Maia SDR

An open-source FPGA-based project for AD936x+Zynq radios

Dr. Daniel Estévez (EA4GPZ / M0HXM)

4 February 2024
FOSDEM, Brussels

What is new since the summer 2023?

- Support for new devices:
  - Pluto+ (similar to the ADI Pluto, but with 1GbE and SD card)
  - Microphase AntSDR E310 and E200 (Zynq 7020, 1GbE, SD card)
- Spectrum peak detect mode (as an alternative to average mode): clever way of using a single DSP48E1 either as a power integrator or as a power comparator.
- Bugfixes, dependency updates, base Pluto firmware updates...

Technical key points of Maia SDR

In this talk hopefully we'll cover:

- FFT
- DMA
- WebGL2 waterfall

FFT

- Flexible pipelined FFT implementation in Amaranth: think of it as Verilog generator in Python if you want.
- Focus in good performance with low resource usage.
- Single-delay-feedback decimation-in-frequency architecture.
- Radix-2, radix-4 and radix-2^2 options. Radix-2^2 gives best results and is used by default.
- Twiddle factor multiplication: can use either 3 DSPs for the complex product, or a single DSP running with a 3x clock.
- Configurable bit widths and truncation schedule.
- Optional window multiplication.
- Bit exact model in Python, compared against Numpy's FFT.
- Use in Maia SDR: 4096 points, Blackman-harris window, 12-bit input, 18-bit output, 62.5 MHz clock. 2.2 kLUT, 1.4 kFF, 9.5 BRAM, 6 DSP.

S. He, M. Torkelson, "A new approach to pipeline FFT processor," *Proceedings of International Conference on Parallel Processing*, 1996, 766-770.

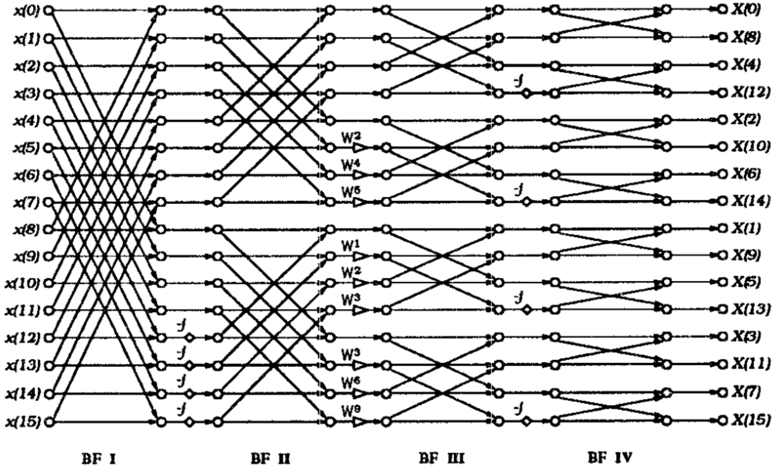


Figure 3: Radix-2^2 DIF FFT flow graph for $N = 16$

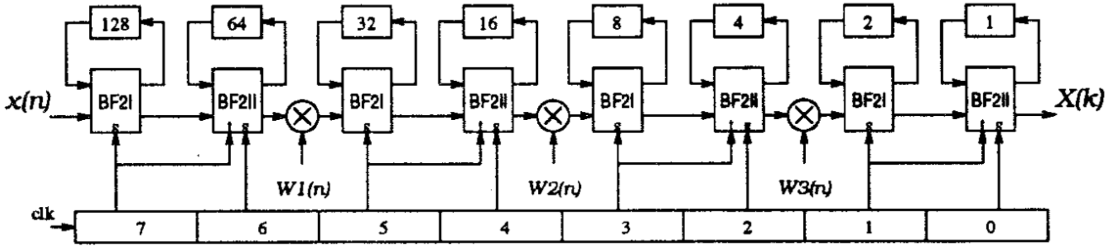
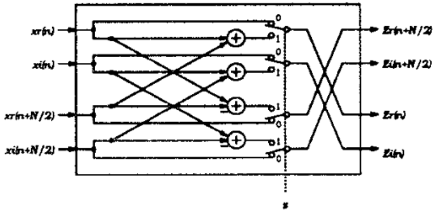
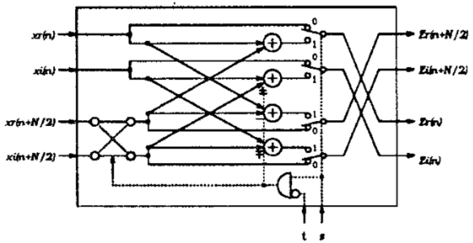


