



Platform-Based Synthesis for Field-Programmable SOCs

Prof. Jason Cong

cong@cs.ucla.edu

UCLA Computer Science Department



Outline

- ◆ Motivation
- ◆ xPilot system framework
- ◆ Behavior-level synthesis in xPilot
 - Advantages of behavioral synthesis
 - Scheduling
 - Resource binding
- ◆ System-level synthesis in xPilot
 - Synthesis for ASIP platforms
 - Design exploration for heterogeneous MPSoCs
- ◆ Conclusions

What about FP-SOC Design Tools

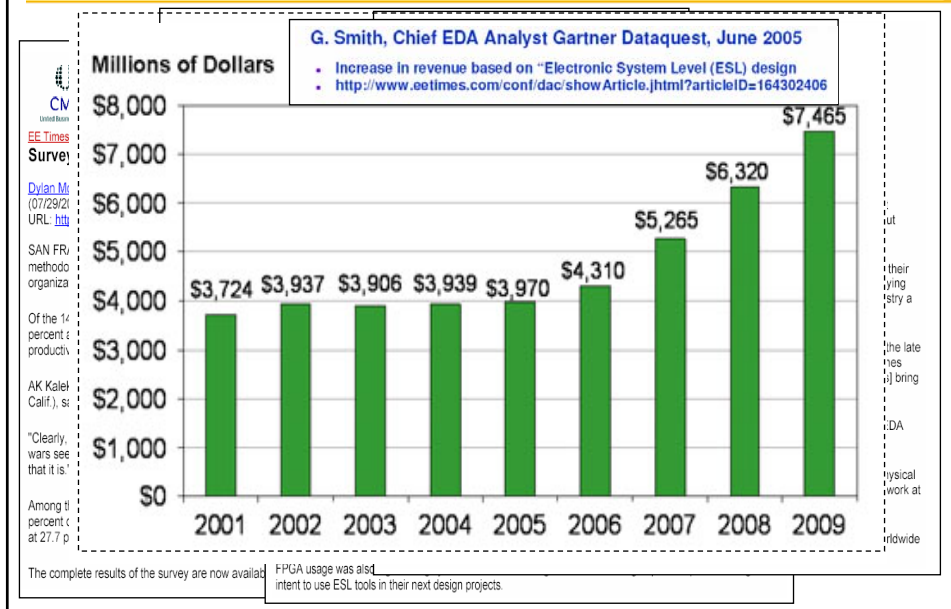
◆ Synthesis

- Behavior-level synthesis: from behavior specification (e.g. C, SystemC, or Matlab) to RTL or netlists
- System-level synthesis: from system specification to system implementation

◆ Verification

- Behavior-level verification
- System-level verification

ESL Tools – A Lot of Interests ...

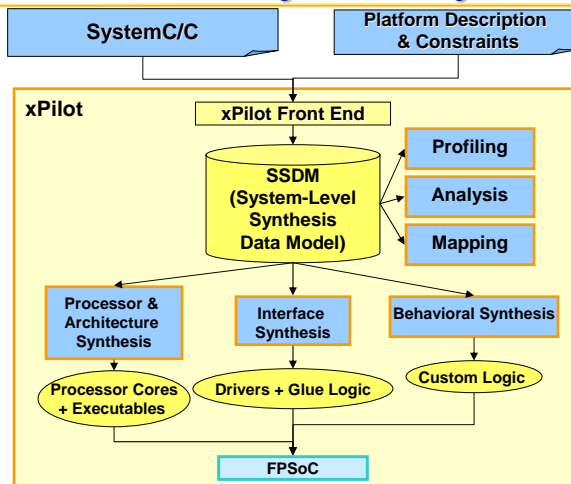


GartnerDataquest's ESL Landscape, 2005

Behavioral Level

Algorithmic Methodology	Processor/Memory Methodology	Control Logic Methodology
Behavioral Algorithmic Design	Behavioral Processor/Memory Design	Behavioral Control Design
Architectural Level — Architectural Design		
Algorithmic Methodology	Processor/Memory Methodology	Control Logic Methodology
Algorithmic Design and Entry Algorithmic Synthesis Application Engine Compiler Algorithmic Power Analysis	Processor/Memory Design and Entry ESL Synthesis Communications Compiler Application Engine Compiler Processor/Memory Power Analysis	Control Logic Design and Entry ESL Synthesis Control Logic Power Analysis
Architectural Level — Platform-Based Design		
Algorithmic Methodology	Processor/Memory Methodology	Control Logic Methodology
Processor Engine Compiler Algorithmic Modeling	Processor/Memory Platform Design Processor/Memory Modeling	Control Logic Platform Design and Entry

xPilot: Platform-Based Synthesis System



◆ Uniqueness of xPilot

- Platform-based synthesis and optimization
- Communication-centric synthesis with interconnect optimization

Outline

- ◆ Motivation
- ◆ xPilot system framework
- ◆ Behavior-level synthesis in xPilot
 - Advantages of behavioral synthesis
 - Scheduling
 - Resource binding
- ◆ System-level synthesis in xPilot
 - Synthesis for ASIP platforms
 - Design exploration for heterogeneous MPSoCs
- ◆ Conclusions

Motivation (1)

- ◆ Design complexity is outgrowing the traditional RTL method
 - Behavioral synthesis – a critical technology for enabling the move to higher level of abstraction
 - Reasons for previous failures
 - Lack of a compelling reason: design complexity is still manageable a decade of ago
 - Lack of a solid RTL foundation
 - Lack of consideration of physical reality

Motivation (2)

- ◆ Behavioral synthesis provides combined advantages
 - Shorter verification/simulation cycle
 - Better complexity management, faster time to market
 - Rapid system exploration
 - Quick evaluation of different hardware/software boundaries
 - Fast exploration of multiple micro-architecture alternatives
 - Higher quality of results
 - Platform-based synthesis & optimization
 - Full consideration of physical reality

