Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Microelectronics

FastIC+ readout system

Bc. Vojtěch Vosáhlo

Supervisor: Ing. Vít Záhlava, CSc.
Field of study: Electronics
May 2023

Acknowledgements

—-

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 25. May 2023

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, května 25, 2023

Abstract

—-

Keywords: PCB, Internet of Things, gateway, BLE, Zigbee

Supervisor: Ing. Vít Záhlava, CSc.

Abstrakt

Tato práce se zabývá návrhem elektronického systému určeného k vyčítání dat z integrovaných obvodů FastIC+ vyvíjených v rámci CERN. Prvně nastiňue vlastnosti zmíněných ASIC a jejich účel. Následně detailně popisuje návrh elektroniky a firmware zařízení. V neposlední řadě popisuje také návrh uživatelského rozhraní.

Klíčová slova: DPS, Internet Věcí, brána, BLE, Zigbee

Překlad názvu: Vyčítací systém pro FastIC+

Contents

Figures

Tables

1

Contents

Chapter 1

Introduction

In order to enhance the time measurement of the ToA and ToT analog pulses, FastIC+ has added a picosecond TDC for digitizing the pulses on the chip itself. This, in turn, has necessitated the need for a high-speed communication channel capable of transferring the resulting data stream. The Aurora 64B/66B protocol was chosen for its sufficient speed and easy interfacing with FPGAs.

Chapter 2

FastIC+

The FastIC+ is a configurable ASIC for fast timing applications, featuring 8-channel front-end for photo detectors such as SiPM, PMT or MCP, capable of precisely measuring the Time-of-Arrival (ToA) and Time-over-Treshold (ToT) of photons hitting the detectors. This feature set finds it's use in applications that require precise photon timestamping, such as Time-of-Flight Positron Emission Tomography, high-energy physics, mass-spectrometry or LIDAR applications. The ASIC is developed in the 65 nm technology by the Institut de Ciencies del Cosmos of the University of Barcelona in close colaboration with CERN.

The ASIC can be mostly used without any additional circuitry. Only power and a 40 MHz low jitter reference clock needs to be provided for correct operation. The chip is configured over a I2C interface.

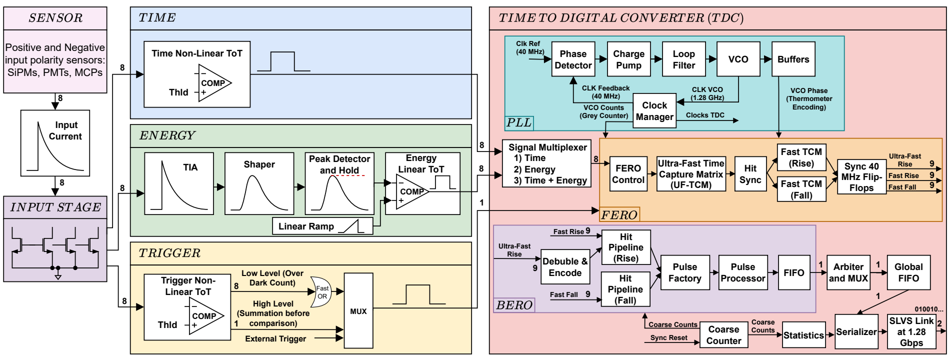


Figure 2.1: Top-level architecture block diagram

2.1 Detection

Each channel of the ASIC consists of a low impedance input stage, working with both positive and negative polarity sensors with dynamic range of 5 µA to 20 mA and stability across sensor capacitances from 10 pF to 1 nF. This stage generates three differently scaled replicas of the input current pulse and forwards them to the three signal paths: time, energy and trigger.

2.1.1 Time path

The time path acts as a simple discriminator which compares the pulse to a programmable treshold, which can be set down to a single photoelectron level. The leading edge of the comparator output pulse provides the ToA timestamp. The logical OR of all time channels can be output via the `TIME` output.

2.1.2 Energy path

The energy path extracts the pulse peak in order to estimate the energy deposited in the sensor. The chain consists of a transimpedance amplifier, shaper, peak detector with hold and a comparator that compares the peak detector level with a linear ramp. The length of the output pulse than directly encodes the energy of the pulse. The input can also be compared to a constant treshold to provide a non-linear ToT instead.

2.1.3 Trigger path

The final path, the trigger, generates either a low-level trigger per every channel or a cluster trigger. The low-level trigger is an logical OR of all the trigger comparators for all channels whereas the cluster trigger results from a analog summation of the input pulses passed through a single comparator. This trigger can be used internally to trigger the conversion FSM or output on a pin. An external trigger can be provided on a pin aswell.

2.2 Digitizing

The time, energy and trigger pulses from each channel are than passed to a Front-End Readout block (FERO) which interpolates the input pulse and obtains a digital representation of the rising edge (with a 25 ps time bin) and the falling edge (with a 390 ps time bin).

After the edge capture, the timing information is passed to the Back-End Readout Block (BERO) which processes the information. It corrects any signal errors, performs trigger validation and filering and encodes the timing information into a suitable binary format. In the end, it stores the encoded packet in the channels FIFO buffer. Arbiter with a MUX than handles the requests from all the channel FIFOs and stores the packets in a global FIFO.

As the last step, an Aurora serializer converts the packets from the FIFO into a Aurora 64B/66B serial stream and sends it over the high speed differential output lines. A statistics information and counter information is also sent if enabled. The stream can be configured to run at speeds from 80 Mb/s to 1.28 Gb/s in the accordance with the Aurora specification.

Chapter 3

Aurora protocol

Aurora 64B/66B is a link-layer, point-to-point protocol that enables high-speed data transfers between two Aurora partners. The Aurora channel, established between the partners, can comprise one or more simplex or full-duplex lanes. Data frames are transmitted in 66-bit blocks, consisting of a two-bit synchronization preamble and 64 bits of data. The channel is shared for both data and control messages.

Frame notation and bit order adopted from the Aurora specification [1] is used in this chapter, the leftmost bit being the first one transmitted to the bus is considered MSB. The rightmost one LSB.

3.1 Encoding

As mentioned above, the Aurora protocol utilizes the commonly used 64B/66B encoding. This encoding scheme converts 64 bits of data into a 66-bit block by appending a two-bit preamble. The `0b01` preamble signifies that the block contains 8 octets of data and is therefore called a *Data block*. If the preamble is `0b10`, the block is recognized as *Control block* with the most significant octet in the block being a *BTF* (Block Type Field). The remaining 7 octets are data. Preambles `0b00` and `0b11` are interpreted as errors. These preambles are than used by the receiver to synchronize to the data stream.

FastIC+ is capable of transmitting the *Data block* and four of the *Control block* types:

- *Separator-7* block indicated by the *BTF* value `0xE1`,
- *Separator* block indicated by the *BTF* value `0x1E`,
- *User K-Block* with *BTF* value indicating the ID of the block (see 3.1.4),
- *Idle* block indicated by the *BTF* value `0x78`.

3.1.1 Data block

A *Data block*, shown in Figure 3.1, consists of eight octets of raw data. Data blocks are transmitted only when eight or more octets of data are available.

For smaller packets, the *Separator-7* and *Separator* blocks are used.

0	1	2 – 9	10 – 17	18 – 25	26 – 33
0	1	D7	D6	D5	D4

34 – 41	42 – 49	50 – 57	58 – 65
D3	D2	D1	D0

Figure 3.1: Data block structure

3.1.2 Separator-7 block

A *Separator-7 block* is transmitted when only seven octets of data remain to be sent over the interface. The block consists of the *BTF* indicating the *Separator-7* type and seven valid octets of data as shown in Figure 3.2.

0	1	2 – 9	10 – 17	18 – 25	26 – 33
1	0	0xE1	D6	D5	D4

34 – 41	42 – 49	50 – 57	58 – 65
D3	D2	D1	D0

Figure 3.2: Separator-7 block structure

3.1.3 Separator block

A *Separator block* is transmitted when 0-6 octets of data remain to be sent. The block, shown in Figure 3.3, consists of the *BTF* indicating the *Separator* type and a field indicating the number of valid data octets. The order of the valid octets is from LSB to MSB.

0	1	2–9	10–17	18–25	26–33
1	0	`0x1E`	`VALID OCTETS`	`D5`	`D4`

34–41	42–49	50–57	58–65
`D3`	`D2`	`D1`	`D0`

Figure 3.3: Separator block structure

3.1.4 User K-block

previouslyto 8, corresponding to *BTFs* `0xD2`, `0x99`, `0x55`, `0xB4`, `0xCC`, `0x66`, `0x33`, `0x4B` and `0x87`, respectively.

0	1	2–9	10–17	18–25	26–33
1	0	`BTF`	`D6`	`D5`	`D4`

34–41	42–49	50–57	58–65
`D3`	`D2`	`D1`	`D0`

Figure 3.4: User K-Block structure

3.1.5 Idle block

Figure 3.5 shows a structure of an *Idle block*. When no data is available for transmission, *Idle* blocks are transmitted instead to maintain the receivers ability to remain in sync and recover the data clock.

