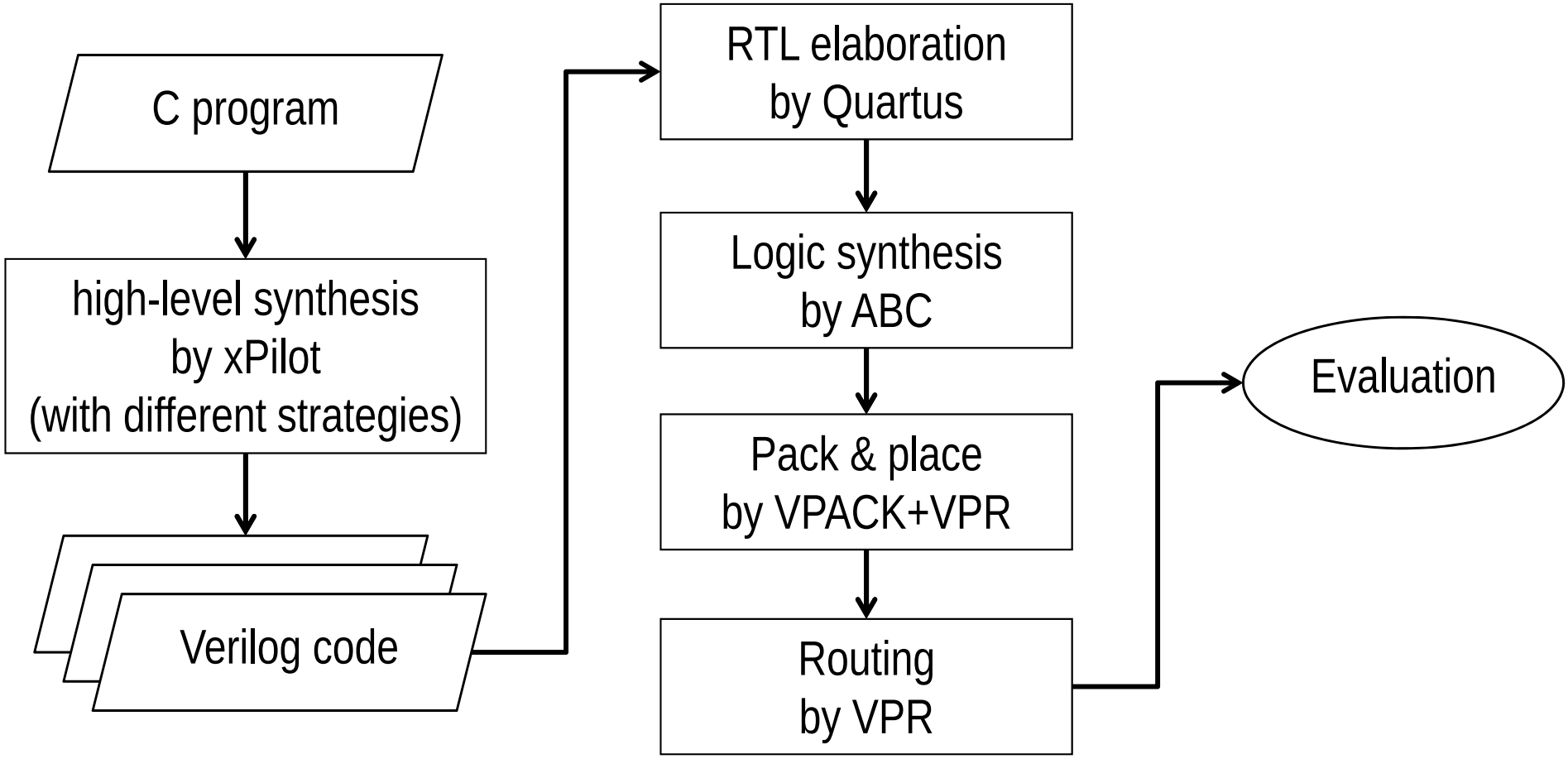
Compiler transformation

- Program -> CDFG

	Binding objective	constraint
1	Total area	None
2	Total area	Mux_input <= 4
3	#R (total number of registers)	Mux_input <= 4
4	#M	None

