GNU Radio with a Rusty FPGA

Brennan Ashton • 02.03.2019

Overview

How feasible is it to write a GNU Radio block in Rust?

What is involved in accelerating a block with an FPGA?

Why Rust?

- Memory Management
- Concurrency Model
- C FFI
- Language ergonomics
- Dependency Management

```
extern crate rand;
use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    let secret_number = rand::thread_rng().gen_range(1, 101);
    loop {
        println!("Please input your guess.");
        let mut guess = String::new();
        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");
        let guess: u32 = match guess.trim().parse() {
            Ok(num) => num,
            Err(_) => continue,
        };
        match guess.cmp(&secret_number) {
            Ordering::Less => println!("To small!"),
            Ordering::Greater => println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            },
        }
    }
}
```

External Package

Explicit Mutability

Match Statements

https://doc.rust-lang.org/book/

```rust
use std::thread;
use std::time::Duration;

fn main() {
    let handle = thread::spawn(|| {
        for i in 1..10 {
            println!("hi number {} from the spawned thread!", i);
            thread::sleep(Duration::from_millis(1));
        }
    });

    for i in 1..5 {
        println!("hi number {} from the main thread!", i);
        thread::sleep(Duration::from_millis(1));
    }

    handle.join().unwrap();
}
```

https://doc.rust-lang.org/book/

How can GNU Radio talk to Rust?

Foreign Function Interface

Rust talks to C →

```
#[link(name = “c_library")]
extern "C" {
    fn library_func(var: i32) -> u32;
}
```

C talks to Rust →

```
#[no_mangle]
pub extern fn hello_rust() -> *const u8 {
    "Hello, world!\0".as_ptr()
}
```

What does a simple GNU Radio block interface look like?

```
template<class T>
  class BLOCKS_API multiply_impl : public  multiply<T>
  {
    size_t d_vlen;

  public:
    multiply_impl (size_t vlen);

    int work(int noutput_items,
             gr_vector_const_void_star &input_items,
             gr_vector_void_star &output_items);
  };
```

How do we build this binding?

- Rust does not support the CPP ABI, and GNU Radio does not provide a C interface. The class for the block implementation will have to be written in CPP.

- In the case of the multiply example the *work* function does not have any side effects, we can implement that directly.

Sharing the heap

- Rust is really good at cleaning up memory that has gone out of scope.

- Passing a pointer to an object was unsafe, Rust cannot know if that object is still in scope and will free it.

The Box

```
struct Dragon {
    gold: u32,
    fire: bool,
}
impl Poke for Dragon {
    fn poke(&self) {
        self.fire = true;
    }
}
#[no_mangle]
pub unsafe extern "C" fn new_dragon() -> *mut Dragon
{
    let dragon = Dragon {
        gold: 0,
        fire: false,
    };
    Box::into_raw(Box::new(dragon))
}
#[no_mangle]
pub unsafe extern "C" fn dragon_free(dragon: *mut Dragon) {
    let _ = Box::from_raw(dragon);
}
```

More details here:

How I Wrote a Modern C++ Library in Rust:

-- Henri Sivonen

https://hsivonen.fi/modern-cpp-in-rust/

Implementing floating point multiply work function

```
typedef void**              gr_vector_void_star;
typedef const void**        gr_vector_const_void_star;

int work(int noutput_items,
         gr_vector_const_void_star &input_items,
         gr_vector_void_star &output_items);
```

The work function for float

```
template<>
 int
 multiply_impl<float>::work(int noutput_items,
                            gr_vector_const_void_star &input_items,
                            gr_vector_void_star &output_items)
{
  float *out = (float *) output_items[0];
  int noi = d_vlen*noutput_items;

  memcpy(out, input_items[0], noi*sizeof(float));
  for(size_t i = 1; i < input_items.size(); i++) {
    /* volk_32f_x2_multiply_32f(out, out, (float*)input_items[i], noi); */
    rust_f_32f_x2_multiply_32f(out, out, (float*)input_items[i], noi);
  }
  return noutput_items;
}
```

