Evaluation of a Feature Extraction Framework for FPGA Firmware Generation During a Beam-test at CERN-SPS

Outline:

- Introduction
- The Feature Extraction Framework
- Overview CERN-SPS TRD Beam-test
- Closing Notes

Presented by Carlos Muñoz Castillo

DPG Frühjahrstagung 2016 HK 29.4 2016.03.15

SPONSORED BY THE

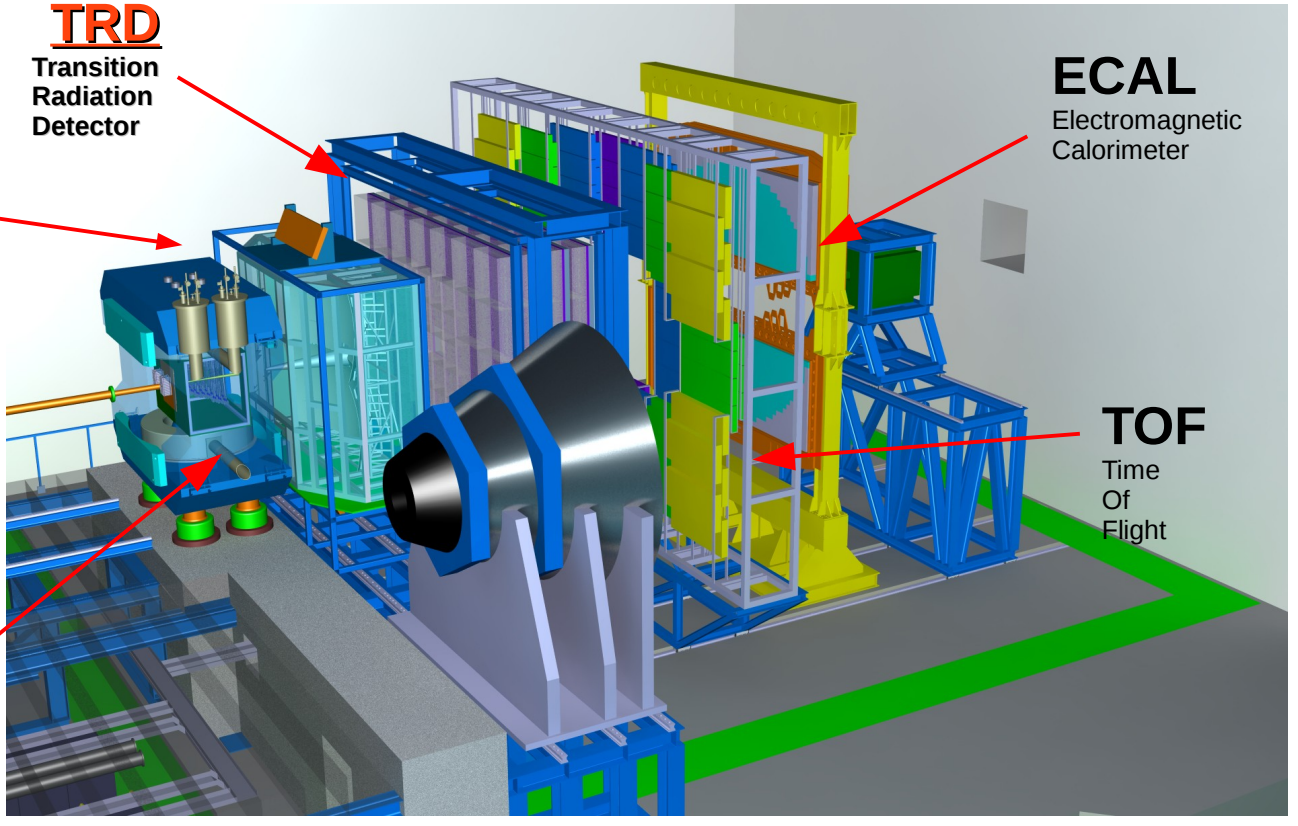
Federal Ministry of Education and Research

05P12RFFCM

Cruz de Jesus García Chávez
garcia@iri.uni-frankfurt.de

IRI - Goethe University
Frankfurt am Main
Prof. Dr. Udo Kebschull

The CBM Experiment



CBM:

- GSI, Darmstadt

TRD Tasks:

- Electron ID
- Tracking of charged particles

The CBM experiment requirements:

- Pion suppression ~100
- Position resolution <300um
- High particles rates (up to > 100 kHz/cm2)
- Data rate ~ 1TB/s

