Circuits and logic in the lab

Toward a coherent picture of computation

Elizabeth Patitsas
Kimberly Voll
Mark Crowley
Steven Wolfman

CS121 - Models of Computation

- Content
  - Discrete math
  - Logic
  - Proof techniques
  - Circuits, basic computer structure
  - Number representation
  - DFAs

Feedback and Redesign

- Perceived disconnect
  - lectures and tutorials on theory, logic, math
  - labs on circuits and hardware
- Labs full of instructions with no discovery by students
- No clear story to lab work
- Redesign began in Jan 2009, changes are ongoing

CS121 – Course Structure

- 3x 1 hour lectures
- 1 hour TA led tutorial on lecture topics
- 2 hour lab led by 1 grad TA, 1 undergrad TA

- Co-requisite with CS1 programming course
- Sept ≈ 100 students Jan ≈ 200 students

The Good and the Bad

- Good
  - Hardware – hands-on learning
  - Simulation – complex models without the mess

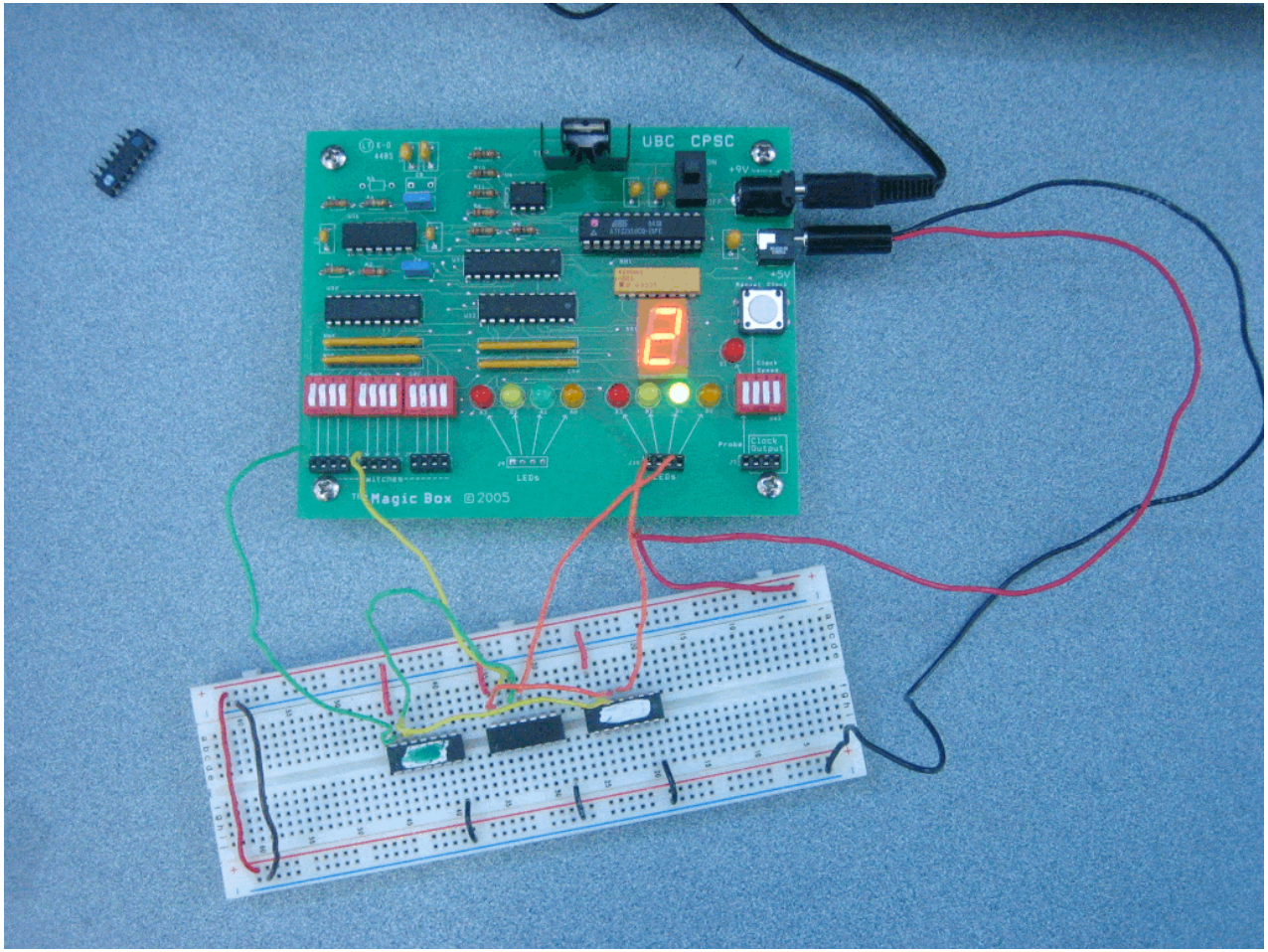
The Good and the Bad

- Good
  - Hardware – hands-on learning
  - Simulation – complex models without the mess
- Bad
  - Hardware – broken circuits
  - Simulation – Non-intuitive UI

