MAMBO: A Simple Soft-core Processor used in the COPPER Mission

Steve Massey

Electrical Engineering 2013

Saint Louis University

SAINT LOUIS UNIVERSITY

Saint Louis University
Space Systems Research Lab

Parks College of Engineering, Aviation and Technology

- 36 full-time faculty, 600 students
- AE, ME, EE, BME, Civil, Aviation, Physics
- SSRL organized in 2009

Joined AFRL's University Nanosatellite in 2009

- COPPER, Nanosat-6, 2009-2010
- Argus-High, Nanosat-7, 2011-2012

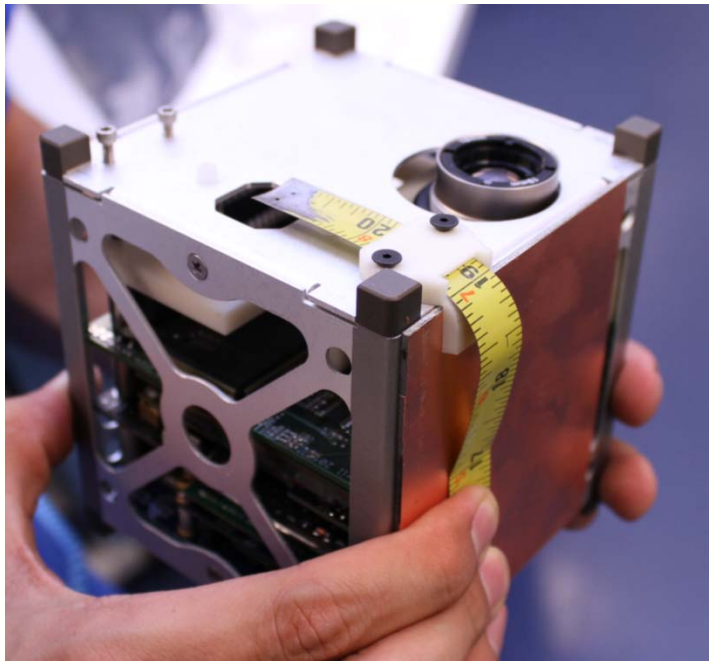
COPPER and Argus manifested through NASA CubeSat Launch Initiative

The COPPER Mission

Imaging Mission:

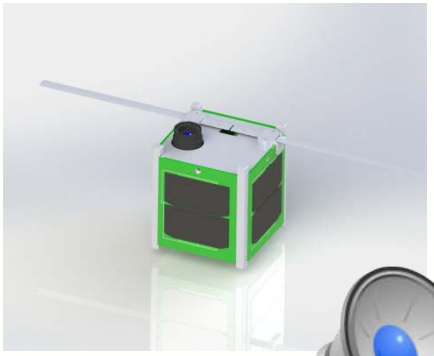
Utilize a commercially available compact uncooled microbolometer array to:

1. Capture infrared video of co-manifested satellites during separation phase
2. Capture infrared images of Earth's oceans and atmosphere



Radiation Mission:

Improve the predictive performance modeling of radiation effects on small, modern space electronics devices by collecting radiation particle collision data from electronic monitoring experiments and relaying the data to the ground



Project Duration: 2009-2013

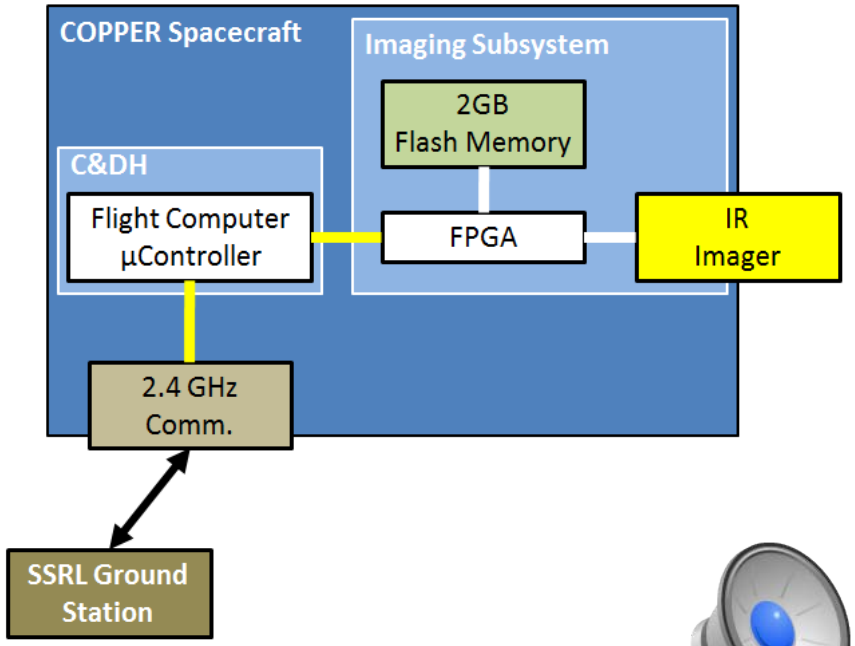
Initial concept: 2009-2010 Nanosat competition