	Number of lines in C	Number of nodes in CDFG
Test1	96	78
Test2	20	90
Test3	97	160
Test4	16	50
Test5	87	390

The RTL Implementation Flow for Routability Evaluation



Implementation Flow Setup

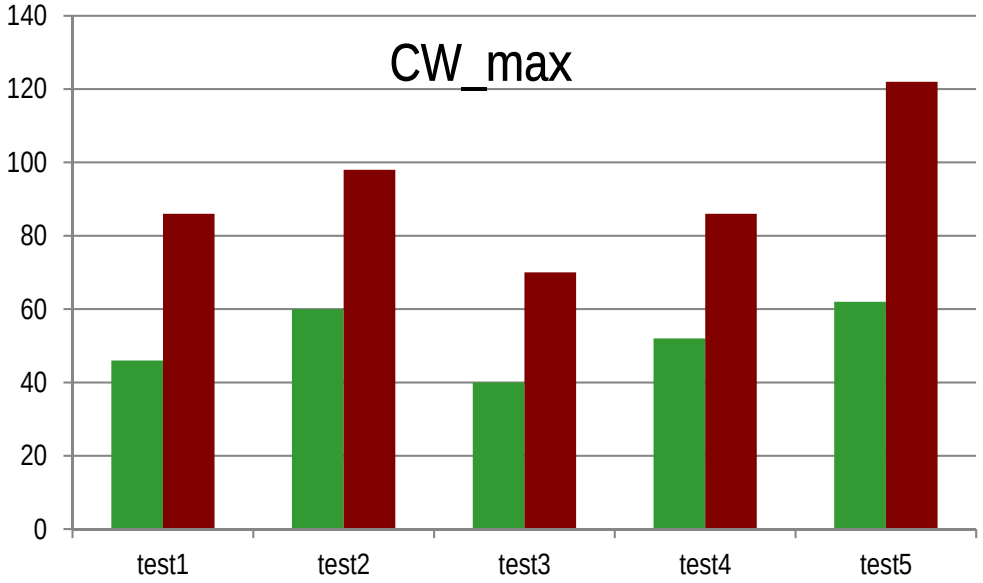
- Target platform: island-style FPGA
  - 10 4-LUTs per CLB, with routing channels between CLBs (span = 1 CLB)
  - The number of routing tracks per channel (*channel width*) is constant
- Configurations of the toolchain
  - Logic synthesis by ABC with default settings
  - Packing by T-VPACK with default settings
  - Wirelength-driven placement by VPR using simulated-annealing
  - Routing by VPR using negotiation-based routing and directed search
    - The channel width is variable and determined by binary search
- Post-layout characteristics
  - Maximum channel width (CW_max)
  - Average wirelength (WL_avg)
    = average #tracks per net

