Custom Sensor-Based Embedded Computing Systems

Frank Vahid
Professor
Dept. of Computer Science and Engineering
University of California, Riverside
Assoc. Director, Center for Embedded Computer Systems, UC Irvine

eBlocks project 2002-present, support provided by the National Science Foundation and Intel

Ph.D. student: Susan Lysecky (2006, now Asst. Prof. at U. Arizona); several MS and undergrad students also

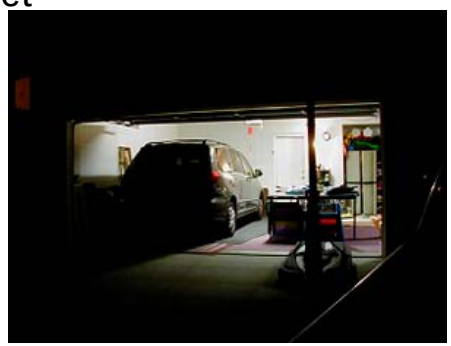
UNIVERSITY OF CALIFORNIA
RIVERSIDE

The Problem

- What do these problems all have in common?

A working adult with an ageing parent at home – did she get out of bed today, is she moving around?

A small store owner with many employees – are they in the storeroom, breakroom, or out back?



A homeowner who sometimes forgets to close the garage at night

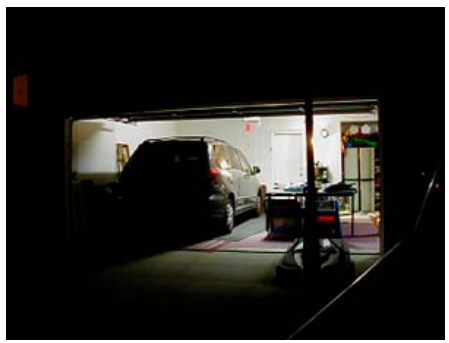
Marines wishing to outfit a building to detect whether someone is inside or when someone was inside

The Problem

- What do these problems all have in common?

Put motion sensors around the house, monitor from the web or cell phone – or even be notified if no motion by certain time in the morning

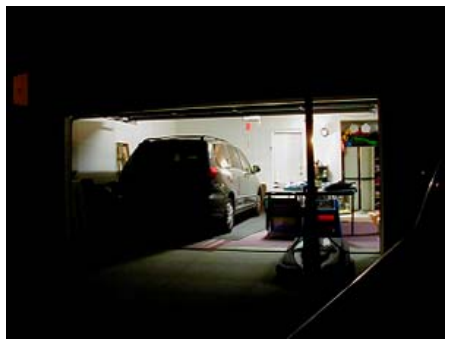
Put motion and sound sensors throughout, small LEDs (lights) near cash register

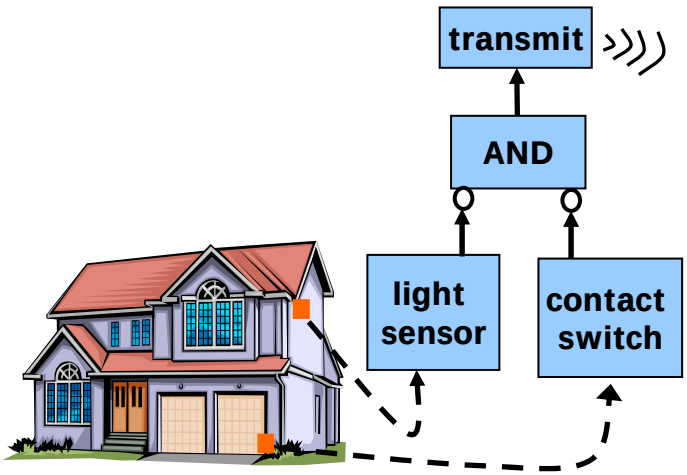


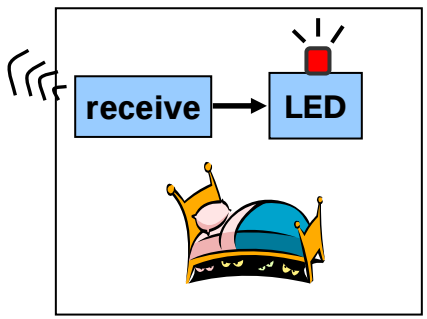
Install contact sensor and light sensor, and indicator next to the bed

Place motion, heat, and sound sensors in rooms, halls, doorways

Why Can't We Just Do This?

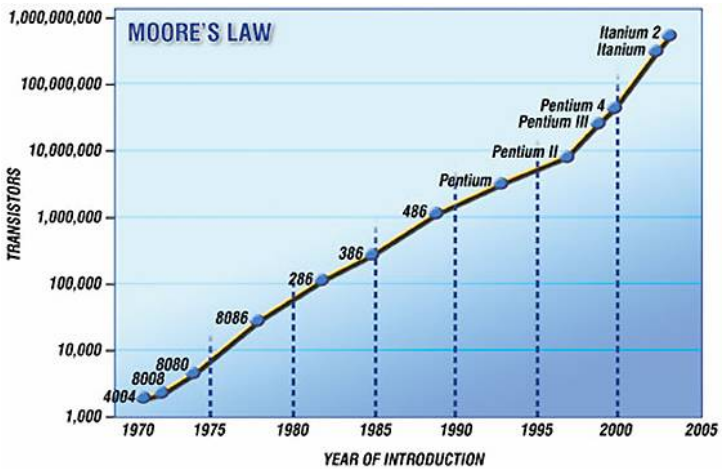


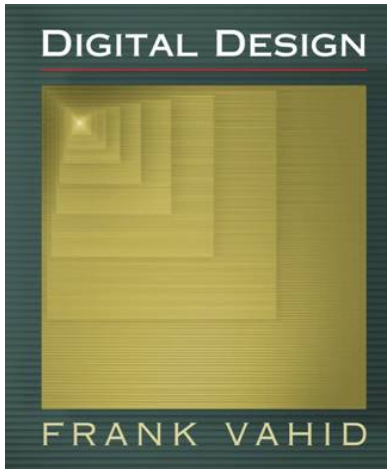




- Widely usable "Lego"-like sensors don't exist today
  - Costly, hard to use, plugged into wall…
- But new technology makes Lego-like sensor blocks possible…

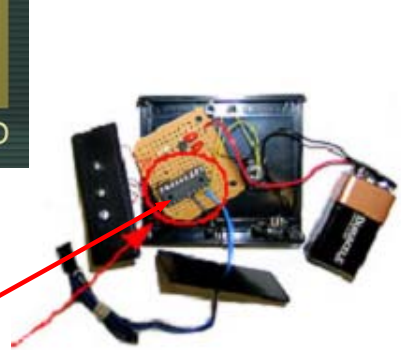
Shrinking Processor Size/Cost Enables New Solution



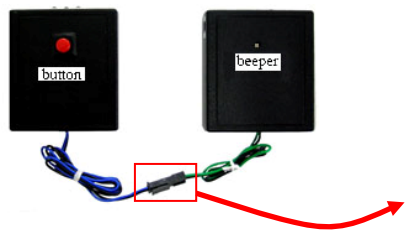


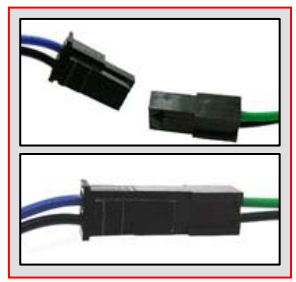
Courtesy of Joe Kahn

http://www.templehealth.org

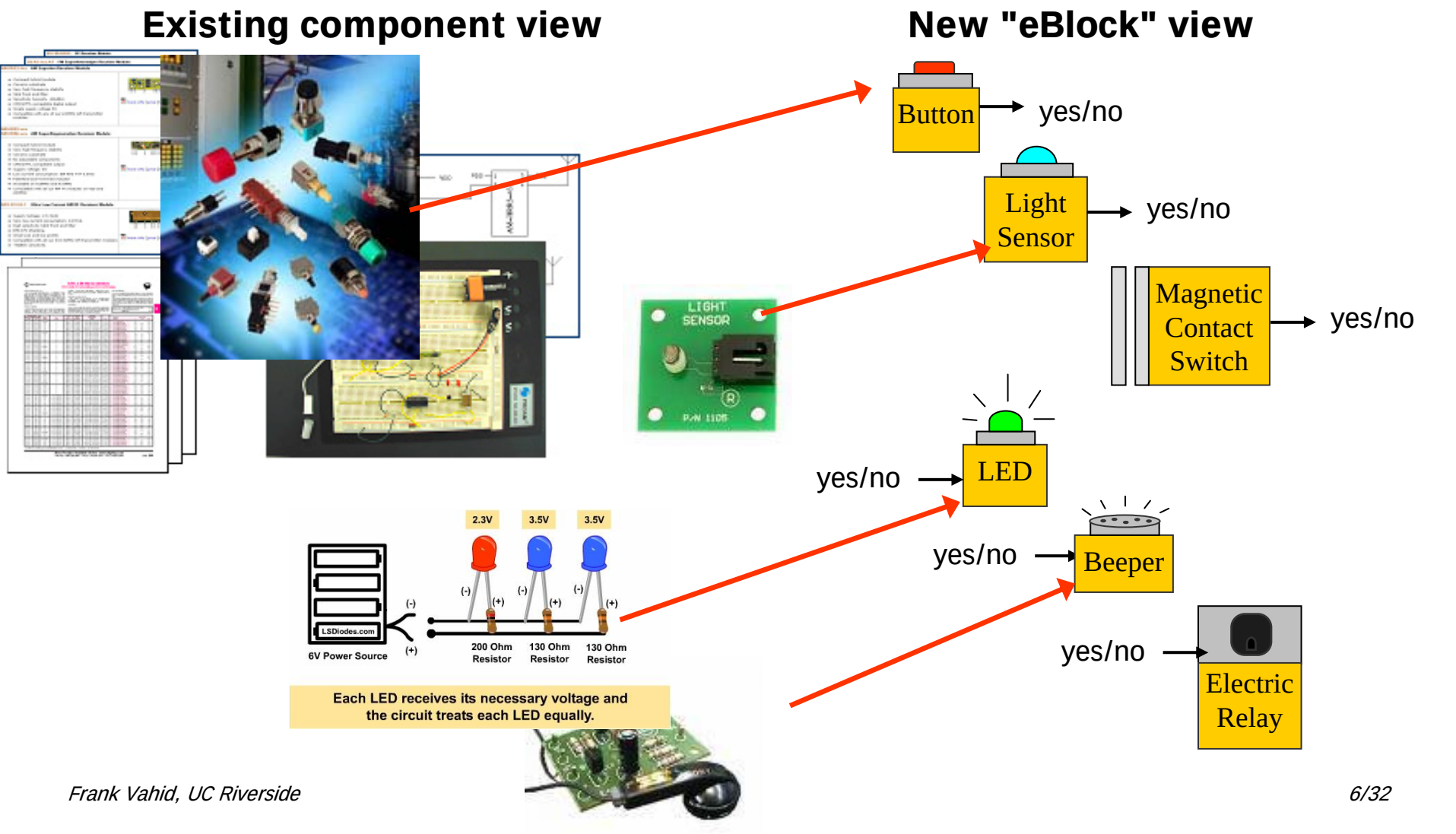


- Make sensors smarter
  - By adding processor+battery
    - Today, tiny and cheap
  - Becomes a "block" easily connected to other blocks