Mission Modified to Fit the CubeSat Launch Initative

Manifested for Launch: August 2013

Solution Requirements

- Video Payload must integrate into the COPPER spacecraft
  - Communicate with flight computer; 9600 baud downlink
  - Limited power budget (generation of 2W-hr/orbit)
  - Survive Low Earth Orbit environment (~500km altitude)
- Video Payload must store required information
  - 3-5 minutes of 320x240x14 video(10FPS min, 30FPS ideal)
  - Individual stills when requested



Solutions

- Looked closely at MicroBlaze and PicoBlaze offerings from Xilinx
- Microblaze
  - Huge logic/BRAM usage requirements.
  - Offered libraries need off-chip storage (data and program RAM) for all but simplest programs
  - Bus transfers super slow, difficulty getting simulation license
- Picoblaze
  - Limited codespace, address space
- Roll-your-own state machine
  - Quickly became too-complex, difficult to manage

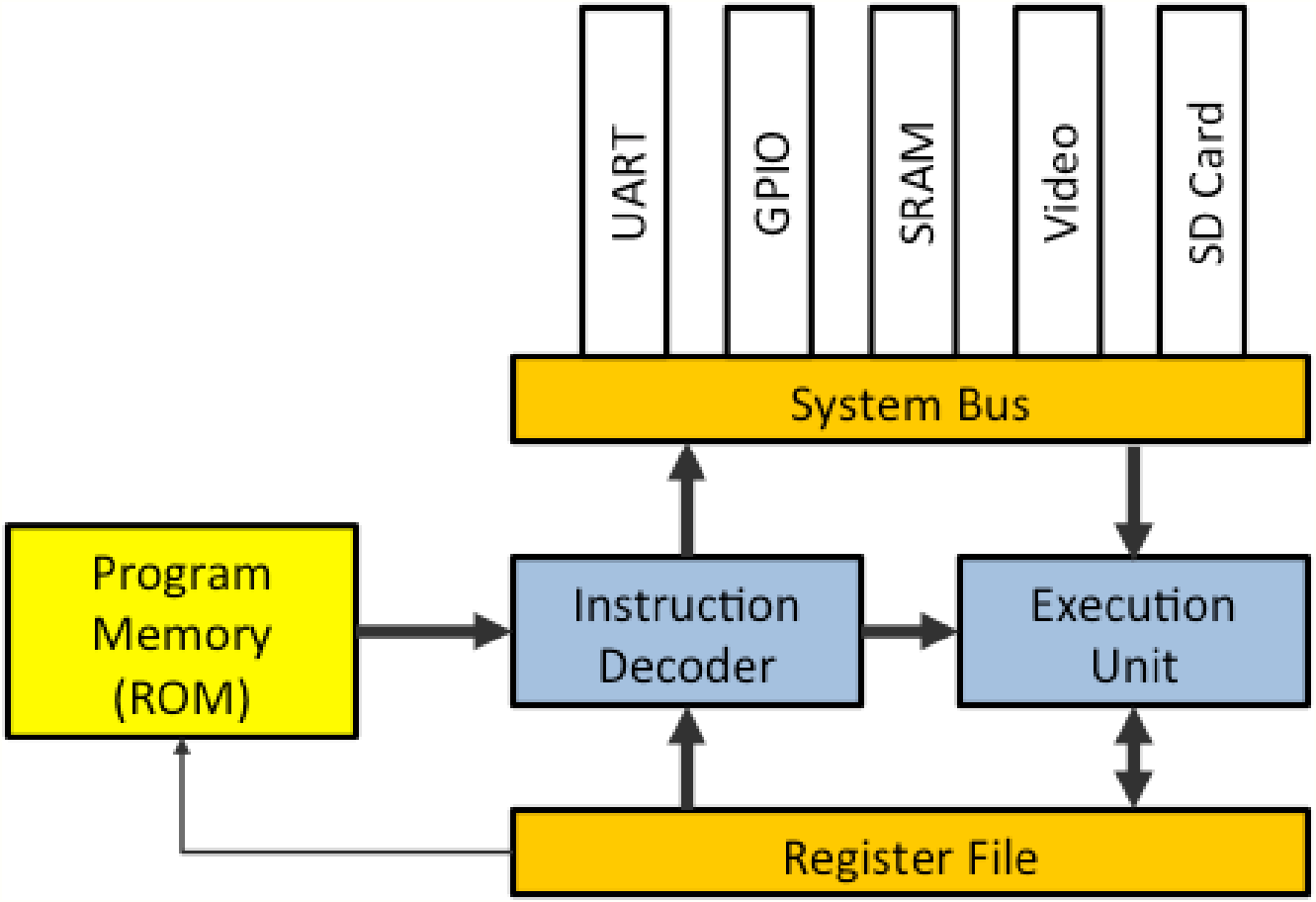
Core Requirements

- Store data at sustained 25MB/s
  - FLIR's CMOS bus is 75% downtime on a 12MHz clock
- Fit in a mid-grade Spartan3E-class part
- Easily extensible using a simple bus

Solution: MAMBO

- MAMBO: Moving And Manipulating Bytes
  - Midpoint between Microblaze and Picoblaze
- Programmable bus arbiter
  - 32-bit instructions focused on moving data across the bus, simple bitwise operations
  - 16-bit bus with 8-bit addressing (up to 32 with 24-bit extension)
- Harvard Architecture

Dataflow


UART
GPIO
SRAM
Video
SD Card
System Bus
Program Memory (ROM)
Instruction Decoder
Execution Unit
Register File

Instruction Overview

- 32-bit instruction
  - 8-bit instruction identifier
  - 24-bit instruction data
- Typical Data Usage:
  - 8-bit standard address, 16-bit immediate data
  - 24-bit extended address

