[redacted]

Notes on ARM & FPGA communications

Table of Contents

- Notes on ARM & FPGA communications
- Table of Contents
- INTERFACE FROM THE ARM TO THE FPGA
  - FPGA
    - FPGA modes
  - ARM FPGA communications
  - ARM GPIO setup
  - FPGA Setup
- HARDWARE OVERVIEW

- ADC (ANALOG TO DIGITAL CONVERTER)
- FIELD PROGRAMMABLE GATE ARRAY, FPGA
- MICROCONTROLLER
-
- To behave like a READER
- To behave like a TAG
- To sniff traffic
- FPGA purpose

https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/original_proxmark3/proxmark3.pdf

INTERFACE FROM THE ARM TO THE FPGA

The FPGA and the ARM can communicate in two main ways: using the ARM's general-purpose synchronous serial port (the SSP), or using the ARM's SPI port. The SPI port is used to configure the FPGA. The ARM writes a configuration word to the FPGA, which determines what operation will be performed (e.g. read 13.56 MHz vs. read 125 kHz vs. read 134 kHz vs...). The SPI is used exclusively for configuration.

The SSP is used for actual data sent over the air. The ARM's SSP can work in slave mode, which means that we can send the data using clocks generated by the FPGA (either from the PCK0 clock, which the ARM itself supplies, or from the 13.56 MHz clock, which is certainly not going to be synchronous to anything in the ARM), which saves synchronizing logic in the FPGA. The SSP is bi-directional and full-duplex.

The FPGA communicates with the ARM through either

1. SPI port (the ARM is the master)
2. SSC synchronous serial port (the ARM is the master).

opamps, (*note, this affects source code in ARM, calculating actual voltage from antenna. Manufacturers never report what they use to much frustration) comparators coil drivers

LF analog path (MCP6294 opamp. This has a GBW of 10 MHz), all 'slow' signals. Used for low frequency signals. Follows the peak detector. Signal centered around generated voltage Vmid.

FPGA

^Top

Since the SPARTAN II is a old outdated FPGA, thus is very limited resource there was a need to split LF and HF functionality into two separate FPGA images. Which are stored in ARM flash memory as bitstreams.

We swap between these images by flashing fpga from ARM on the go. It takes about 1sec. Hence its usually a bad idea to program your device to continuously execute LF alt HF commands.

The FPGA images is precompiled and located inside the /fpga folder.

- fpga_hf.bit
- fpga_lf.bit

There is very rarely changes to the images so there is no need to setup a fpga tool chain to compile it yourself. Since the FPGA is very old, the Xilinx WebPack ISE 10.1 is the last working tool chain. You can download this legacy development on Xilinx and register for a free product installation id. Or use mine `11LTAJ5ZJK3PXTUBMF0C0J6C4` The package to download is about 7Gb and linux based. Though I recently managed to install it on WSL for Windows 10.

In order to save space, these fpga images are LZ4 compressed and included in the fullimage.elf file when compiling the ARM SRC. `make armsrc` This means we save some precious space on the ARM but its a bit more complex when flashing to fpga since it has to decompress on the fly.

FPGA modes

^Top

- Major modes
- Minor modes

ARM FPGA communications

^Top

The ARM talks with FPGA over the Synchronous Serial Port (SSC) rx an tx.

ARM, send a 16bit configuration with fits the select major mode.