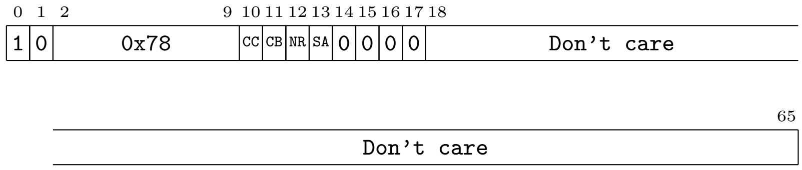


Figure 3.5: Idle block structure

The *Idle block* contains four flag bits determining the type of the block:

- `CC` bit indicating that the block is *Clock Compensation* idle,
- `CB` bit indicating that the block is *Channel Bonding* idle,
- `NR` bit indicating that the block is a *Not Ready* idle,
- `SA` bit indicating that the receivers obey the *Strict Alignment* rules.

3.2 Scrambling

Whenever either *Data Block* or *Control Block* is transmitted, the eight octets in the block have to be scrambled with the polynomial shown in Equation 3.1. The scrambling serves as a way to randomize the data stream so that a receive can later recover the clock from the bit transitions and synchronize to the stream. Note that the two bit preamble is never scrambled, otherwise, the synchronization could not be acquired.

$$P(x) = 1 + x^{39} + x^{58} \tag{3.1}$$

Chapter 4

FastIC+ packet structure

As mentioned in section 2.2, the ASIC encodes the timing and other information into a format suitable for transmission. The structure of these packets is described

4.1 Data packet

Each detection event on each channel of the FastIC+ is represented by a 48 bit data packet containing the digitized *ToA* and *ToT* values. The stream of packets is then encoded into Aurora data blocks. A structure of the packet is depicted in Figure 4.1.

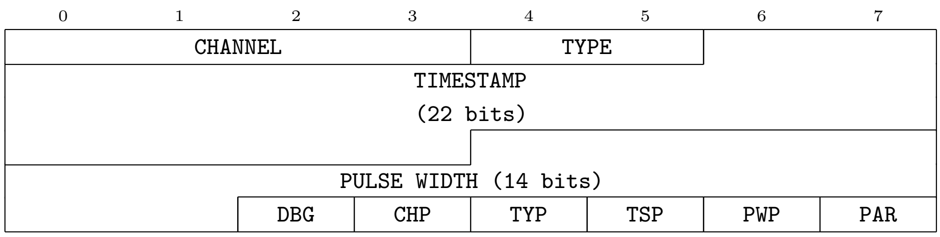


Figure 4.1: FastIC+ data packet structure

The `CHANNEL` field specifies the number of the channel that the event was detected on (0-7) or the trigger channel (8). The `TYPE` field indicates the packet type where the following types are recognized:

- *ToA + non-linear ToT* type represented by the value `0b00`,
- *ToA only* type represented by the value `0b01`,
- *Linear ToT only* type represented by the value `0b10`,
- *ToA + linear ToT* type represented by the value `0b11`.

`TIMESTAMP` and `PULSE WIDTH` fields contain the digitized ToA and ToT values. `DBG` is a flag used in debug mode. The rest of the bits are even parity bits, namely:

- `CHP` parity bit of the `CHANNEL` field,
- `TYP` parity bit of the `TYPE` field,
- `TSP` parity bit of the `TIMESTAMP` field,
- `PWP` parity bit of the `PULSE WIDTH` field,
- and the overall parity bit `PAR` computed from all fields.

4.2 Statistics packet

In addition to the data packets, FastIC+ also periodically transmits a statistics packet that contains basic information about the past events. Structure of the packet is shown in Figure 4.2. This packet is sent onto the Aurora bus in two *K-Blocks* due to the packet size being bigger than the *K-Block* capacity. The first 56 bits are sent in a block with the *BTF* of `0x99`, the rest is sent in a second block with the *BTF* of `0x55`.

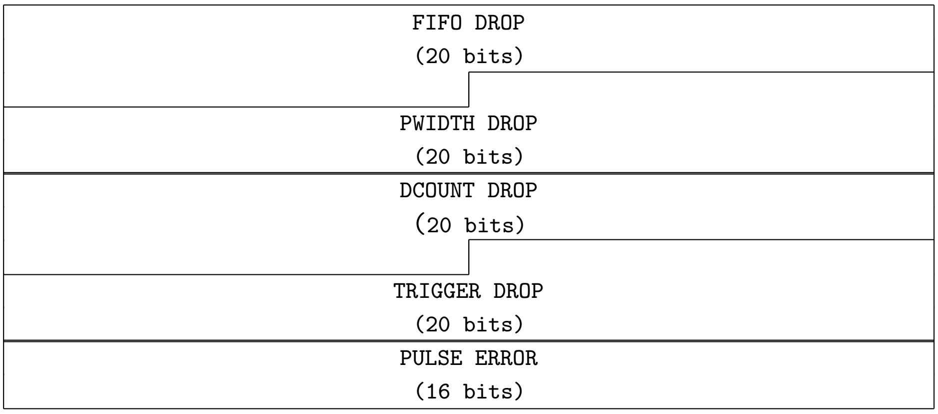


Figure 4.2: FastIC+ statistics packet structure

The packet consists of the following fields:

- `FIFO DROP` field containing a number of events dropped in the FIFO buffer.
- `PWIDTH DROP` field containing a number of events dropped due to the pulse width being outside of the specified range.
- `DCOUNT DROP` field holding the value of dark counts. This field is valid only if *High-Energy Resolution* mode and *Bandwidth optimization* is enabled.
- `TRIGGER DROP` field holding the number of packets dropped due to external trigger pulse not being validated.

- `PULSE ERROR` field being a malformed pulse counter. Malformed pulse is detected when two consecutive edges are read by the TDC. This may occur due to the limitation of one leading+falling edge per clock period.

4.3 Counter Extension + Event Counter packet

When no detection events are processed during the *Coarse Counter Extension* inactivity period, the timestamp counter would overflow the timestamp field sent in the data packet. The following packet, specified in Figure 4.3, is transmitted periodically so the receiver can base the data timestamp on the most recent coarse counter value, thus drastically increase the dynamic range of the time detection. The packet is encoded in an *K-Block* with the *BTF* of `0xD2`.

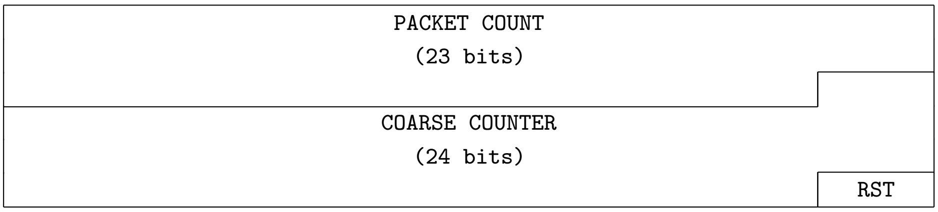


Figure 4.3: FastIC+ counter extension and event counter packet structure

The packets fields are:

- `PACKET COUNT` field, which holds the number of data packets transmitted since the last reset event.
- `COARSE COUNTER` field holding the timestamp from the coarse counter.
- `RST` field indicating, that the coarse counter has been reset during the coarse counter packet period.

Chapter 5

Device concept

The ultimate goal was to develop a system utilizing the FastIC+ chips that could be used at CERN to quickly assess measurements, by companies to simplify prototyping with the FastIC+ chips or by schools to use them in experiments. This implied the base requirements for the system to be:

- versatile to allow for both simple and demanding tasks,
- easy to use for both novices and experts,
- standalone all-in-one device requiring no external scientific instruments,
- miniature and portable to be taken anywhere,
- accessible for schools and companies.

To allow for versatility, it was decided that two FastIC+ chips will be used, resulting in a total of 16 detection channels. The trigger inputs and outputs were decided to be exposed to the user to allow the synchronization of multiple readout boards. On the other hand, it was decided to read the data from the FastIC+ chips at the lowest possible speed of 80 Mb/s, as it was found sufficent for most of the applications and would greatly simplify the development.

To make the readout easy to use and standalone, USB interface was proposed to both power the device and transfer the data to a host computer for processing. This in turn required the device to be power efficent and be capable of sufficient USB bandwith.

Considering all of the above, it was decided to split the device into two parts:

- the readout board containing all the necessary electronics except the sensors,
- the easily replacable userboard containing mainly the sensors.

Chapter 6

Readout board

In the usual use case, a system integrating the FastIC+ chip would be based on an FPGA with dedicated Aurora receiver, or an FPGA with a custom HDL definition of the receiver. This approach results in easier implementation, as the FPGAs integrate the necessary deserializers and are capable of synchronizing to the data stream and reconstructing the data clock. The disadvantage is that capable enough FPGAs are usually expensive and power hungry, which goes against the requirements of the device. The second thing being, even though Aurora receiver implementation is quite simple, the implementation of other interfaces like USB is complicated or requires expensive IP cores to be purchased. These downsides led to the decision of using a microcontroller for the readout system. When a proper microcontroller is selected all the interfaces as

- ADCs for monitoring voltages,
- DACs for providing voltage feedback,
- SPI and I2C for communicating with other digital devices,
- USB for connection to the host PC,

are implemented in hardware and do not have to be defined in HDL code. A simple firmware in C/C++ can than be programmed to control those interfaces, possibly speeding up and simplifying the development and allowing more users to easily customize the device functionality. The main issue with this approach is, that no microcontroller on the market implements the receivers or deserializers needed to recover the clock from the Aurora stream and synchronize to it, thus, the clock recovery has to be done externally and a suitable alternative peripheral has to be chosen to serve as the receiver.