Instruction Overview

- Immediate data write

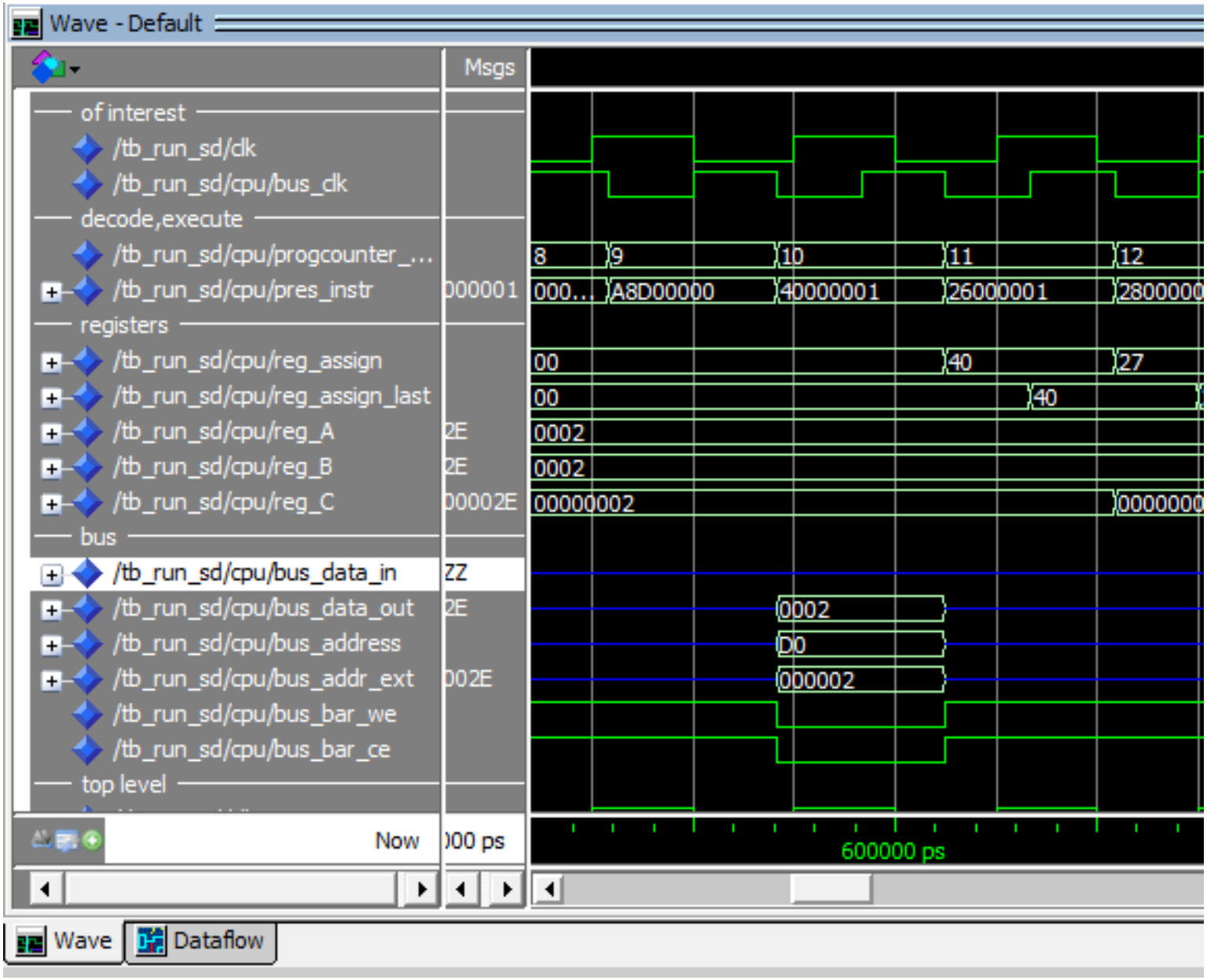
xA0		xC5		x003E	
Identifier		Standard Addr		Immediate Data	