The Good and the Bad

- Good
  - Hardware – hands-on learning
  - Simulation – complex models without the mess
- Bad
  - Hardware – broken circuits
  - Simulation – Non-intuitive UI
  - Instruction vs. discovery
  - Training of Teaching Assistants
  - Lab material disconnected from Lectures

Hardware – Magic Box



Hardware

- In half the labs, students use “The Magic Box” to implement simple circuits in hardware
- Collaboration – work in teams of 2-3
- Hands-on, discovery oriented learning
- Diagnosing bugs in circuits
- Identifying mystery chips
- TAs make connections to the computers they use everyday

Hardware - Ownership

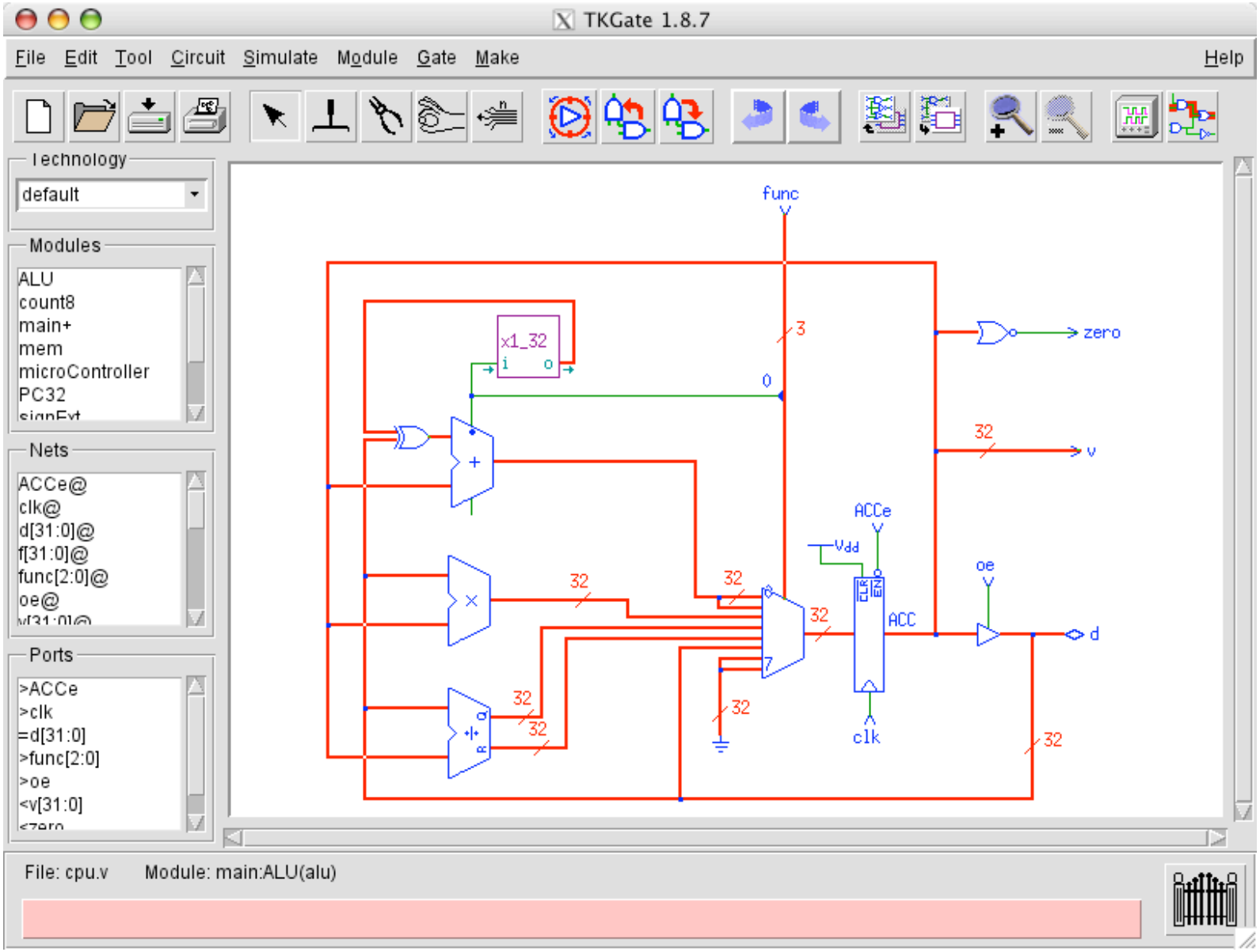
- Students used to buy their own $80 box
- Hope was that students would do extra work with box
- Reality:
  - No time
  - Not that powerful
  - Frustration at cost
  - Reliability
- Department owns all boxes now