Advantages – Better Complexity Management

- ◆ Shorter verification/simulation cycle
 - Simulation speed 100X faster than RTL-based method [NEC, ASPDAC04]
- ◆ Significant code size reduction
 - RTL design ~300KL → Behavioral design 40KL [NEC, ASPDAC04]

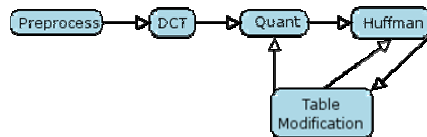
C VS. RTL VHDL CODE SIZES

Design	C lines	VHDL lines	LE	Fmax(MHz)
PR	90	600	1349	178.7
MCM	161	1260	2402	152.6
CACHE	295	1277	371	161.6
MOTION	130	1200	888	161.2
IDCT	236	7388	9351	162.9
DWT	180	1371	1862	147.3
EDGELOOP	329	7296	7440	100.1

- VHDL code generated by UCLA xPilot targeting Altera Stratix platform
- Over 10x code size reduction can be achieved

Advantages – Rapid System Exploration (1)

- ◆ Quick evaluation of various amounts of process level concurrency and different hardware/software boundaries



Example: Motion-JPEG implementation

- All HW implementation
- All SW implementation (using embedded processors)
- SW/HW co-design: optimal partitioning?
 - Repeated manual RTL coding is not solution!

Advantages – Rapid System Exploration (2)

- ◆ Fast exploration of multiple micro-architecture alternatives
 - Different hardware implementations can be easily obtained by varying the high-level spec. and applying different design constraints

Target cycle time	State#	Fmax (MHz)	Cycle#	Latency (ns)	LE#	DSP#
9ns	34	123.56	4830	39.1	1777	128
7ns	36	147.28	5211	35.4	1862	128
5.5ns	51	183.62	6926	37.8	1926	128

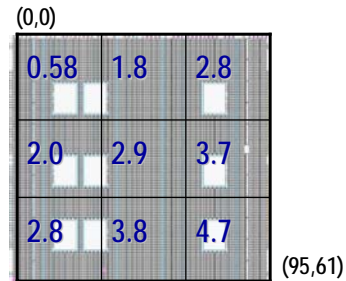
- Platform: Altera Stratix
- RTL synthesis & place-and-route: Altera QuartusII v5.0
- Simulation: Mentor ModelSim SE6.0

Advantages – Higher Quality of Results (1)

◆ Platform-based synthesis & optimization

- The quality of a RTL design is platform-dependent
- Designers often lack the complete and detail knowledge of the target platform

Resource	Area	Delay (ns)
ADDSUB-24b	25 LUTs	2.27
ADDSUB-32b	33 LUTs	2.61
MUX8to1-24b	120 LUTs	2.92
<i>MUX16to1-24b</i>	<i>264 LUTs</i>	<i>4.658</i>
DSPMUL-18bx18b	2 DSP Blocks	3.833
DSPMUL-24bx24b	8 DSP Blocks	7.688



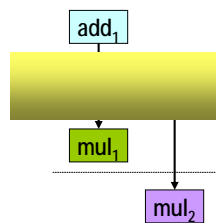
3X3 Delay Matrix

- Platform: Altera Stratix
- RTL synthesis & place-and-route: Altera QuartusII v5.0

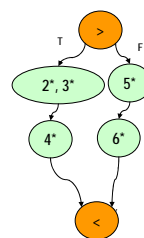
Motivation – Higher Quality of Results (2)

◆ Communication-centric synthesis & optimization with full consideration of physical reality

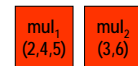
- System performance & power is dominated by interconnect
- It is difficult for designers to consider physical layout at the RT level