Our Rust function call

Create a Rust library

- Create the library boilerplate with cargo:
  - $ cargo new rust-lib --lib
- Update the Cargo.toml file:

```
[package]
name = "rust-lib"
version = "0.1.0"
authors = ["Brennan Ashton <email>"]
edition = "2018"

[lib]
crate-type = ["staticlib"]

[dependencies]
```

Create a the multiply function

```
void volk_32f_x2_multiply_32f(
        float* cVector,
        const float* aVector,
        const float* bVector,
        unsigned int num_points)
```

$$c[i] = a[i] * b[i]$$

Create a the multiply function

```
use std::slice;
#[no_mangle]
pub unsafe extern "C" fn rust_32f_x2_multiply_32f(
    cvec: *mut f32,
    avec: *const f32,
    bvec: *const f32,
    points: u32,
) {
    let buffa: &[f32] = slice::from_raw_parts(avec, points as usize);
    let buffb: &[f32] = slice::from_raw_parts(bvec, points as usize);
    let buffc: &mut [f32] = slice::from_raw_parts_mut(cvec, points as usize);
    for i in 0..points as usize {
        buffc[i] = buffa[i] * buffb[i];
    }
}
```

Link the build systems

- Rust normally uses cargo to perform the builds, but under that it is using rustc which we can call using Make

- https://github.com/Devolutions/CMakeRust

Link the build systems

- Rust normally uses cargo to perform the builds, but under that it is using rustc which we can call using Make

```
./gr-blocks/rust
├──  include
│   └── rust_lib.h
└── rust-lib
    ├── Cargo.toml
    ├── CMakeLists.txt
    └── src
    └── lib.rs
```

Link the build systems

```
--- a/gr-blocks/CMakeLists.txt
+++ b/gr-blocks/CMakeLists.txt
@@ -21,6 +21,8 @@
 # Setup dependencies
 ########################################################################
 include(GrBoost)
+include(CMakeCargo)
+enable_language(Rust)

 ########################################################################
 # Register component
@@ -38,6 +40,7 @@ GR_SET_GLOBAL(GR_BLOCKS_INCLUDE_DIRS
         ${CMAKE_CURRENT_BINARY_DIR}/lib
         ${CMAKE_CURRENT_BINARY_DIR}/include
         ${VOLK_INCLUDE_DIRS}
+        ${CMAKE_CURRENT_SOURCE_DIR}/rust/include
 )
```

The build

```
[ 16%] Built target grc_generated_yml
[ 16%] Built target pygen_grc_scripts_3143f
[ 16%] running cargo
   Compiling rust-lib v0.1.0 (/home/bashton/repos/gnuradio/gr-blocks/rust/rust-lib)
        Finished release [optimized] target(s) in 0.36s
[ 16%] Built target rust-lib_target
[ 16%] Linking CXX shared library libgnuradio-blocks-3.8git.so
[ 27%] Built target gnuradio-blocks
```

Still use SIMD

```
#[no_mangle]
pub unsafe extern "C" fn rust_f_32f_x2_multiply_32f(
    cvec: *mut f32,
    avec: *const f32,
    bvec: *const f32,
    points: u32,
) {
    let buffa: &[f32] = slice::from_raw_parts(avec, points as usize);
    let buffb: &[f32] = slice::from_raw_parts(bvec, points as usize);
    let buffc: &mut [f32] = slice::from_raw_parts_mut(cvec, points as usize);
    (buffa.simd_iter(f32s(0.0)), buffb.simd_iter(f32s(0.0)))
        .zip()
        .simd_map(|(a, b)| a + b)
        .scalar_fill(buffc);
}
```