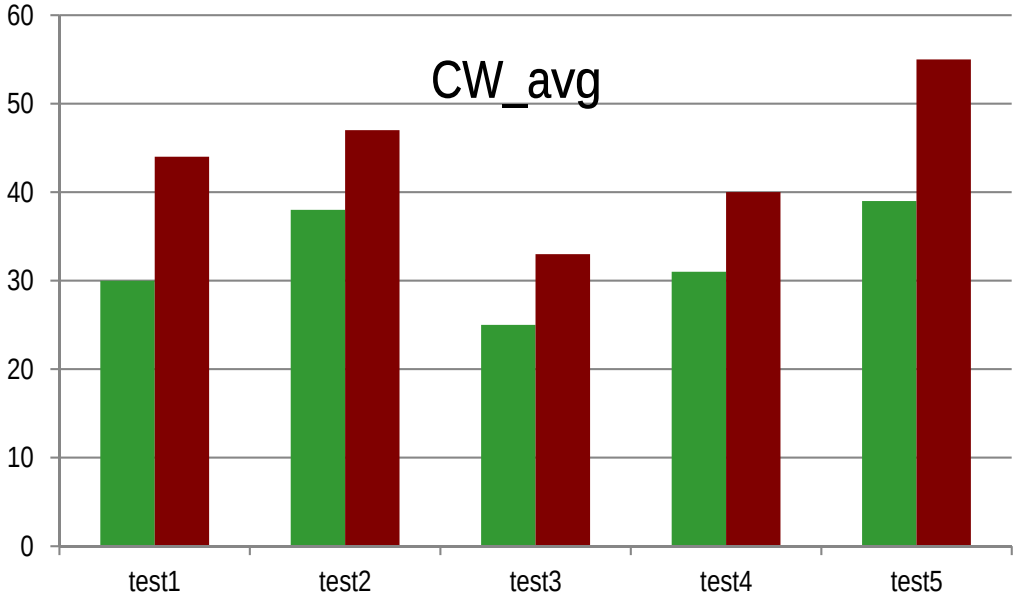
Impact of the Synthesis Engine

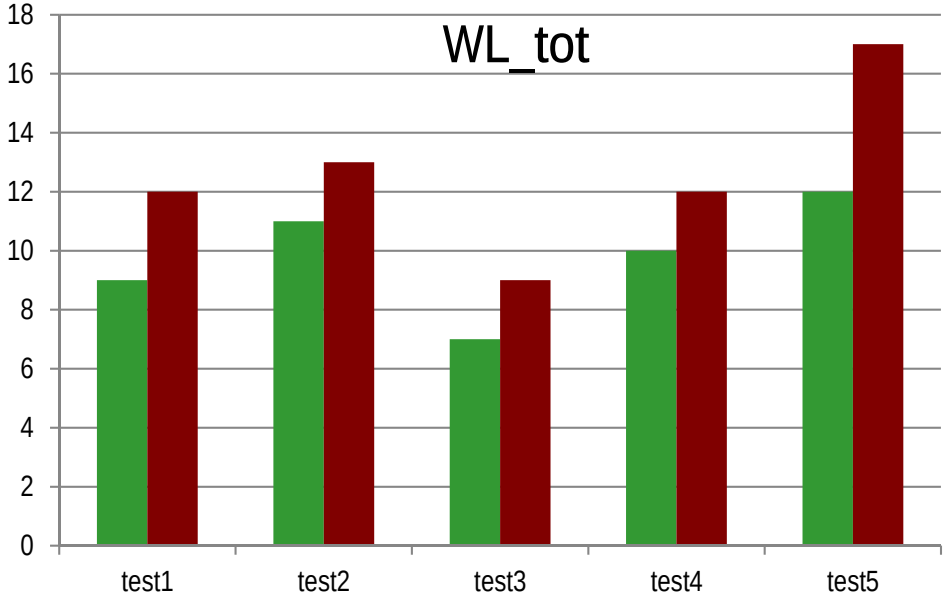
- 60 RTLs generated for each design
  - 6 scheduling strategies, 10 binding strategies
  - Some are equivalent
- Results: min/max for each metric
  - Clearly, very different although behaviorally equivalent

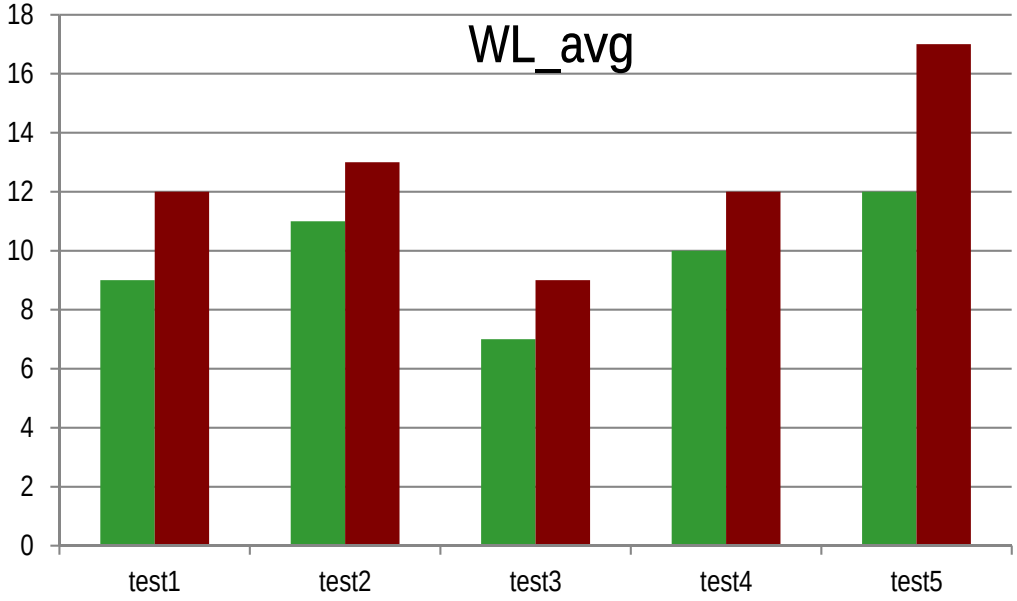
design	CW_max	CW_avg	WL_tot	WL_avg
test1	46/86	30/44	36K/166K	9/12
test2	60/98	38/47	61K/251K	11/13
test3	40/70	25/33	25K/103K	7/9
test4	52/86	31/40	47K/196K	10/12
test5	62/122	39/55	94K/562K	12/17