Layout-aware performance optimization
Overlap computation with communication



Layout-aware power optimization

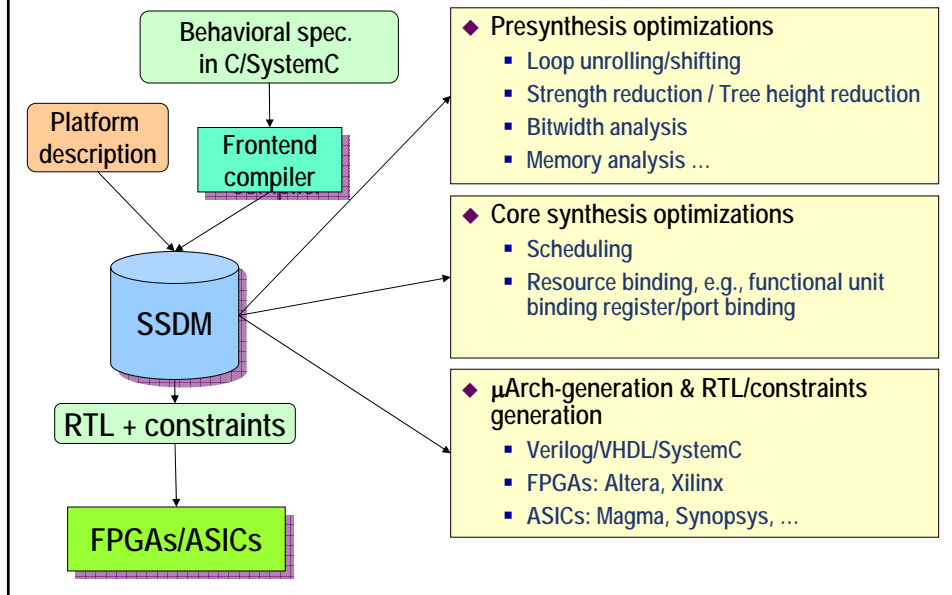


Binding solution 1:
Both multipliers keep active

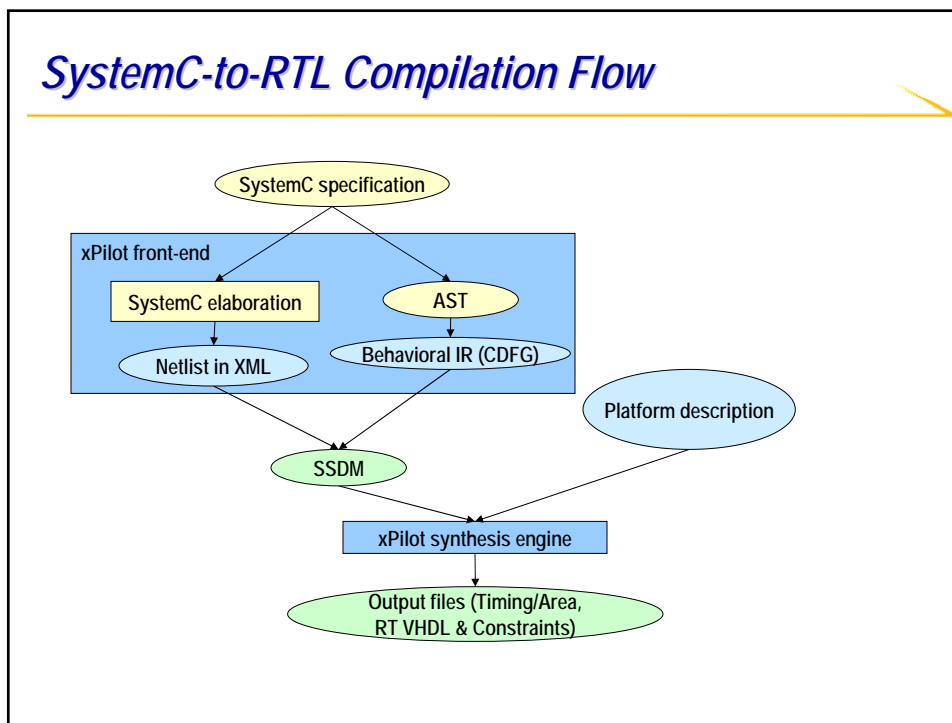


Binding solution 2:
mul₂ can be powered off when false branch is taken

xPilot: Behavioral-to-RTL Synthesis Flow



SystemC-to-RTL Compilation Flow



Restricted Behavioral C Subset

◆ Data types:

- Primitive integer types: char, byte, short, int, long...
- One-dimension arrays of primitive integer types

◆ Operations:

- All arithmetic and logic operations: +, -, *, /, >>, &, ...

◆ Control flow statements:

- while, for, switch-case, if-then-else, break, continue, return, ...

Restricted Behavioral C Subset (cont.)

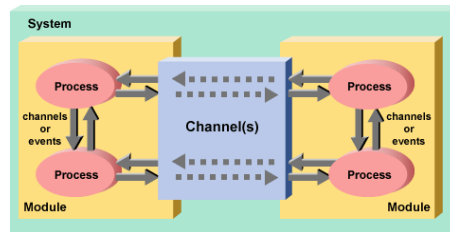
◆ Unsynthesizable

- Recursions
- Pointers
- Dynamic memory allocations and system calls
- Irregular jumps, e.g., gotos

System-level Synthesis Data Model

◆ SSDM (System-level Synthesis Data Model)

- Hierarchical netlist of concurrent processes and communication channels



- Each leaf process contains a sequential program which is represented by an extended LLVM IR with hardware-specific semantics
 - Port / IO interfaces, bit-vector manipulations, cycle-level notations

Hardware-Specific SSDM Semantics

◆ Process port/interface semantics

- FIFO: `FifoRead()` / `FifoWrite()`
- Buffer: `BuffRead()` / `BuffWrite()`
- Memory: `MemRead()` / `MemWrite()`

◆ Bit-vector manipulation

- Bit extraction / concatenation / insertion
- Bit-width attributes for every operation and every value

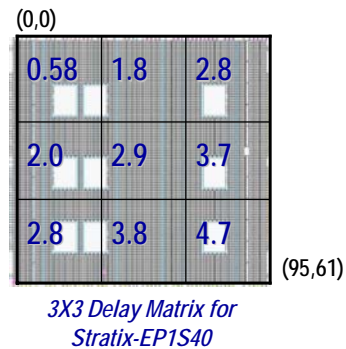
◆ Cycle-level notation

- Clock: `waitClockEvent()`

Platform Modeling & Characterization

◆ Target platform specification

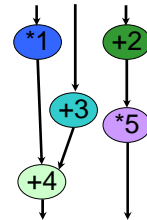
- High-level resource library with delay/latency/area/power curve for various input/bitwidth configurations
 - Functional units: adders, ALUs, multipliers, comparators, etc.
 - Connectors: mux, demux, etc.
 - Memories: registers, synchronous memories, etc.
- Chip layout description
 - On-chip resource distributions
 - On-chip interconnect delay/power estimation



Scheduling– Problem Statement

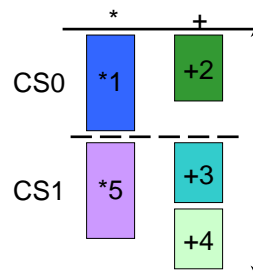
◆ Scheduling problem in behavioral synthesis

- Given:
 - A control data flow graph (CDFG) which captures the behavior of the input description
 - A set of scheduling constraints: resource constraints, latency constraints, frequency constraints, relative IO timing constraints, etc.
- Goal:
 - Assign the operations to control states so that a particular design objective (performance / power) is optimized while all the constraints are satisfied.



◆ Highlights of our scheduling engine

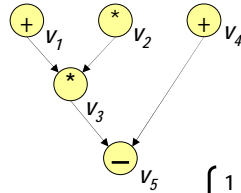
- Applicable to a wide range of application domains
 - Computation-intensive, memory-intensive, control-intensive, partially timed, etc.
- Offers a variety of optimization techniques in a unified framework
 - Operation chaining, behavioral template, relative scheduling, physical layout consideration, etc.



