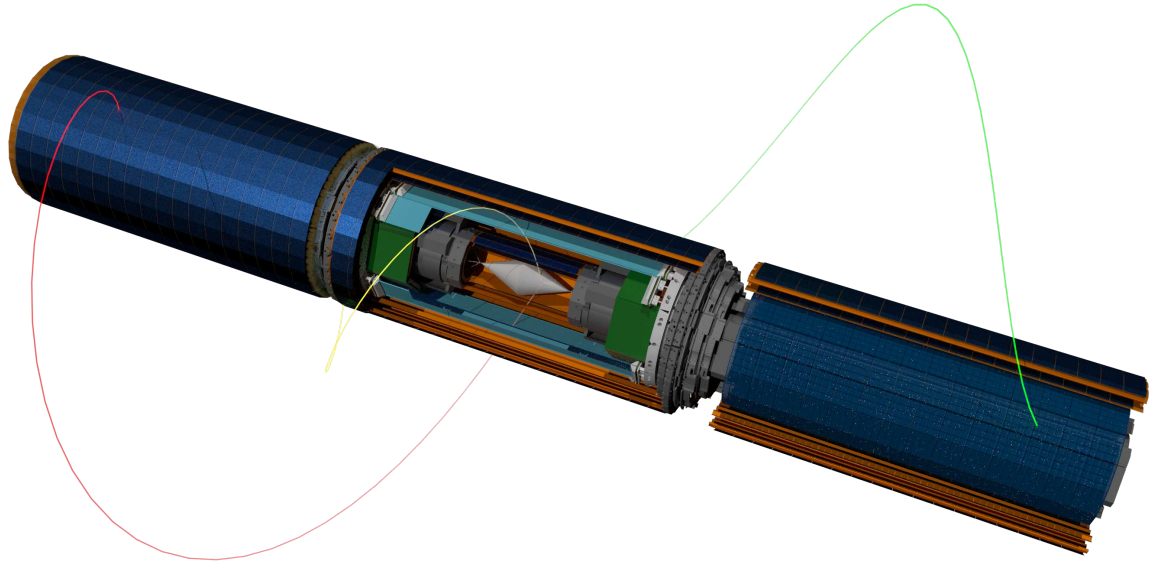
The build system for the *Mu3e* DAQ firmware

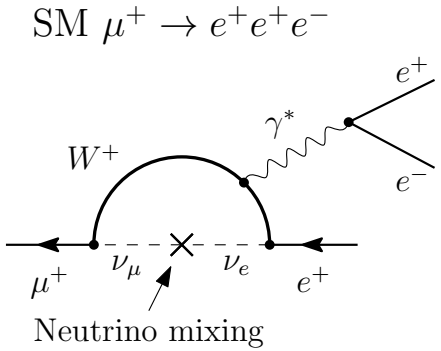
Alexandr Kozlinskiy, Mainz



Mu3e Experiment

Search for charged Lepton Flavor Violation

- Through a decay $\mu^+ \to e^+e^+e^-$
- Allowed in the Standard Model
  but not observable (Br $< 10^{-54}$)
- *Any observation will point to New Physics*