31	24	23	16	15	0

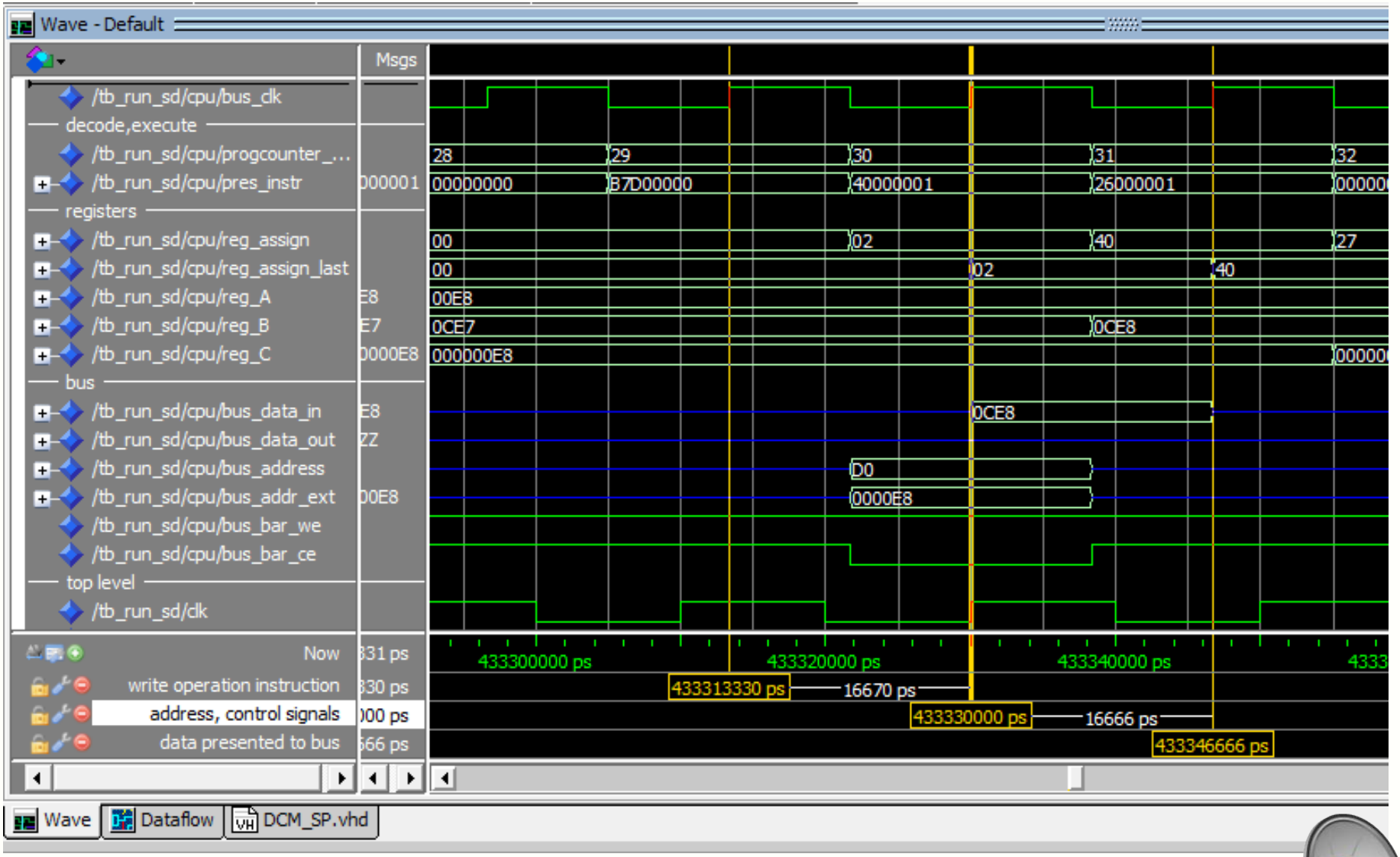
- Relative branch if rA is less than immediate value