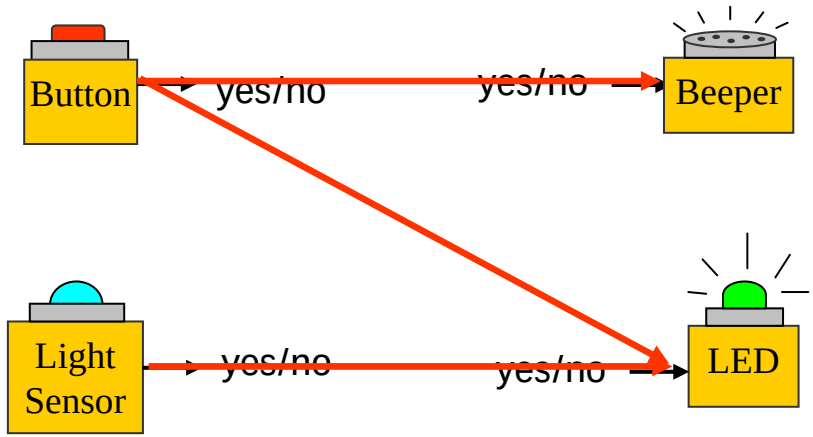


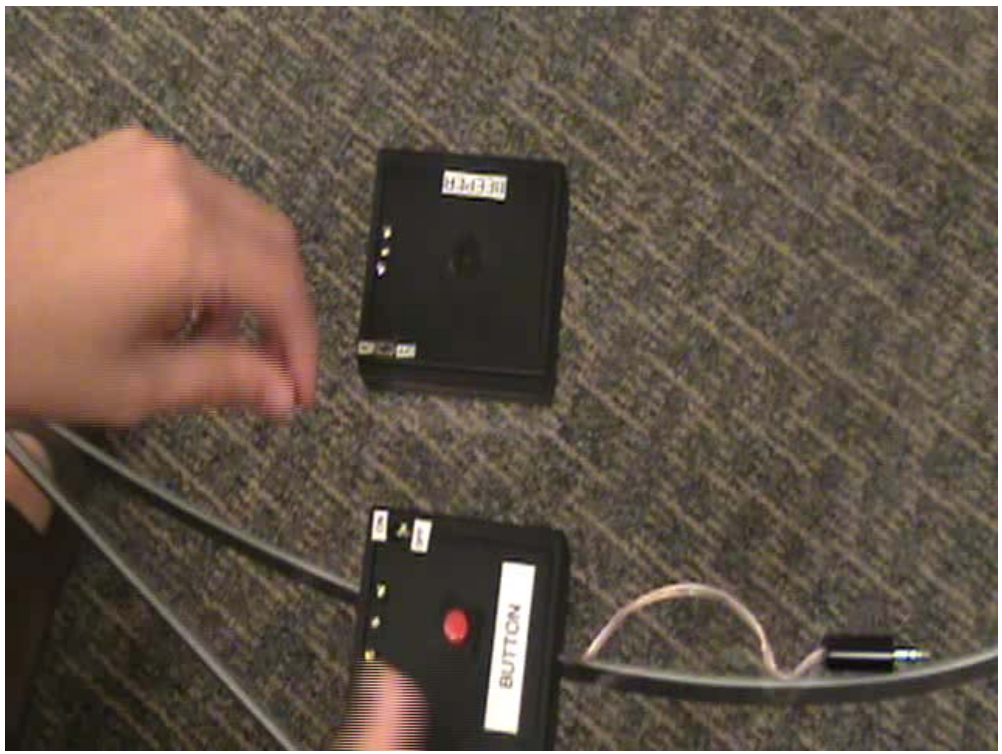
Shrinking Processor Size/Cost Enables New Solution – eBlocks



eBlocks

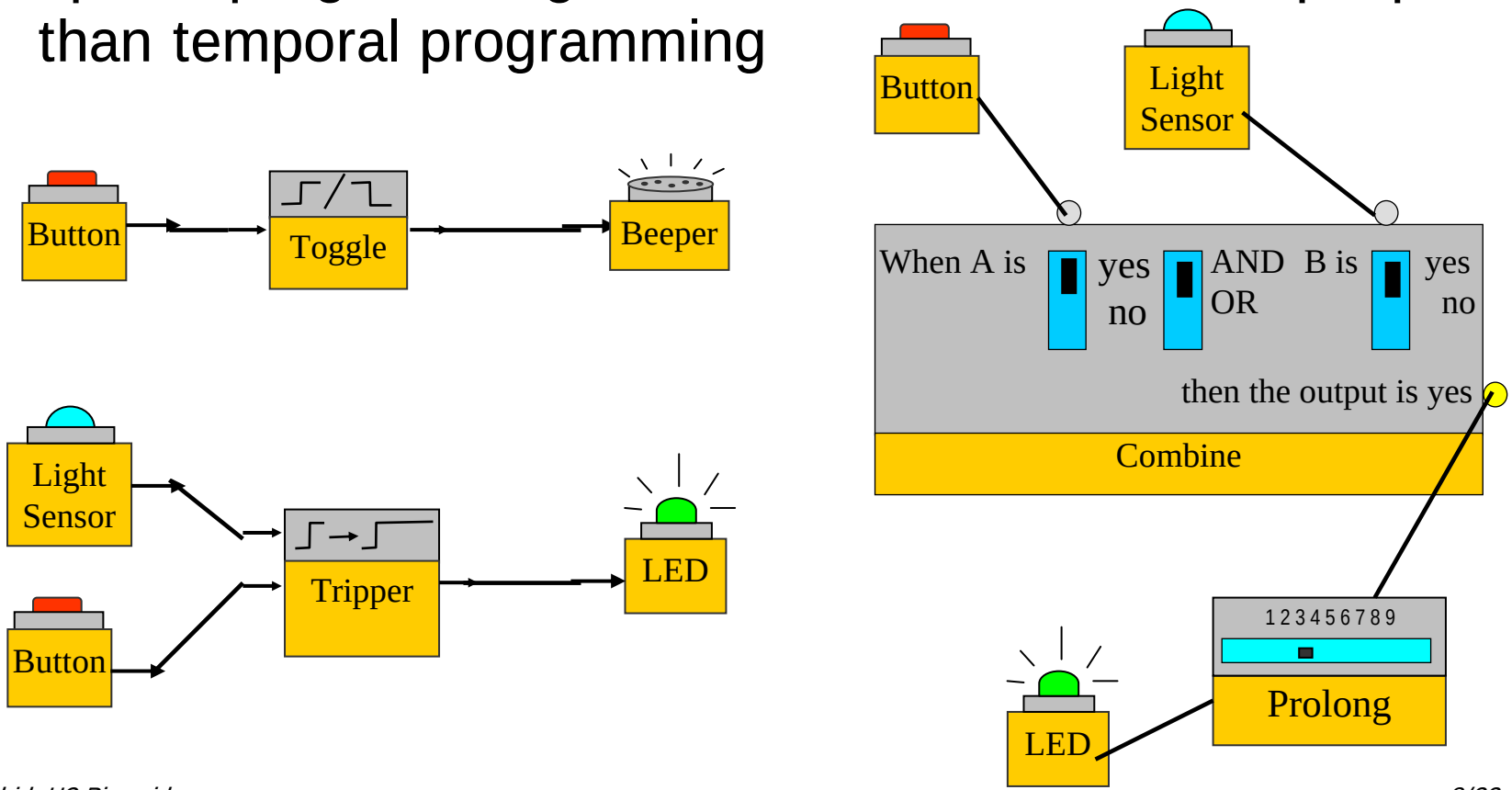
- Just connect blocks, and they work
  - *No* programming knowledge, *no* electronics knowledge





eBlocks

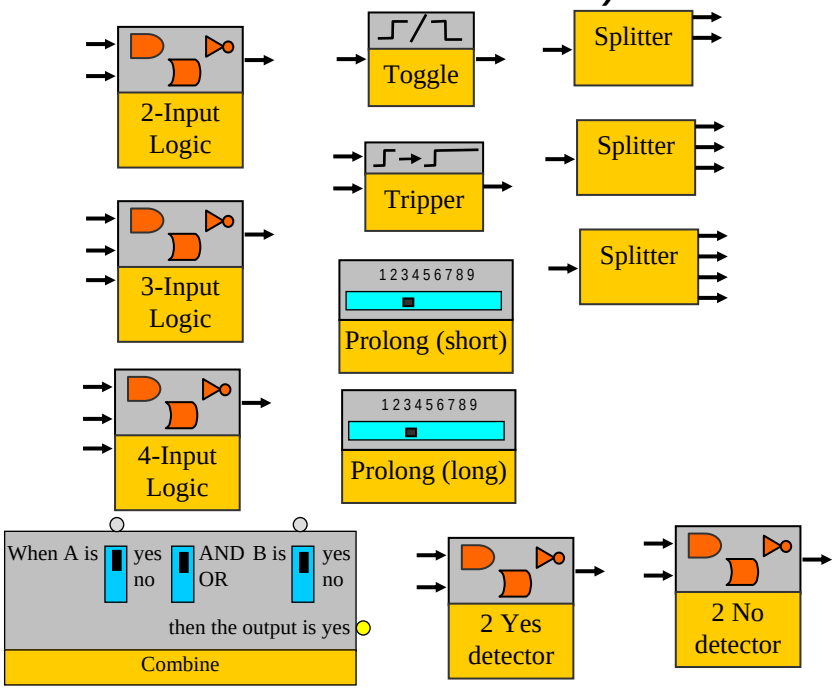
- Add intermediate blocks that compute and maintain state
  - Spatial programming – more intuitive to non-CS people than temporal programming



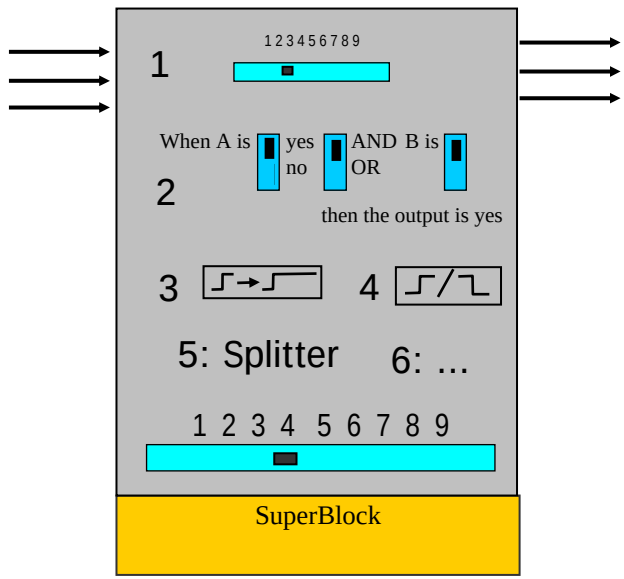
What's Hard

- (1) Finding right set of building blocks

Too many – Overwhelming (too much choice)



Too few – Overwhelming (too much configuration)