Scheduling – Overall Approach

◆ Overall approach

- Current objective: high-performance
 - Use a system of integer difference constraints to express all kinds of scheduling constraints
 - Represent the design objective in a linear function
- Dependency constraint
 - $V_1 \rightarrow V_3: x_3 - x_1 \geq 0$
 - $V_2 \rightarrow V_3: x_3 - x_2 \geq 0$
 - $V_3 \rightarrow V_5: x_4 - x_3 \geq 0$
 - $V_4 \rightarrow V_5: x_5 - x_4 \geq 0$
 - Frequency constraint
 - $\langle V_2, V_5 \rangle: x_5 - x_2 \geq 1$
 - Resource constraint
 - $\langle V_2, V_3 \rangle: x_3 - x_2 \geq 1$

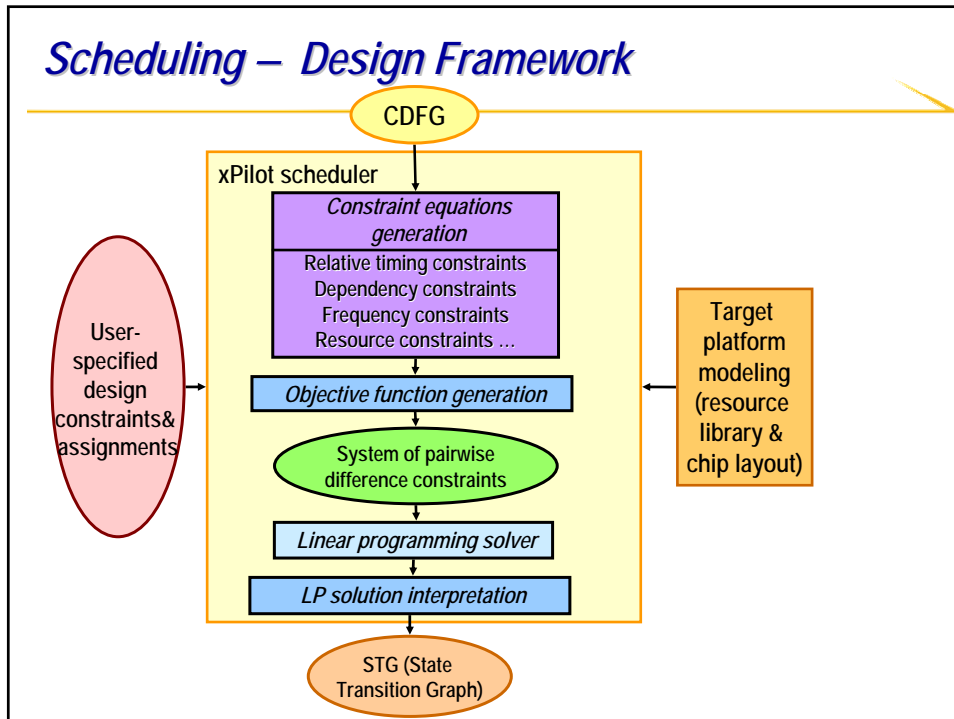


- Platform characterization:
 - adder (+/-) 2ns
 - multiplier (*): 5ns
- Target cycle time: 10ns
- Resource constraint: Only ONE multiplier is available

$$\begin{matrix}
 \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 & -1 \end{bmatrix} & \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} & \leq & \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ -1 \end{bmatrix} \\
 A & x & & b
 \end{matrix}$$

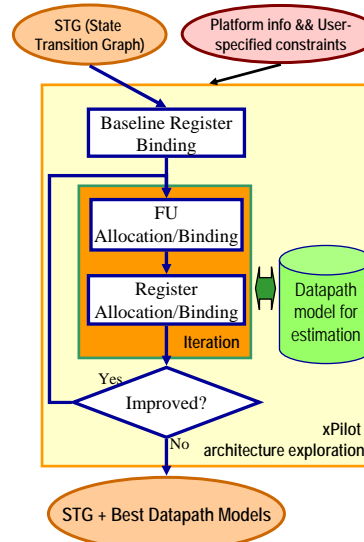
Totally unimodular matrix: guarantees integral solutions

Scheduling – Design Framework



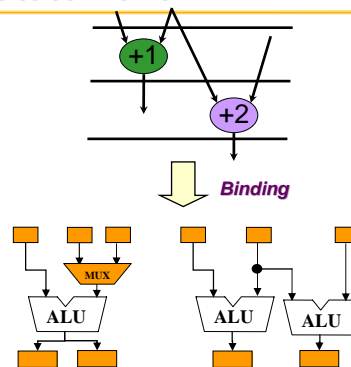
Unified Resource Binding

- ◆ An efficient architectural exploration framework
- ◆ Simultaneous functional unit, register, and port binding
- ◆ Emphasize on the interconnect and steering logic networks
- ◆ Guided by a flexible cost evaluation engine to achieve different objectives, e.g., performance, area, power, etc.
- ◆ Extendable to exploit physical layout information



Resource Binding– Problem Statement

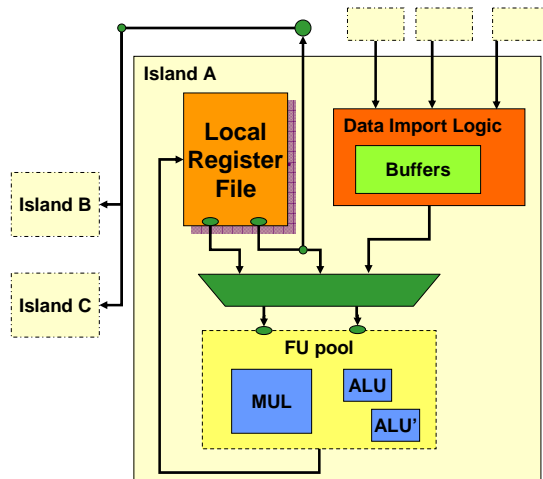
- ◆ Resource binding problem
 - **Given:** (1) A scheduled control data flow graph, i.e., STG; (2) Design constraints: performance, delay, or power, etc.
 - **Goal:** Assign the operations and variables to functional units and register, respectively, so that their executions or lifetimes are not conflicted, and all of the design constraints are satisfied.
- ◆ Properties of the problem
 - FU and register binding are highly correlated
 - Simultaneous FU and register binding considering interconnection is very difficult