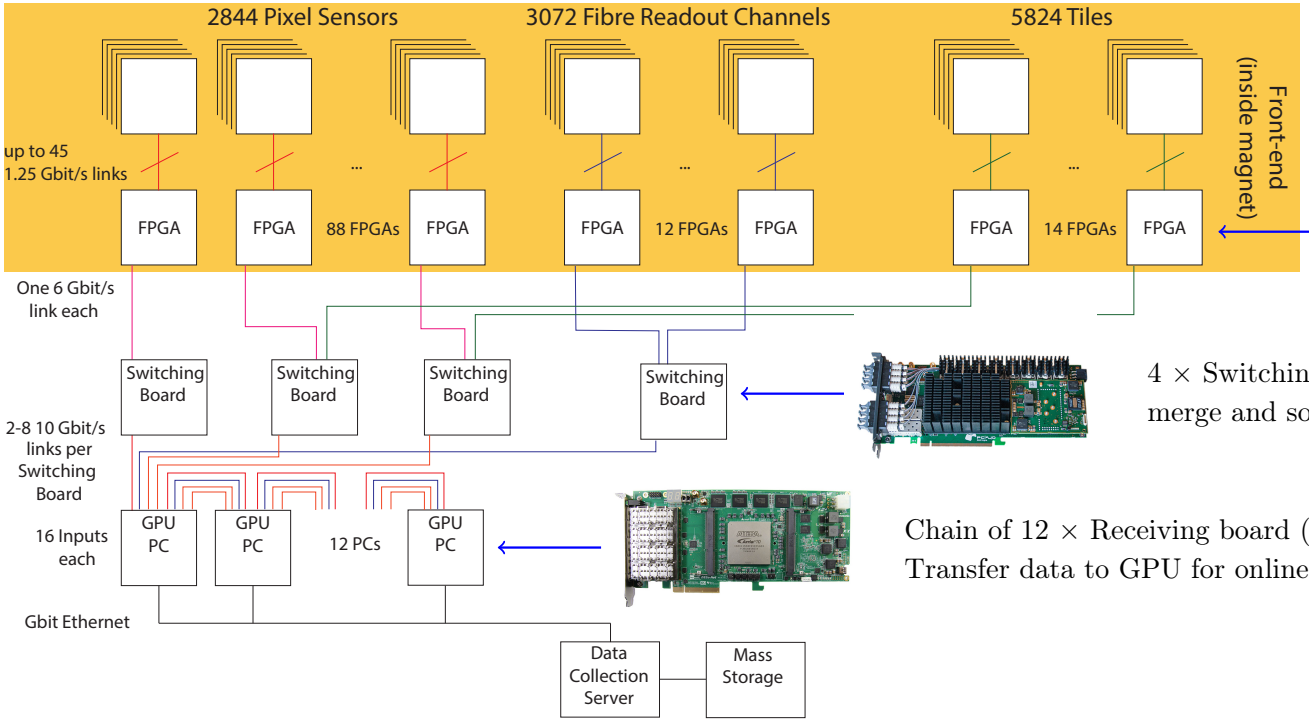
Current experimental status:

- SINDRUM (1988) *Nucl.Phys.B299(1988)1*
- Br $< 10^{-12}$ at 90% C.L.

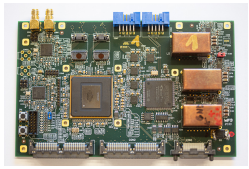
Mu3e aims for sensitivity of $2 \cdot 10^{-15}$ (Phase 1)

- Under construction at Paul Scherrer Institute
  using existing beam line (πE5, $10^8 \mu$/s)
- Further improve to 10^{-16} (Phase 2) with new
  beam line (HIMB)

Readout



114 × Front-End Board (Arria V)
readout for pixels, fibres and tiles
lvds to optical links



4 × Switching board (LHCb PCIe40, Arria 10)
merge and sort data streams

Chain of 12 × Receiving board (DE5a NET, Arria 10)
Transfer data to GPU for online reconstruction

- Intel FPGAs: two main (Arria V and Arria 10), one auxilary (MAX 10)
- Three boards (FEB, SWB/PCIe40 and Farm/DE5a)
- Several firmware configurations (mupix/scifi/scitile, DDR3/4, test stands)

Overview

- Several firmwares base on Intel FPGAs
- Many developers and users in different locations
  (MZ, HD, ZH, PSI and UK)
- Development in Git (10k commits in 5 years)
  (want to avoid generated files to keep repository manageable)
- Use CI (Continuous Integration) to track status of firmwares
  (build, simulation)

Overview

Firmware compiled with Quartus Prime software:

- Different Quartus versions
- Different Intelectual Property (IP) blocks in different FPGAs (RAM, transceivers, NIOS, etc.)
- Most actions are done via GUI and are not suitable for CI
- Typycal workflow requires to keep generated IP files in a project (e.g. 1000 files for FEB firmware)

Solution:

- Command line build system:
  - Make target for each part of the build process
  - Scripts to assemble NIOS and other IPs
    (no need to keep generated files)
  - Simulation scripts for GHDL and Modelsim
- This system allow to perform all steps (build, etc.) on each commit in CI

IP Components

IPs:

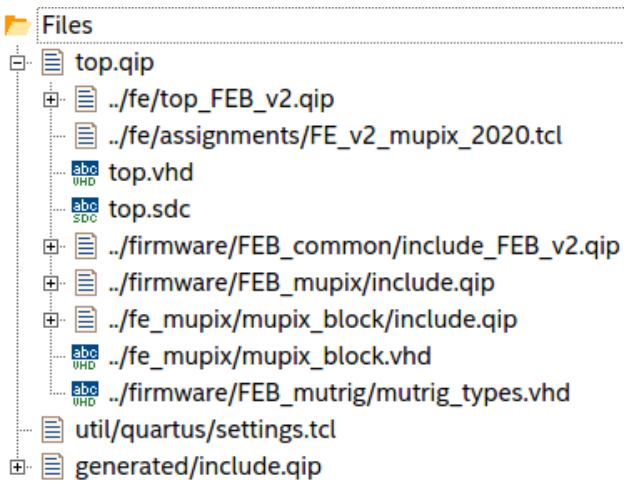
- LVDS - connection to pixel sensors
- Optical transceivers - communication between boards
- NIOS CPU - initialization of on board components, monitoring during development