Hardware - Reliability

- Its great…when it works
  - Reliability issues
  - Importance of clear planning to aid debugging
- If frustration is too high it hampers learning
- Improved clarity of instructions
- Well trained TAs can avoid solve these problems more quickly

Circuit Simulator - TKGate

- Good
  - Play with a more complex system
  - No hardware failures
  - Can work from home
- Not so good
  - Clunky, unintuitive interface
  - Requires complex instructions
  - Tends towards micromanaging learning


TKGate 1.8.7
File Edit Tool Circuit Simulate Module Gate Make Help
Technology
default
Modules
ALU
count8
main+
mem
microController
PC32
Nets
ACCe@
clk@
d[31:0]@
f[31:0]@
func[2:0]@
oe@
Ports
>ACCe
>clk
=d[31:0]
>func[2:0]
>oe
<v[31:0]
func
x1_32
i o
3
0
zero
32
v
ACCe
Vdd
oe
ACC
d
clk
File: cpu.v
Module: main:ALU(alu)

Circuit Simulator

- Made an effort to simplify instructions
  - Try to focus on discovery tasks rather than endless detailed instructions
- Removed finicky circuit errors
- Focus on strengths of TKGate
  - Simple circuits
  - Modules
  - A full simulated computer

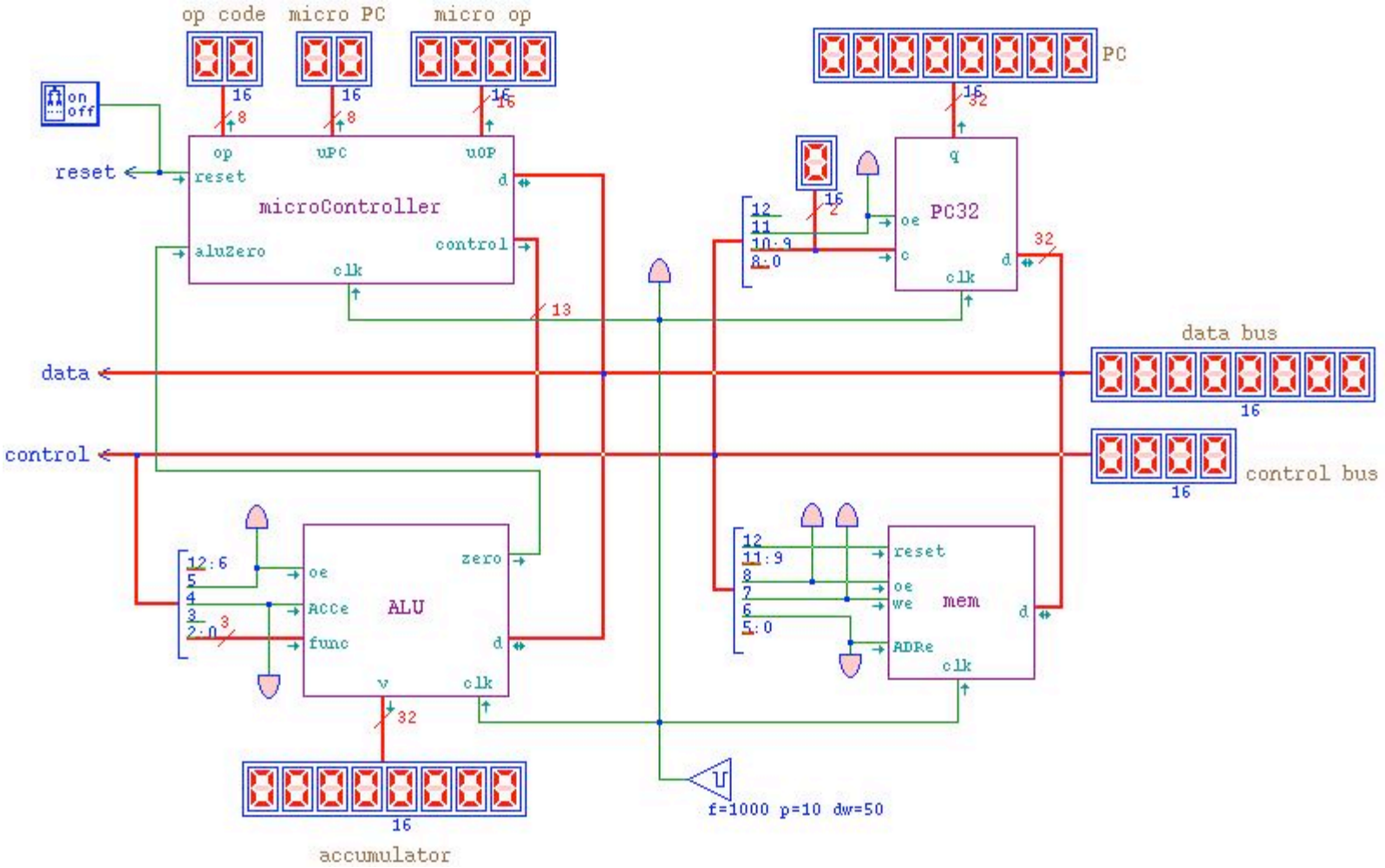
Losing Sight of the Big Picture

- Goal of Labs was lost over the years

  "Understand a computer from ones and zeros on wires all the way up to working computer built from basic gates"