Impact of the Synthesis Engine (min vs max)









Impact of Compiler Transformations

- A matrix multiplication example

```
outer_loop: for (i = 0; i < 8; i++) {
middle_loop:   for (j = 0; j < 8; j++) {
                  Result[i][j] = 0;
inner_loop:       for (k = 0; k < 8; k++)
                     Result[i][j] += X[i][k] * Y[k][j];
               }
            }
```

- Different ways to transform/pipeline the code, partition memory

	loop	memory
1	Keep all loops, pipeline inner loop	As is
2	Unroll inner loop, pipeline middle loop	Partition X into columns and Y into rows to allow simultaneous accesses
3	Unroll inner and middle loop, pipeline outer loop	Partition X and Y into scalars, partition Result into columns

Impact of Compiler Transformations

solution	CW_max	CW_avg	WL_tot	WL_avg
1	30	18	4543	5
2	44	24	43610	7
3	80	36	223042	10

Outline

- High-level synthesis and layout-friendly architecture
- Evaluation of the impact of high-level decisions
- Evaluation of metrics for scheduling/binding
- Conclusion

Structural Metrics in High-Level Synthesis

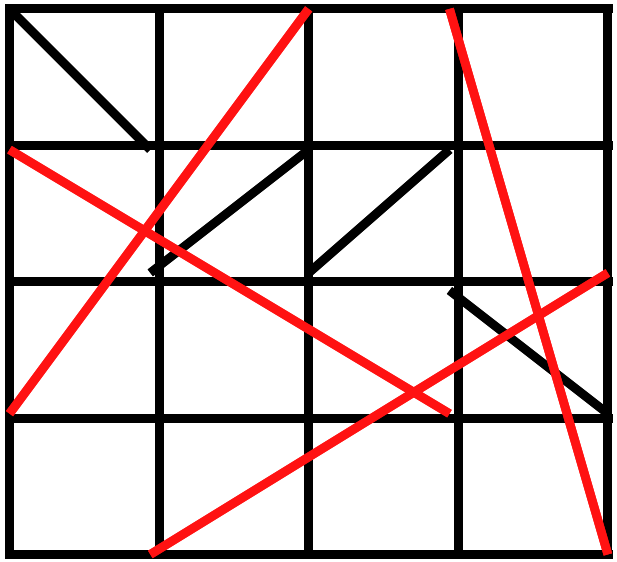
- Can we estimate layout-friendliness from netlist structure?
  - How effective are they?
- Number of datapath nets
  - A straightforward estimation of interconnect complexity
- Total multiplexor inputs
  - Extensively in HLS systems since the 1980s
  - M. C. McFarland. Reevaluating the design space for register transfer hardware synthesis. *ICCAD' 87*.
  - Big multiplexor often cause congestion/timing issues
  - Less multiplexors -> less interconnects
  - Regarded as a good first-order metric