6.1 Microcontroller

The STM32H753XIH6 was chosen as a great microcontroller candidate for the system. It is based on an Arm Cortex-M7 core running at up to 480 MHz which allows for fast computation required for processing of the two continuous 80 Mb/s streams. It integrates 2 MB of flash memory and 1 MB of RAM which is plenty for buffering the data. From the peripheral side, it supports USB High Speed with a maximum throughput of 480 Mb which is needed for transferring the large amount of data to the host. It also features multiple SPI peripherals supporting input clocks of up to 120 MHz which are an ideal choice for sampling the Aurora stream running at the 80 Mb/s. The internal data buses of the microcontroller are running at half of the core clock and support DMA which also dramatically increases the performance with such a big amount of data. Aside from these main features, the chips implementation of two ADCs, one DAC and I2C for digital communication is useful for the readout aswell. The chip is packaged in a compact 14 mm × 14 mm TFBGA 240+25 package.

6.1.1 Receiving the Aurora stream

As noted above, the most suitable peripheral that can be used for receiving the Aurora stream with this microcontroller is the SPI peripheral. This peripheral is an industry standard serial interface, implemented in all of today MCUs, meant for receiving a serial stream with a dedicated clock. In a simplex slave, receiver only mode, which fits this usecase well, the peripheral exposes the `CLK` and `MOSI` pins. The `MOSI` pin is used for inputting the data stream to the peripheral. The clock, present on the `CLK` pin, is than used to sample the data stream. The sampled data is than shifted into a register on a selected clock edge and sent over the internal microcontroller buses for processing. The only issue with using SPI is that the need to recover the clock from the data stream persists.

6.1.2 Omitting clock recovery with the FastIC+

As noted in section 2, the FastIC+ requires a 40 MHz reference clock to function. The chip features an internal PLL, which synchronizes to this clock and distributes it to other peripherals including the Aurora transmitter. Since the PLL phase is locked to the input clock, the Aurora transmitter, and thus the Aurora output stream, is also phase locked to the input clock, just delayed by the transmitter propagation delay t_{pd}. By measurement, it has been found that this delay is very stable for temperature range of 0 °C - 100 °C at a value $t_{pd} = (3.48 \pm 0.08)$ ns. If this delay is combined with an additional controlled delay, the digital stream can be aligned such that the data can be sampled with an 80 MHz sampling clock thats in phase with the FastIC+ reference clock.

The delay could be implemented by a variable delay line. However it turns out that this delay in combination with the typical propagation delay of a common LVDS to CMOS receiver (typ. 5 ns) equals approximately 9 ns. The Aurora data stream is double data rate, meaning that a valid symbol is transfered on both rising and falling edge of the reference clock. This results in the symbol duration of 12.5 ns. The 9 ns delay, being almost exactly 3/4 of the symbol duration, shifts the data such that both sampling clock edges are contained in the data symbol as seen in figure 6.1. Thus the data symbol can be sampled on either the rising or falling edge and the edge can be chosen programatically on the fly, based on the receiver propagation delay, to avoid a possibility of sampling in a metastable state of the data stream. Since the FastIC+ outputs an differential stream and a MCU with CMOS inputs is used to capture the stream, this receiver has to be in place anyways and if carefully chosen, it can at the same time be used as the delay line to achieve the correct sampling clock phase and thus completly avoid recovering the clock from the data stream.

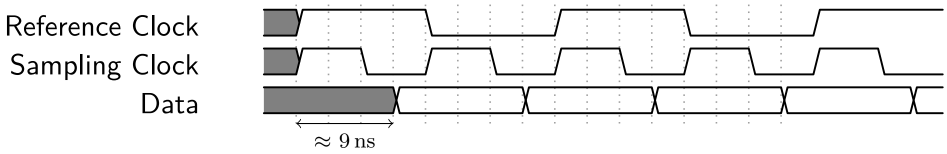


Figure 6.1: Alignment of the sampling clock and data stream

6.2 FastIC+

As mentioned above, the FastIC+ requires almost no additional components aside from 1.2 V power source and the 40 MHz reference clock. The XVDD (PLL) power domain has been isolated from the TVDD (treshold circuitry) by a ferrite bead, to reduce noise coupling between the two. However, the PLL is only expected to produce noise at startup, while the treshold circuitry is used only after the PLL has locked, so any noise coupling between the two domains should have little to no impact. All of the power domains have been thourougly decoupled as seen in figure 6.2.

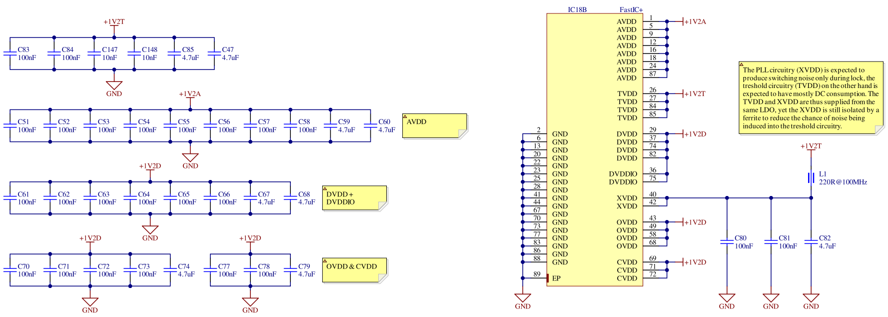


Figure 6.2: Schematic of the FastIC+ power

To increase the stability of the internal band gap reference, a 10 nF capacitor has been added to the `VBG` pin.

Both `nRST` (reset of the FastIC+) and `SRST` (reset of the synchronous counter) have been pulled up to the digital supply so that the microcontroller pins in open drain mode can be used to control these pins without the need for voltage translation.

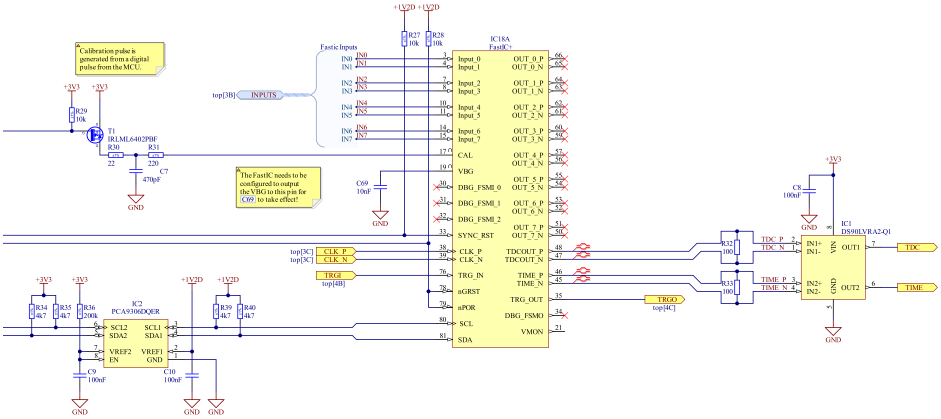


Figure 6.3: Schematic of the FastIC+ logic

6.2.1 I2C communication

As the FastIC+ voltage domains all run at 1.2 V, a voltage shifter had to be implemented for communication with the microcontroller over I2C. For this, the PCA9306DQER has been chosen for its miniature X2SON package and sufficent 400 kHz speed.

6.2.2 Calibration pulse generator

The FastIC+ features a `CAL` input pin used for injecting a test pulse into any of the eight input channels. This pin converts the pulse to current with an internal 70 Ω resistor. The usual shape of such a pulse should resemble a real sensor output, thus a decaying exponential with an amplitude of a few milliamperes and length under a microsecond. To create this pulse, a high side switch has been implemented with series resistance to limit the current and parallel capacitance to recreate the decaying exponential. The transistor gate is driven by a quick digital pulse from the microcontroller, whos length can be adjusted to adjust the current pulse width and by some degree also the amplitude.

6.2.3 Voltage monitoring

The `VMON` pin on the ASIC serves for monitoring of the internal analog tresholds and DAC outputs. Since the microcontrollers internal voltage reference has been selected to run at 1.8 V, a simple non inverting amplifier, with a voltage gain of

$$A_V = \left(1 + \frac{R41}{R44}\right) = \left(1 + \frac{10\,\mathrm{k}\Omega}{20\,\mathrm{k}\Omega}\right) = 1.5 \tag{6.1}$$

has been implemented to amplify the 1.2 V output to the microcontrollers full-scale range and improve the performance. Because the `VMON` output is used only for treshold monitoring, thus only DC voltages, a slow RC filter has been used to reduce the noise coupled to the analog signal.