Problem:

- Different FPGA and sub-projects may need different configuration or tools (Quartus System Integration QSYS or Quartus Mega Wizard)
- Only specific parameters need adjustment (difficult to keep track within GUI)
- Quartus produces alot of artifacts (thousands of vhdl/verilog and other files)

IP Components

- Use Quartus TCL scripts to generate and configure components (LVDS 1250 Mbps, XCVR 10 Gbps and 6250 Mbps, etc.)
- Use QIP files to include generated files and to create project structure



Soft CPU - NIOS

- NIOS provides set of components (I2C, SPI, RAM interface, etc.) that can be controlled from software (C/C++)
- Software uploaded from internal RAM or externaly via JTAG
- NIOS can be configured via QSYS GUI, but difficult to track configuration for sub-projects

⊞ cpu	Nios II Processor		clk	0x7000_0000
⊞ ram	On-Chip Memory (RAM or ROM) I...		clk	0x1000_0000
⊞ sysid	System ID Peripheral Intel FPGA IP		clk	0x700f_0000
⊞ jtag_uart	JTAG UART Intel FPGA IP		clk	0x700f_0010
⊞ timer	Interval Timer Intel FPGA IP		clk	0x700f_0100
⊞ timer_ts	Interval Timer Intel FPGA IP		clk	0x700f_0140
⊞ i2c	Avalon I2C (Master) Intel FPGA IP		clk	0x700f_0200
⊞ spi	SPI (3 Wire Serial) Intel FPGA IP		clk	0x700f_0240
⊞ pio	PIO (Parallel I/O) Intel FPGA IP		clk	0x700f_0280
⊞ i2c_mask	PIO (Parallel I/O) Intel FPGA IP		clk	0x700f_02a0
⊞ clk_156	Clock Source		*exported*	
⊞ clk_125	Clock Source		*exported*	
⊞ irq_bridge	IRQ Bridge		clk_156	IRQ 0
⊞ avm_mscb	avalon_proxy		clk_156	0x7003_0000
⊞ avm_sc	avalon_proxy		clk_156	0x7008_0000
⊞ avm_qsfp	avalon_proxy		clk_156	0x7001_0000
⊞ avm_pod	avalon_proxy		clk_125	0x7002_0000
⊞ spi_si	SPI (3 Wire Serial) Intel FPGA IP		clk	0x700f_0260

Soft CPU - NIOS

- Use base TCL script with common configuration for all sub-projects
- Can specify amount of memory, additional RAM interfaces, etc.
- Mostly same component interface, also generated via command line

```
# base configuration (common for all sub-projects)
source {util/nios_base.tcl}
# configure total RAM size
set_instance_parameter_value ram {memorySize} {0x00010000}
# configure SPI lines count
set_instance_parameter_value spi numberOfSlaves 16
# SPI lines for FEB SI clock chips
source {fe/nios_spi_si.tcl}
# A10 external flash configuration
source {util/flash1616.tcl}
```

Hardware compilation - Flow

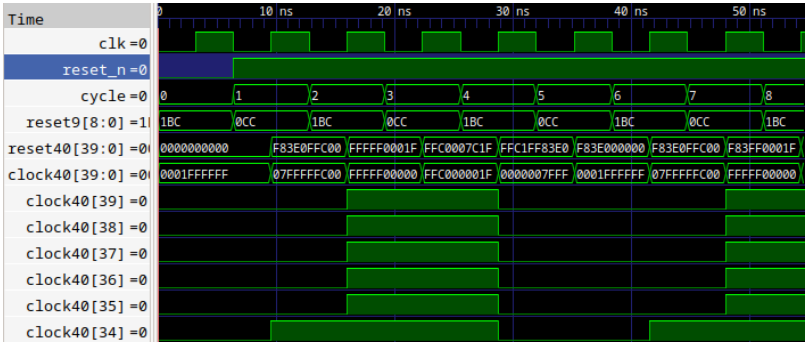
- Typical steps for firmware compilation:
  - Generate IPs via QSYS and qmegawizard
  - Compile NIOS software (BSP and application)
  - Generate components and QIP files
  - Compile firmware (analysys, synthesis, assembler)
- Set of MAKE targets for each steps with proper dependencies (starting from TOP file down to all generated and source files)
- Supports Quartus version from 18.0 to 22.1

```
# init project and generate IPs
make top.qpf
# NIOS software
make app
# flow
make flow
```