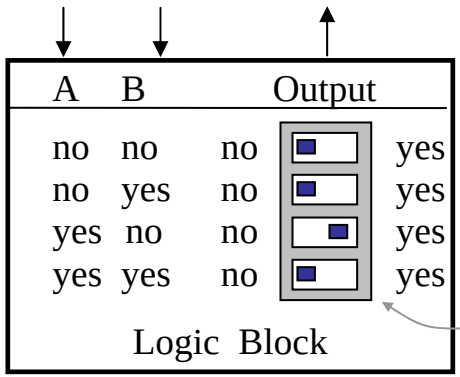


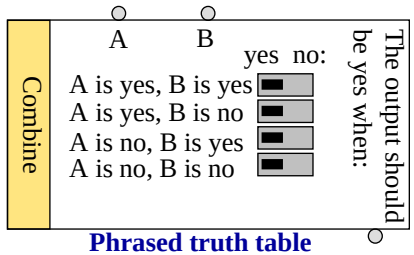
What's Hard

- (2) Making the blocks understandable
  - People NOT likely to read directions
    - Those that do are unlikely to understand

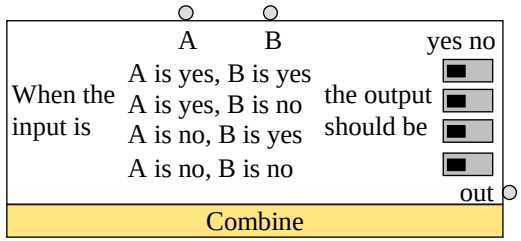
Performed extensive user testing (over 500 students, kids, and adults) over two years

Example: *Combine* block

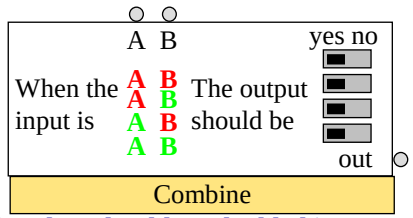




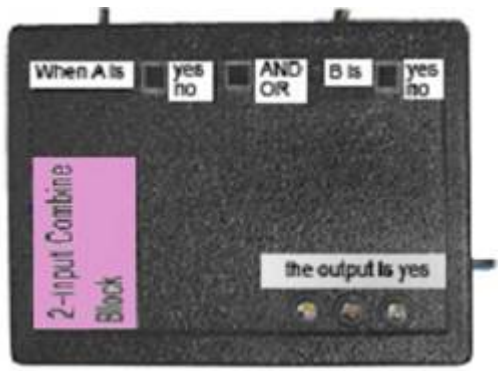
Phrased truth table



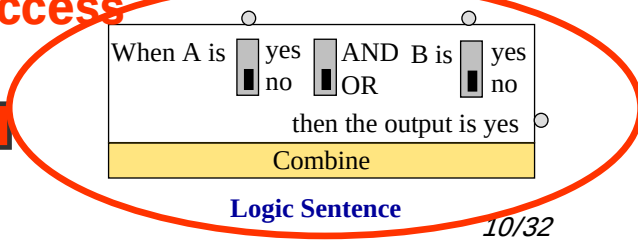
Phrased truth table embedded in sentence



Colored truth table embedded in sentence



Most success

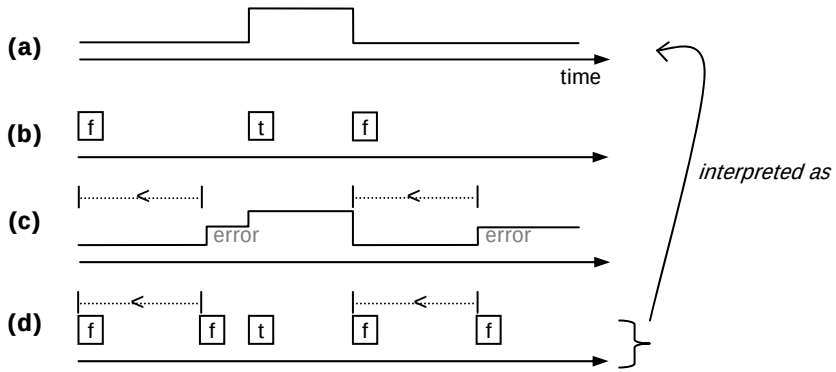


Logic Sentence

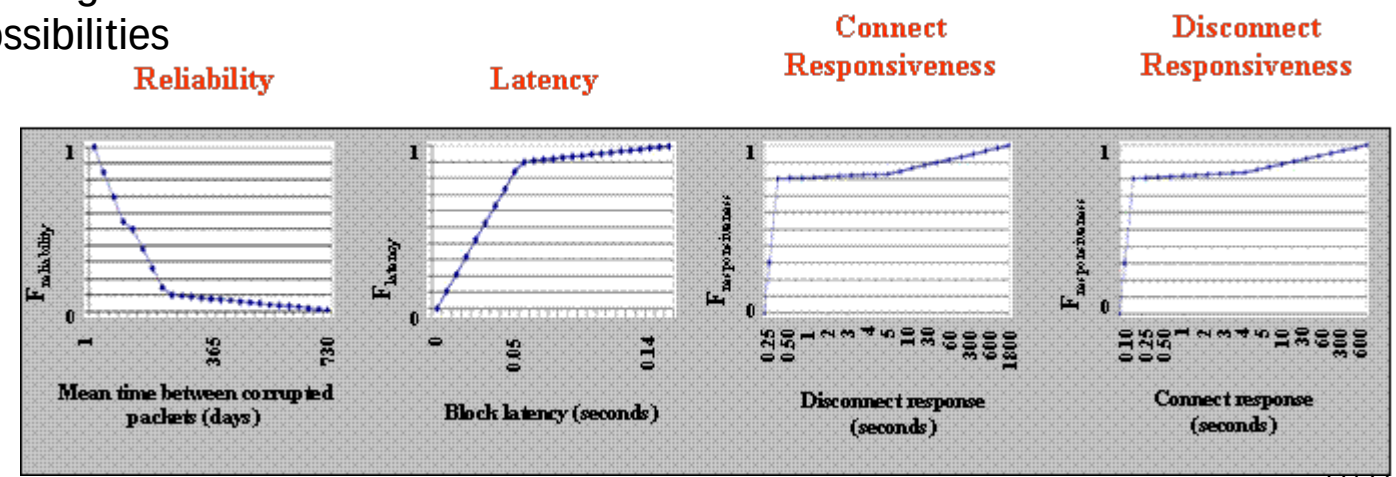
What's Hard

- (3) Batteries must last years, yet performance should appear continuous
  - Blocks are off 99.9% of the time

Developed theory to map eBlock events to continuous time

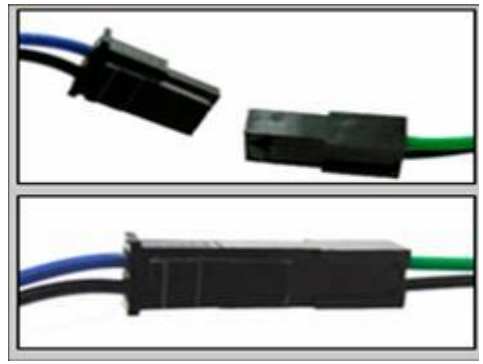


Developed custom CAD tool to automatically find the best block parameter settings out of the billions of possibilities

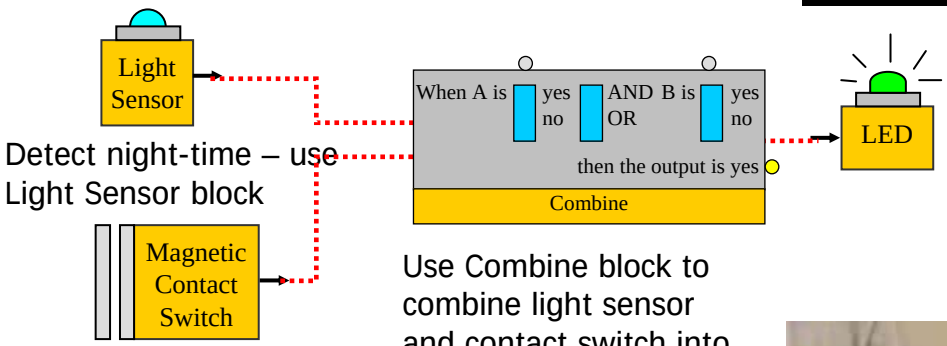


eBlocks Example

- "Garage Open at Night" detector
- <10 minutes to build



Plug pieces together and the system is done!



Detect night-time – use Light Sensor block

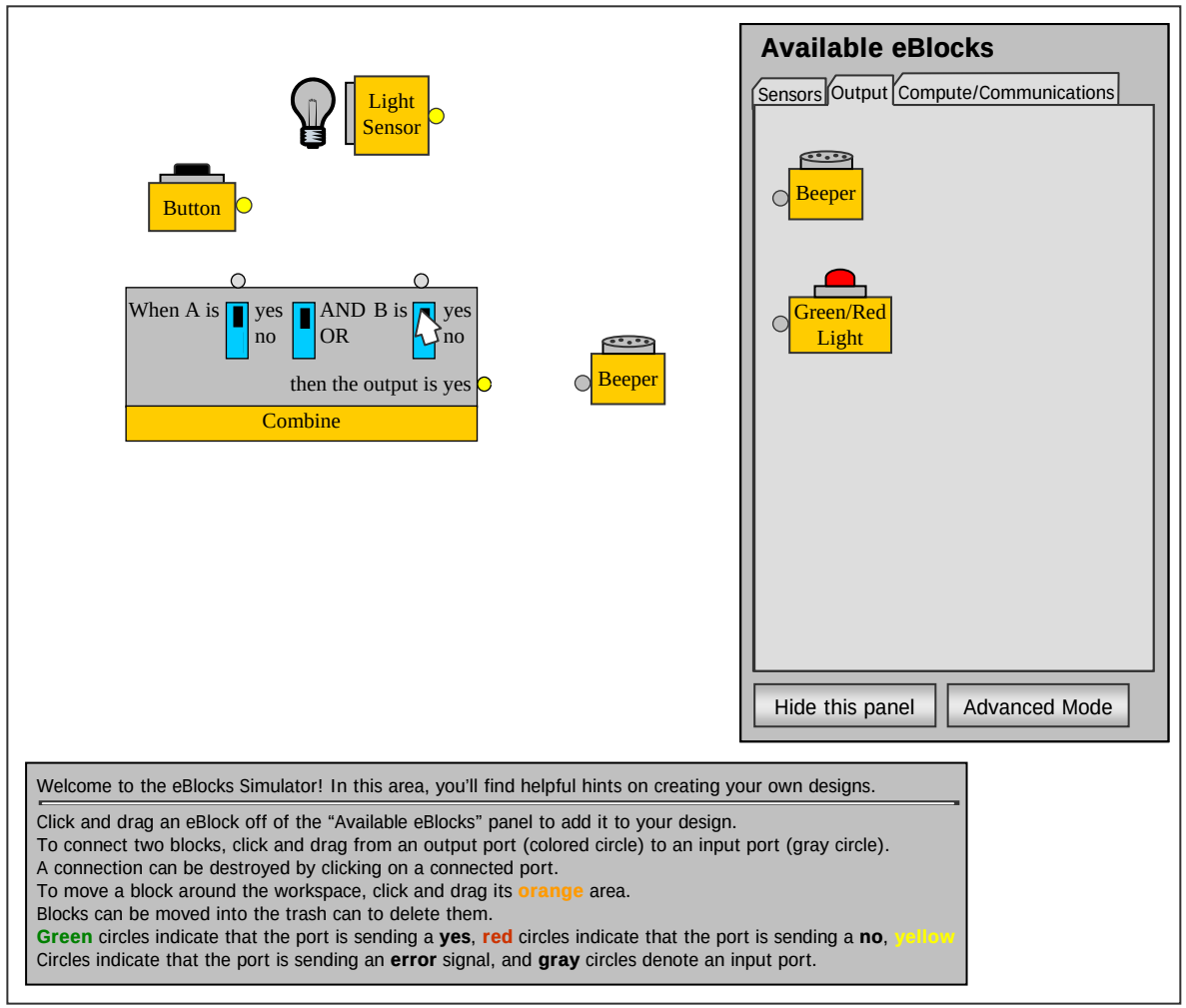
Detect garage door open – use Contact Switch block

