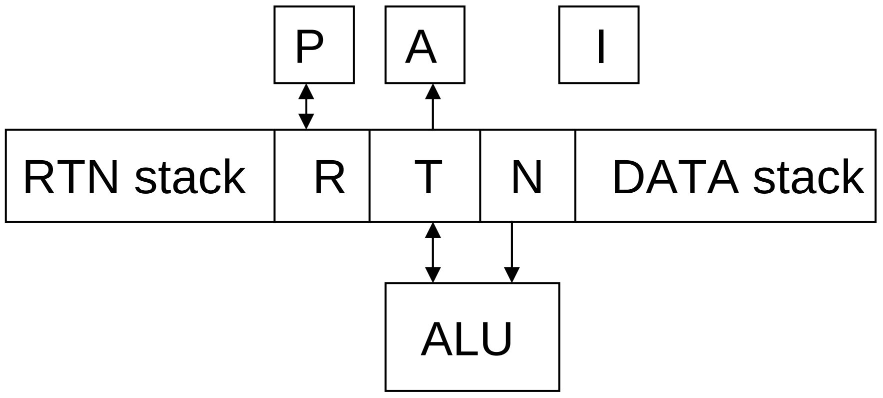
FEFFF

Forth Engine Factored For FPGA

Don Roberts
jamesdroberts@comcast.net
SVFIG – 7/28/2007

Why? … Why Not?

- Always wanted to implement my own CPU
- Had some applications needing several orders of magnitude speedup
- Profiled application inner loops, range of architectures… realized core was essentially P16



Current FPGAs

- Have large on-chip memory, roughly 64 K BYTES!
- Register/stack access time about the same as block memory
  - 8-9 ns for 16 x 16 bits of local memory
  - 10 ns for 1K x 16 bit dual-ported block memory

So Let’s Re-Factor the P16

- On-chip memory *only*
- Separate data and program memories
  - Not really separate – dual ported block memory
  - Busses rather than muxes
- KISS
  - Not pre-fetching several instructions in one word
  - Hardware blocks idle –vs– pipeline and other complications

Revised Instruction Set

- Execute all opcodes in 2 clock cycles with “simple” clocking
- Several P16 operations broken into two opcodes:
  - Proceed LD, LDP, LDA, POP with a DUP
  - Follow ST, STP, STA, PUSH with a DROP
- PASS
  - TOS $\leftarrow$ Tbuss
- OVER conveniently omitted

Problems with LIT

- P16 grabs next (program) memory word for LIT
- In my implementation, critical path is instruction fetch and decode
- Would have had to latch instruction (and other complications) slowing *all* instructions by some 30%
- So, like JMP, JZ, JNZ, literal is embedded in the instruction word and becomes
  - DUP (preserve current TOS)
  - LITL (load low-byte of literal into TOS)
  - LITH (load high-byte of literal into TOS

