Ways to reduce RISC-V soft processor footprint

万瑞罡 (Ruigang Wan) Chengdu University

E-mail: h@iloli.bid

GitHub account: rgwan

RISC-V Shanghai Day

June 30th 2018

Why RISC-V?

- The RISC-V architecture provided a simple, elegant and totally free instruction set, ABI and ecosystem.

- Some vendor provided soft processor is covered by patent, e.g. Altera's Nios processor and Xilinx's MicroBlaze or PicoBlaze processor. You can't realize it on every platform you want, legally.

- And, you can't perform modifications on these vendor provided soft processor at all.

A handy tiny FPGA soft processor needs:

- Small footprint, allows it fit in FPGA's unused logic elements.
- Reasonable performance.
- Open source, free, can be easy to extend.
- High fmax, make it doesn't becomes bottleneck.

How to optimize footprint?

- Know your target FPGA platform's architecture and technology, e.g., LUT inputs, RAM width, FPGA capacity, etc.

- Use one-hot code instead of binary code for FSM. On FPGA it can make logic faster and smaller.

- Use as less registers as possible, or pair combinational logic and register together as possible. Because one logic element often contains a LUT and a register in modern FPGA.

How to optimize footprint?

- Use short combinational logic as possible. Because long combinational logic not only add amounts of MUXes into design, but also add delay in data path, which downgrades timing.

- Share same function block in the processor. e.g. You can use ALU to generate next PC-address, load-store memory address, even interrupt entry address and several state!

How to optimize footprint?

Question: The RISC-V ISA defines a load-store scheme, which makes the processor have to implement a large register file.

Solution:

- Instead of decoder + DFFs (which costs about 1K logic elements) or distributed RAM (about 100 logic elements), use system RAM as register file.
- Benefits: The FPGA has almost all block RAM can be work as dual-port RAM. And we can easily have more than 1 bank of register file, which allows the processor handle interrupts quickly than standard implementation. It turns this register based processor just like a stack-based processor.

How to optimize footprint?

Question: If the ALU is too wide, costs too much logic elements, how to reduce it?

Solution:

- Most FPGA has fast 1-logic-element ALU, the typical width of this ALU is around 4-bit. If you're not very interested in processor performance, but in high fmax and low resource usage. You can use a shift register to shift in the operand to the ALU, and shift out the result from the ALU. It can significant reduce the processor's footprint.

Bapi-RV32I processor's specification

- “扒皮”(Bapi) is a Chinese word, which means 'mean'. The RTL is released on my GitHub.

- Designed to use less than 800LUTs (typical 573, min 400), 4 x 32K block RAMs on Anlogic Eagle4-20 FPGA. No pipeline, but FSM:
  - IF-ID-REGR-EX-MEM-REGW

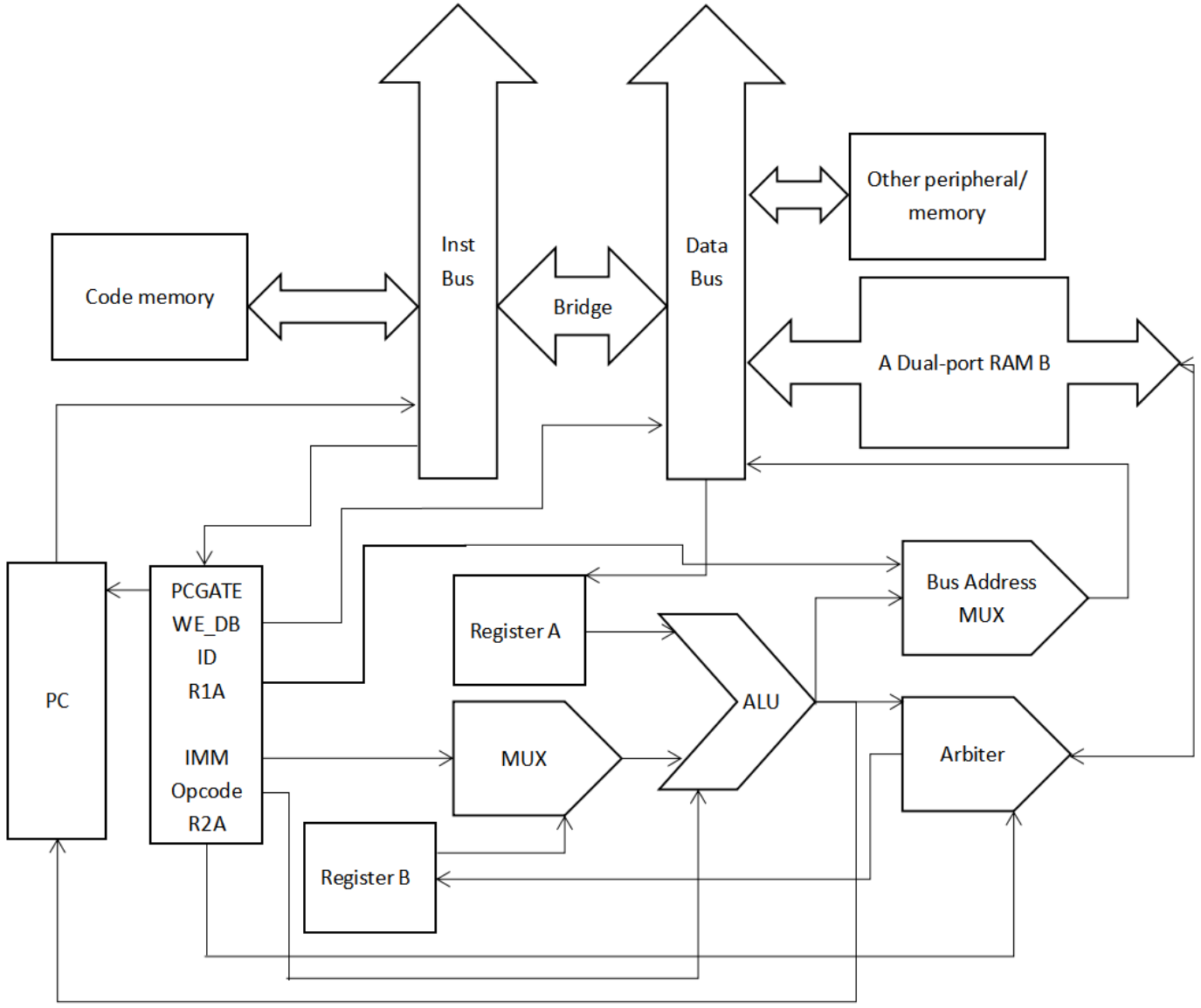
- IPC is exactly 6. Just like Clifford Wolf's PicoRV32, but Bapi-RV32I is more smaller than it. fmax is around 110MHz. It performances better than same frequency 6T-8051 soft processor.