Use Combine block to combine light sensor and contact switch into one

Need to indicate garage open at night – use LED block

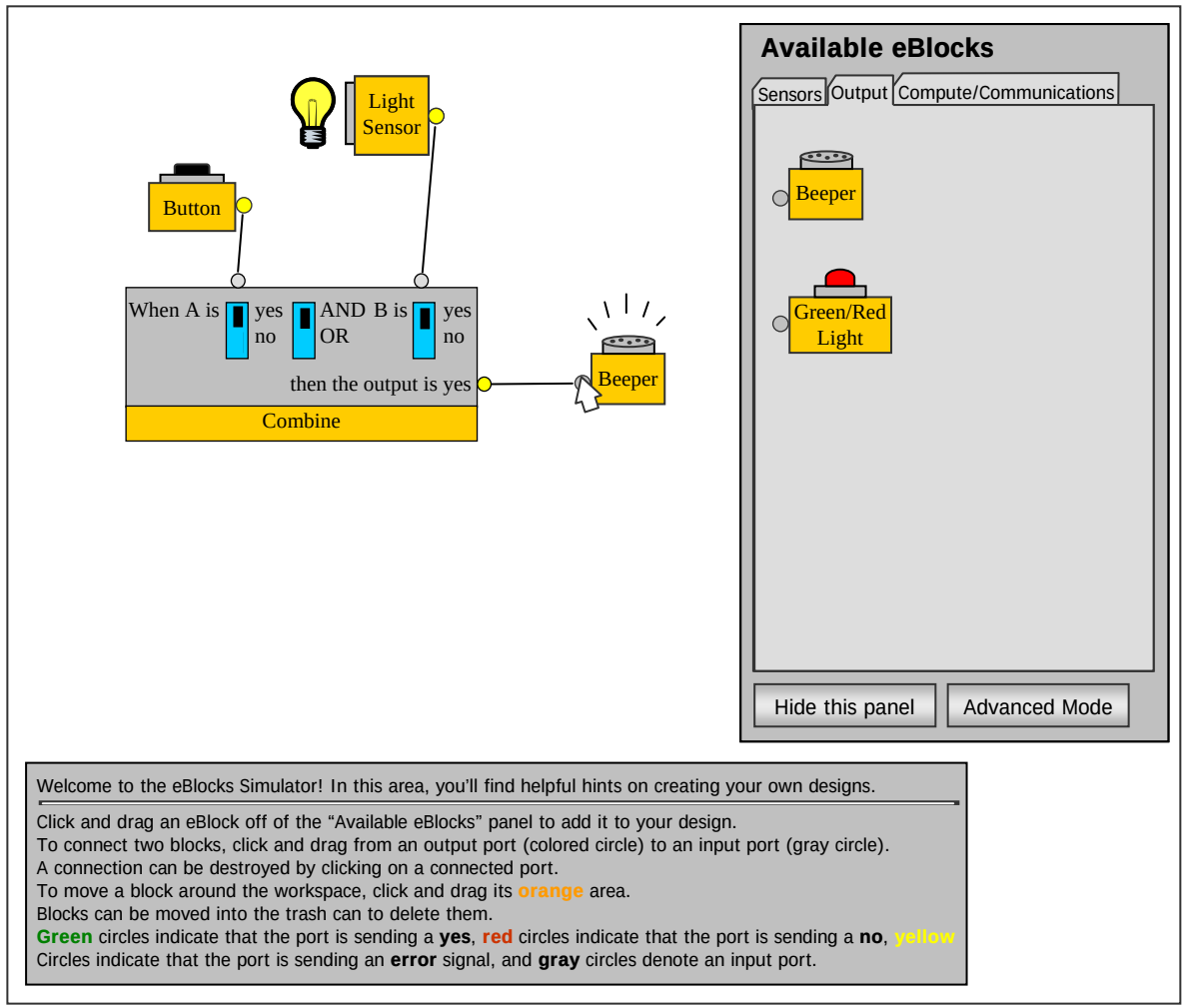
Graphical Simulator

- User specifies and tests block design
- Java-based simulator
  - User chooses between pallets
  - Blocks added by dragging
  - User is able to configure various blocks by clicking on switches
  - Connections created by drawing lines between blocks



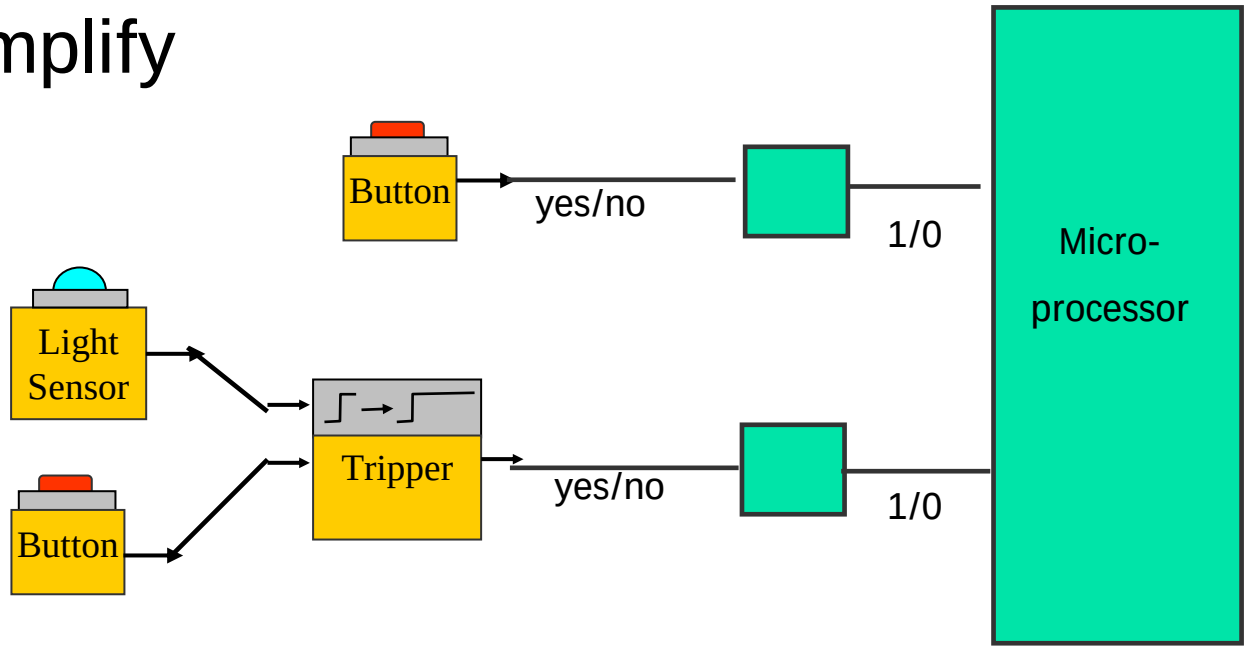
Graphical Simulator

- User specifies and tests block design
- Java-based simulator
  - User chooses between pallets
  - Blocks added by dragging
  - User is able to configure various blocks by clicking on switches
  - Connections created by drawing lines between blocks
  - User can create, experiment, test and configure design



eBlocks and Embedded Microprocessors

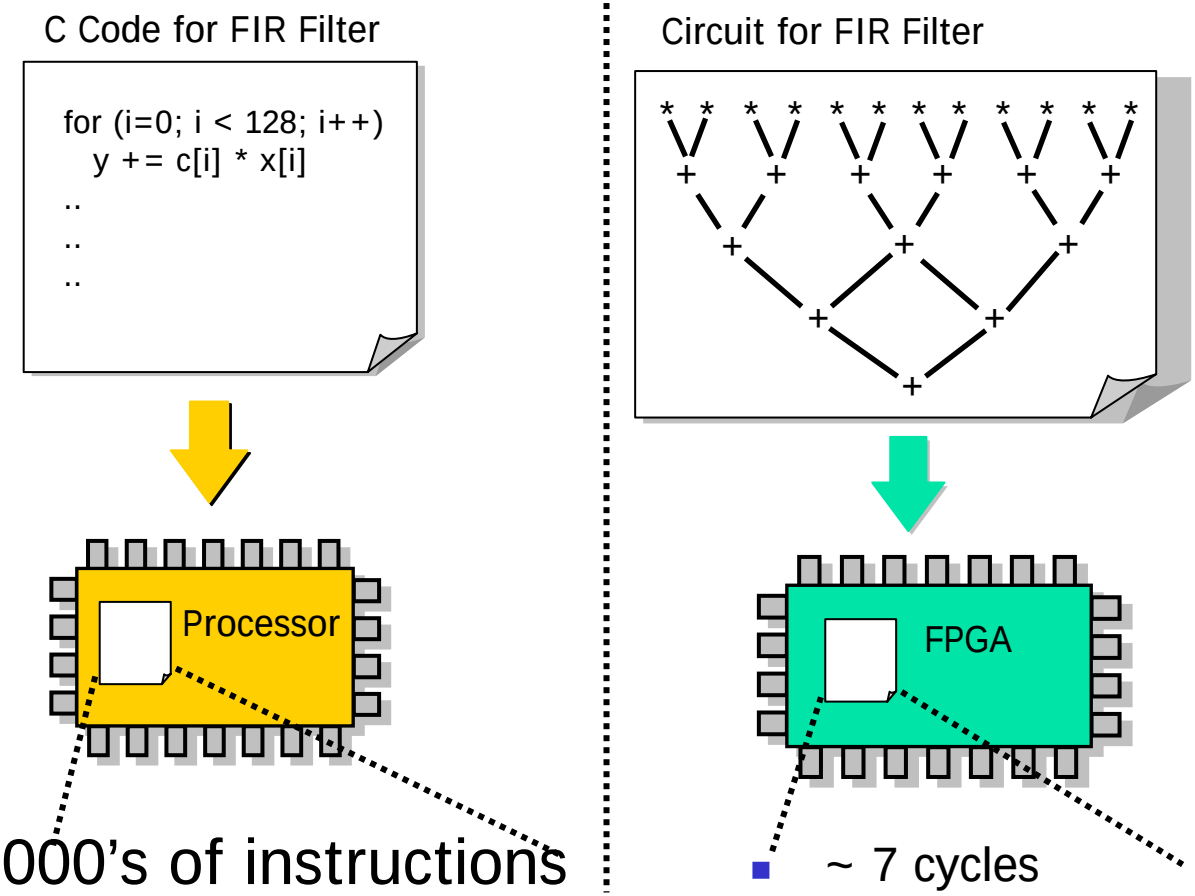
- Can greatly simplify coding



And now for something completely different…

- Warp processing
  - 2001-present, supported by SRC, Intel, IBM, Freescale, Xilinx
  - Ph.D. students
    - Roman Lysecky – Ph.D., June 2005, now Asst. Prof. at Univ. of Arizona
    - Greg Stitt – Ph.D. June 2007, now Asst. Prof. at Univ. of Florida, Gainseville
    - Ann Gordon-Ross – Ph.D. June 2007, now Asst. Prof. at Univ. of Florida, Gainseville