SIMD performance

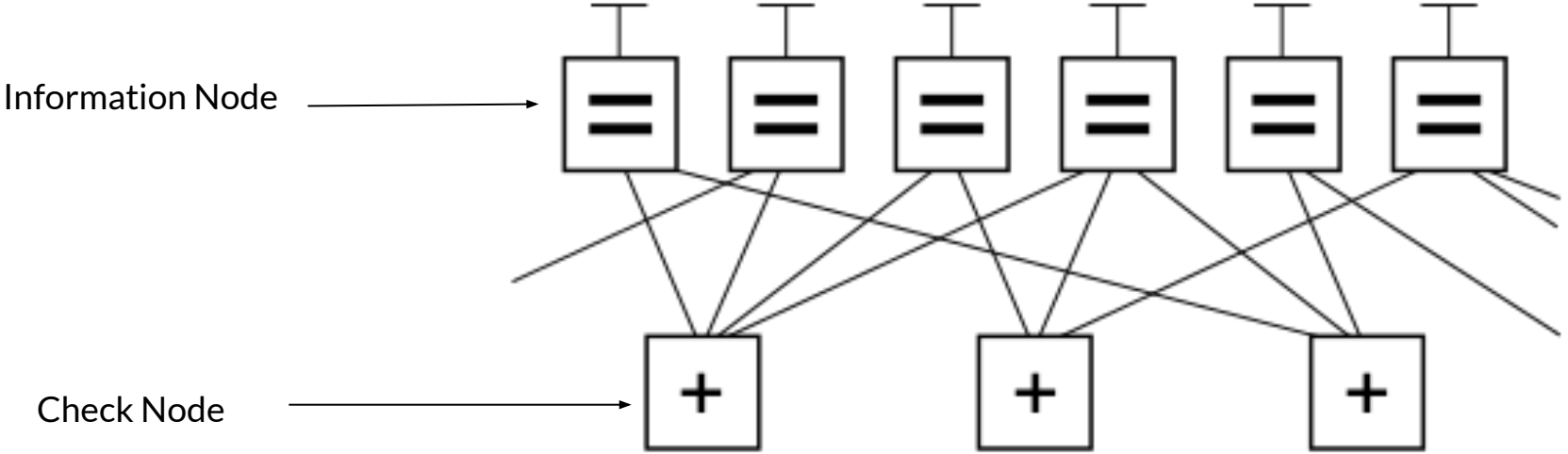
```
$  RUSTFLAGS="-C no-vectorize-loops -C target-cpu=native" cargo  bench
        Finished release [optimized] target(s) in 0.01s
        Running target/release/deps/rust_lib-7ae080876e912313

running 4 tests
test tests::bench_mult          ... bench:    28,694 ns/iter (+/- 1,289)
test tests::bench_mult_faster ... bench:    26,152 ns/iter (+/- 2,106)
```

What does this have to do with an FPGA?

What does this have to do with an FPGA?

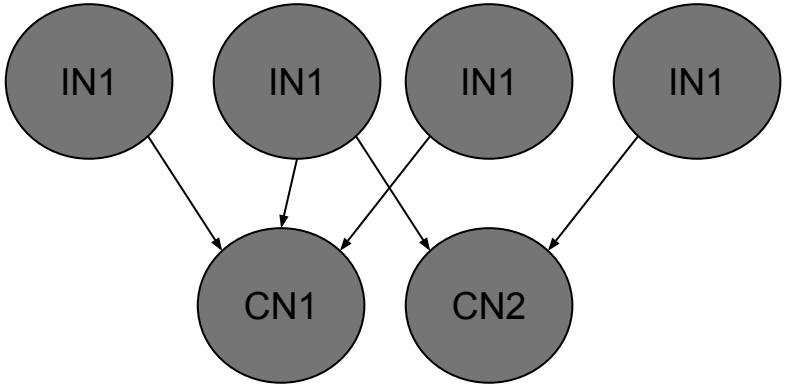
- The decoder was first implemented in Rust and portions of the code removed as the functionality was moved into the FPGA.