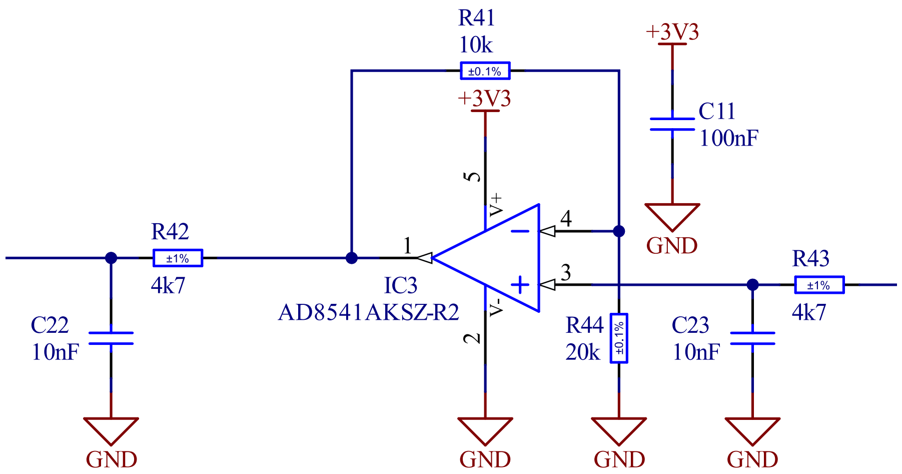


Figure 6.4: Schematic of the voltage monitoring amplifier

6.2.4 High speed outputs

The FastIC+ outputs are realized using the Scalable Low Voltage Signaling standard. SLVS is a alternative to the LVDS, utilizing lower voltages. The transmitter common mode voltage V_{CMTX} being 0.6 V and the differential voltage $|V_{OD}|$ being 0.2 V as seen in figure 6.5.

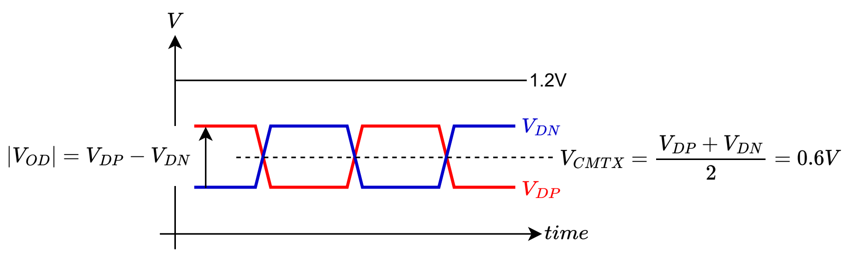


Figure 6.5: Voltage definition for SLVS transmitter.

A differential output capable of transmitting a pulse whos beginning timestamps the ToA of a photon and length resembles the ToT is present for each channel separately. However, as the readout uses the data digitalized by the internal TDC, these channels are left unconnected and disabled in the chip.

The `TIME` output can either be used to generate a digital pulse whenever any input is received on any of the channels or can be internally connected to the trigger comparator and used for trigger calibration routine, the latter being used in this case. When the pin is not used for calibration, it is disabled to reduce any EMI generated by the fast edges. The `TDCOUT` is the output of the Aurora stream from the transmitter.

Both of the above mentioned pins are connected to the DS90LVRA2 dual channel differential line receiver and terminated by a 100 Ω resistance. This receiver, used to convert the SLVS output to CMOS, has been chosen specifically for its small size, high enough speed but also for its typical propagation delay $t_{pd} = 4.4\,\text{ns}$ to correctly align the data to the sampling clock. Even though it is made to convert LVDS signals, the treshold voltage levels work well with the SLVS standard mentioned above, thus no other special circuitry is needed.

6.2.5 Trigger input and output

The trigger inputs (used for externally triggering the ASIC) and outputs (outputting the signal from the internal comparator) have been exposed to the user on two SMA connectors. Termination has been placed on these to mitigate any reflections and ESD protection diodes have been added to protect the chip from electrostatic discharge. It's important to note that the ESD diodes add capacitance to the trigger lines, thus degrading (slowing) the trigger edge and introducing a slight delay in the trigger. This either needs to be accounted for when using the readout or the diodes need to be left unassembled at the expense of worse ESD immunity.

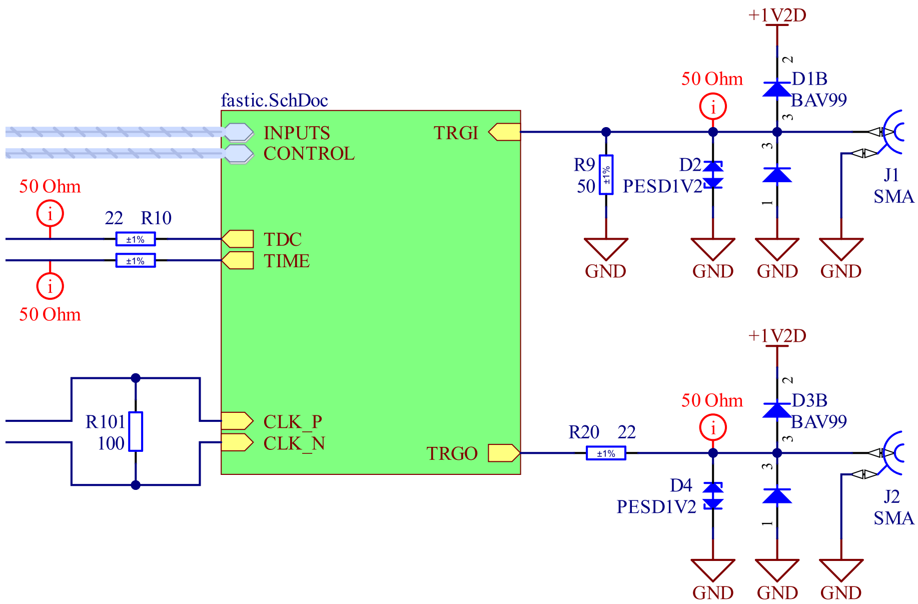


Figure 6.6: Schematic of the FastIC+ trigger circuitry

6.3 USB

To supply power to the device and handle communication with the host computer, a USB type C connector has been used.

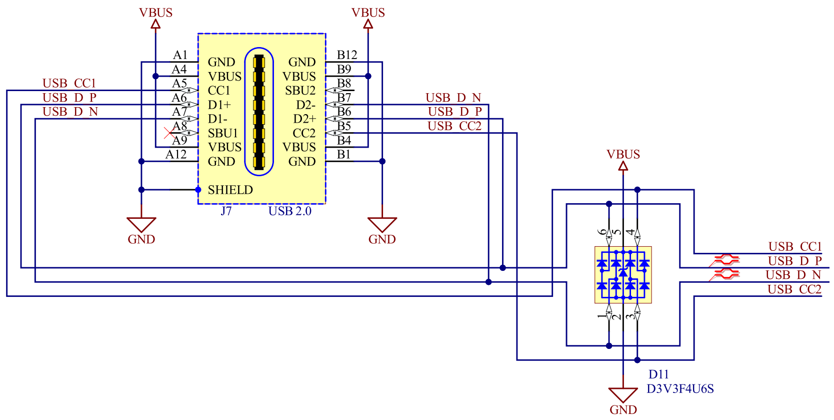


Figure 6.7: Schematic of the USB connector with ESD protection

External PHY

Because the combined data rate of the FastIC+ chips of 160 Mb/s exceeds the USB 1.1 specification, a USB 2.0 (High Speed) was implemented, supporting up to 480 Mb/s throughput. Unfortunately, the used microcontroller does not integrate the USB 2.0 PHY directly. Instead, it only integrates the necessary logic and interfaces with an external PHY via the ULPI interface.

A USB3320 interface was chosen as it is well supported by the microcontroller and widely used in countless designs. A 48 MHz crystal oscillator has been used as the external reference clock required for the device to operate. Different crystal frequency selection has been made possible with 0 Ω resistor jumpers. The ULPI connections have been series terminated, to better match the driver output impedance to the cotrolled impedance of the line.

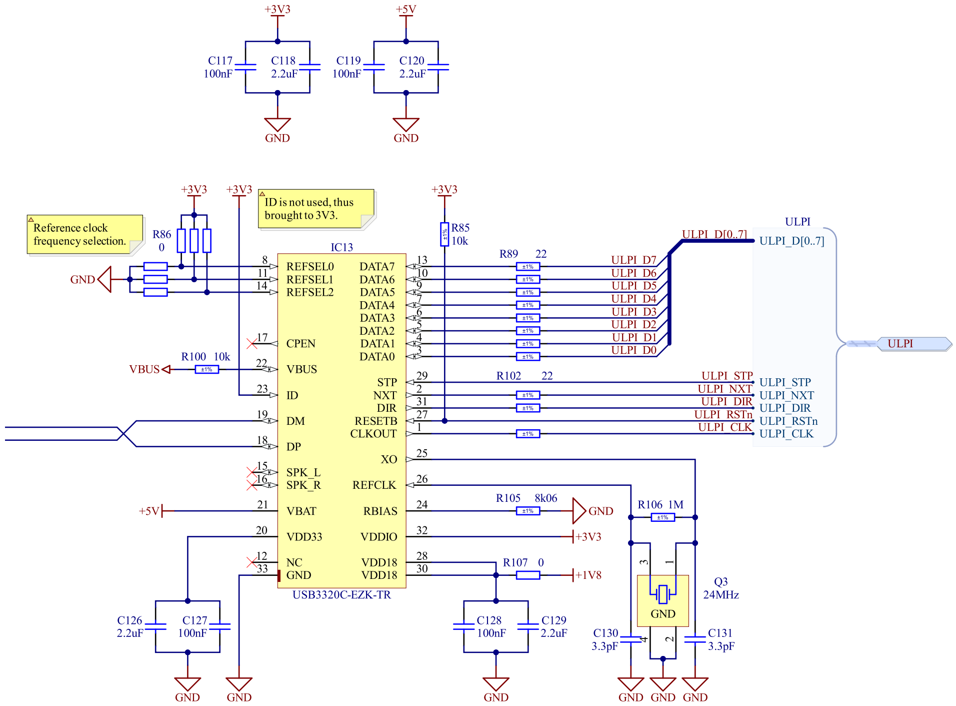


Figure 6.8: Schematic of the USB

Power Delivery

A FUSB302 chip was added to allow for power limit negotiation with a UCPD capable power source. The chip handles all the necessary signaling and communicates with the MCU via a I2C interface. Optional 5.1 kΩ resistors have been added to passively negotiate the highest power limit, if it would be decided to omit the power delivery functionality and not assemble the FUSB302.