Figure 4: R2^2SDF pipeline FFT architecture for $N = 256$



(i). BF2I



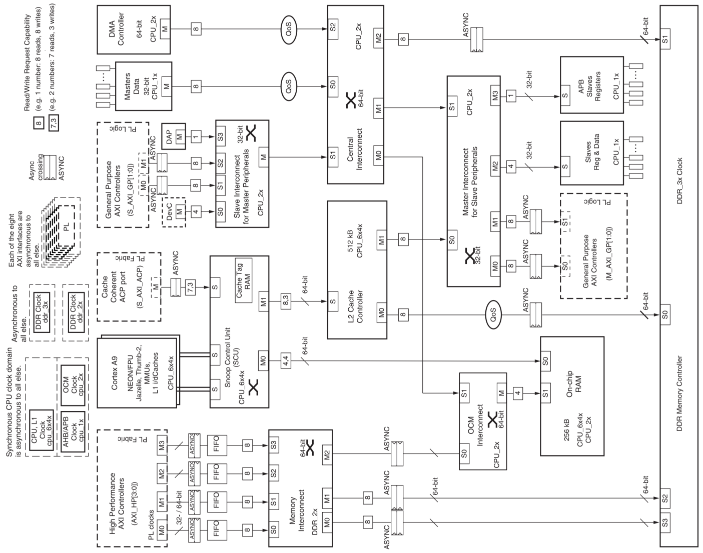
(ii). BF2II

Figure 5: Butterfly structure for R2^2SDF FFT processor

DMA: FPGA side

- Two use cases:
  - Waterfall. Write a waterfall line into a ring buffer as soon as it is ready.
  - IQ recorder. Write samples continuously to a very large buffer until the end of buffer or “stop” is pressed.
- Two flavours:
  - `DmaBRAMWrite`. Copies the contents of a BRAM into a buffer in a ring.
  - `DmaStreamWrite`. Reads from an AXI4-Stream-like interface and writes to a buffer.
- Implement an AXI3 Manager interface. Xilinx favours AXI4, but the FPGA ports into the SoC are AXI3, so we save us a protocol converter.
- Most stupid implementation ever: Addresses hardcoded at synthesis time. Almost no need for software control.
- Use one SoC AXI port per DMA. No need for an interconnect in the FPGA.
- Very low resource usage. Waterfall DMA: 26 LUTs, 46 flip-flops.

To be coherent or not to be coherent?



The Good, the Bad

- ACP port. **Good:** Fully coherent. **Bad:** only one port, will evict cache lines that are actively used by the software (round-robin/random eviction policy).
- HP ports. **Good:** 4 ports, doesn't disturb the L2 cache. **Bad:** the software needs to manage coherency.

Maia SDR uses the HP ports, so it must manage coherency.

Coherency management methods:

- Mark memory as uncacheable. **Good:** simple solution. **Bad:** painfully slow accesses.
- Mark memory as cacheable. Manage cache invalidation manually. **Good:** fast accesses. **Bad:** complex.

Maia SDR uses manual cache invalidation.

and The Ugly

Managing caches manually from Linux requires a few **ugly** things.

- The L1 and L2 caches are physically indexed and tagged. L1 invalidation must be done by virtual address. The pages to invalidate must be mapped in the MMU. L2 is invalidated by physical address.
- Cache invalidation must be done line by line (32-byte lines). This is very slow, but it's much slower to read uncacheable memory.
- The Linux kernel cannot map a 400 MiB buffer (used by the IQ recorder) into its address space. There is the 3G/1G split, and mapping a large buffer with the Linux DMA framework exceeds the 1G kernel address space. We cannot use any of the Linux DMA framework to help us.

Solutions:

- Only map the DMA buffers into userspace.
- Implement a kernel module that performs cache invalidation by calling the relevant low-level functions, using userspace virtual addresses. These low-level functions are not exported for modules to use. The kernel must be patched (**very ugly**).

DMA: software side

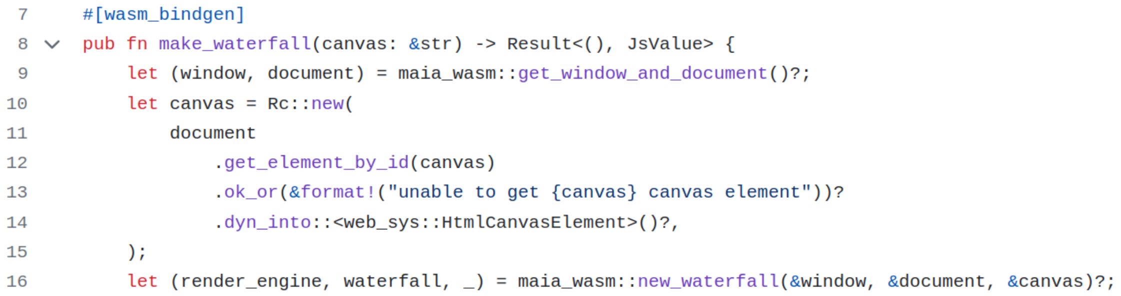
- `maia-kmod`, a Linux kernel module. Manages waterfall and IQ recorder DMA buffers.
- IQ recorder: userspace only mmaps the buffer when it needs to read the recording, after the recording has finished. mmap'ing calls L1 and L2 cache invalidation functions, using the virtual addresses just obtained for the mapping.
- Waterfall: the ring buffer is permanently mmap'ed by userspace. An ioctl is used by userspace to indicate that a buffer in the ring must be invalidated (the buffer number is the ioctl argument).
- A word of caution: typically, mmap'ing `/dev/mem` and similar techniques do not give you regular cacheable memory. Using `remap_pfn_range()` gives us full control of memory attributes.