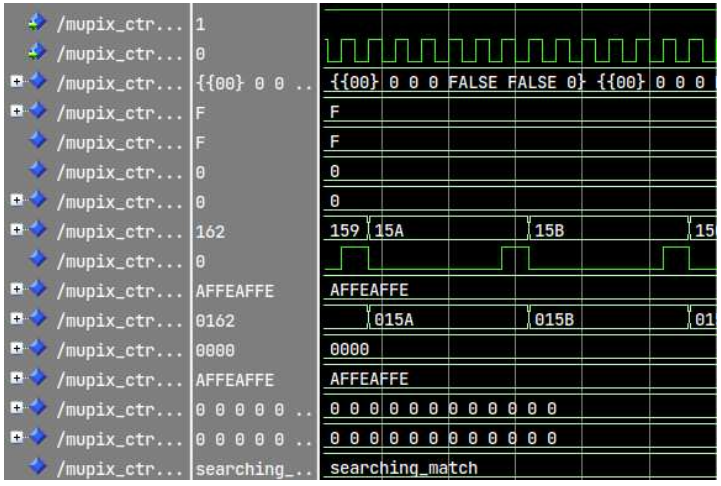
Hardware Simulation

- GHDL and Modelsim for simulation with and without GUI
- Wrappers for both to perform simulation with the same options (only need to specify source VHDL/Verilog files and testbench parameters/generics)

GHDL - reset/clock-system gen:
(via `./sim.sh *.vhd`)



Modelsim - mupix TDAC config:
(via `./vsim.do *.vhd`)



Continuous Integration

- The ability to run wihtout GUI allows to use common CI setups
  and perform build after each commit
  (limitted by firmware compilation time - up to one hour per sub-project)
- CI pipelines in Jenkins
  - Compile firmware for most sub-projects
  - Run specific simulations (mupix TDAC config)
  - Export status to Bitbucket
- Plan to also use Bitbucket pipelines
  (which is used by Mu3e software)

Continuous Integration

Jenkins pipeline for mupix:

	Declarative: Checkout SCM	notify INPROGRESS	Firmware compile	NIOS Software compile	Declarative: Post Actions
Average stage times: (Average full run time: ~36min 33s)	851ms	839ms	22min 42s	4s	836ms
#546 Feb 28 13:37 — 7 commits	853ms	844ms	26min 37s	6s	847ms
#545 Feb 27 15:37 — 2 commits	861ms	843ms	23min 41s	6s	837ms
#544 Feb 26 21:37 — 30 commits	850ms	844ms	24min 22s	7s	792ms
#543 Feb 26 19:37 — No Changes	821ms	846ms	26min 11s	300ms failed	841ms

Bitbucket status (per commit):

Summary

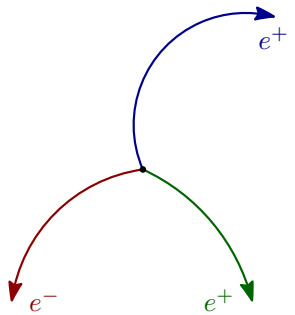
- All sub-projects use same structure and build scripts that allow to compile firmware and software on all versions of Quartus Prime software (which seem to be very unusual, especially in industry)
- Mostly transparent use of IPs
  (plan to also adapt to Vivado, e.g. for P2 experiment in Mainz)
- Possible to build on CI build machines 'without' any GUI requirement with easy fail/success monitoring
- More tools are still in development
  (e.g. show only necessary information in log files)
- All scripts are available at
  `https://github.com/akozlins/vhdutil/util/quartus/`

Backup

Signal

Signal ($\mu \to 3e$):