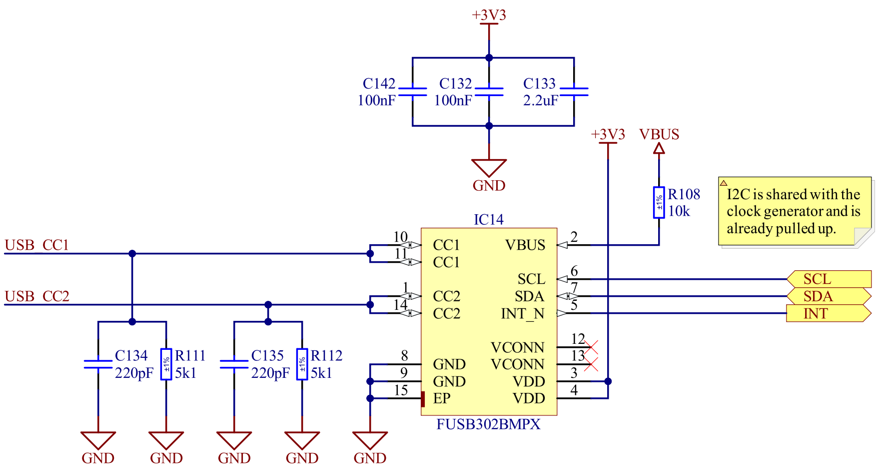


Figure 6.9: Schematic of the USB

6.3.1 Clock generation

For generating the 40 MHz reference clock for both of the FastIC+ chips and the 80 MHz sampling clock for the SPI peripherals, the SI5340 clock synthetizer has been used. It features four outputs with 90 fs RMS jitter, supporting both CMOS and differential output. Additionally, each output can be powered by an independent power supply allowing for mixing the 3.3 V CMOS signals for the microcontroller and differential signals for the FastIC+. A precise 48 MHz crystal oscillator has been used as the reference clock. The chip features both I2C and SPI interface for configuration. I2C was selected to be used in this design.

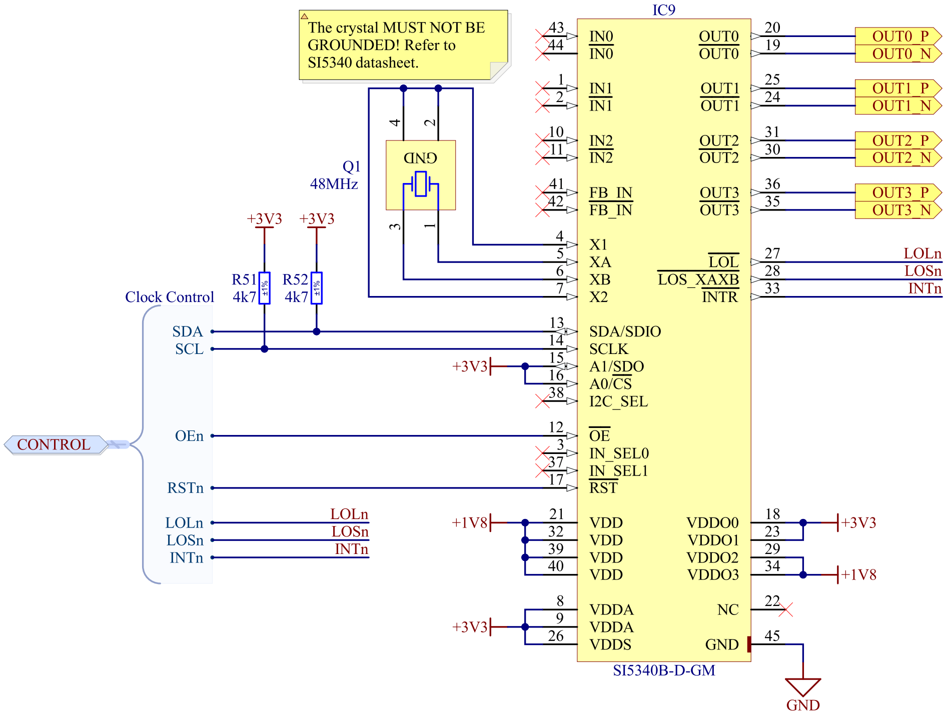


Figure 6.10: Schematic of the clock generator

Since the FastIC+ requiers an SLVS input, which is not supported by the generator a signal divider had to be implemented. This circuit, shown in figure 6.11, was provided in the datasheet as a good way to lower the voltage levels while keeping the impedance of the lines well matched.

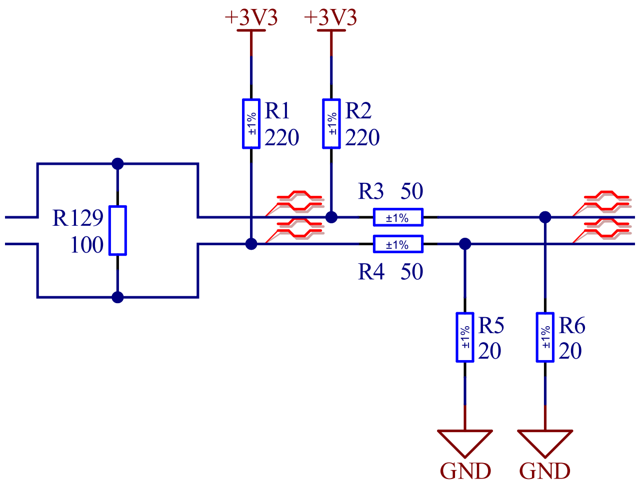


Figure 6.11: Divider for the LVDS to SLVS conversion

6.3.2 High Voltage

All the sensors, which the FastIC+ shall interface with, like SiPMs, PMTs or MCPs need high voltage biasing for their function. To eliminate the need for an external HV supply, an internal one has been implemented using the LT3571. This DC/DC converter, intended for biasing of avalanche photodiodes, is capable of generating up to 75 V output from a low voltage input. It fully integrates the power switch and regulation along with soft-start and variable switching frequency. This has been set to approximately 1 MHz via `R61`. Higher switching frequency than allows for use of smaller inductance and thus keep the size of the device small.

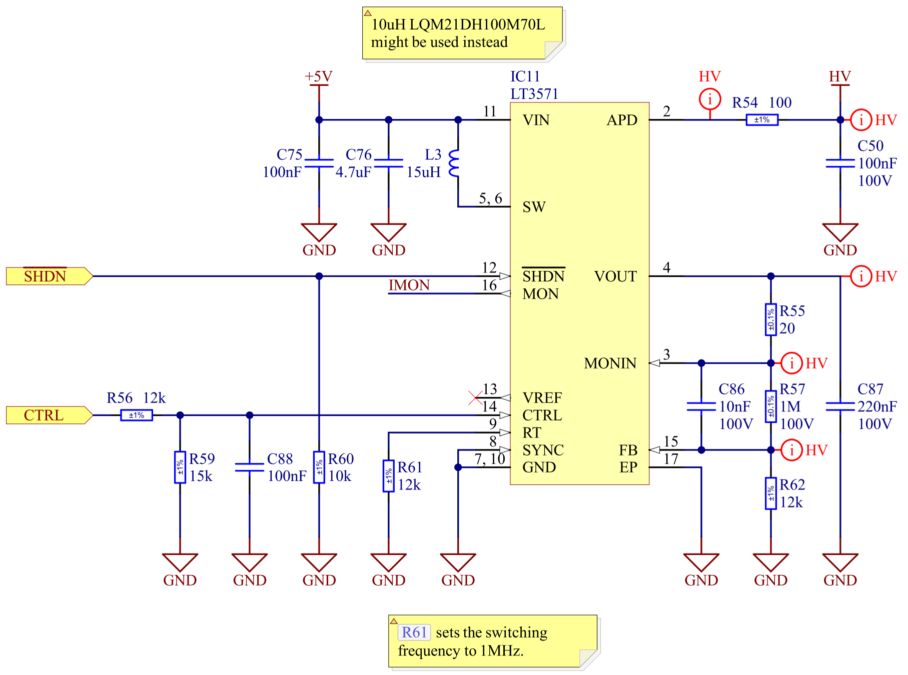


Figure 6.12: High voltage power supply schematic

A CTRL input is available for adjusting the output with a control voltage. This reference is generated by the MCU DAC and divided by a voltage divider to a suitable 0 V - 1 V range. The feedback resistor divider was chosen such that the theoretical maximum output voltage with the CTRL pin held at 1 V is 84.3 V according to equation 6.2.

$$V_{MONIN} = (\frac{R57}{R62} + 1) \cdot V_{CTRL} = (\frac{1000\,\mathrm{k\Omega}}{12\,\mathrm{k\Omega}} + 1) \cdot 1\,\mathrm{V} = 84.3\,\mathrm{V} \qquad (6.2)$$

The current limit resistor has been chosen to limit the APD current to approximately 8 mA based on the equation 6.3 mentioned in the datasheet.

$$R_{SENSE} = \frac{200\,\mathrm{mV}}{1.2 \cdot I_{APD} + 0.3\,\mathrm{mA}} \approx 20\,\Omega \qquad (6.3)$$

where I_{APD} is the APD current limit in milliamperes.