Structural Metrics in High-Level Synthesis

- Graph *adhesion*: a metric based on cut size
  - P. Kudva, A. Sullivan, and W. Dougherty. Measurements for structural logic synthesis optimizations. *IEEE Trans. CAD*, 2003.
  - Originally developed for use in logic synthesis (gate-level)
  - Measured as sum of all-pair min-cut size (SAPMC)
    - Smaller cut size implies better structure
- The adhesion metric grows quadratically with circuit size
- To avoid a bias on area, we can do normalization
  - Average min-cut size (AMC) between all pairs *n(n-1)/2*

Structural Metrics in High-Level Synthesis

- Spreading score: a metric based on graph embedding
  - Observation: long wires are often trouble makers
  - If a netlist can be placed without introducing long wires, it is good
- Example
  - Mesh: known to be layout-friendly
  - Mesh with local interconnects
  - Mesh with long interconnects
    - Hard to make the wires short
- Intuition: prefer topologies that spread easily



Structural Metrics in High-Level Synthesis

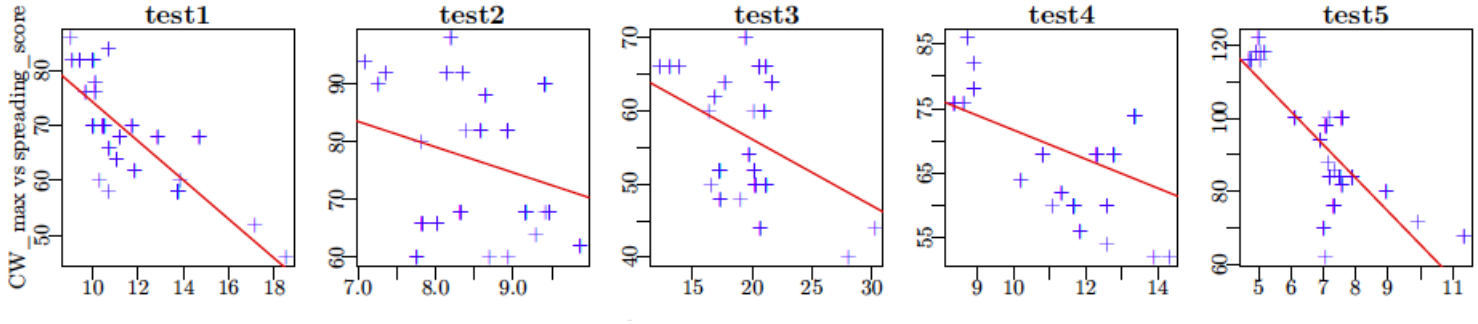
- Spreading score is defined as the optimal value of the following problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} w_i \|p_i\|^2 \\ \text{subject to} & \sum_{i=1}^{n} w_i p_i = 0 \\ & \|p_i - p_j\| \leq l_{ij} \quad \forall (i,j) \in E \end{array}$$

- p_i is the location of the ith component. l_{ij} is the length budget for the edge (I,j), w_i is the weight (area)
- The formulation asks the following question
  - How far can the components spread away from their center of gravity without introducing wires longer than l_{ij}?
  - Relaxed version can be solved optimally in polynomial time
  - Also normalize by *n(n-1)/2*

Single-Variable Regressions (CW_max)