xE3		xFE		xCD14	
Identifier		Relative PC		Immediate Data	
31	24	23	16	15	0

Bus Overview

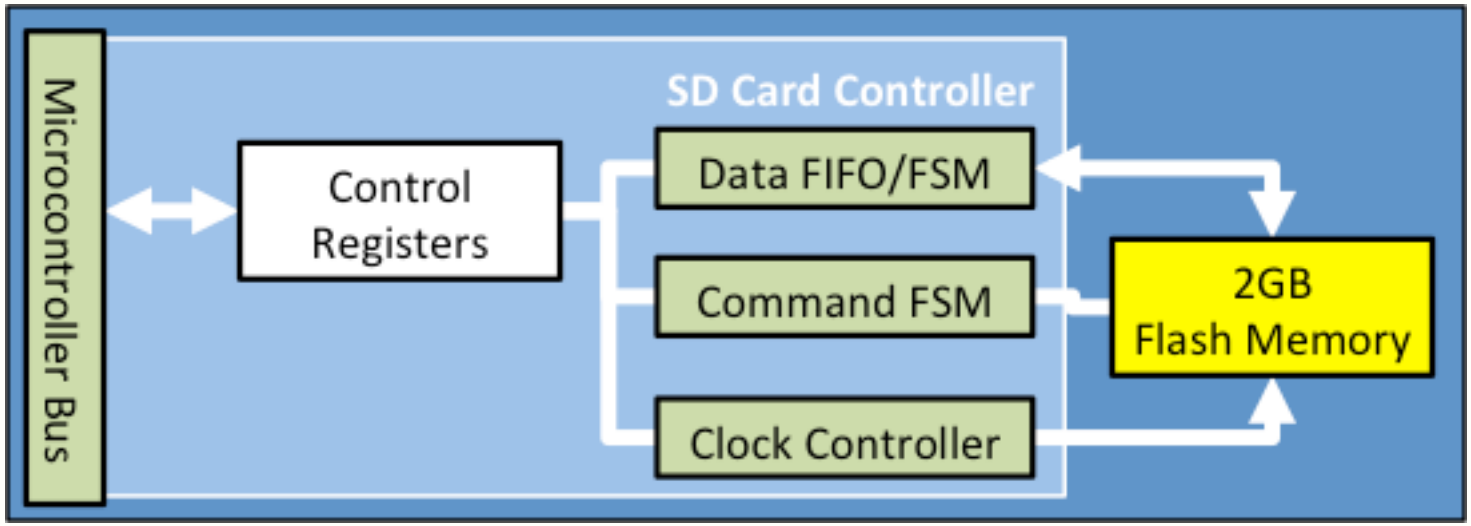
- 8-bit address line
- 32-bit optional extended address line
- Two control signals:
  - Chip Enable
  - Write Enable
- 16-bit data “to-CPU” line
- 16-bit data “from-CPU” line