To allow for closed loop control with the microcontroller, the output voltage on the APD pin is monitored via a suitable voltage divider with an operational amplifier buffer. For current monitoring, the LT3571 offers the IMON pin which sources a current $I_{IMON} = 0.2 \cdot I_{APD}$. This current is than converted to voltage with a 1 kΩ shunt resistor and buffered with an operational amplifier.

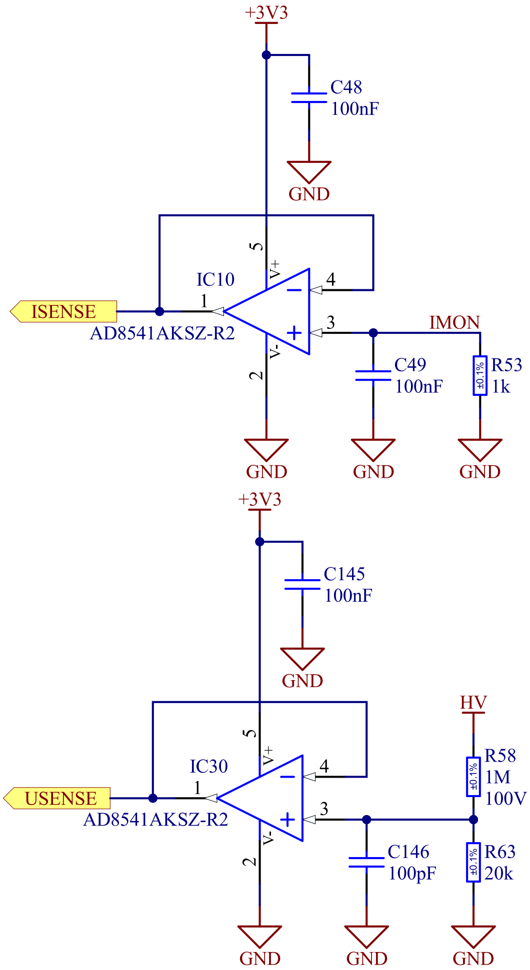


Figure 6.13: Sensing of the high voltage and current

6.3.3 Power

A DC/DC converter has been implemented to efficiently lower the 5 V input voltage to the 3.3 V used by the microcontroller. The 3.3 V for the analog domain has been generated with a low ripple LDO. The 1.8 V domain for the USB interface has been derived from the 3.3 V using an LDO aswell, as efficiency is not required with the low power consumption of 29.4 mA of the 1.8 V domain.

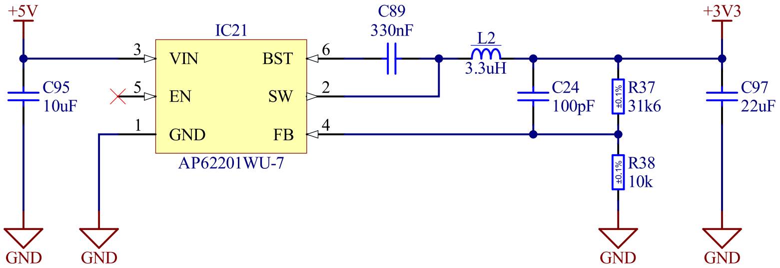


Figure 6.14: Step down regulator schematic

Power sequencing has been implemented for all the three FastIC+ voltage domains. First, the digital domain supply is activated, followed by the supply for the treshold circuitry and PLL and lastly, the analog domain is supplied. All of these domains are derived from the 3.3 V using a 1.2 V LDOs which are sufficient for the low power consumption (113 mA per chip) and provide a ripple-free supply for the sensitive treshold circuitry.

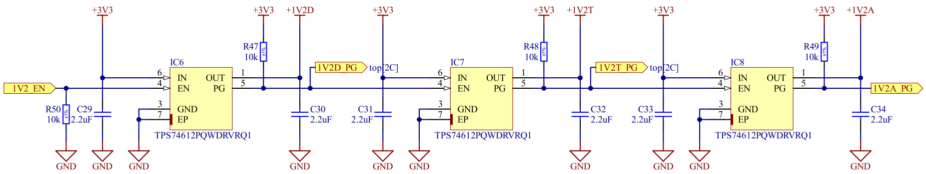


Figure 6.15: FastIC+ power domain sequencing

6.4 Connector

For interfacing with the user board, the 9 mm high version of ERM8-020 has been used for its durability, up to 1000 connection cycles, good high speed performance and suitable pin count. This connector carries all the sixteen input channels alongside the high voltage biasing supply. The 3.3 V supply is also exposed and four pins are dedicated to the user board identification.

Identification pins

The identification pins serve as an easy way to assign a four bit short ID to a specific user board. This ID can than be used by the software to load a configuration preset defined for the user board. If the userboard designer decides to, the pins ID0 and ID1 can be used as I2C communication lines. A compatible EEPROM can than be assembled on the user board, making it possible to save the full configuration on the user board itself, as well as additional data such as name or a unique ID. If the I2C interface is to be used, all of the ID pins need to be pulled up high by a suitable resistor. At least one of the ID pins has to be high at all times, this means that an ID of `0b0000` is not allowed and in this case, the readout will not recognize a valid user board.

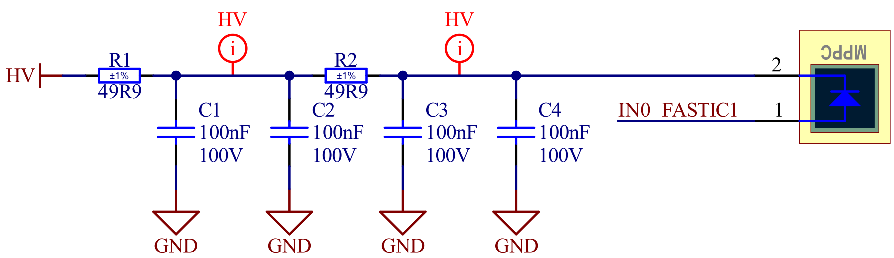
Chapter 7

Userboard

The userboard has been developed as a template for the users to get easily started with the readout system. It contains a matrix of 4 × 4 SiPM sensors as well as an EEPROM for storing the board configuration. The 9 mm high ERF8-020 connector has been chosen for the user board to match the counterpart present on the readout and allow enough clearance between the assembled readout board and user board.

7.1 Sensors

The footprint for a generic THT SiMPs has been implemented on the board. Each SiPM is powered from the HV plane and the input is filtered with a dual RC low pass to eliminate a possibility of cross triggering of the sensors.



7.2 EEPROM

The 24AA04T-I/OT 4 kb EEPROM has been used on the user board for the configuration storage. It has been selected for the low price and sufficient capacity.

Chapter 8

PCB design

Special care had to be taken when designing the PCB to keep in mind the correct design practices for propper power integrity, signal integrity and high speed design.

First a suitable stackup was chosen. In this case, it was decided to proceed with an eight layer stackup as shown in table 8.1.

Table 8.1: PCB stackup

Layer	Name	Thickness
1	L1 - Top copper	0.018 mm
	Dielectric - PR2116	0.120 mm
2	L2 - Ground plane	0.035 mm
	Dielectric - FR4 core	0.200 mm
3	L3 - Power plane	0.035 mm
	Dielectric - PR2116	0.120 mm
	Dielectric - PR2116	0.120 mm
4	L4 - Signal layer	0.035 mm
	Dielectric - FR4 core	0.200 mm
5	L5 - Signal layer	0.035 mm
	Dielectric - PR2116	0.120 mm
	Dielectric - PR2116	0.120 mm
6	L6 - Power plane	0.035 mm
	Dielectric - FR4 core	0.200 mm
7	L7 - Ground plane	0.035 mm
	Dielectric - PR2116	0.120 mm
8	L8 - Bottom copper	0.018 mm

The layers 2 and 3 and layers 6 and 7 were completly dedicated as ground and power planes. This, in combination with relatively thin dielectric in between, creates an inherent capacitive coupling between the planes proportional to the plane area and spacing which helps the power integrity of the device. For a board of size 50 mm × 50 mm, which is the case for the readout, and the stackup mentioned, this capacitance can be estimated with an equation for a parallel plate capacitor to be

$$C = \varepsilon_0 \cdot \varepsilon_r \cdot \frac{S}{d} = 8.85 \times 10^{-12}\,\mathrm{F/m} \cdot 4.2 \cdot \frac{2500\,\mathrm{mm}^2}{0.2\,\mathrm{mm}} = 46.5\,\mathrm{pF} \tag{8.1}$$

Leaving the planes for solid planes only also keeps the impedance of the planes minimal which reduces the voltage drops in the planes. The layers 4 and 5 were also mostly kept as planes, although some connections needed to be routed through these because of the very high board density. While routing in these layers, care was taken not to cross possible return current paths of the planes with another trace, which could result in a much bigger current loop and an increase in EMI. Layers 1 and 8, the top and bottom copper, were used mainly for routing of all the signals. Since both of the layers have a ground reference underneath, both of them can and were used for routing of the high speed lines with defined impedance.