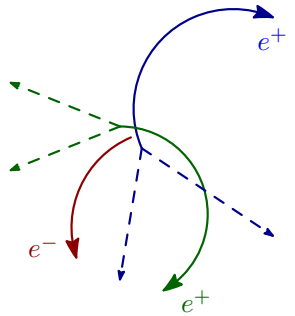
- Decay at rest to two positrons and one electron
  - Common vertex & time
  - Invariant mass: $M_{e^+e^+e^-} = m_\mu$
  - Total (missing) momentum: $\sum \mathbf{p}_e = 0$
- Require good momentum, vertex and time resolution
- Tracks with maximum momentum of 53 MeV/c
  - Large Multiple Scattering (MS) $\to$ minimize material budget



Background sources

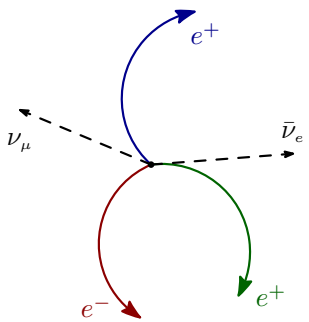
1st - random combinations:

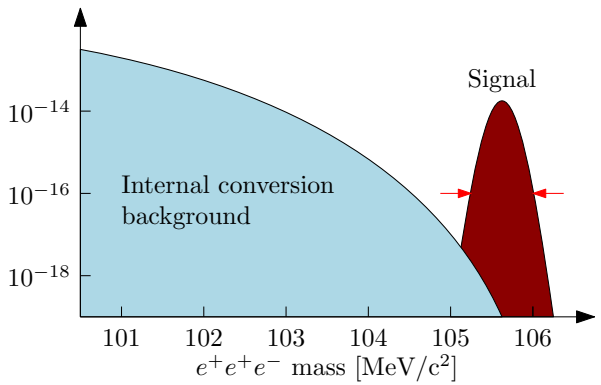
- Overlap of several $\mu^+ \to e^+ + 2\nu$ and/or $e^\pm$ scattering
- Contribution from *fake* tracks
- Signature: not same vertex, time, etc.



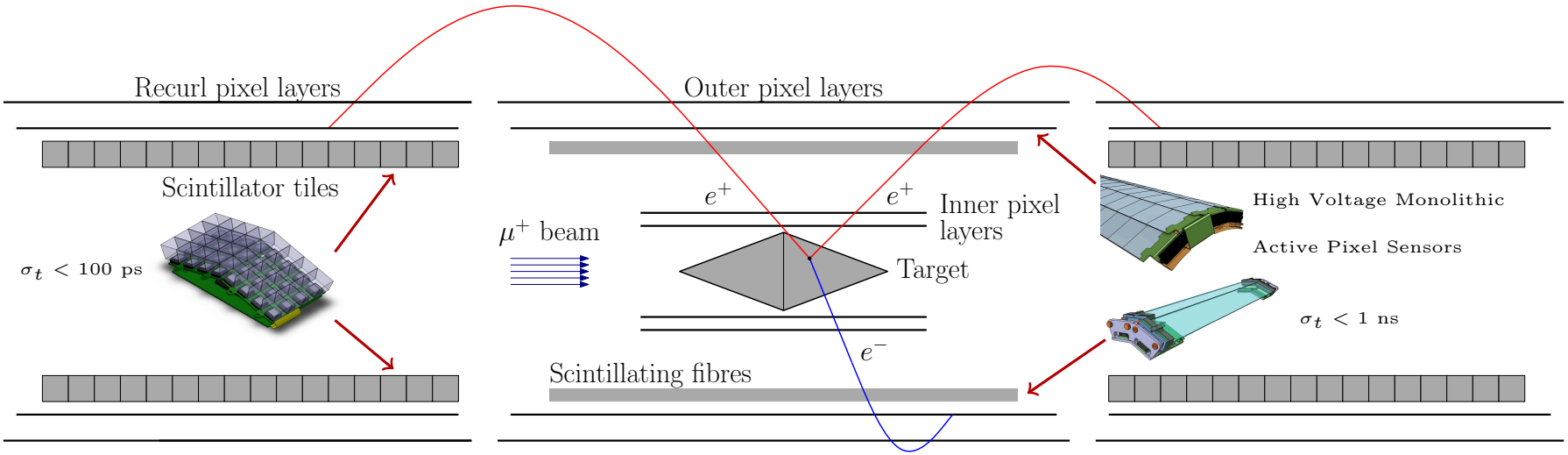
2nd - internal conversion:

- $\mu^+ \to e^+e^+e^- + 2\nu$
- Signature: missing momentum & energy





Detector



- Muons stop on target and decay at rest
- 4 pixel layers provide hits for track reconstruction
- Two recurl stations to improve acceptance
- SciTile and SciFi for timing