Wave - Default
Msgs
of interest
/tb_run_sd/clk
/tb_run_sd/cpu/bus_clk
decode,execute
/tb_run_sd/cpu/progcounter_...
/tb_run_sd/cpu/pres_instr
registers
/tb_run_sd/cpu/reg_assign
/tb_run_sd/cpu/reg_assign_last
/tb_run_sd/cpu/reg_A
/tb_run_sd/cpu/reg_B
/tb_run_sd/cpu/reg_C
bus
/tb_run_sd/cpu/bus_data_in
/tb_run_sd/cpu/bus_data_out
/tb_run_sd/cpu/bus_address
/tb_run_sd/cpu/bus_addr_ext
/tb_run_sd/cpu/bus_bar_we
/tb_run_sd/cpu/bus_bar_ce
top level
8
9
10
11
12
000...
A8D00000
40000001
26000001
2800000
00
40
27
0002
0002
00000002
0000000
ZZ
0002
D0
000002
Now
600000 ps
Wave
Dataflow


Wave - Default
Msgs
/tb_run_sd/cpu/bus_clk
decode,execute
/tb_run_sd/cpu/progcounter_...
/tb_run_sd/cpu/pres_instr
registers
/tb_run_sd/cpu/reg_assign
/tb_run_sd/cpu/reg_assign_last
/tb_run_sd/cpu/reg_A
/tb_run_sd/cpu/reg_B
/tb_run_sd/cpu/reg_C
bus
/tb_run_sd/cpu/bus_data_in
/tb_run_sd/cpu/bus_data_out
/tb_run_sd/cpu/bus_address
/tb_run_sd/cpu/bus_addr_ext
/tb_run_sd/cpu/bus_bar_we
/tb_run_sd/cpu/bus_bar_ce
top level
/tb_run_sd/clk
28
29
30
31
32
00000000
B7D00000
40000001
26000001
02
40
27
00E8
0CE7
0CE8
000000E8
D0
0000E8
Now
write operation instruction
address, control signals
data presented to bus
433300000 ps
433320000 ps
433340000 ps
433313330 ps
16670 ps
433330000 ps
16666 ps
433346666 ps
Wave
Dataflow
DCM_SP.vhd

Current Peripherals

- SD Card interface
  - Standard capacity 4-bit 50MHz operation, 25MB/s
  - Single Read, Single Write, Burst Write



Current Peripherals

- UART
  - 9600bps, N81
- General Purpose I/O
  - 16-bit width, assignable
  - Separate Input and Output registers
- Cypress SRAM Controller
  - 4MB off-chip RAM