8.1 High speed signals

To mitigate any possible reflections on the high speed lines which carry high frequency signals, the charasteristic impedance of all of these was calculated to match the driver impedance. In case of the differential signals, the differential impedance of the lines was optimized to be $100\,\Omega$, aside from the USB which requires a $90\,\Omega$ differential impedance. For the single ended lines, the target impedance was $50\,\Omega$. The widths of the lines along with the spacing for the differential lines were calculated using the Altium Designer impedance calculation tool and double checked using the manufacturers impedance calculation tool. These calculations yilded the width $W_{USB} = 11\,111\,\mathrm{mm}$ and spacing $S_{USB} = 11\,111\,\mathrm{mm}$ for the USB differential pair, the width $W_{DIFF} = 0.15\,\mathrm{mm}$ and spacing $S_{DIFF} = 0.2\,\mathrm{mm}$ for the rest of the differential pairs and the width $W_{SE} = 0.182\,\mathrm{mm}$ for all of the single-ended high speed lines. The widths and spacings were obeyed during the routing to keep the tracks impedance as close to the target impedance as possible.

8.2 BGA fanout

8.3 Power integrity

The two concepts described above were designed and manufactured keeping in mind the required performance. The final PCBs can be seen on the images bellow.

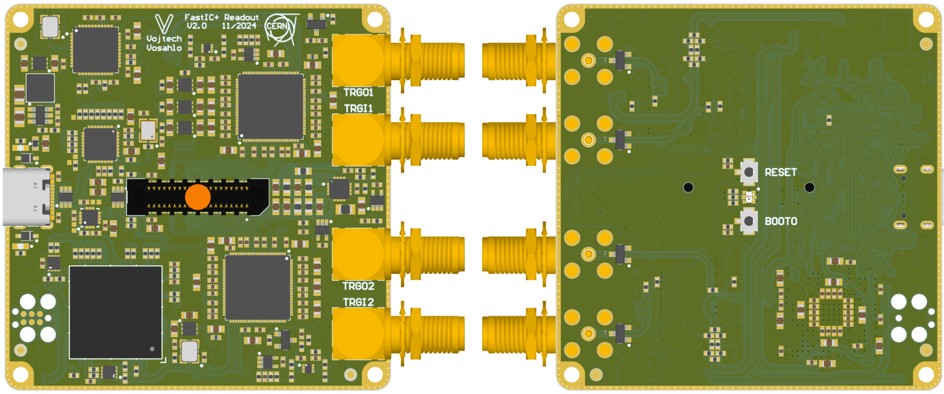


Figure 8.1: Readout PCB

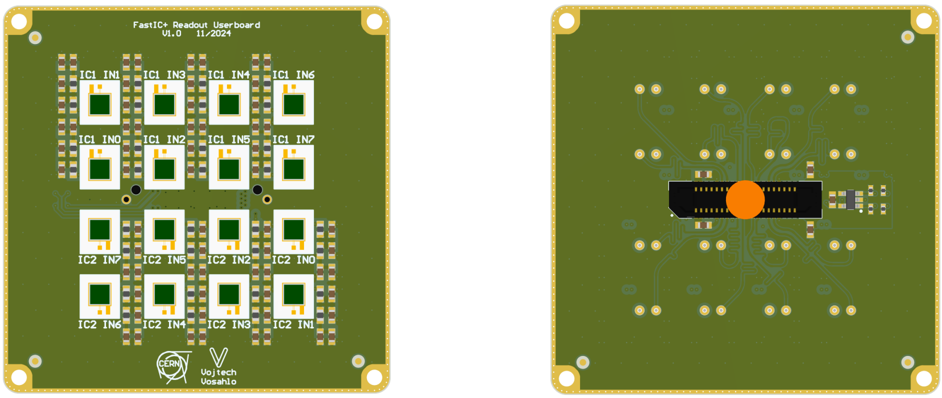


Figure 8.2: Userboard PCB

Chapter 9

Enclosure design

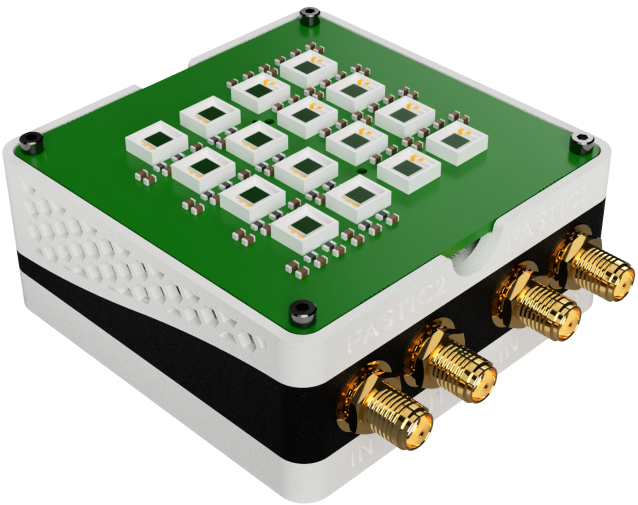


Figure 9.1: Readout PCB

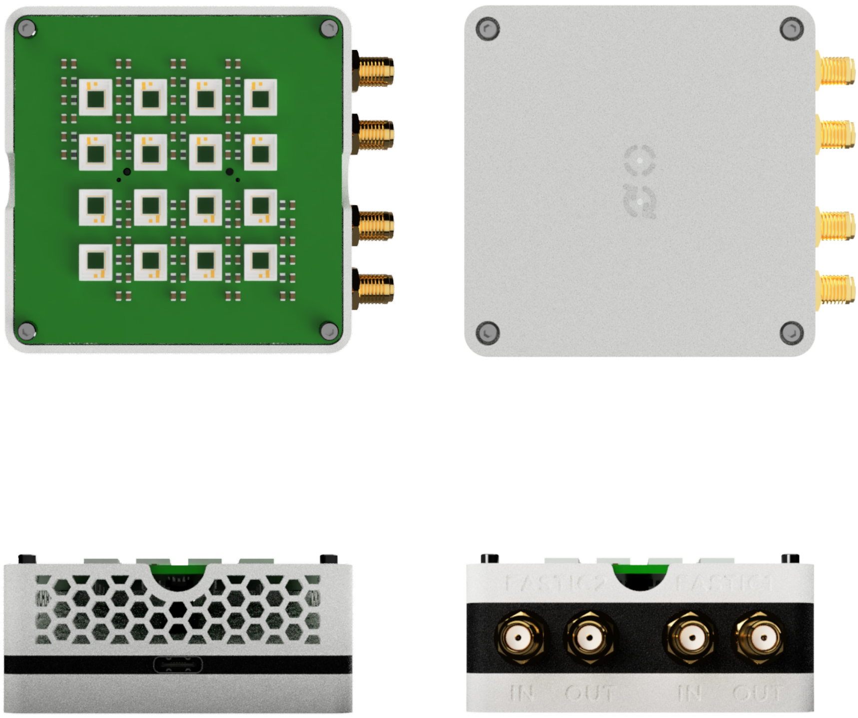


Figure 9.2: Readout PCB

Chapter 10

Firmware

A firware in the C++ programming language has been developped for the readout aiming for best performance and reliable functionality.

10.1 USB

The TinyUSB library has been used for implementing the USB stack on the device. It is an open-source cross-platform USB stack for embedded systems, designed to be memory-safe with no dynamic allocation and thread-safe. It provides support for both Host and Device roles and implements all the common usb classes such as CDC, HID, DFU, Vendor specific and others. On the STM32H7 series specifically, it is capable of working with the ULPI PHY in HS mode and make use of the internal DMA to allow for very high throughput and good latency. All of these specification make it ideal for this application.

Three interfaces have been implemented in order to allow for configuration of the device via human readable protocol, binary protocol and readout of the two data streams.

10.1.1 CDC interface

A Communication Device Class has been implemented on the first interface (endpoints 0x81, 0x82 ans 0x02). This interface emulates a COM port over USB and allows for easy, human-readable interaction with the device. A simple text protocol has been impemented to serve all the required functions of the device.

The following commands have been implemented, where square brackets indicate a choice between the options separated by slash and curly braces indicate value in the specified format:

- `get readout status` - returns the status of the readout
- `get readout uid` - returns a UID of the readout
- `get hv enable` - returns the state of the HV supply

```
set hv enable [true/false]
get hv current
get hv voltage
set hv voltage {float: voltage}
get fastic register [1/2] {hex byte: address}
set fastic register [1/2] {hex byte: address} {hex byte: value}
get fastic voltage [1/2]
get fastic syncreset [1/2]
set fastic syncreset [1/2] [high/low]
set fastic calpulse [1/2] [enable/disable]
get fastic time [1/2]
get fastic aurora [1/2]
set fastic aurora [1/2] [enable/disable]
get userboard status
get userboard uid
get userboard name
set userboard name {string: name}
get userboard writeprotect
set userboard writeprotect [true/false]
get userboard init
set userboard init
get userboard voltage
set userboard voltage {float: voltage}
get userboard register [1/2] {hex byte: address}
set userboard register [1/2] {hex byte: address} {hex byte: value}
set userboard tomemory
set userboard frommemory
```

A detailed description of the commands and their usage is provided in the device user manual.