(Low Density Parity Check) LDPC Decoder



DVB-S2 R=(½) :
64800 Information Nodes
32400 Check Nodes
Bit rates up to 85Mbps

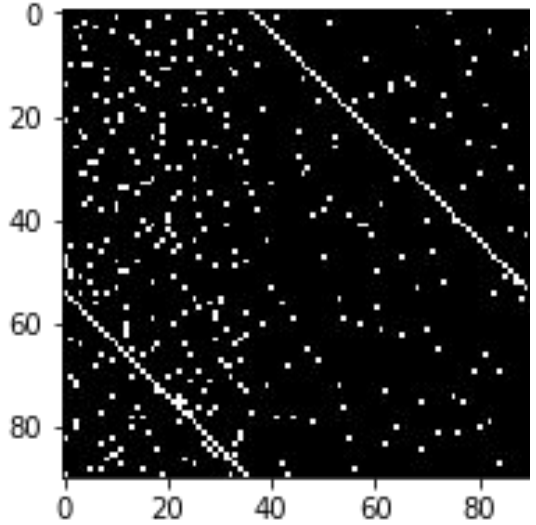
Matrix Representation



$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

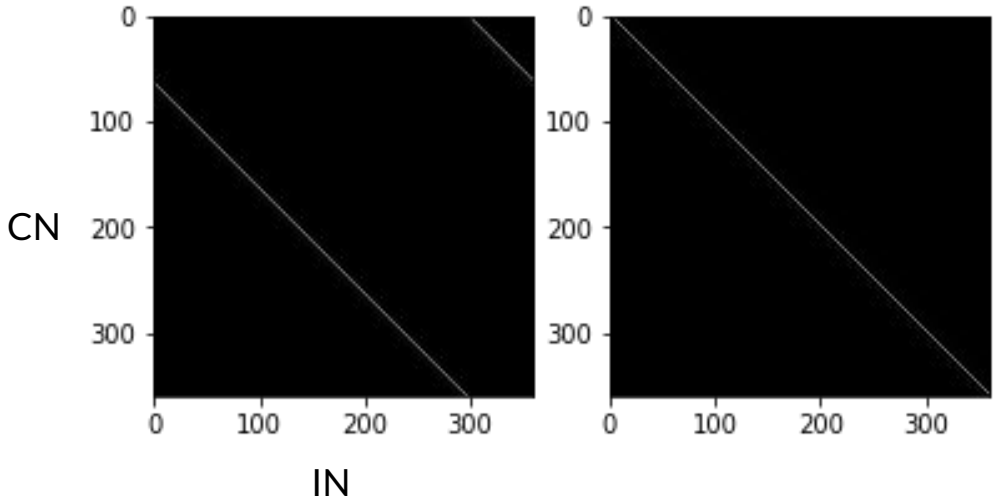
There is a intentional pattern

- Each white dot is a shifted identity matrix

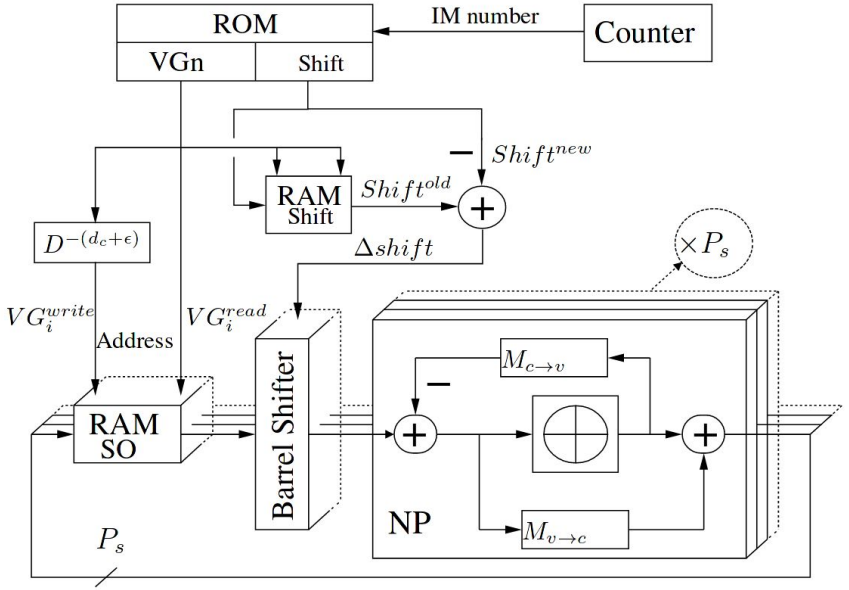


There is a intentional pattern

- We can process these 390 connections in parallel.
- The memory is aligned.



This enables implementation of the layered decoder



Cédric Marchand. Implementation of an LDPC decoder for the DVB-S2, -T2 and -C2 standards. Electronics. Université de Bretagne Sud, 2010. English. <tel-01151985>

How can we connect a FPGA block to GNU Radio?