Circuits on FPGAs Can Sometimes Give Big Speedups



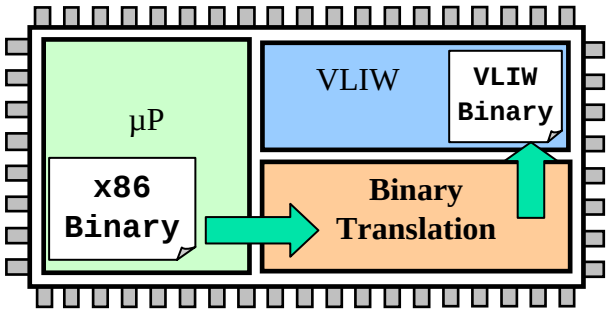
- 1000's of instructions
  - Several thousand cycles
- ~ 7 cycles
  - Speedup > 100x
  - Pipelined -- >500x

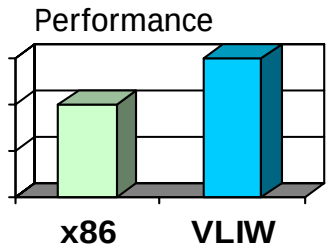
Circuit parallelism/pipelining can yield big speedups

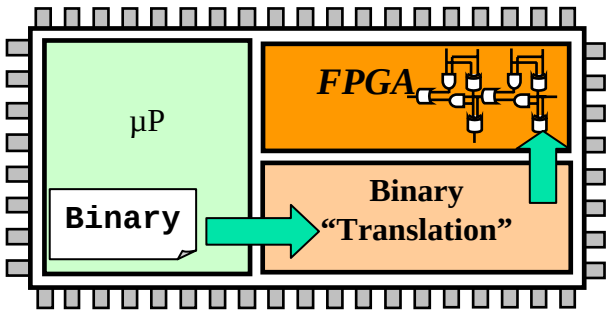
Dynamic Translation

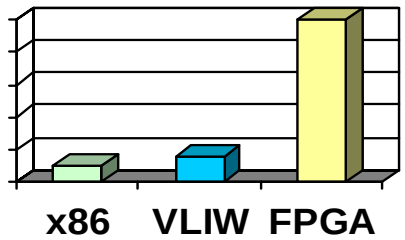
- Motivated by commercial dynamic binary translation of early 2000s

e.g., Transmeta Crusoe "code morphing"





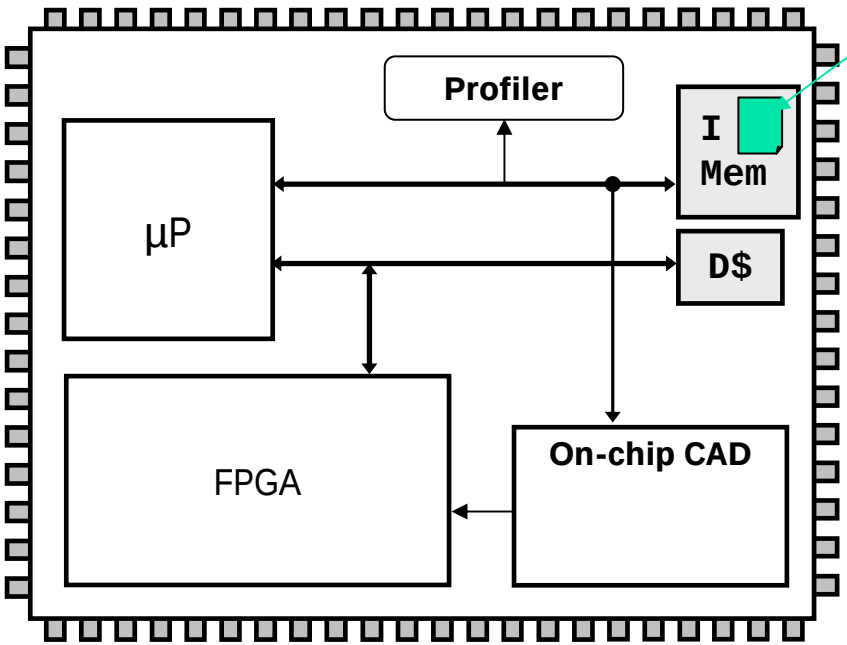




- ***Warp processing*** (Lysecky/Stitt/Vahid 2003-2007): dynamically translate binary to circuits on FPGAs

Warp Processing Background

1 *Initially, software binary loaded into instruction memory*

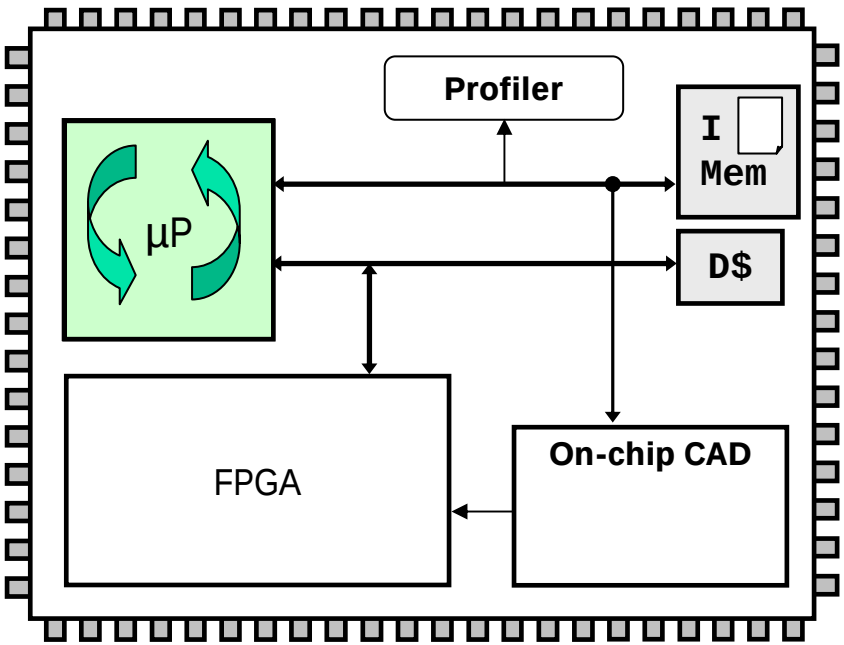


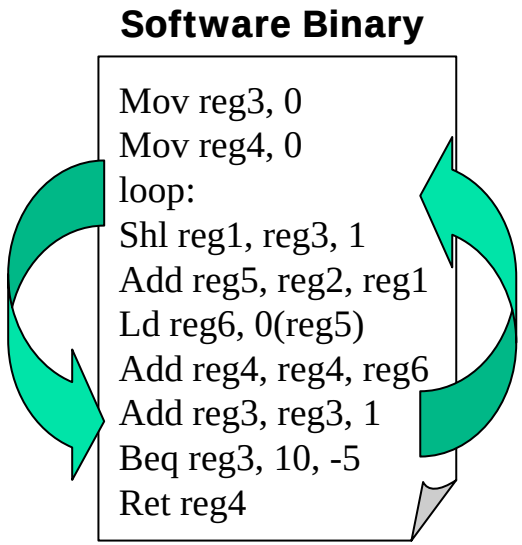
Software Binary

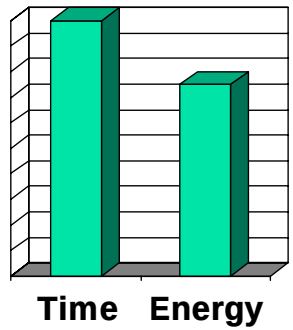
```
Mov reg3, 0
Mov reg4, 0
loop:
Shl reg1, reg3, 1
Add reg5, reg2, reg1
Ld reg6, 0(reg5)
Add reg4, reg4, reg6
Add reg3, reg3, 1
Beq reg3, 10, -5
Ret reg4
```