10.1.2 Vendor control

The USB specification describes a way to communicate with a USB device in a bursty matter by Control Transfers. A control transfer is typically a short random packet, containing up to 64 B of data, which is delivered to the default endpoint with the best effort delievery (no retransmissions). These packets are, for example, used to control the flow of the CDC interface but can be very easilly adopted to transfer auxilary vendor data and thus allow for binary communication with the device.

The same commands as in the CDC interface have been implemented using control transfers to allow for easier interactions with the device via software on the PC as the text communication adds unneccessary overhead in thios regard.

A control transfer is started by an eight byte long Setup Packet, which contains the following fields:

0–7	8–15	16–31
`bmType`	`bRequest`	`wValue`

32–47	48–63
`wIndex`	`wLength`

Figure 10.1: Setup packet structure

`bmType` indicates the direction of the communication, the type, in this case a Vendor transfer, and the recipient, in this case a Device. `bRequest` field indicates the request number. The readout text commands have been each mapped to a unique number which is used in this field in the binary communication. `wValue` and `wIndex` allow for other parameters to be passed with the request, in this case parameters such as the index of the FastIC+ chip to work with. If there is more data to be transfered, the `wLength` field is used to specify the length of an additional data packet sent after the Setup Packet.

10.1.3 Vendor interfaces

For transfering the Aurora data stream from the FastIC+ chips to the computer, two vendor interfaces have been implemented, one for each FastIC+ chip. The sampled bitstream is transfered using Bulk Transfers over these two interfaces.

10.2 Clock generation

The configuration for the Si5340 clock synthetizer was generated using the Clock Builder application provided by Skyworks. This software generates a register map as a C/C++ array that is later parsed by the software and the configuration is applied to the synthetizer over I2C.

10.3 HV power supply

For controlling the HV power supply, a DAC peripheral has been used to provide the control voltage. Two channels of an ADC peripheral with 256 times oversampling, resulting in a 1000 Hz sample rate, were than used to acquire feedback for voltage and current monitoring.

10.3.1 PID controller

A closed control loop was implemented, using the ADC inputs and DAC output called a PID controller. This controller calculates the DAC value in order to minimize the difference between a required output voltage and a feedback from the ADC. On every cycle, which takes place after the ADCs finished sampling, an error value e is calculated, which represents the difference between the setpoint and the measured voltage. The error is than integrated into variable I and a derivation of the measured value is calculated, denoted D. In the end, a corrective output setting is calculated, to compensate for possible error, using the following equation:

$$O = K_P \cdot e + K_I \cdot I + K_D \cdot D \tag{10.1}$$

where O is the output value and K_P, K_I and K_D are the progressive, integral and derivative constants respectively.

The afformentioned constants can either be determined analytically after modelling the control loops transfer function or they can be determined heuristically. A heuristical method, namely the Ziegler-Nichols method was used in this case.

The tuning of the constants begins with setting K_I and K_D both equal to zero. K_P is than increased until the output of the PID controller starts to oscillate consistantly. The value of K_P at this point is called the ultimate gain K_U and the period of oscillations is called T_U. Based on the acquired constants, the values of K_P, K_I and K_D are than set.

For a classic PID, the values of the constants should be set to $K_P = 0.6 \cdot K_U$, $K_I = 1.2 \cdot \frac{K_U}{T_U}$ and $K_D = 0.075 \cdot K_U \cdot T_U$

10.4 FastIC+

Both of the FastIC+ chips are mainly controlled over the I2C interface. Only a very basic configuration is done by the software, setting the propper clock dividers to be able to receive the Aurora stream. Rest of the settings are left for the user to modify, as the registers of the FastIC+ can be accessed directly over the communication interface and most of them need to be tuned for a specific application of the readout.

10.4.1 Aurora stream

The Aurora stream is continuously sampled by an SPI interface on the rising edge of the sampling clock. The SPI interface buffers the received bytes in its internal FIFO until a DMA peripheral, configured in double-buffer circular mode, transfers the data to a buffer. In the double buffer mode, the programmer provides the DMA wit two separate buffers. The DMA is than configured to switch back and forth between the buffers every time one of them is full. This allows the rest of the code to transfer data from one of the buffers while the other one is being filled, not causing any bit drops.

10.4.2 Pulse injection

The pulse injection circuit is fed by a fast timer output. The width of the timer pulse directly influences the width and amplitude of the generated pulse which is used for testing the frontends. The size of this pulse has been fixed and the pulses are generated with a period of 100 Hz.

10.5 Userboard

The userboard detection is achieved with the four exposed GPIO pins. When a command, such as `get userboard status` tries to access the userboard, the short ID is first obtained. If the short ID equals `0b1111`, the pins are switched into a I2C mode and the microcontroller checks if a configuration header is present in the EEPROM.

The following structure of a configuration header has been implemented:

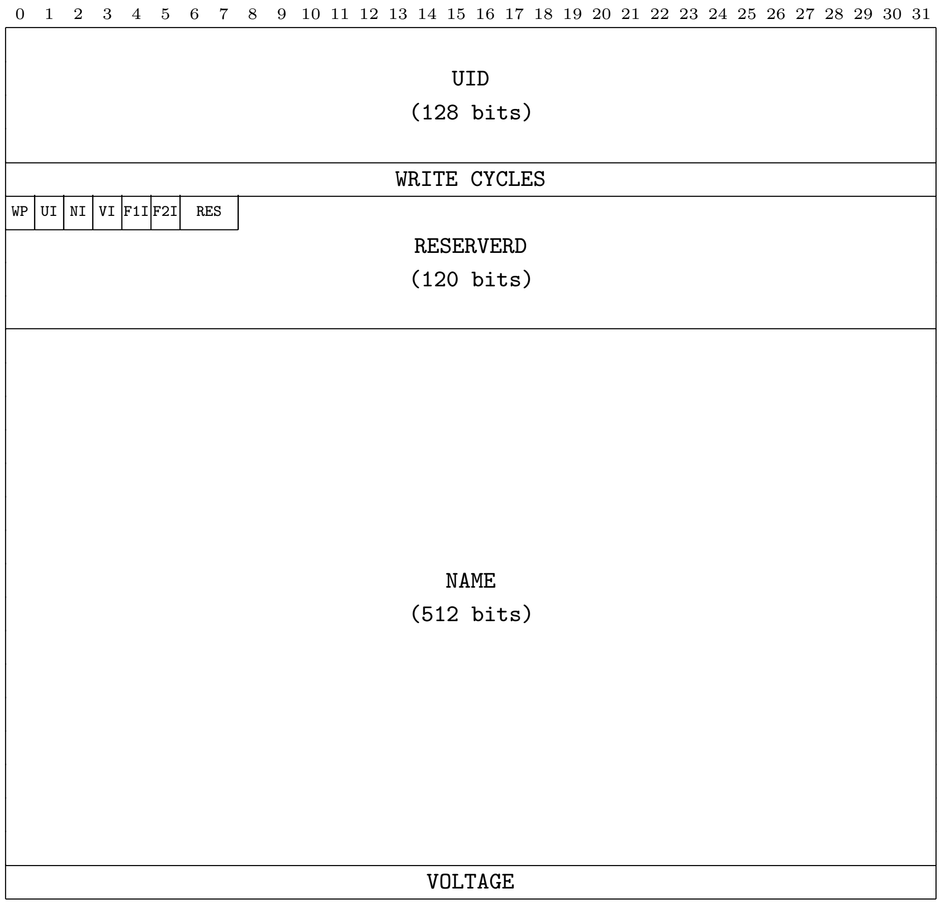


Figure 10.2: EEPROM configuration header structure

where `UID` is a unique ID generated on first initialization of the userboard, `WRITE CYCLES` stores the number of times that the EEPROM has been written. `WP` is a write protect bit and the `UI`, `NI`, `VI`, `F1I`, `F2I` signal if the UID, name, voltage value, FastIC+ 1 and FastIC+ 2 registers have been initialized respectively. `NAME` contains up to 64 character user name and `VOLTAGE` is a floating point value that stores the high voltage preset of a given userboard.

Chapter 11

Software

11.1 Device control

11.2 Stream reception

11.3 Packet parsing

Chapter 12

Measurements

12.1 Power supplies

12.2 ADC precision

12.3 High speed signals

12.4 High Voltage power supply

The output voltage ripple of the high voltage power supply was measured at two voltage settings. The transient response to a load of 100 kΩ was also measured. The performance of the PID regulator was also evaluated.

12.5 Detection emulation test

12.5.1 Pulse injection

12.5.2 Data analysis

12.5.3 Results

Bibliography

[1] *Aurora 64B/66B Protocol Specification*. `https://docs.xilinx.com/v/u/en-US/aurora_64b66b_protocol_spec_sp011`. 2014.