The Zynq MPSoC

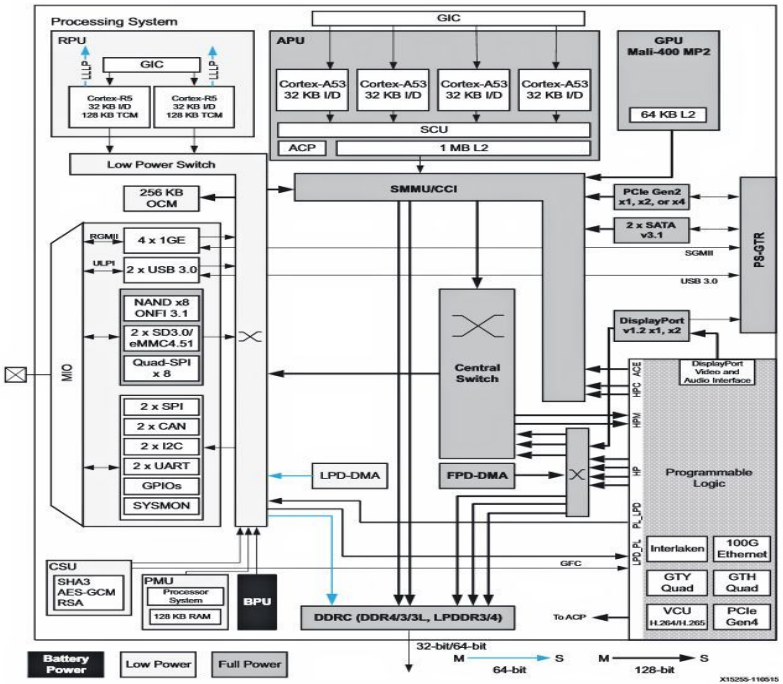


Figure 1-1: **Zynq UltraScale+ MPSoC Top-Level Block Diagram**

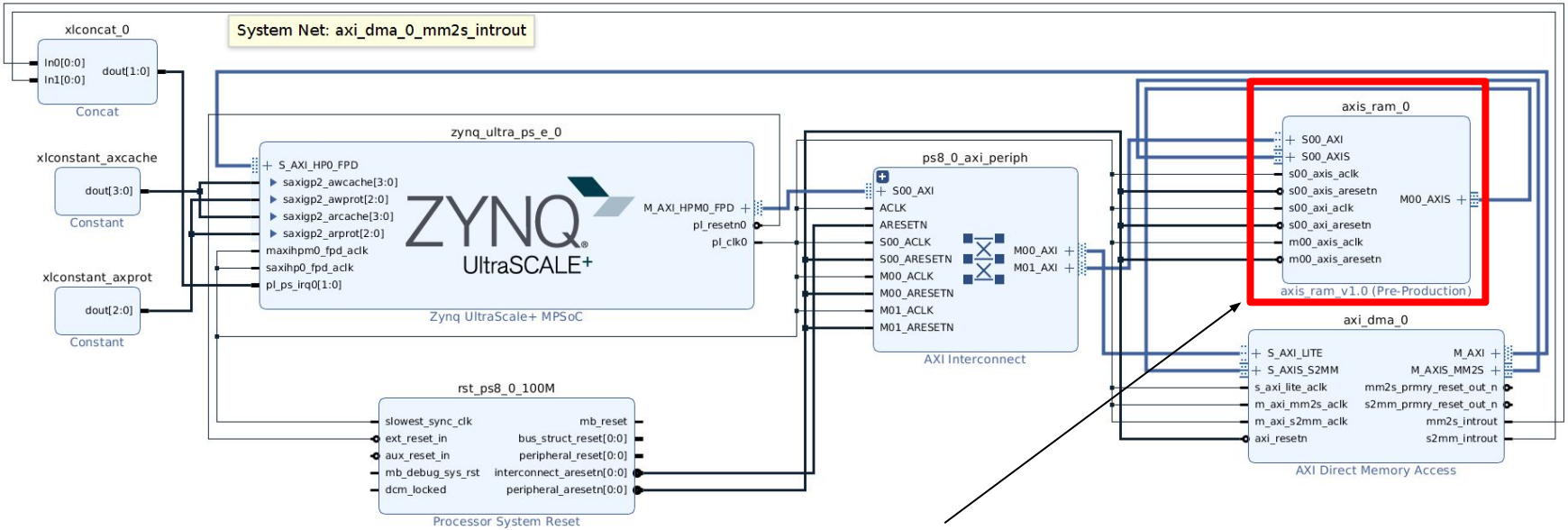
High Level Function

- The application running on the ARM processor performs a DMA transfer of a data frame. It also requests a DMA transfer for the frame out.