Two binding solutions:

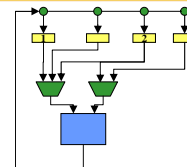
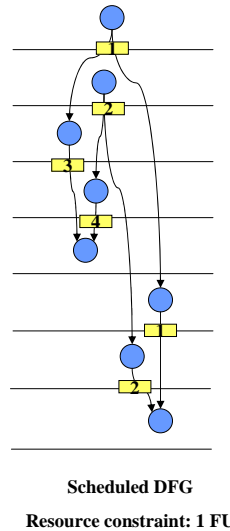
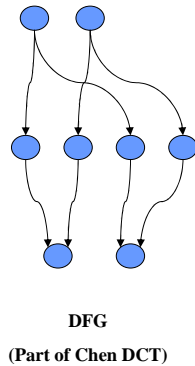
- ◆ Which one is better?
- ◆ The answer depends on:
 1. How large are the MUX and ALU (platform-dependent)
 2. Performance and area constraints

Distributed Register-File (DRF) Microarchitecture



- ◆ Regular datapath structure
- ◆ Provides opportunities to hide large MUX into register-files
- ◆ Computations and communications are localized
 - Allow replicated values among islands
 - Enables efficient optimizations to control interconnects among islands

Advantages of DRF Microarchitecture



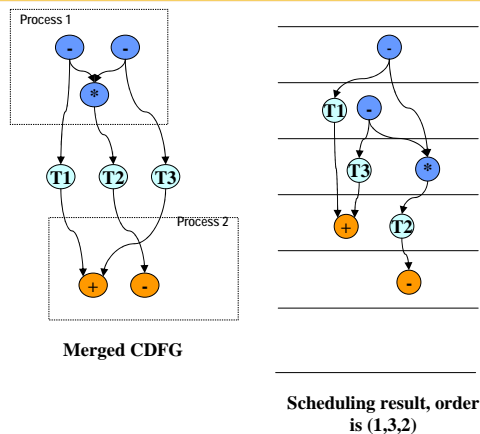
- ◆ Discrete register result
 - ◆ MUX implementation may be very expensive (e.g., on FPGAs)
- DRF result:
- Datapath with more regularity
 - Hide MUX into the register file
 - Especially effective for FPGA designs

Platform-Based Interface Synthesis

- ◆ Focus on sequential communication channels
 - Data must be read and written in the same order
 - Example: FIFO (FSL in VirtexII), Bus (in both Stratix and Virtex)
 - Order may have dramatic impact on performance
 - Best order should guarantee that no data transmission on critical path are delayed by non-critical transmission
- ◆ Interface synthesis for sequential communication channels
 - Consider both the behavior model and communication topology to detect the optimal transmission order
 - Automatically do interface generation for sequential communication units, as well as code transformation for behavior models

Overall Approach to Interface Synthesis

- ◆ Reduce the order detection problem to a min-latecncy scheduling problem:
 - Merge the CDFGs of all processes
 - Each element to be transferred on FIFO are transformed to a special operation T
 - Only one T can be scheduled at each step.
- ◆ Example shown on right, assuming only 1 cycle is needed for FIFO operation

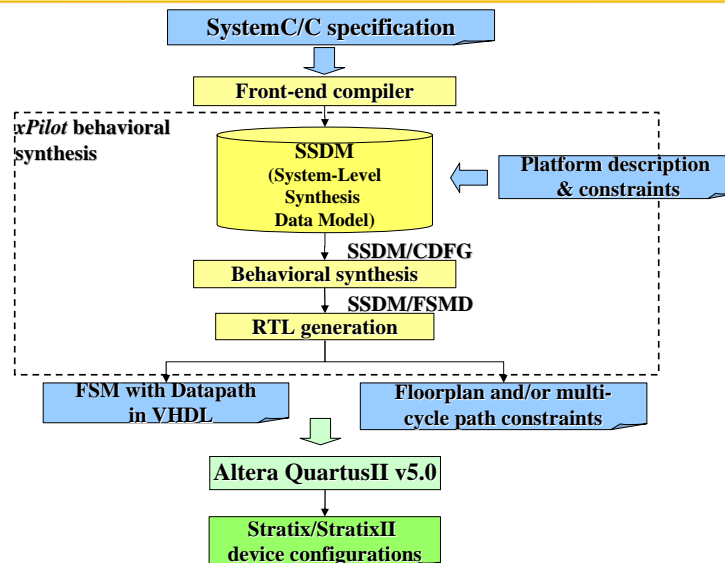


Experimental Results – Benchmark Suite

◆ Benchmark suite

- PR, MCM:
 - DSP kernels: pure additions/subtractions and multiplications
- CACHE
 - Cache controller: control-intensive designs with cycle-accurate I/O operations
- MOTION:
 - Motion compensation algorithm for MPEG-1 decoder: control-intensive with modest amount of computations
- IDCT:
 - JPEG inverse discrete cosine transform: computation intensive
- DWT:
 - JPEG2000 discrete wavelet transform: computation intensive with modest control flow
- EDGLOOP:
 - Extracted from H.264 decoder: a very complex design, features a mix of computation, control, and memory accesses

SystemC/C-to-FPGA Design Flow (Altera)