Warp Processing Background

2 *Microprocessor executes instructions in software binary*

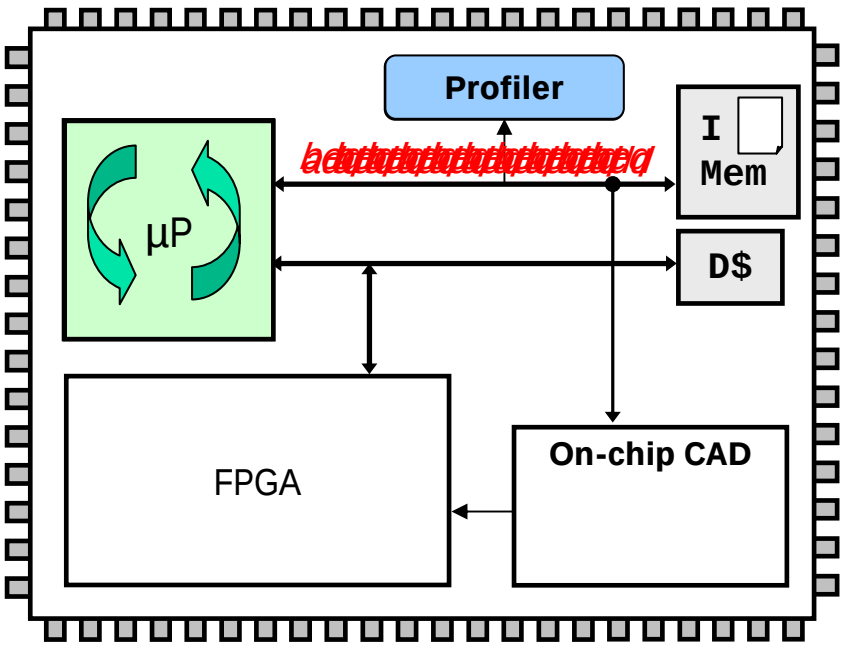


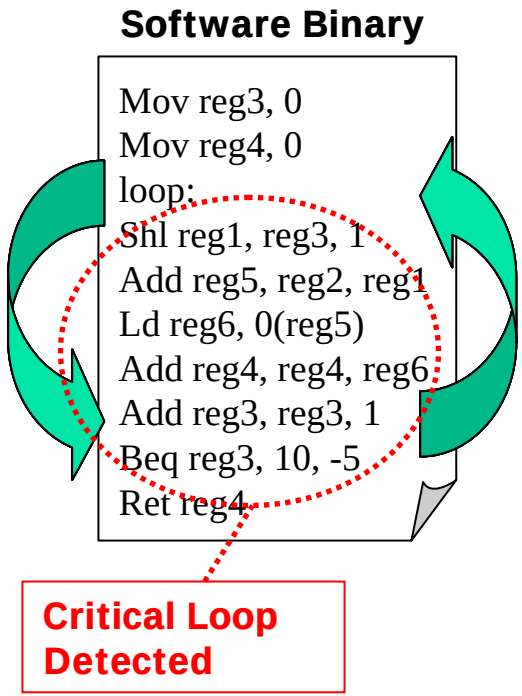


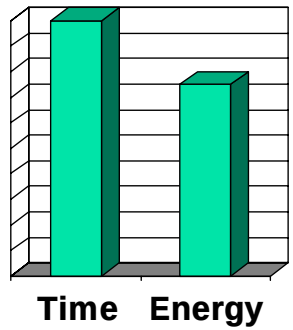


Warp Processing Background

3 *Profiler monitors instructions and detects critical regions in binary*

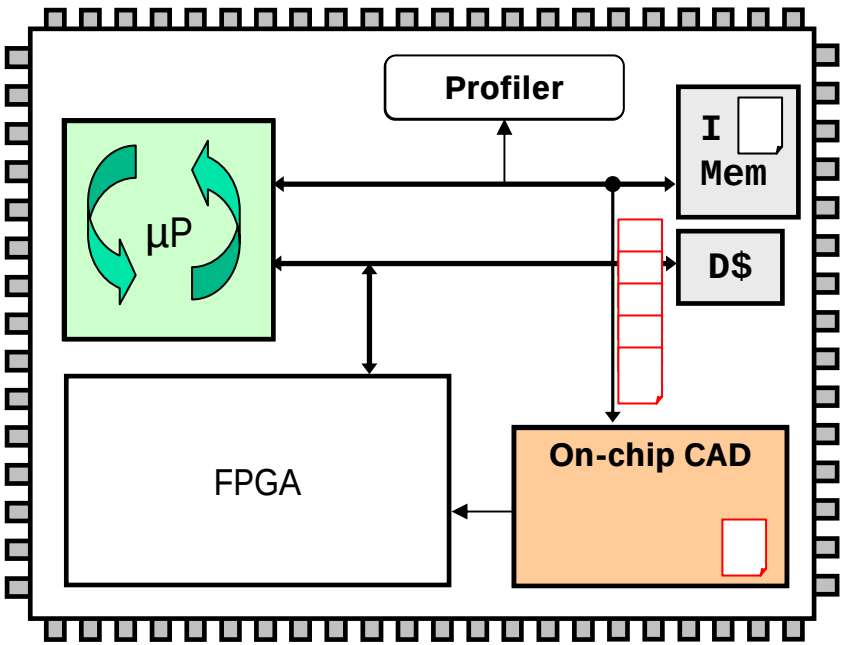


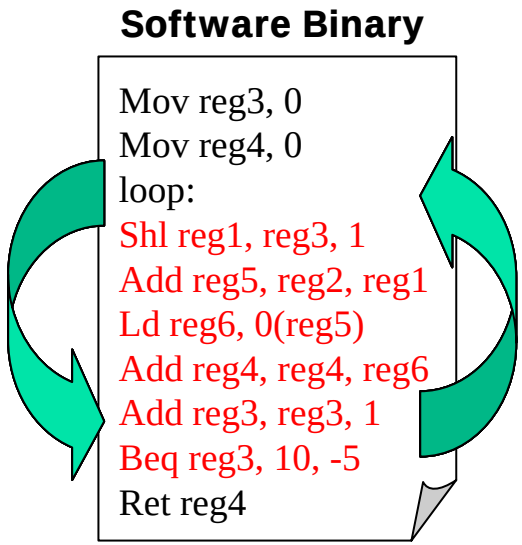


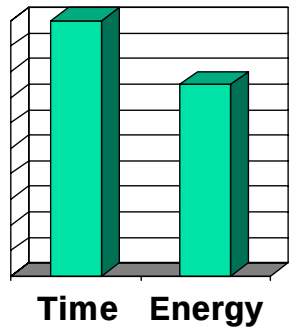


Warp Processing Background

4 *On-chip CAD reads in critical region*

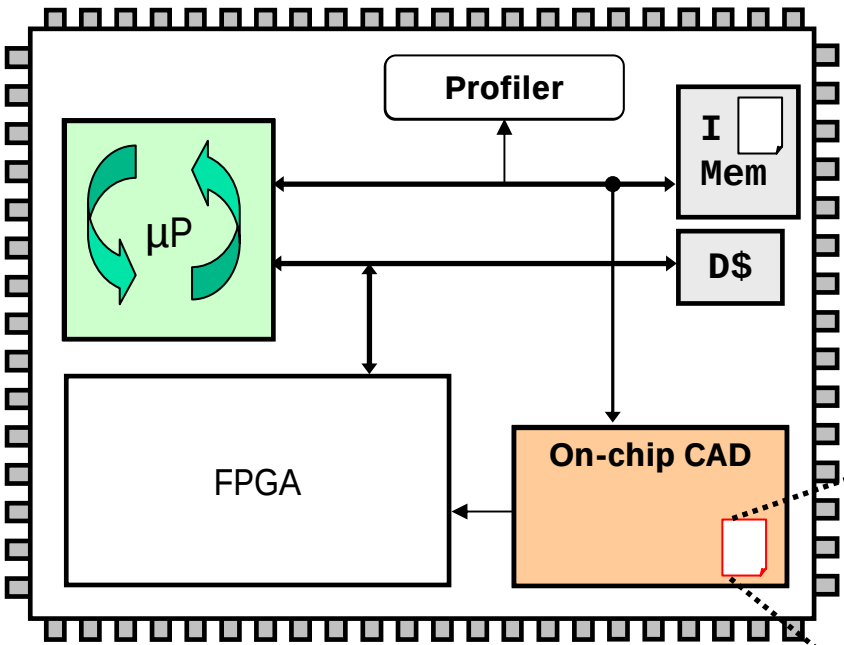






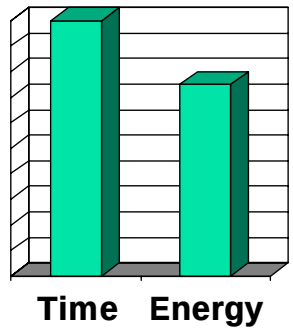
Warp Processing Background

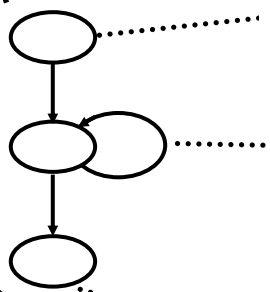
5 *On-chip CAD decompiles critical region into control data flow graph (CDFG)*



Software Binary

```
Mov reg3, 0
Mov reg4, 0
loop:
Shl reg1, reg3, 1
Add reg5, reg2, reg1
Ld reg6, 0(reg5)
Add reg4, reg4, reg6
Add reg3, reg3, 1
Beq reg3, 10, -5
Ret reg4
```





```
reg3 := 0
reg4 := 0
-----------------
loop:
reg4 := reg4 + mem[
    reg2 + (reg3 << 1)]
reg3 := reg3 + 1
if (reg3 < 10) goto loop
-----------------
ret reg4
```

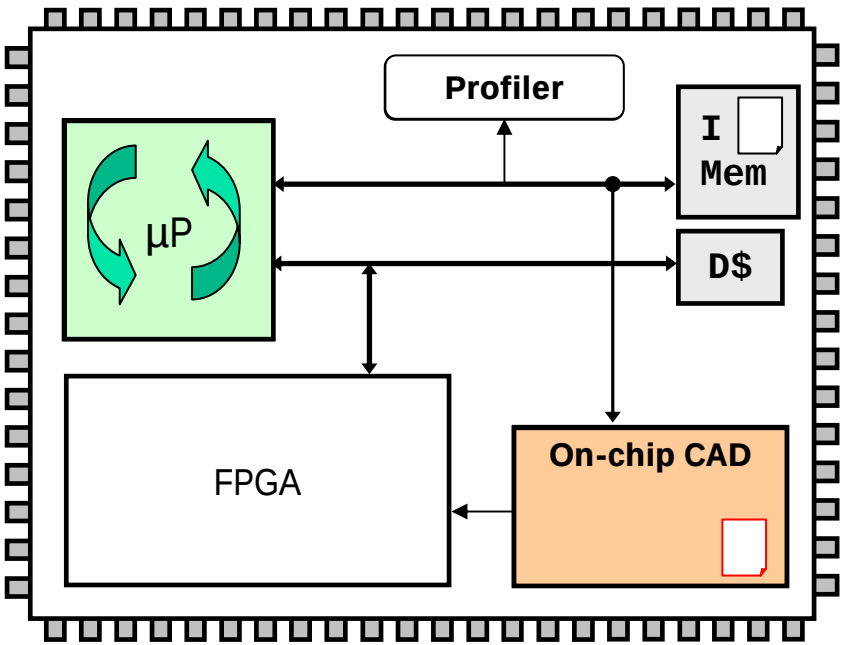
Recover loops, arrays, subroutines, etc. – needed to synthesize good circuits

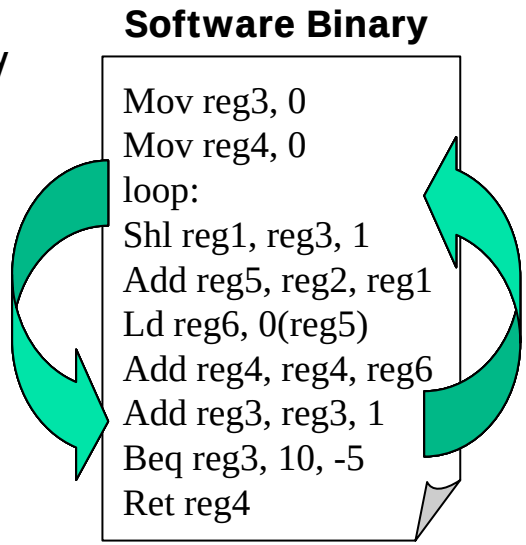
Decompilation surprisingly effective at recovering high-level program structures

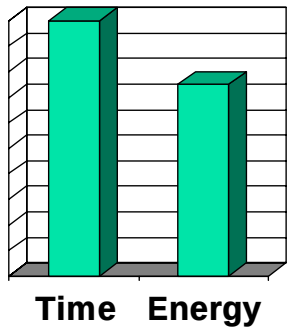
Stitt et al ICCAD'02, DAC'03, CODES/ISSS'05, ICCAD'05, FPGA'05, TODAES'06, TODAES'07