ARM GPIO setup

^Top

```
    // First configure the GPIOs, and get ourselves a clock.
    AT91C_BASE_PIOA->PIO_ASR =
        GPIO_SSC_FRAME  |
        GPIO_SSC_DIN    |
        GPIO_SSC_DOUT   |
        GPIO_SSC_CLK;
    AT91C_BASE_PIOA->PIO_PDR = GPIO_SSC_DOUT;

    AT91C_BASE_PMC->PMC_PCER = (1 << AT91C_ID_SSC);

    // Now set up the SSC proper, starting from a known
state.
    AT91C_BASE_SSC->SSC_CR = AT91C_SSC_SWRST;

    // RX clock comes from TX clock, RX starts on Transmit
Start,
    // data and frame signal is sampled on falling edge of RK
    AT91C_BASE_SSC->SSC_RCMR = SSC_CLOCK_MODE_SELECT(1) |
SSC_CLOCK_MODE_START(1);
```

```
    // 8, 16 or 32 bits per transfer, no loopback, MSB first,
1 transfer per sync
    // pulse, no output sync
    if ((FPGA_mode & FPGA_MAJOR_MODE_MASK) ==
FPGA_MAJOR_MODE_HF_READER && FpgaGetCurrent() ==
FPGA_BITSTREAM_HF) {
        AT91C_BASE_SSC->SSC_RFMR =
SSC_FRAME_MODE_BITS_IN_WORD(16) | AT91C_SSC_MSBF |
SSC_FRAME_MODE_WORDS_PER_TRANSFER(0);
    } else {
        AT91C_BASE_SSC->SSC_RFMR =
SSC_FRAME_MODE_BITS_IN_WORD(8) | AT91C_SSC_MSBF |
SSC_FRAME_MODE_WORDS_PER_TRANSFER(0);
    }

    // TX clock comes from TK pin, no clock output, outputs
change on rising edge of TK,
    // TF (frame sync) is sampled on falling edge of TK,
start TX on rising edge of TF
    AT91C_BASE_SSC->SSC_TCMR = SSC_CLOCK_MODE_SELECT(2) |
SSC_CLOCK_MODE_START(5);

    // tx framing is the same as the rx framing
    AT91C_BASE_SSC->SSC_TFMR = AT91C_BASE_SSC->SSC_RFMR;
```

FPGA Setup

^Top

// Set up DMA to receive samples from the FPGA. We will use the PDC, with
// a single buffer as a circular buffer (so that we just chain back to

HARDWARE OVERVIEW

^Top

ADC (ANALOG TO DIGITAL CONVERTER)

^Top

The analogue signal that comes from the antenna circuit is fed into an 8-bit Analogue to Digital Converter (ADC). This delivers 8 output bits in parallel which represent the current voltage retrieved from the field.

FIELD PROGRAMMABLE GATE ARRAY, FPGA

^Top

The 8 output pins from the ADC are connected to 8 pins of the Field Programmable Gate Array (FPGA). An FPGA has a great advantage over a normal microcontroller in the sense that it emulates hardware. A hardware description can be compiled and flashed into an FPGA.

Because basic arithmetic functions can be performed fast and in parallel by an FPGA it is faster than an implementation on a normal microcontroller. Only a real hardware implementation would be faster but this lacks the flexibility of an FPGA.

The FPGA can therefore be seen as dynamic hardware. It is possible to make a hardware design and flash it into the memory of the FPGA. This gives some major advantages:

- "Hardware" errors can be corrected; the FPGA can be flashed with a new hardware design.
- Although not as fast as a real hardware implementation, an FPGA is faster than its equivalent on microprocessor. That is, it is specialized for one job.

The FPGA has two main tasks. The first task is to demodulate the signal received from the ADC and relay this as a digital encoded signal to the ARM. Depending on the task this might be the demodulation of a 100% Amplitude Shift Keying (ASK) signal from the reader or the load modulation of a card. The encoding schemes used to communicate the signal to the ARM are Modified Miller for the reader and Manchester encoding for the card signal.

The second task is to modulate an encoded signal that is received from the ARM into the field of the antenna. This can be both the encoding of reader messages or card messages. For reader messages the FPGA generates an electromagnetic field on power hi and drops the amplitude for short periods.

MICROCONTROLLER

^Top

The microcontroller is responsible for the protocol management. It receives the digital encoded signals from the FPGA and decodes them. The decoded signals can just be copied to a buffer in the EEPROM memory. Additionally, an answer to the received message can be send by encoding a reply and communicating this to the FPGA.

The microcontroller (ARM) implements the transport layer. First it decodes the samples received from the FPGA. These samples are stored in a Direct Memory Access (DMA) buffer. The samples are binary sequences that represent whether the signal was high or low. The software on the ARM tries to decode these samples. When the Proxmark is in sniffing mode this is done for both the Manchester and Modified Miller at the same time. Whenever one of the decoding procedures returns a valid message, this message is stored in another buffer (BigBuf) and both decoding procedures are set to an un-synced state. The BigBuf is limited to the available memory on the ARM. The current firmware has 2 KB of memory reserved for traces (Besides the trace, the buffer also stores some temporary data that is needed in the processing). When the BigBuf buffer is full the function normally returns. A new function call from the client is needed to download the BigBuf contents to the computer. The BigBuf is especially useful for protocol investigation. Every single message is stored in this buffer. When a card is emulated or when the Proxmark is used as a reader the BigBuf can be used to store status messages or protocol exceptions.

```
HF PATH
 -- ANTENNA -> rectifying -> lowpass filter -> ADC -> FPGA ->
ARM -> USB/CDC | FPC -> CLIENT
       |             |                          |      |
     induct     peak detect                  (8bit)   --
modes:
                via circuit                                HF
- peak-detected
                                                           HF
- RAW
                                                           HF
-
```

```
LF PATH

 -- ANTENNA -> rectifying -> lowpass filter -> ADC -> FPGA ->
ARM -> USB/CDC | FPC -> CLIENT
```

```
        |             |                              |      |
      induct     peak detect                     (8bit)   --
modes:
                  via circuit                                  LF
- peak-detected
                                                               LF
- RAW
```