Experimental Results – Altera

Designs	Line Count		Resource Usage					F _{max}
	C	VHDL	LE	COMB	Lonely-Reg	Comb-Reg	DSP	(MHz)
PR	90	600	1349	713	84	552	0	178.7
WANG	90	727	1105	527	62	516	8	166.11
LEE	141	696	1585	691	207	687	4	166.61
MCM	161	1260	2402	981	73	1348	0	152.56
DIR	190	1352	3489	1752	110	1627	4	146.8

- ◆ Device setting: Stratix
- ◆ Target frequency: 200 MHz

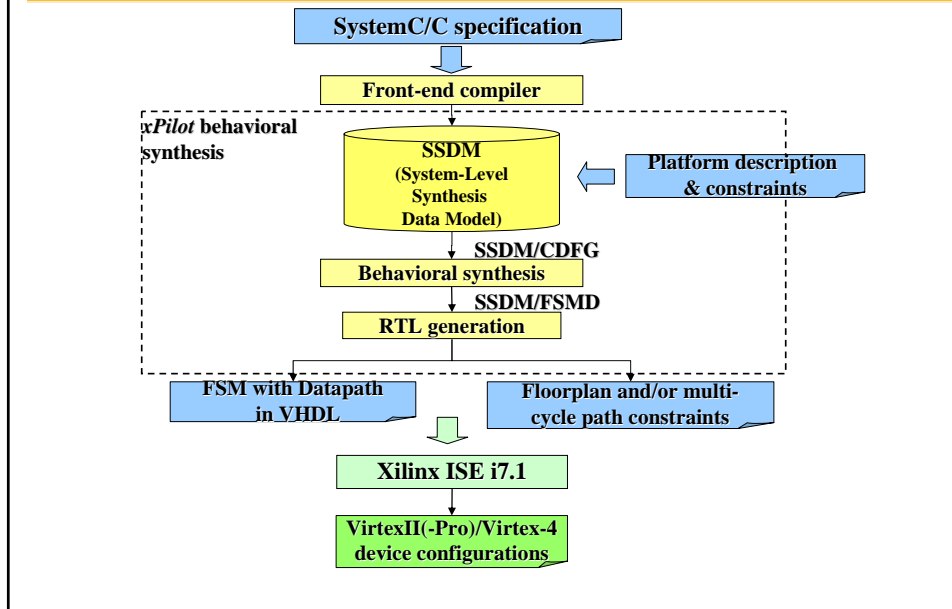
Experimental Results – Comparison with SPARK on Altera Stratix FPGA

- ◆ SPARK [UCI/UCSD, 2004], a state of the art academic high-level synthesis tool

Designs	SPARK						xPilot					
	Resource Usage					F _{max}	Resource Usage					F _{max}
	LE	COMB	Lonely-Reg	Comb-Reg	DSP	(MHz)	LE	COMB	Lonely-Reg	Comb-Reg	DSP	(MHz)
PR	1108	815	0	293	0	123.5	1349	713	84	552	0	178.7
WANG	1217	942	0	275	0	118.9	1105	527	62	516	8	166.1
LEE	1367	1052	0	315	0	119.3	1585	691	207	687	4	166.6
MCM	2808	2248	0	560	0	74.87	2402	981	73	1348	0	152.6
DIR	2425	2034	0	391	6	69.38	3489	1752	110	1627	4	146.8
Ave Ratio	1	1	1	1	1	1	1.12	0.68	n/a*	2.50	n/a*	1.68

- ◆ On average, xPilot resource binding achieves designs with similar area, and 1.68x higher frequency over Spark

SystemC/C-to-FPGA Design Flow (Xilinx)



Experimental Results – Xilinx

Designs	Line Count		Resource Usage				Fmax
	C	VHDL	Slices	(LUT)	(FF)	DSP	(MHz)
PR	90	600	331	416	564	16	146.84
WANG	90	727	357	464	588	15	133.51
LEE	141	696	356	484	659	19	131.93
MCM	161	1260	887	1207	1282	30	110.38
DIR	190	1352	979	1002	1732	56	98.81

◆ Device setting: xc2vp30 -7

◆ Target frequency: 200 MHz

Synthesis from Behavior to DRF

- ◆ Data import logic is the most “critical”
 - Operations bound to an island form a “chain” in DFG
 - Optimize complexity of inter-island connections
 - Min-cut chain partitioning → improve design quality

	Micro-Architecture	Slices	LUT	FF	RAM Blk#	MUL	Clock Period (ns)
CHEN-24	DRF (6 islands)	425	808	86	6	6	12.04
	Baseline	798	928	1,235	0	66	10.37
PR-24	DRF (5 islands)	457	860	63	5	3	10.62
	Baseline	701	858	1,076	0	48	11.69

- ◆ Device: Xilinx Virtex II -6; Target clock period: 10ns
- ◆ Observations: Large area (Slices and Multiplier blocks) reduction by using on-chip RAM blocks to implement register files, with small impact on Fmax

Initial Results of Interface Synthesis

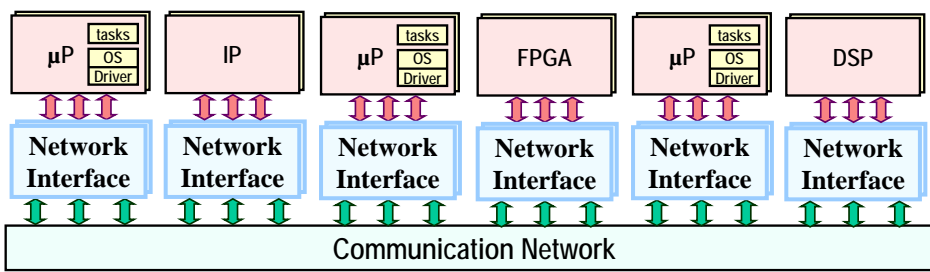
- ◆ Target for sequential communication channels
 - In particular, FSL in VirtexII
- ◆ Consider two communicating processes
- ◆ 20+% performance improvement on average by optimizing communication ordering