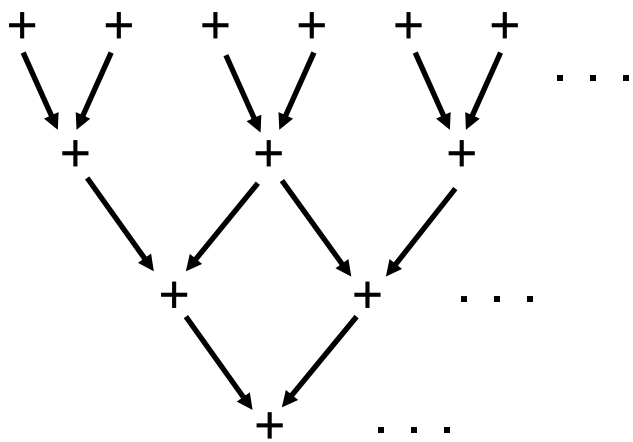
Warp Processing Background

6 *On-chip CAD **synthesizes** decompiled CDFG to a custom (parallel) circuit*



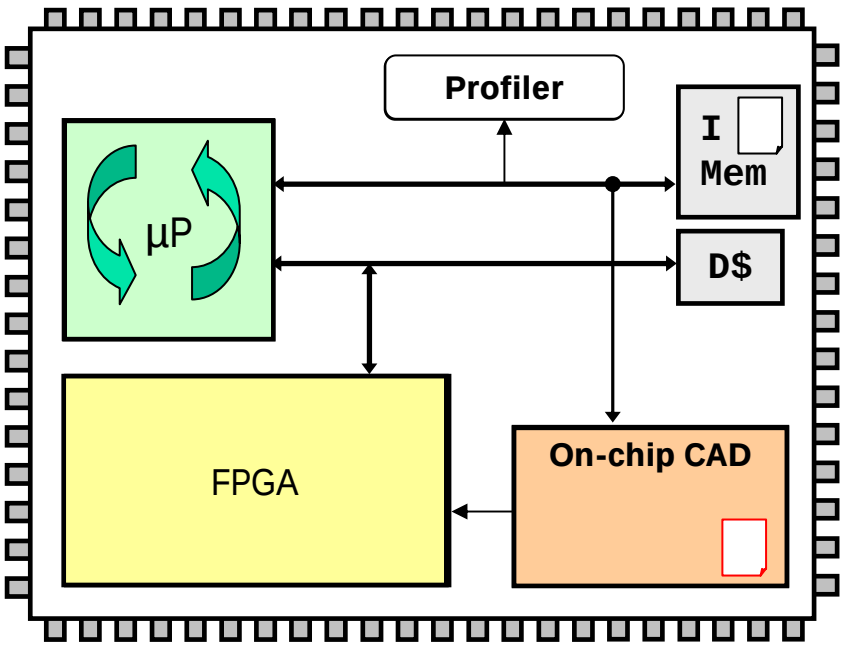


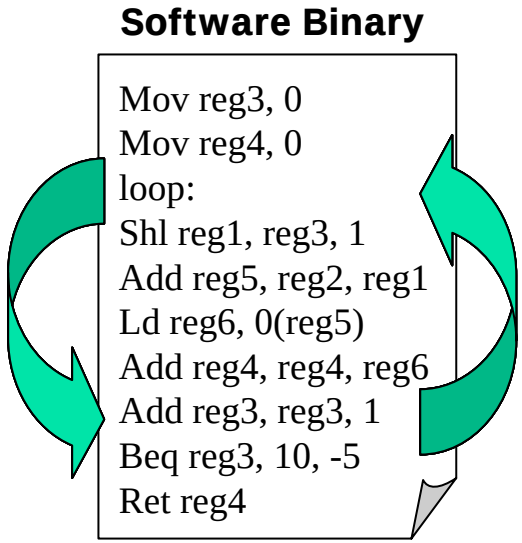


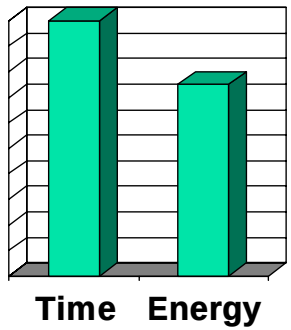


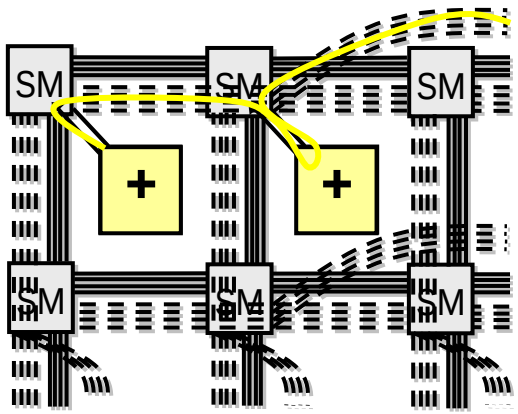
Warp Processing Background

- *On-chip CAD maps circuit onto FPGA*







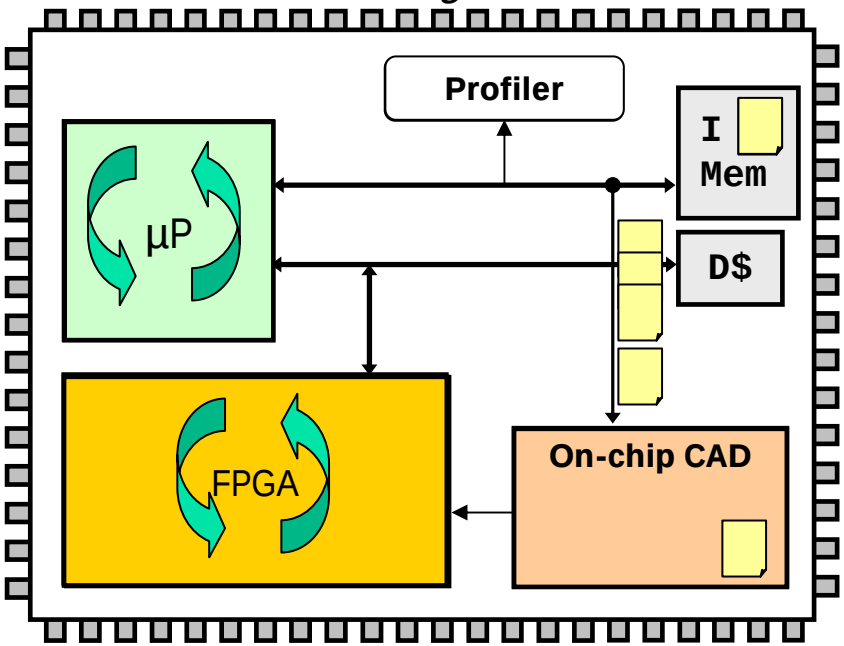


Lean place&route/FPGA →10x faster CAD
(Lysecky et al DAC'03, ISSS/CODES'03, DATE'04, DAC'04, DATE'05, FCCM'05, TODAES'06)

Multi-core chips – use 1 powerful core for CAD

Warp Processing Background

8 *On-chip CAD replaces instructions in binary to use hardware, causing performance and energy to "warp" by an order of magnitude or more*

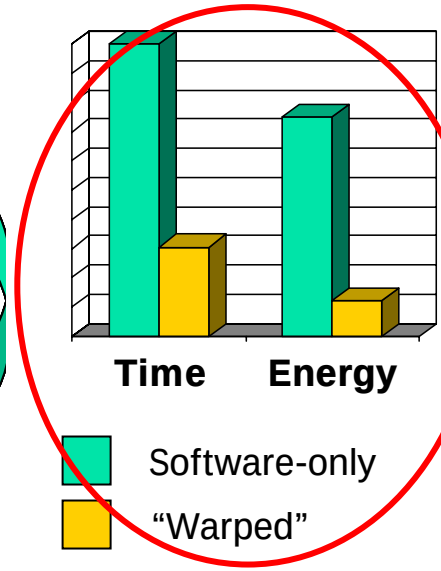


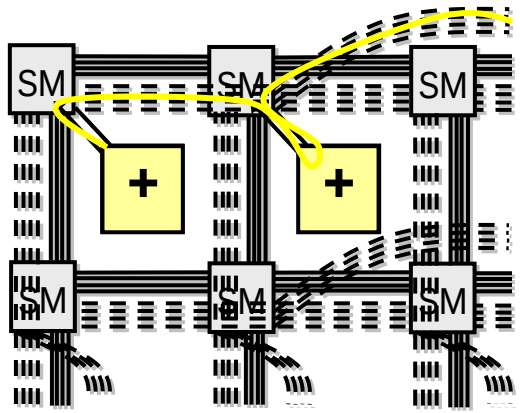
Software Binary

```
Mov reg3, 0
Mov reg4, 0
loop:
// instructions that
interact with FPGA

Ret reg4
```

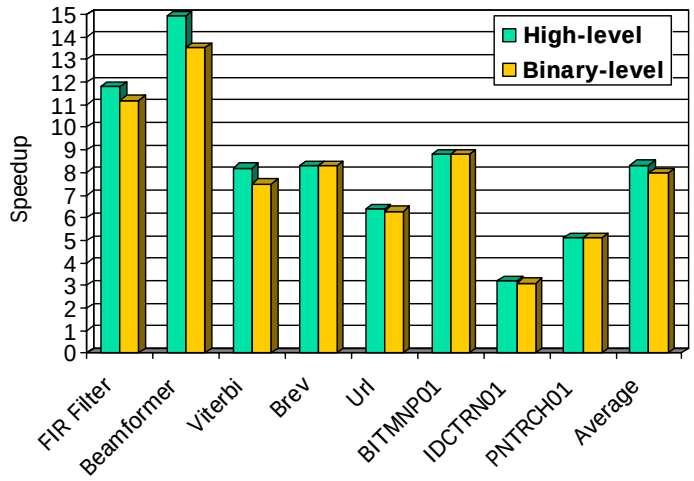
>10x speedups for some apps





Challenge: Decompilation

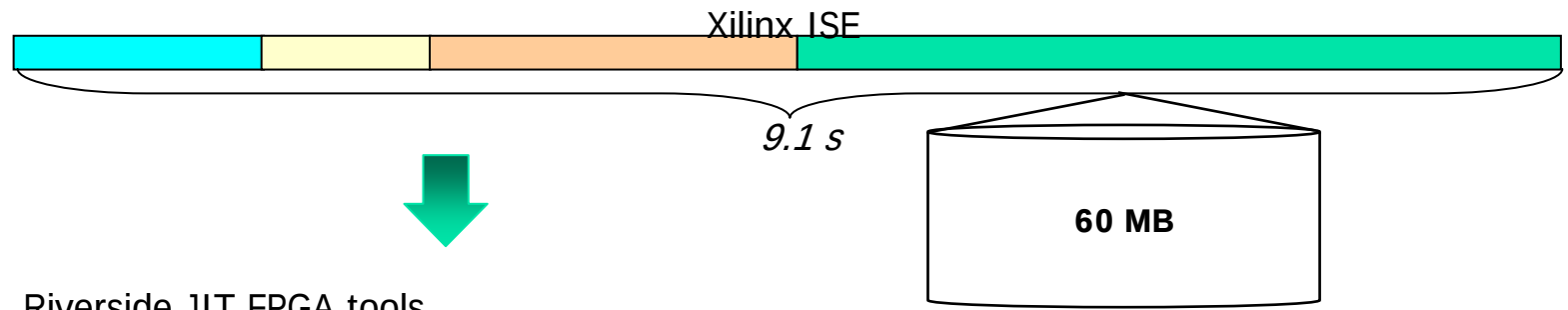
- Requires aggressive decompilation to recover loops, arrays, ..., from binaries
- Results: Competitive with synthesis from C



	Synthesis from C Code				Synthesis after Decompiling Binary					
Example	Cycles	ClkFrq	Time	Area	Cycles	ClkFrq	Time	Area	$\%Time_{Overhead}$	$\%Area_{Overhead}$
bit_correlator	258	118	2.19	15	258	118	2.186	15	0%	0%
fir	129	125	1.03	359	129	125	1.032	371	0%	3%
udiv8	281	190	1.48	398	281	190	1.479	398	0%	0%
prewitt	64516	123	525	2690	64516	123	524.5	4250	0%	58%
mf9	258	57	4.5	1048	258	57	4.503	1048	0%	0%
moravec	195072	66	2951	680	195072	70	2791	676	-6%	-1%
								Avg:	**-1%**	**10%**

Challenge: JIT Compile to FPGA

- Developed ultra-lean CAD heuristics for synthesis, placement, ***routing***, and technology mapping; simultaneously developed CAD-oriented FPGA
  - e.g., Our router (ROCR) 10x faster and 20x less memory, at cost of 30% longer critical path. Similar results for synth & placement
  - (EDAA Outstanding Dissertation Award 2006)



