FPGABoy Documentation

Luke Wren

February 12, 2018

Contents

1 What?

FPGABoy is...

- An open source games console
- An open source PCB layout
- Designed with KiCAD open source PCB editor
- An open source CPU, graphics and bus architecture
- Based on the RISC-V open source instruction set
- Synthesised and taped out with iCEStorm open source FPGA toolchain

  If you say open source one more time I swear I'm gonna nut instantly - Oscar Wilde

1.1 PCB

The board is a 4-layer stackup:

1. Signal + GND Fill
2. GND plane
3. Power planes
4. Signal + GND Fill

It is intended to be suitable for low-cost PCB prototyping services such as iTead. Board dimensions are 50mm × 50mm, which fits into the cheapest size category on iTead. For the most part, it sticks to the following minimum specifications:

- Track width 0.15mm
- Copper-copper clearance 0.15mm
- Soldermask-copper clearance 0.1mm
- Soldermask width 0.1mm
- Via drill 0.3mm
- Annular ring 0.15mm (i.e. via diameter 0.6mm)

The only exception is some 0.5mm vias underneath the BGA. Strictly this is out of specification for iTead, but they claim to have a 90 μm drill registration, so we'll see how it goes.

The iCE40-HX8k FPGA is packaged in a 256-pin 0.8mm BGA, which *can be* reflowed by a hobbyist with a hot air gun or a frying pan (best to choose a HASL finish so that contacts are pretinned). The 132-pin 0.5mm BGA has sufficient IO for our needs, but iTead does not manufacture at the tolerance required for such a fine pitch.

1.2 Logic Design

The heart of the design is a Lattice iCE40-HX8k FPGA, containing 7680 LUT4s and flipflops. The logic was designed from scratch, using synthesisable Verilog. This includes:

- RV32EC-compatible 32-bit CPU design
  - E: embedded profile (reduced register set)
  - C: compressed instruction extension, for higher code density
- Graphics pipeline
  - Don't expect much, it's about as powerful as a Gameboy Advance

  - Includes some MODE7-like functionality which allows drawing perspective-mapped textured planes, by providing per-scanline affine texture transformation. Think MarioKart

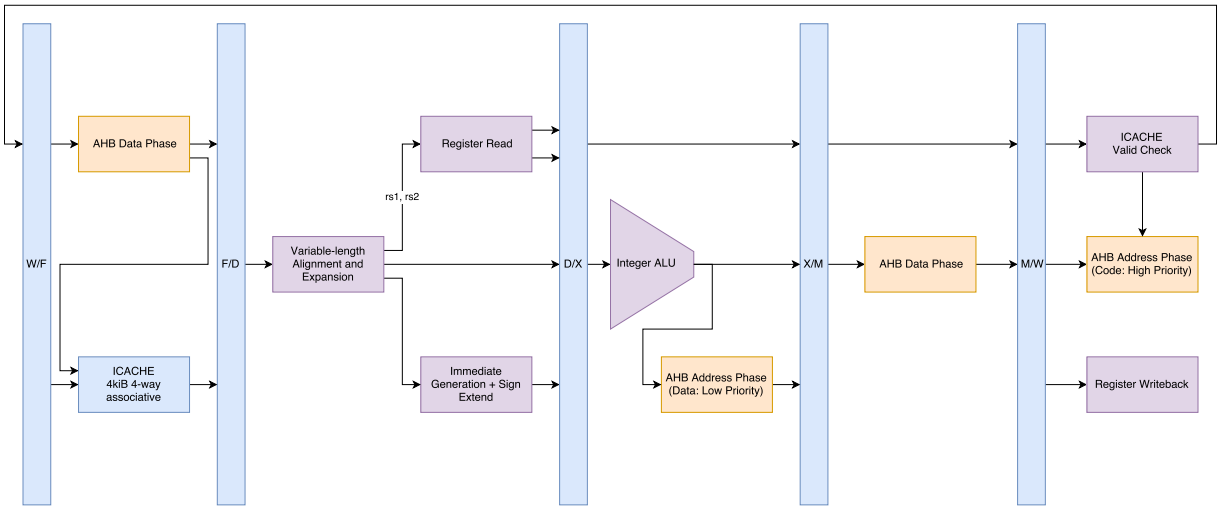
- AHB Lite 3.0-compatible multi-master busfabric
- Other peripherals
  - External SRAM controller
  - Display controller
  - DMA master
  - Interrupt controller
  - GPIO controller (buttons!)
  - Serial port etc.

Some attempt is made in this document to describe the operation of these hardware blocks, but if you are looking for nitty-gritty detail, the best documentation is the files ending with `.v`.

That a free synthesis tool can cram all this logic into one of the cheapest FPGAs on the market is tremendously impressive. Hopefully we will one day have a situation similar to software compilers, where free tools such as GCC are industry standards.

2 CPU Architecture

Figure 1: ReVive architectural block diagram



ReVive is a 32-bit processor based on the RISC V (hereafter RV) instruction set architecture. The name seemed appropriate for what is supposed to be a return to the glory days of games programming, when NULL pointers were just pointers to the start of ROM, and programmers were real programmers who wrote their *own* polygon rasterisers, dammit.

The diagram shown here is a simplified block diagram of the core. Not shown is:

- The full and specific contents of the pipeline registers
- The forwarding/bypass network which shortens data hazards to improve pipeline throughput
- Additional hardware in ICACHE which helps to support unaligned code fetches
- Interrupt entry/exit hardware
- The ready/valid handshakes on pipe stages which allow them to NOP later stages, and stall prior ones
- Hardware registers such as PC, and PC update logic

- Pipeline flushing signals for branch mispredict

Overall this is a fairly standard RISC-style processor pipeline. The tall blue boxes represent the registers at the boundaries of two pipe stages. The nomenclature used in the diagram is:

- F: fetch
- D: decode
- X: execute
- M: memory access (load/store)
- W: register writeback

The processor has a single AHB-lite busmaster; the instruction fetcher and the load/store stage need to share it, and the instruction fetch takes priority.

Branches are speculated, according to the static prediction scheme described in the RV ISA manual:

- Backward branches are predicted to be taken, on the assumption that loops will run at least twice on average.
- Forward branches are predicted not taken; the ISA manual advises that compilers should put more likely code on this path.

RISC-V compressed instructions achieve respectable code density, but it's no Thumb-2. The instruction cache helps to tame the fetch bandwidth, so that load/stores and other system masters will still see some appreciable fraction of the bus bandwidth.