CW_max vs. Spreading



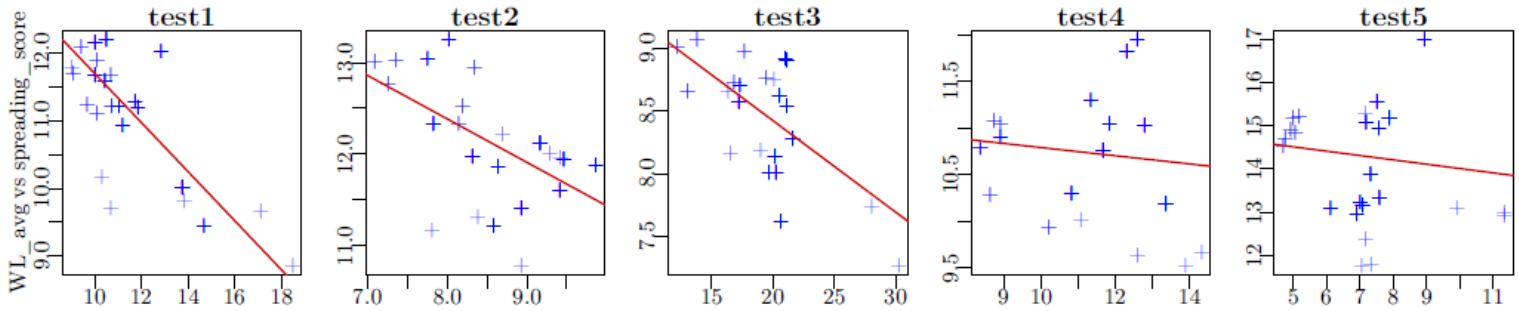
CW_max vs. AMC

CW_max vs. #net

CW_max vs. mux_input

Single-Variable Regressions (WL_avg)

WL_avg vs. Spreading



WL_avg vs. AMC

WL_avg vs. #net

WL_avg vs. mux_input

Multi-Variable Linear/Quadratic Regressions

- Randomly select 70% data as the training data, and 30% data as the testing data
- Predicting CW_max
  - (2-var) CW_max = regress(Spreading, AMC)
    - Linear: $CW\_max = k_1 \times Spreading + k_2 \times AMC + k_0$
    - Quadratic: $CW\_max = k_1 \times Spreading + k_2 \times AMC + k_{11} \times Spreading^2 + k_{22} \times AMC^2 + k_{12} \times Spreading \times AMC + k_0$
  - (3-var) CW_max = regress(Spreading, AMC, num_nets)
- Predicting WL_avg
  - (2-var) WL_avg = regress(Spreading, AMC)
  - (3-var) WL_avg = regress(Spreading, AMC, num_nets)

Predicting CW_max = regress(Spreading, AMC)

training	linear		quadratic	
	error	%	error	%
test1	4.76	7.0%	3.77	5.6%
test2	8.15	10.7%	6.41	8.6%
test3	3.73	6.8%	3.09	5.5%
test4	4.48	7.0%	3.81	5.9%
test5	5.50	6.4%	4.36	4.9%

testing	linear		quadratic	
	error	%	error	%
test1	6.10	9.3%	8.69	12.8%
test2	10.37	14.4%	11.00	14.7%
test3	7.20	13.7%	7.39	14.3%
test4	5.16	6.9%	6.21	8.9%
test5	11.33	15.9%	10.75	15.0%

Predicting CW_max = regress(Spreading, AMC, num_nets)

training	linear		quadratic	
	error	%	error	%
test1	3.16	4.6%	2.48	3.6%
test2	1.47	1.9%	0.92	1.2%
test3	2.30	4.2%	1.64	2.8%
test4	3.80	6.0%	2.62	4.1%
test5	3.08	3.5%	2.28	2.6%

testing	linear		quadratic	
	error	%	error	%
test1	4.33	6.1%	6.10	8.8%
test2	3.55	5.1%	3.08	4.4%
test3	2.66	4.9%	4.65	8.8%
test4	4.42	6.3%	5.38	7.7%
test5	4.66	6.8%	8.81	11.7%

Predicting WL_avg = regress(Spreading, AMC)

training	linear		quadratic	
	error	%	error	%
test1	0.39	3.5%	0.33	3.0%
test2	0.34	2.9%	0.27	2.3%
test3	0.18	2.1%	0.18	2.1%
test4	0.57	5.3%	0.41	3.8%
test5	0.81	5.7%	0.51	3.6%

testing	linear		quadratic	
	error	%	error	%
test1	0.54	5.1%	0.73	7.1%
test2	0.44	3.7%	0.46	3.8%
test3	0.49	5.9%	0.60	7.2%
test4	0.38	3.6%	0.45	4.4%
test5	0.89	7.0%	0.93	7.0%

Predicting WL_avg = regress(Spreading, AMC, num_nets)

training	linear		quadratic	
	error	%	error	%
test1	0.39	3.5%	0.17	1.5%
test2	0.33	2.8%	0.04	0.3%
test3	0.18	2.1%	0.14	1.6%
test4	0.53	5.0%	0.27	2.6%
test5	0.80	5.6%	0.30	2.1%

testing	linear		quadratic	
	error	%	error	%
test1	0.56	5.2%	0.69	6.5%
test2	0.47	3.9%	0.41	3.5%
test3	0.44	5.3%	0.91	10.7%
test4	0.45	4.3%	0.66	6.2%
test5	0.89	6.9%	0.81	6.3%