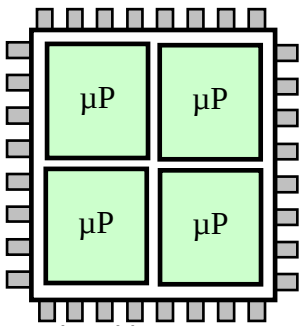
Riverside JIT FPGA tools **on a 75MHz ARM7**

Experiments

- Benchmarks: Image processing, DSP, scientific computing
  - Highly parallel examples to illustrate thread warping potential
  - We created multithreaded versions
- Base architecture – 4 ARM cores
  - Focus on recurring applications (embedded)
- TW: FPGA running at whatever frequency determined by synthesis

Multi-core

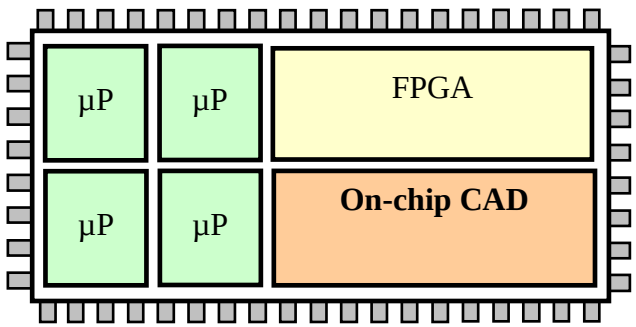
4 ARM11 400 MHz



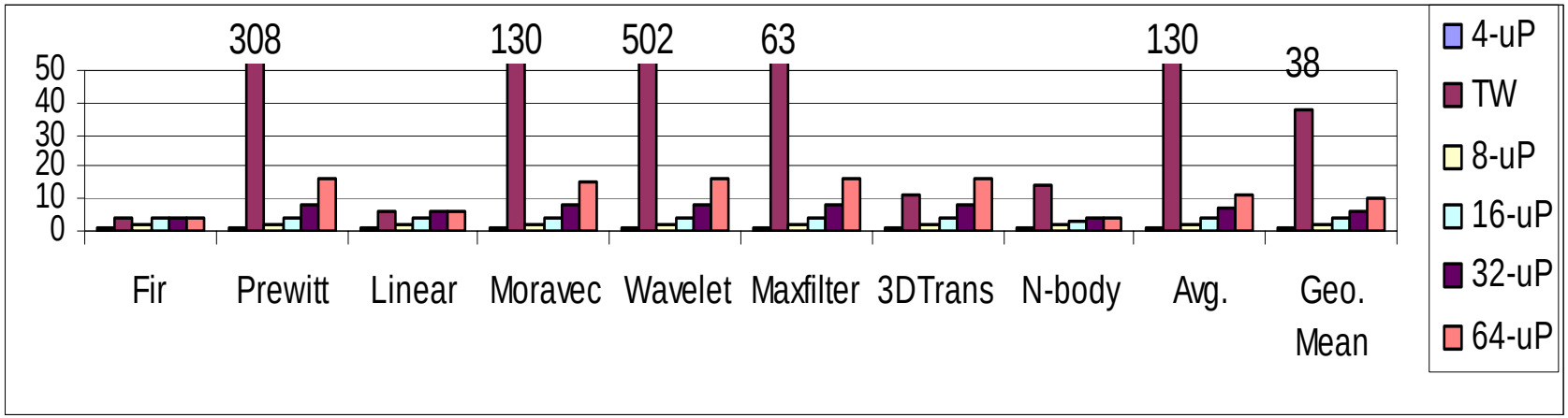
Compared to

Thread Warping

4 ARM11 400 MHz + FPGA (synth freq)



Speedup from Thread Warping



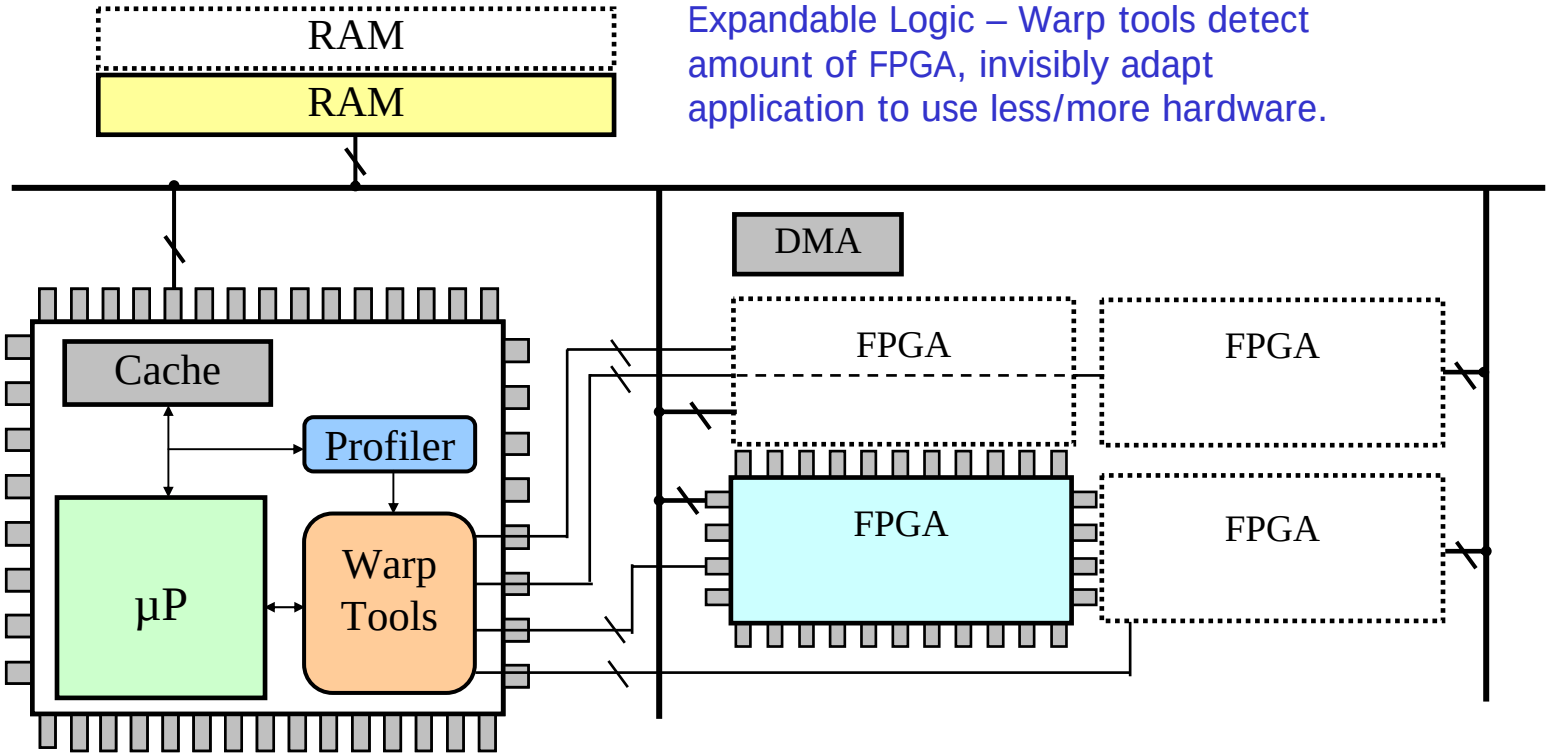
- Average 130x speedup

But, FPGA uses additional area

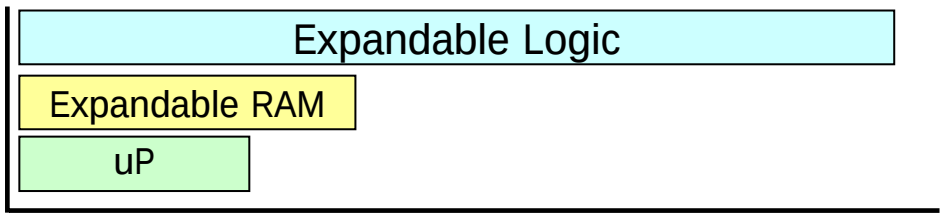
So we also compare to systems with 8 to 64 ARM11 uPs – FPGA size = ~36 ARM11s

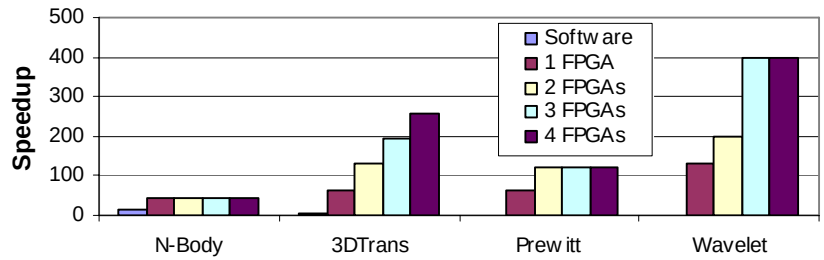
- 11x faster than 64-core system
- Simulation pessimistic, actual results likely better

Dynamic Enables *Expandable Logic* Concept



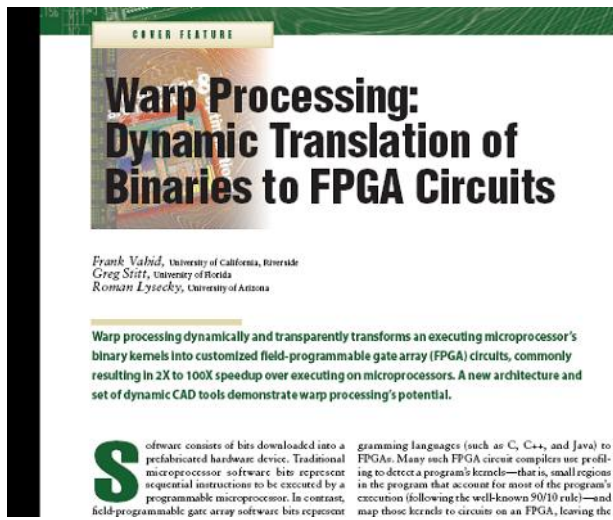
Expandable Logic – Warp tools detect amount of FPGA, invisibly adapt application to use less/more hardware.





Virtual Immersion

- eBlocks: Enables customized sensor-based system design by *non-experts*
  - May lead to …? "Wood and nails of the sensor world"
  - Currently working with hearing-impaired, aging
- Warp processing (featured in this month's IEEE Computer)
  - Enables large speedups on certain applications (e.g., image processing), user can expand hardware without changing software



COVER FEATURE

Warp Processing: Dynamic Translation of Binaries to FPGA Circuits

Frank Vahid, University of California, Riverside
Greg Stitt, University of Florida
Roman Lysecky, University of Arizona

Warp processing dynamically and transparently transforms an executing microprocessor's binary kernels into customized field-programmable gate array (FPGA) circuits, commonly resulting in 2X to 100X speedup over executing on microprocessors. A new architecture and set of dynamic CAD tools demonstrate warp processing's potential.

Software consists of bits downloaded into a prefabricated hardware device. Traditional microprocessor software bits represent sequential instructions to be executed by a programmable microprocessor. In contrast, field-programmable gate array software bits represent gramming languages (such as C, C++, and Java) to FPGAs. Many such FPGA circuit compilers use profiling to detect a program's kernels—that is, small regions in the program that account for most of the program's execution (following the well-known 90/10 rule)—and map those kernels to circuits on an FPGA, leaving the