- Building circuits without a single goal tying them together
- Even simple circuits seem difficult at the beginning
- Spring 2009 reintroduced a new "big picture"
  - Show them the destination first, even if they don't understand it all
  - Show a full CPU in TKGate in first week

The Big Picture



The Big Picture

- In Lab 1
  - Asked to explore, label the components
  - Given a running program (sums 1-10)
  - Guess the meaning of outputs
- Now they have a standard, a mystery they will explore throughout the term
- New lab learning goal:
  - *Trace the execution of any instruction through the computer down to individual wires and gates*

The Big Picture

- Grounds all later examples
- Week 2 seems simple by comparison
- Learn about:
  - Logic gates
  - Multiplexers
  - Counters
  - Flip-flops / memory
  - Arithmetic / ALU
- TAs always tie back to how current module fits into Big Picture in the computer
- Capstone lab

Changing the Lab Experience

- Goal:
  - Makes labs about discovery instead of following instructions
- Solution Part 1: modify content
- Solution Part 2:
  - **Train** TAs thoroughly on the lab
  - change **marking scheme**
    - reward sound reasoning and observation
  - core lab + challenge problems (9 + 2)/10

Flexible Learning

- Everyone learns the basic lesson
- Marking Scheme:
  - 9/10 - Core done thoroughly
  - 10/10 - Exceptional work or one challenge
  - 11/10 – Multiple or one extra hard challenge
- Students trade off their time vs challenge
- Result:
  - Most groups try at least one challenge problem
  - Some students do core ahead of time and spend lab on challenge problems

Continuous TA Training

- Course requires TAs to use:
  - challenging, rarely used skills
  - adapt to a dynamic lab with no 'right' answers
- So TAs need to know material very well
  - New lead lab TA position
  - Lead TA runs weekly lab TA meeting
  - Full run-through of lab *two weeks* beforehand
  - Modifications/corrections then made by lead TA

Conclusions

- Improved feedback
  - Students see more relevance of labs to course
  - TAs report better experience
  - Workload balance for instructors improved
- Powerful Combination
  - Focus on discovery
  - Unifying story
  - Continuous discussion and training

Future Work

- Collecting more data on impact of changes
- Reinforcing the connection to labs in lectures
- More emphasis on end goal of understanding the computer
- Improving the quality and clarity of the simulated computer

Thank You

Questions?

Code

```
#  Machine language   Exact meaning          Meaning in our program
0/ 20000200          # ACC <- M[200]        ; load n's value for use
1/ 11000101          # ACC <- ACC - M[101]  ; calculate n - 1
2/ 41000008          # BZ 8                 ; if n - 1 = 0, skip to the end
3/ 21000200          # M[200] <- ACC        ; n = n - 1
4/ 12000102          # ACC <- ACC * M[102]  ; calculate 2*n
5/ 10000201          # ACC <- ACC + M[201]  ; calculate numIntros + 2*n
6/ 21000201          # M[201] <- ACC        ; numIntros = numIntros + 2*n
7/ 40000000          # B 0                  ; go back to the start
8/ 20000201          # ACC <- M[201]        ; "return" numIntro's value
9/ ffffffff          # LOOP FOREVER         ; end the program

100/ 0           # constant 0   We use 100, 101, and 102 for constants
101/ 1           # constant 1   0, 1, and 2 to make them easy to remember,
102/ 2           # constant 2   but they could go anywhere.

200/ 2           # n; 2 is its initial value (but you can change it)
201/ 0           # numIntros; 0 is the initial value required for the
                 # algorithm
```

Code

```
# reset
0/ 3000 # set PC to zero
1/ 0840 # memory address register <- 0
2/ 8100 # instruction register <- M[0]; PC <- 1; uPC <- 0

#ALU instructions
# op code 10: ACC <- ACC + M[imm]
1000/ 4040 # memory address register <- imm
1001/ 0110 # ACC <- ACC + M[imm]
1002/ 0840 # memory address register <- PC
1003/ 8100 # fetch next instruction and increment PC

# op code 11: ACC <- ACC - M[imm]
1100/ 4040 0111 0840 8100
```