Concluding Remarks

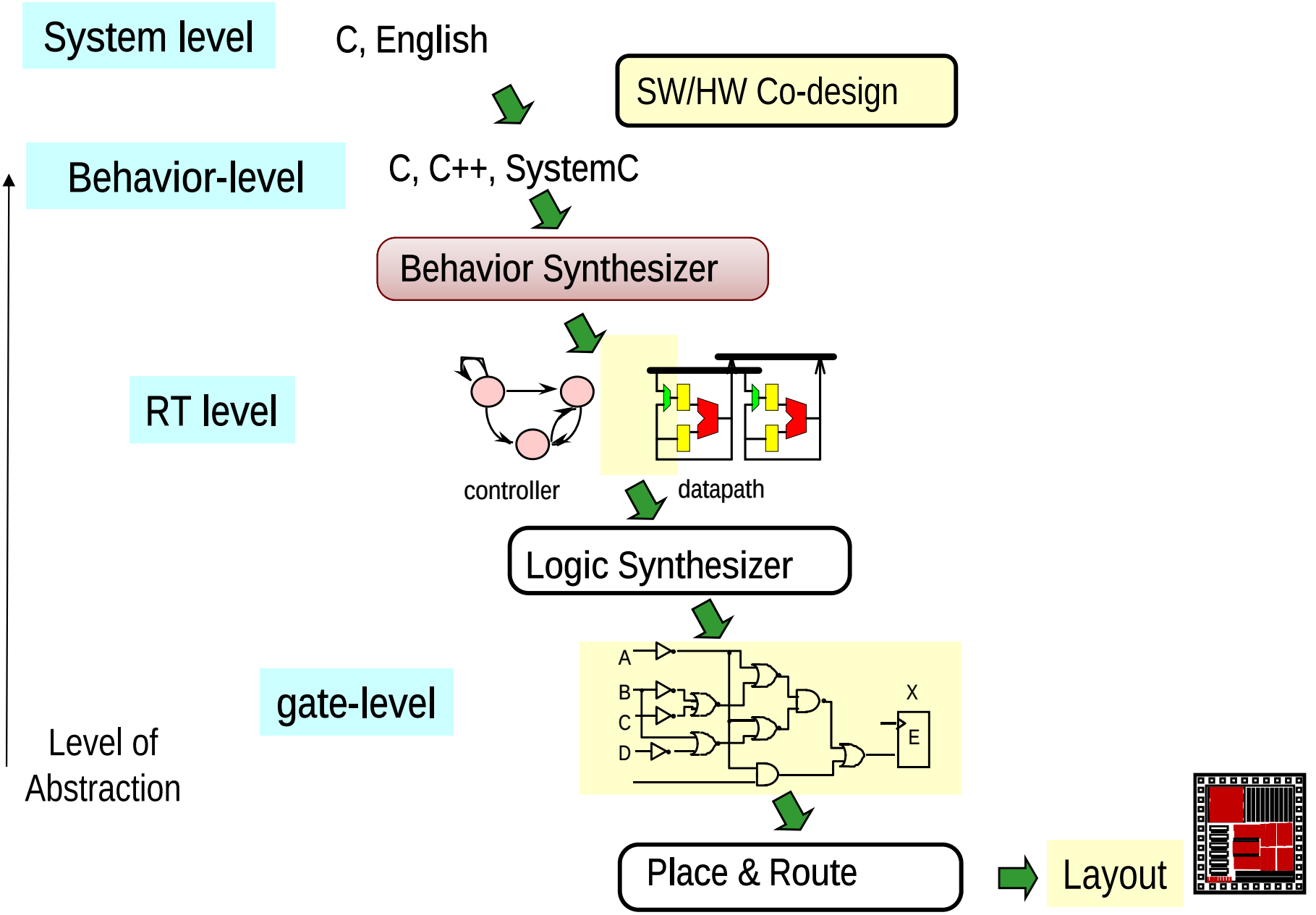
- Predicting impact on layout in HLS is hard
- Some structural metrics show good correlation with results after layout in some cases.
  - But consistency is an issue
- We would like explicit models to guide HLS perturbation
- Layout-friendly compiler transformation is even harder
  - Structural metrics not applicable
  - Machine learning?