- The DMA transfer of the decoded frame is blocked until the decoder is complete.

- The AXI-DMA block converts this data into a stream that is read by the encoder block.

The Zynq MPSoC Implementation



Custom Block

Making the DMA Request

- Since the core application is running in user space, it would be ideal to also be able to take care of the DMA transfer there as well.

- Issues:
  - **User Space Cache Control**
  - **User Space Memory Allocation**

VFIO "Virtual Function I/O"

- This is the same framework that is used to allow virtual machines to directly access hardware devices with low IO penalty.

https://www.kernel.org/doc/Documentation/vfio.txt

VFIO Memory Allocation

```
/* Enable the IOMMU model we want */
ioctl(container, VFIO_SET_IOMMU, VFIO_TYPE1_IOMMU);
/* Get addition IOMMU info */
ioctl(container, VFIO_IOMMU_GET_INFO, &iommu_info);

// Get a buffer for the source of the DMA transfer and then map into the IOMMU
dma_map_src.vaddr = (u64)((uintptr_t)mmap(NULL, size_to_map, PROT_READ | PROT_WRITE,
                          MAP_PRIVATE | MAP_ANONYMOUS, 0, 0));
dma_map_src.size = size_to_map;
dma_map_src.iova = 0;
dma_map_src.flags = VFIO_DMA_MAP_FLAG_READ | VFIO_DMA_MAP_FLAG_WRITE;
ret = ioctl(container, VFIO_IOMMU_MAP_DMA, &dma_map_src);
```

VFIO Device Register Access

```
/* Get a file descriptor for the device */
device = ioctl(group, VFIO_GROUP_GET_DEVICE_FD, AXIDMA_DEVICE);
/* Test and setup the device */
ret = ioctl(device, VFIO_DEVICE_GET_INFO, &device_info);
reg.index = 0;
ret = ioctl(device, VFIO_DEVICE_GET_REGION_INFO, &reg);
base_regs = (uchar *)mmap(NULL, reg.size, PROT_READ | PROT_WRITE, MAP_SHARED,
                          device, reg.offset);
```

VFIO Device Control

```
// Start the tx transfer

*(u32 *)(base_regs) = 1;
*(u32 *)(base_regs + 0x18) = (u32)(u64)dma_map_src.iova;
*(u32 *)(base_regs + 0x1c) = (u32)((u64)dma_map_src.iova >> 32);
*(u32 *)(base_regs + 0x28) = size_to_map;
```

VFIO IRQ

- Since we have access to the device registers we can simply poll the status register, but this is wasteful.

```
while ((*(u32 *)(base_regs + 0x34) & 0x1000) != 0x1000);
```

VFIO IRQ

- Instead we can create an eventfd which allows the use of select, poll, epoll.

```
e_fd = eventfd(0, 0)
ret = ioctl(device, VFIO_DEVICE_SET_IRQS, irq_setup);
irq_set = malloc(argsz);
irq_set->argsz = argsz;
irq_set->flags = VFIO_IRQ_SET_DATA_EVENTFD | VFIO_IRQ_SET_ACTION_TRIGGER;
irq_set->index = VFIO_PCI_MSIX_IRQ_INDEX;
irq_set->start = 0;
irq_set->count = 1;
pfd = (int32_t *)&irq_set->data;
pfd[0] = e_fd;
ret = ioctl(vdev->fd, VFIO_DEVICE_SET_IRQS, irq_set);
```

https://elixir.bootlin.com/linux/v5.0-rc4/source/include/uapi/linux/vfio.h#L453

Device Interface

- Userspace Driver
  - Memory allocation
  - Control
  - Interrupt

- Rust provides a libc interface to build this driver, but be aware that most of the interfaces are marked as unsafe.