WebGL2 waterfall

- Coded in Rust using WebAssembly and WebGL2.
- Uses a custom render engine.
- The waterfall data is stored as a 2D texture that shades a scrolling rectangle that uses a "doubly-mapped buffer"-like trick. The fragment shader uses the GLSL ES 3.0 function `texture()` to do a look-up in a 1D texture colormap.
- Only the waterfall data that changes is updated with `texSubImage2D()`.
- The "ticks" in the frequency scale are rendered using the `LINES` draw mode.
- The labels in the frequency scale are pre-rendered onto a texture generated using `CanvasRenderingContext2D` text rendering features.
- Frequency ticks and labels drawn are selected dynamically according to zoom level.

Waterfall example: demo

Waterfall example: Rust code

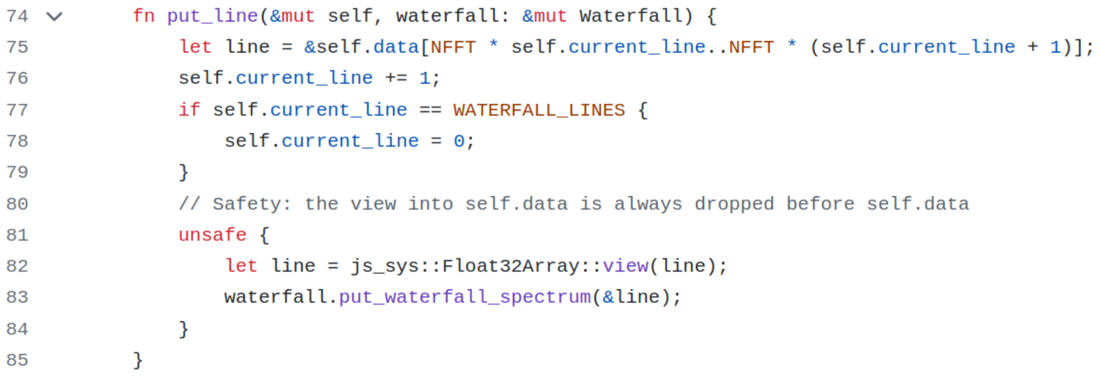


```rust
#[wasm_bindgen]
pub fn make_waterfall(canvas: &str) -> Result<(), JsValue> {
    let (window, document) = maia_wasm::get_window_and_document()?;
    let canvas = Rc::new(
        document
            .get_element_by_id(canvas)
            .ok_or(&format!("unable to get {canvas} canvas element"))?
            .dyn_into::<web_sys::HtmlCanvasElement>()?,
    );
    let (render_engine, waterfall, _) = maia_wasm::new_waterfall(&window, &document, &canvas)?;
```

```
        let center_freq = 915e6;
        let samp_rate = 960e3;
        // An averaging of 8 and FFT size of 4096 were used to construct the
        // waterfall data
        let waterfall_averaging = 8;
        let waterfall_rate = samp_rate / ((NFFT * waterfall_averaging) as f64);
        {
            let mut waterfall = waterfall.borrow_mut();
            waterfall.set_freq_samprate(center_freq, samp_rate, &mut render_engine.borrow_mut())?;
            waterfall.set_waterfall_min(20.0);
            waterfall.set_waterfall_max(280.0);
            waterfall.set_waterfall_update_rate(waterfall_rate as f32);
        }
```

```rust
        let mut generator = WaterfallGenerator::new();
        let handler = Closure::<dyn FnMut()>::new({
            let waterfall = Rc::clone(&waterfall);
            move || {
                generator.put_line(&mut waterfall.borrow_mut());
            }
        });
        let interval_ms = (1000.0 / waterfall_rate).round() as i32;
        window.set_interval_with_callback_and_timeout_and_arguments_0(
            handler.into_js_value().unchecked_ref(),
            interval_ms,
        )?;

        maia_wasm::setup_render_loop(render_engine, waterfall);
        Ok(())
```



```rust
    fn put_line(&mut self, waterfall: &mut Waterfall) {
        let line = &self.data[NFFT * self.current_line..NFFT * (self.current_line + 1)];
        self.current_line += 1;
        if self.current_line == WATERFALL_LINES {
            self.current_line = 0;
        }
        // Safety: the view into self.data is always dropped before self.data
        unsafe {
            let line = js_sys::Float32Array::view(line);
            waterfall.put_waterfall_spectrum(&line);
        }
    }
```

Waterfall example: HTML and JavaScript code

```html
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, height=device-height, initial-scale=1">
    <title>Maia SDR waterfall example</title>
    <script type="module" src="./index.js"></script>
  </head>
  <body>
    <canvas id="waterfall" style="width: 100%; height: 80vh; touch-action: none;"></canvas>
  </body>
</html>
```

```javascript
import init, { make_waterfall } from "./pkg/waterfall_example.js";

async function run() {
    await init();
    make_waterfall("waterfall");
};

run();
```