Problems:

1. dynamic range of signal. Ie: High Carrier signal (reader) and low

---

To behave like a READER

^Top

By driving all of the buffers LOW, it is possible to make the antenna look to the receive path like a parallel LC circuit; this provides a high-voltage output signal. This is typically what will be done when we are not actively transmitting a carrier (i.e., behaving as a reader).

To behave like a TAG

^Top

On the receive side, there are two possibilities, which are selected by RLY1. A mechanical relay is used, because the signal from the antenna is likely to be more positive or negative than the highest or lowest supply voltages on-board. In the usual case (PEAK-DETECTED mode), the received signal is peak-detected by an analog circuit, then filtered slightly, and then digitized by the ADC. This is the case for both the low- and high-frequency paths, although the details of the circuits for the two cases are somewhat different. This receive path would typically be selected when the device is behaving as a reader, or when it is eavesdropping at close range.

It is also possible to digitize the signal from the antenna directly (RAW mode), after passing it through a gain stage. This is more likely to be useful in reading signals at long range, but the available dynamic range will be poor, since it is limited by the 8-bit A/D.

In either case, an analog signal is digitized by the ADC, and from there goes in to the FPGA. The FPGA is big enough that it can perform DSP operations itself. For some high-frequency standards, the subcarriers are fast enough that it would be inconvenient to do all the math on a general-purpose CPU. The FPGA can therefore correlate for the desired signal itself, and simply report the total to the ARM. For low-frequency tags, it probably makes sense just to pass data straight through to the ARM.

The FPGA communicates with the ARM through either its SPI port (the ARM is the master) or its generic synchronous serial port (again, the ARM is the master). The ARM connects to the outside world over USB.

To sniff traffic

^Top

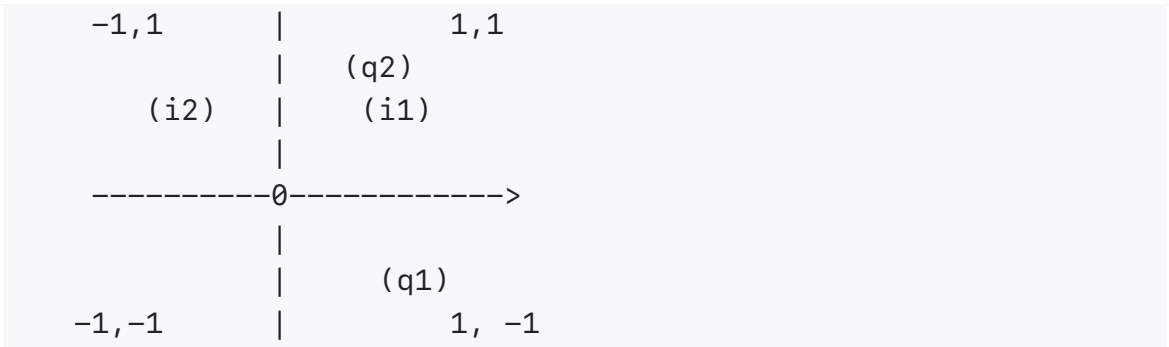
FPGA purpose

^Top

Digital signal processing. In short, apply low pass / hi pass filtering, peak detect, correlate signal meaning IQ pair collecting.

IQ means measure at In-phase and 90 phase shift later Quadrature-phase, with IQ samples you can plot the signal on a vector plan.

```
IQ1 =   1,1 :  1, -1  (rising)
IQ2 =  -1,1 :  1, 1   (falling)
```


−1,1
1,1
(q2)
(i2)
(i1)
0
(q1)
−1,−1
1, −1