- No RISC-V compression instruction support, maybe I will realize it in the future.

Bapi-RV32I processor's specification

- Shifter realized by serial shifter, not barrel shifter.
- Register file is realized by processor's scratchpad memory, at address 0x00 to 0x7F.
- Currently no interrupt support.
- Currently no systick and CSRs. When I realize interrupt controller, some CSRs related to interrupt will be implement.
- Byte or half-byte load-store can be enabled or disabled.

Bapi-RV32I's block digram

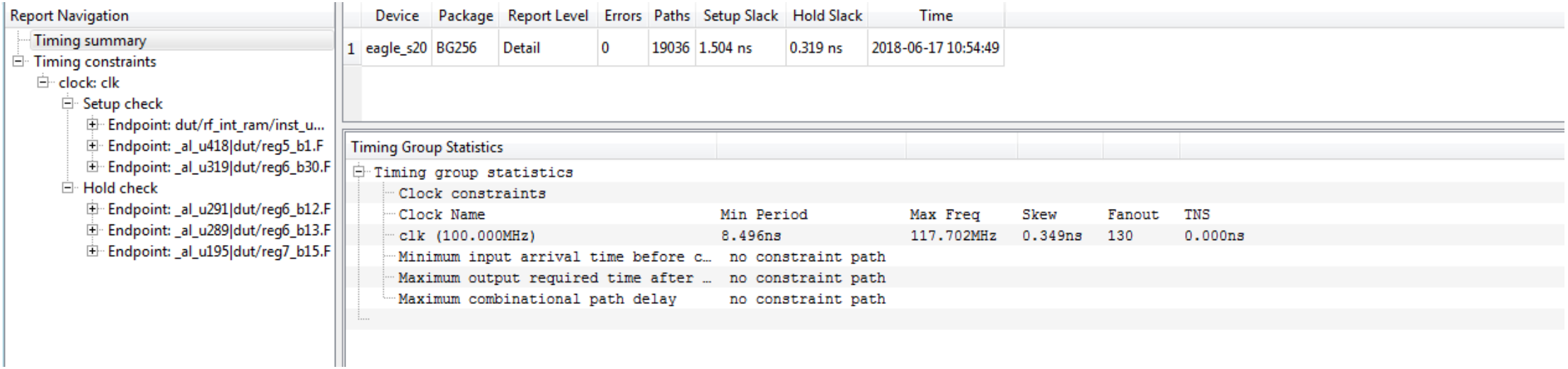


Resource usage in typical configuration

```
Utilization Statistics
#lut                   573    out of  19600     2.92%
#reg                   128    out of  19600     0.65%
#le                    573
  #lut only            445    out of    573    77.66%
  #reg only              0    out of    573     0.00%
  #lut&reg             128    out of    573    22.34%
#dsp                     0    out of     29     0.00%
#bram                    0    out of     64     0.00%
  #bram9k                0
  #fifo9k                0
#bram32k                 4    out of     16    25.00%
#pad                     5    out of    243     2.06%
  #ireg                  0
  #oreg                  3
  #treg                  0
#pll                     0    out of      4     0.00%
#bufg                    1    out of     16     6.25%
```

- This FPGA platform is provided and supported by Anlogic Technologies.

Timing report in typical configuration



- This FPGA platform is provided and supported by Anlogic Technologies.

Future work

- Implement RVC support, to improve code density.
- Implement simple Debug Interface, to help development.
- Optimize timing, help it runs faster.
- Implement Interrupt controller and multi-bank register file.
  - If this processor has multi-bank register file(if the processor use scratchpad memory as register file, it will be very convenient to realize). We can decimate stack-accessing overhead. Some function call overhead, even task-scheduling overhead.

Future work

- Design a processor behaviour description language, aims at performing auto platform specific optimization for tiny soft RISC-V processor.

- Implement coprocessor interface, help user to design their own instruction.

- Implement common tiny peripherals like QSPI Flash controller, SPI controller, LCD-framebuffer, UART interface, USB controller. It can help user quickly startup the entire system.

Q&A

Thank you!

RISC-V Shanghai Day

June 30th 2018