backups

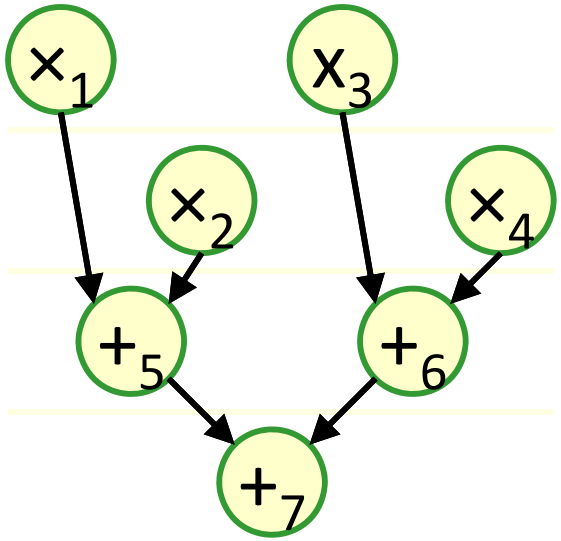
Why High-Level Synthesis?

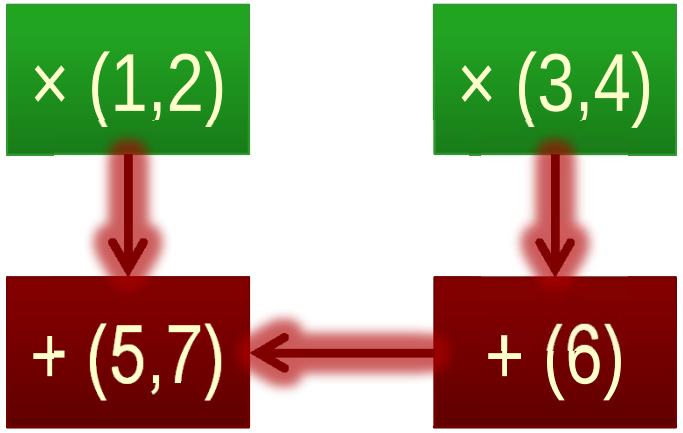
- Management of design complexity
  - Higher level abstraction -> less details ->more efficiency in design and verification
  - C/C++ easier to code than VHDL/Verilog
    - Sequential code is easier to create/maintain/verify
- Design reused and exploration
  - The same code can be synthesized on different platforms, targeting different frequency, resource limits, etc.
  - It is easy to generate different achitectures
- Easier integration
  - Specify software/hardware in a unified model

Levels of Abstraction in IC Design

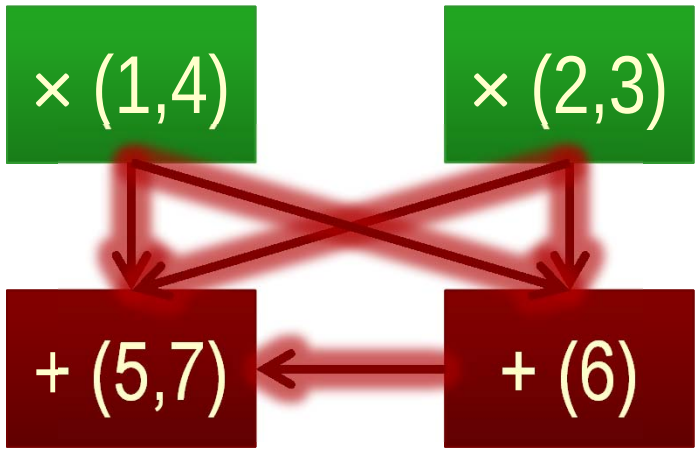


Example





AMC = 1
Mux size = 0
Spreading score = 0.417



AMC = 2.2
Mux size = 4
Spreading score = 0.083