Outline

- ◆ Motivation
- ◆ xPilot system framework
- ◆ Behavior-level synthesis in xPilot
 - Advantages of behavioral synthesis
 - Scheduling
 - Resource binding
- ◆ System-level synthesis in xPilot
 - Synthesis for ASIP platforms
 - Design exploration for heterogeneous MPSoCs
- ◆ Conclusions

Design Exploration for Heterogeneous MPSoC Platforms

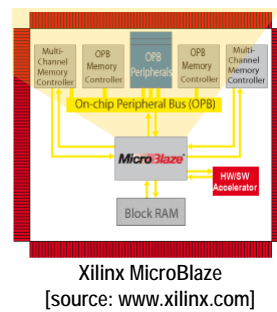
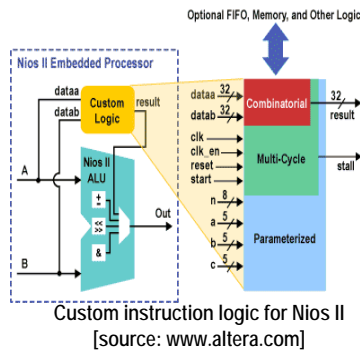
- ◆ Heterogeneous MPSoCs exploration
 - Processors
 - Heterogeneous vs. homogeneous
 - General-purpose vs. application-specific
 - On-chip communication architecture (OCA)
 - Bus (e.g. AMBA, CoreConnect), packet switching network (e.g. Alpha 21364)
 - Memory hierarchy



Configurable SoC Platforms

◆ General purpose processor cores + programmable fabric

- Tight integration using extended instructions (ASIPs)
 - Example: Altera Nios / Nios II
- Loose integration using FIFOs/busses for communications
 - Example: Xilinx MicroBlaze, etc.



ASIP Compilation: Problem Statement

◆ Given:

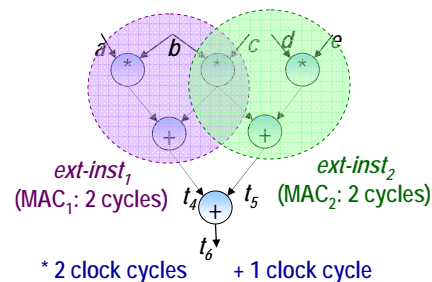
- CDFG $G(V, E)$
- The basic instruction set I
- Pattern constraints:
 - Number of inputs $|PI(pi)| \leq Nin$;
 - Number of outputs $|PO(pi)| = 1$;
 - Total area $\sum_{1 \leq i \leq N} area(p_i) < A$

◆ Objective:

- Generate a pattern library P
- Map G to the extended instruction set $I \cup P$, so that the total execution time is minimized

$$\begin{aligned}
 t_1 &= a * b; & t_4 &= ext-inst_1(a, b, c); \\
 t_2 &= b * c; & t_5 &= ext-inst_2(b, c, d, e); \\
 t_3 &= d * e; & t_6 &= t_4 + t_5; \\
 t_4 &= t_1 + t_2; \\
 t_5 &= t_2 + t_3; \\
 t_6 &= t_5 + t_4;
 \end{aligned}$$

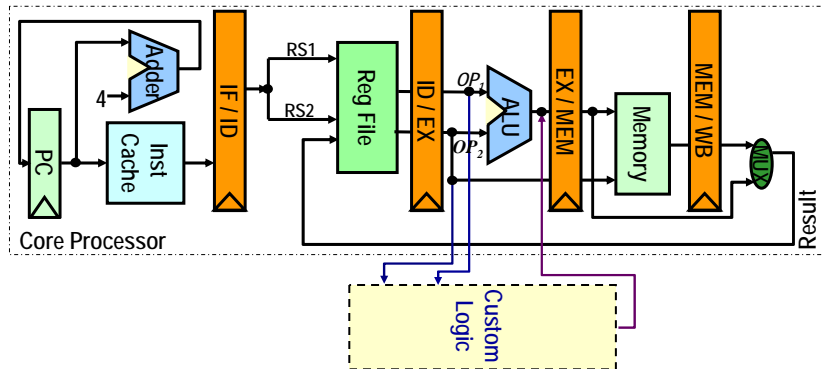
Performance speedup = $9 / 5 = 1.8X$



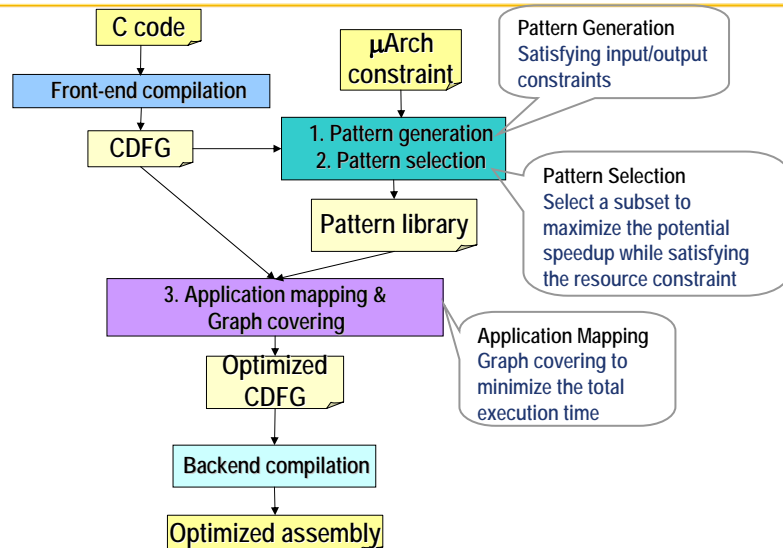
Target Core Processor Model

◆ Core processor model

- Classic single-issue pipelined RISC core (fetch / decode / execute / mem / write-back)
 - The number of input and output operands of an instruction is pre-determined
 - An instruction reads the core register file during the execute stage, and commits the result during the write-back stage