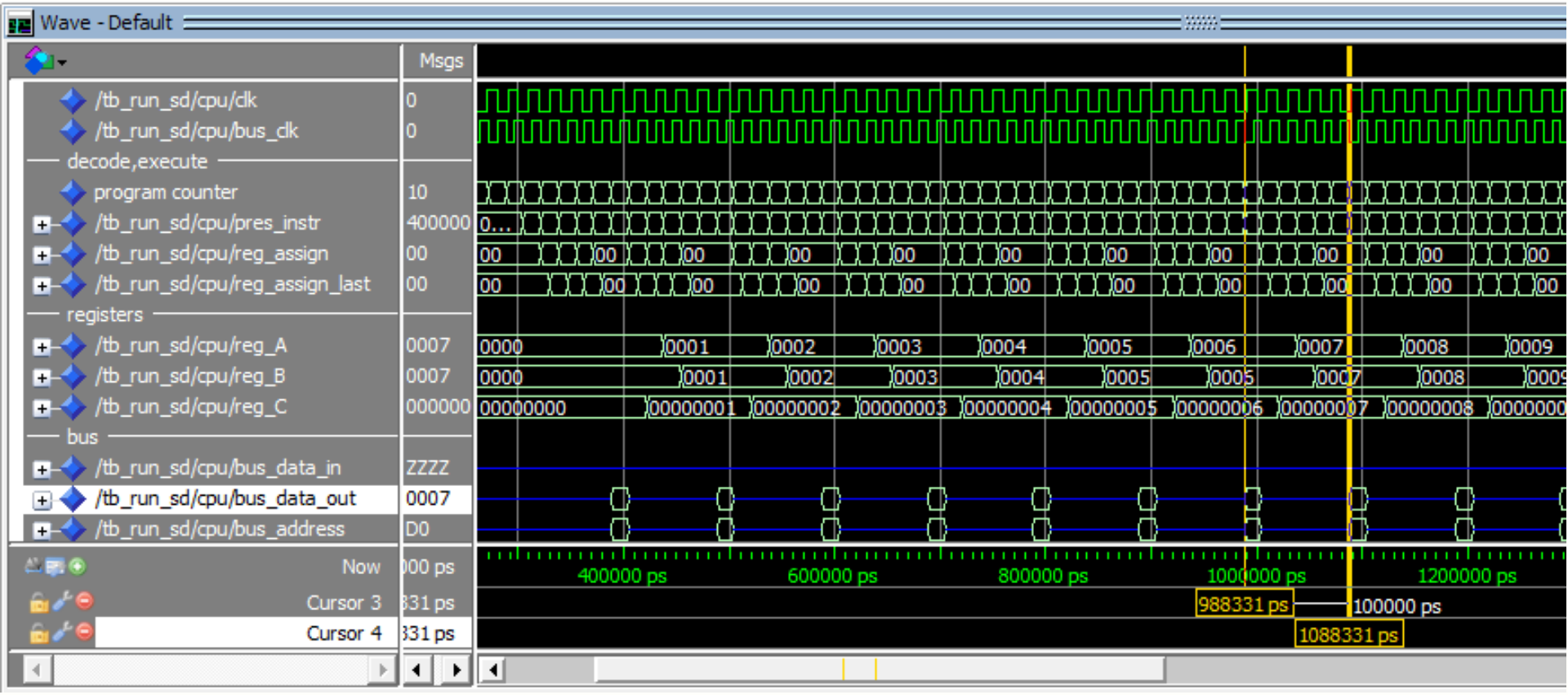
Current Peripherals

- Block RAM
  - 256 16-bit words available when using extended addressing
- Additional Registers
  - Supplement processor registers with 16x 16-bit words addressable
- FLIR Tau 320
  - 320x240x14 single-plane CMOS imager
  - Long-Wave Infrared

Example Code

```
65      -- program memory banks
66      type prog_mem is array ( 0 to num_instr ) of std_logic_vector(31 downto 0);
67
68      constant rom_instr : prog_mem := (
69              0   => x"FF000000", -- reset
70              1   => x"FF000000", -- reset
71
72              5   => x"10000000", -- rA = 0
73              6   => x"14000000", -- rC = 0
74              7   => x"11000000", -- rB = 0
75              9   => x"A8d00000", -- write rB to (xE2, rC)
76              10  => x"40000001", -- rC = rC + 1
77              11  => x"26000001", -- rA = rA + 1
78              12  => x"28000001", -- rB = rB + 1
79              13  => x"e3fb0fff", -- rA brlt xFF
80
81              17  => x"b0f40000", -- rA <= cache5
82              19  => x"b0f50000", -- rA <= cache6
83              21  => x"b0f60000", -- rA <= cache7
84              23  => x"b0f70000", -- rA <= cache8
85
86              25  => x"10000000", -- rA = 0
87              26  => x"14000000", -- rC = 0
88              27  => x"11000000", -- rB = 0
89              29  => x"b7d00000", -- read (xE2, rC) to rB
90              30  => x"40000001", -- rC = rC + 1
91              31  => x"26000001", -- rA = rA + 1
92              33  => x"e3fb01ff", -- rA brlt xFF
93
94              50 => x"e0000002", -- restart system
95              others => x"00000000");
96
```

Example Code



Debugging

- Simulation: ModelSim SE
- On-Chip: ChipScope Pro

Future Roadmap

- FORTH Interpreter
- Interrupts
- Testing, testing, testing!
  - Wrap up for COPPER
  - Explore additional uses

MAMBO: A Simple Soft-core Processor used in the COPPER Mission

Steve Massey

Electrical Engineering 2013

Saint Louis University

SAINT LOUIS UNIVERSITY