26 Opcodes

JMP to 10-bit address
JZ
JNC
Spare
RET
CALL
Spare
Spare

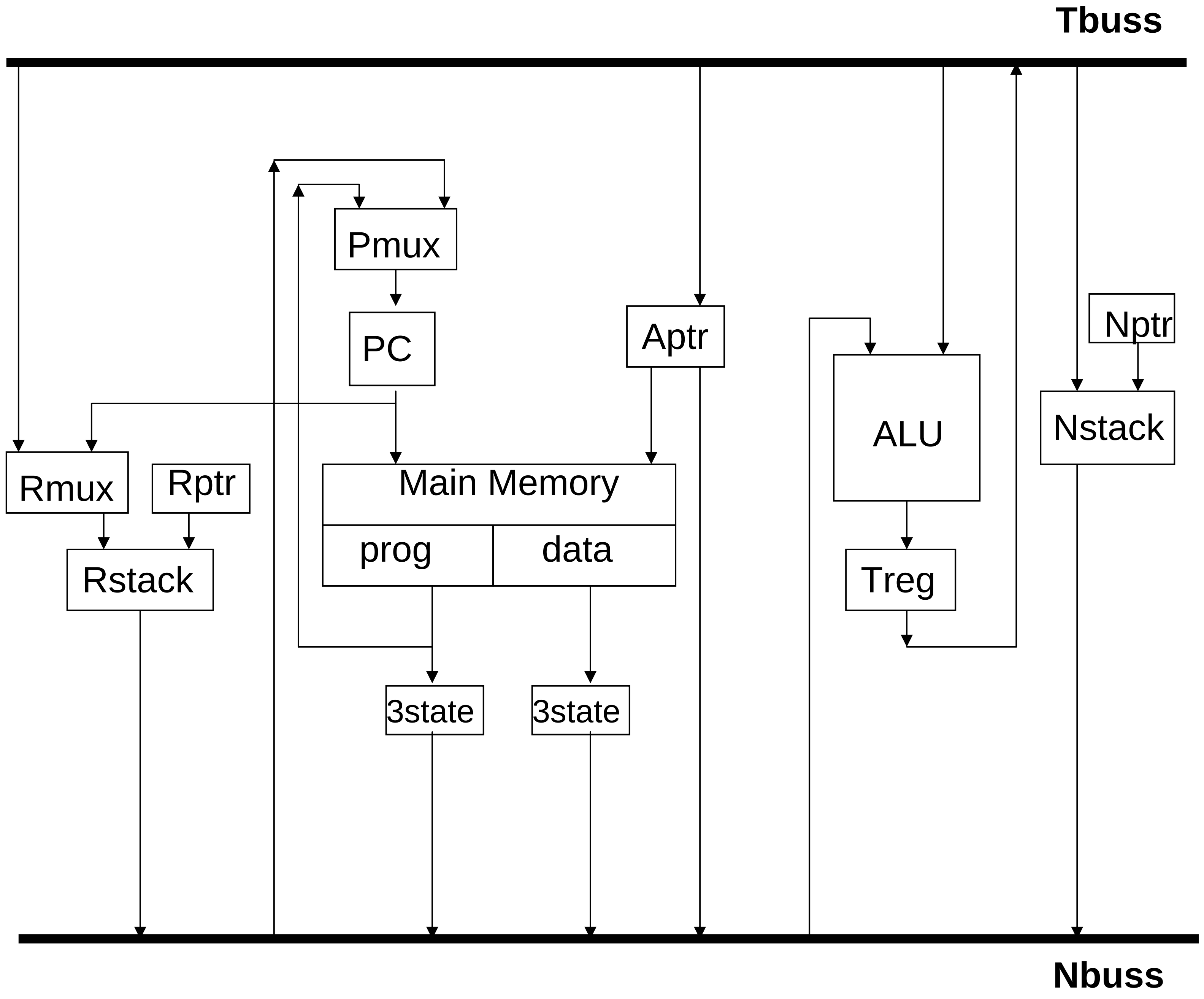
ADD
SUB
COM
XOR
AND
SHR
SHL (multiply)
PASS (no ADDC)

‘LD
‘LDP
ST’
STP’
LITL
LITH
Spare
Spare

‘LDA
STA’
‘POP
PUSH’
DROP
DUP
(OVER)
NOP

The Implementation

- Targeting Spartan XC3S1000
  - 48 KByte on-chip memory
  - Future multiprocessing
- 16-bit data and program words
- Return and data stacks 16 words each
- No TMUX
  - PASS opcode sends Tbuss through ALU unchanged


Tbuss
Pmux
PC
Aptr
Nptr
ALU
Nstack
Rmux
Rptr
Main Memory
prog
data
Rstack
Treg
3state
3state
Nbuss

Putting It Together

- 8 pages VHDL + 1 page schematic
- Key data paths
  - Nstack + Nptr = 16 nS
  - Rstack + Rptr + Rmux = 20/25 nS
  - ALU + Treg = 17/20 nS
  - Data memory + Aptr = 23 nS
  - Program memory + Pmux + PC = 25 nS
  - Program memory + Instruction decode = 25 nS

Design Comments

- Xilinx free tools surprisingly good
  - Good synthesis from behavioral description for data path
  - limited simulation time (1000 ns), but good enough
- The “big burden” is the controller block
- Ting’s “bootstrapped debug” *very* helpful
  - Test JMP, NOP, LIT 1st – you’ll need ‘em for other tests
  - Then test ADD and JZ, then DUP and JNC
  - Then CALL and RET
  - End tests with a opcode that jumps to itself – which one (address) well tell you results of conditionals
- Simulation + full-speed hardware “good enough”
  - Hardware single-step and VGA “monitor” very useful

Results & Status

- “Works” in simulation
  - 40 MHz, <= 20 MIPS
  - Ignoring main memory, CPU uses 3% of chip resources!
- Works in real-world!
  - Tested at 25 MHz
  - Single-step and VGA character generator added
    - 4% of chip logic
    - 8% of chip memory

Single-Step Demo

Loc	**Hex**	**Instruction**	
0	FC00	NOP	
1	2804	CALL 4	
2	0002	JMP 2	
3	0003	JMP 3	
4	6755	LITL 55	
5	2000	RET