ASIP Compilation Flow



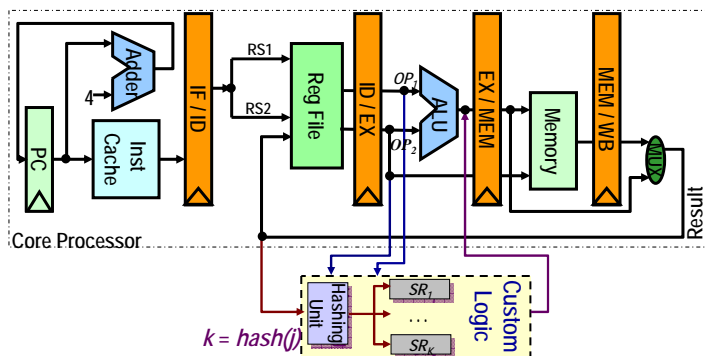
Experimental Results on Altera Nios

- ◆ Altera Nios is used for ASIP implementation
 - 5 extended instruction formats
 - up to 2048 instructions for each format
- ◆ Small DSP applications are taken as benchmark

	Extended Instruction#	Speedup		Resource Overhead				
		Estimation	Nios	LE		Memory		DSP Block
fft_br	9	3.28	2.65	408	6.06%	65,536	9.79%	16
iir	7	3.18	3.73	255	3.79%	4,736	0.71%	40
fir	2	2.40	2.14	51	0.76%	1,024	0.15%	8
pr	2	1.57	1.75	71	1.05%	0	0.00%	14
dir	2	3.28	3.02	54	0.80%	0	0.00%	16
mcm	4	4.75	3.22	186	2.76%	0	0.00%	56
Average		3.08	2.75	-	2.54%	-	1.77%	-

Architecture Extension for ASIPs

- ◆ Data bandwidth problem
 - Limited register file bandwidth (two read ports, one write port)
 - ~40% of the ideal performance speedup will be lost
- ◆ Shadow-register-based architectural extension
 - Core registers are augmented by an extra set of shadow registers
 - Conditionally written during write-back stage
 - Low power/area overhead
 - Novel shadow-register binding algorithms are developed



Problem Statement: Mapping for Heterogeneous Integration with Multiple Processing Cores

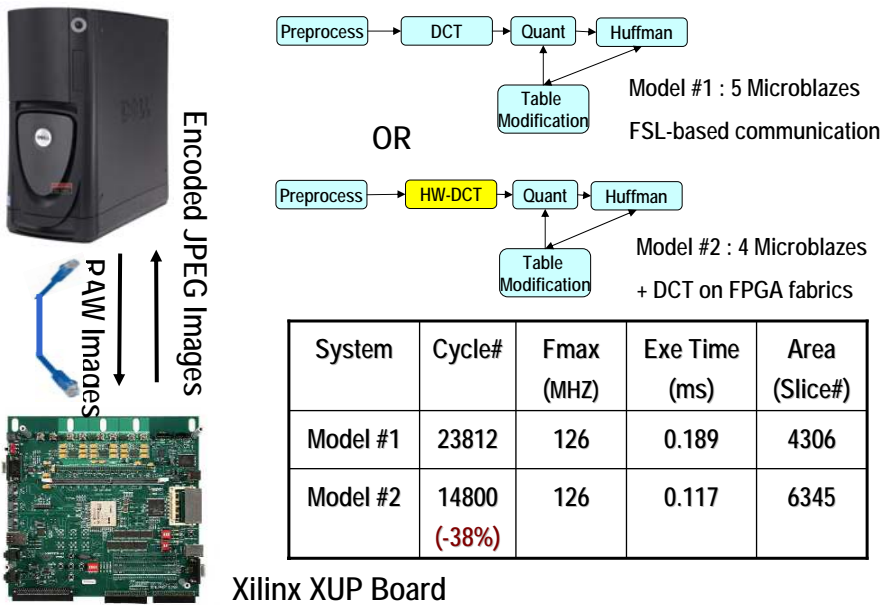
◆ Given:

- A library of processing cores L
- Task graph $G(V, E)$
 - For each v in V , execution time $t(v, p_i)$ on p_i
 - For each (u, v) in E , communication data size $s(u, v)$
- Cost (area/power) constraint C

◆ Problem:

- Select and instantiate the processing elements from L
- Generate the on-chip communication architecture and topology
- Map the tasks onto the processing elements so that
 - The total latency is minimized while the final implementation cost is less than C

Preliminary Results on Motion-JPEG Example



Conclusions

- ◆ xPilot can automatically synthesize behavior level C or SystemC presentation to RTL code with necessary design constraints
- ◆ Platform-based synthesis with physical planning provides
 - Shorter verification/simulation cycle
 - Better complexity management, faster time to market
 - Rapid system exploration
 - Higher quality of results
- ◆ xPilot can help to explore the efficient use of (multiple) on-chip processors
- ◆ xPilot can efficiently optimize the software for reconfigurable processors
- ◆ We are interested to engage with selected industrial partners to further validate and enhance the technology

Acknowledgements

- ◆ We would like to thank the supports from
 - National Science Foundation (NSF)
 - Gigascale Systems Research Center (GSRC)
 - Semiconductor Research Corporation (SRC)
 - Industrial sponsors under the California MICRO programs (Altera, Xilinx)
- ◆ Team members:



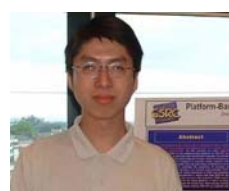
Yiping Fan



Guoling Han



Wei Jiang



Zhiru Zhang