Source www.gsi.de

The CBM DAQ Architecture

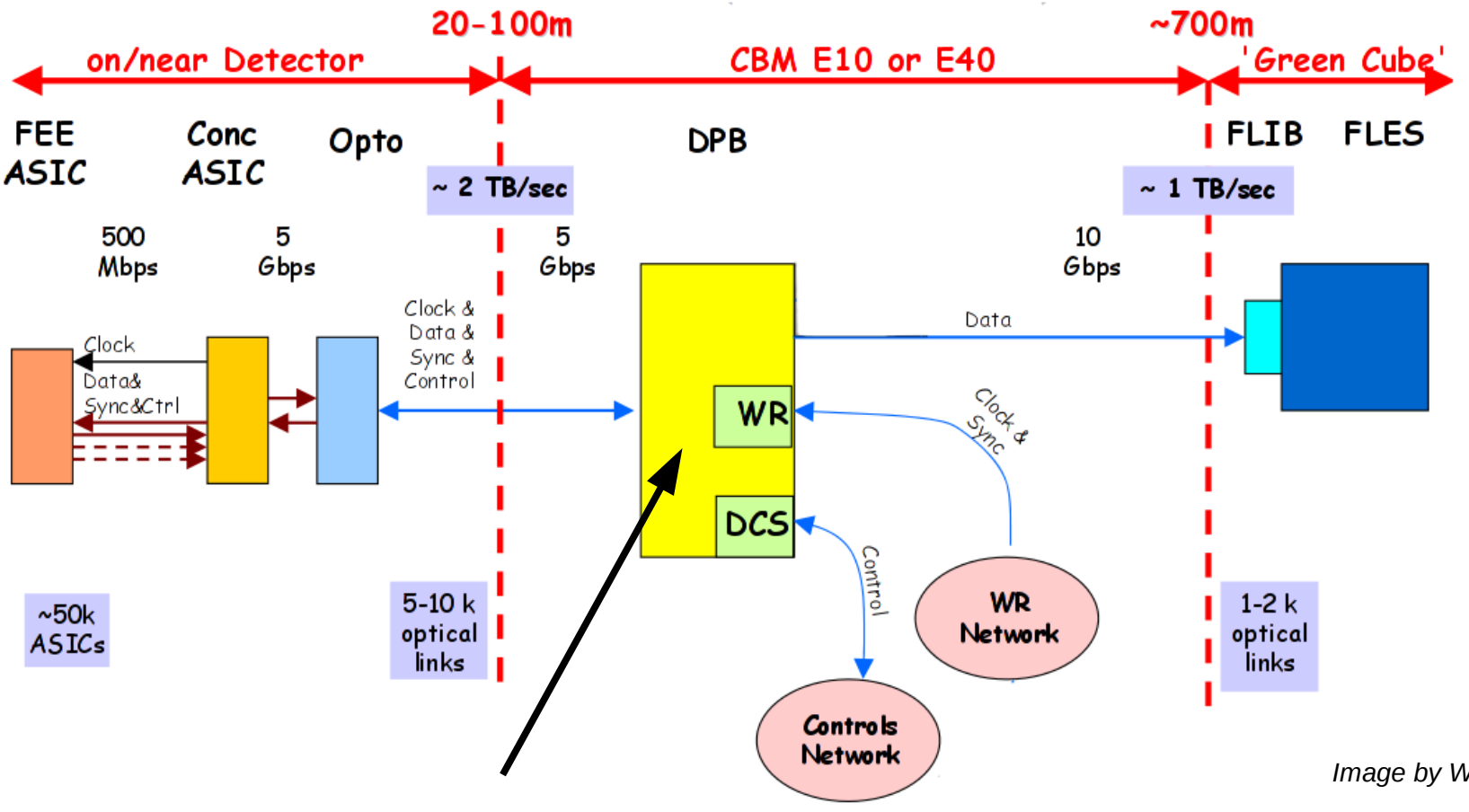


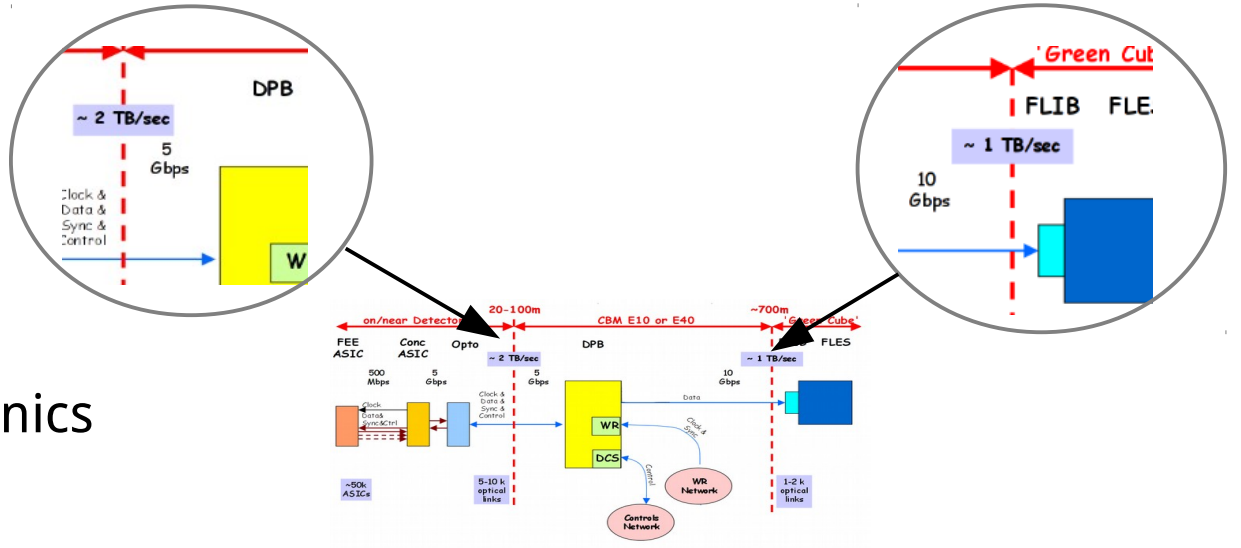
Image by Walter Müller

Data Processing Board (DPB):

- Data concentrator
- Performs data pre-processing
- Feature Extraction
- Sorting

Why Feature Extraction?

→ Total foreseen data rate input to FLES ~ 1TB/s



→ Detector front-end electronics would generate **~ 2TB/s**

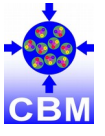
→ *Data reduction is mandatory!*

→ *Extract most relevant features, discard others*

→ Required extraction algorithms are highly parallelizable

→ *Good for FPGAs*

The Feature Extraction Framework



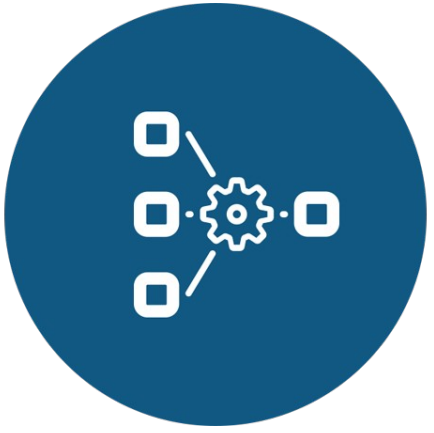
Feature Extraction: Problem to Solve

- Feature extraction is a common problem in the DAQ at high-energy particle experiments.
- Processing algorithms tend to be common among experiments.
- Development time take some years (~1 - 3).
- High level of expertise for firmware development

  → VHDL/Verilog are time consuming!

- New alternatives are still *not performance enough!*
  - → HLS Vivado
  - → Matlab
  - → Simulink
  - → System-C

The Framework Concept



Stream-processing paradigm

→ Exploits parallel processing
→ Stream : sequence of data
→ Kernel function : set of operations

User interaction

→ Domain specific language
→ High-level encapsulation of:
- *Processing Stream Networks*
- *Kernel Functions*

→ No low-level coding knowledge needed

Automatic code generation

→ Optimized VHDL source code
→ Synthesis project for 3rd party applications *(Xilinx)*

Framework's Tool-Flow

Parser:

→ Compiles the DSL file

→ Populates a Semantic Model

FW Synthesizer:

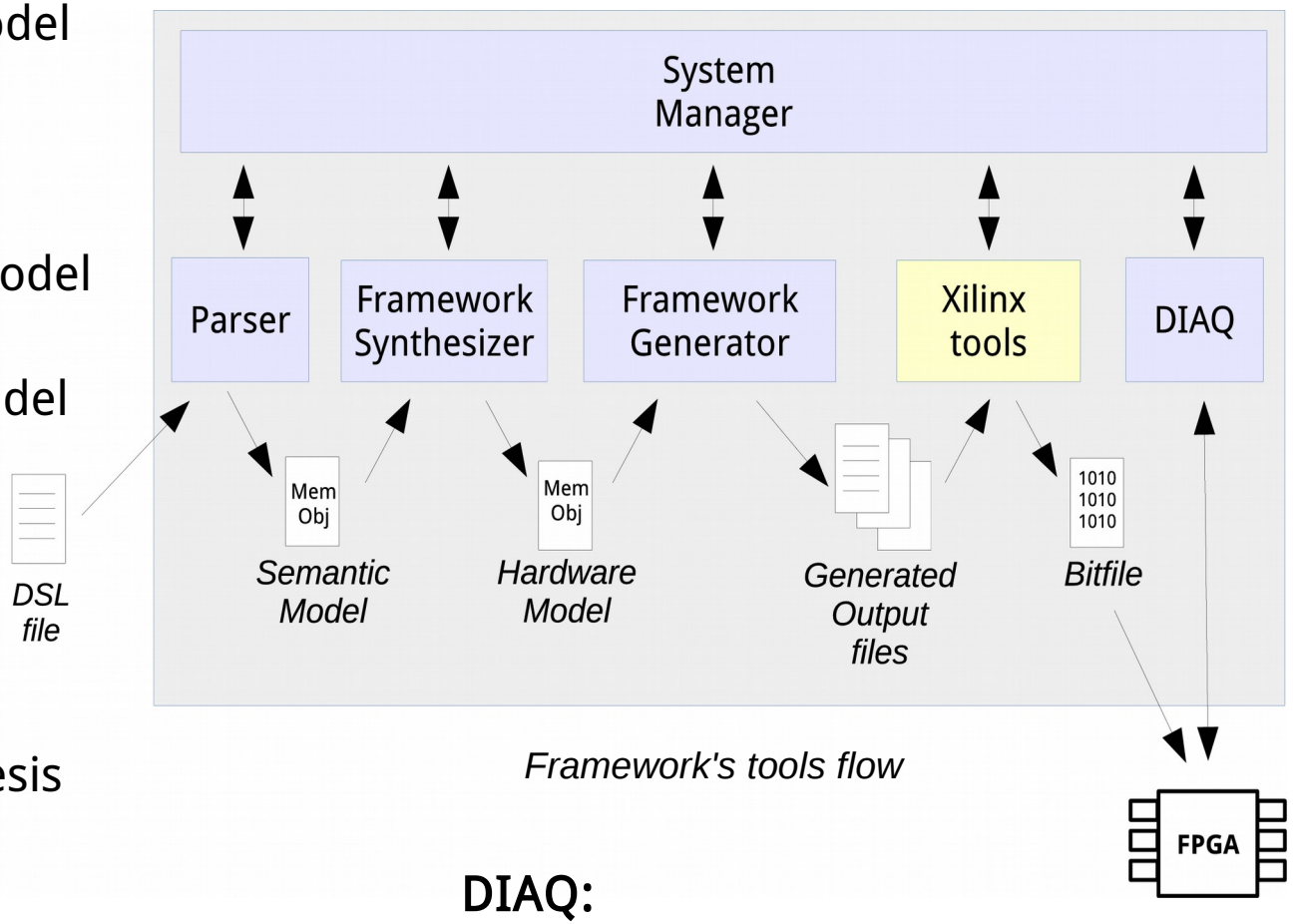
→ Generates a Hardware Model Description

→ Hardware dependent model

FW Generator:

→ Generates:

- VHDL sources
- Project files for synthesis



Framework's tools flow

System Manager:

→ Orchestrates framework's internal processes

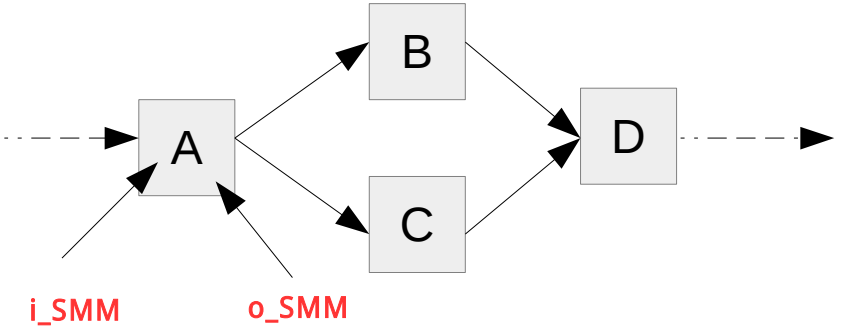
DIAQ:

→ Data Injection and Acquisition

Stream Message Modeling – Representation

- **Stream Message Model (SMM)**

  → Model that defines:
  - Name
  - Identifier
  - Type (Metadata/Payload)
  - Payload type



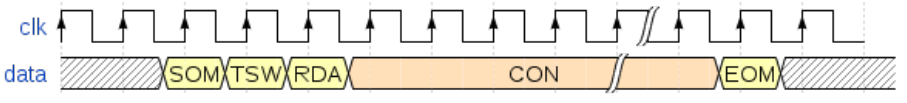
  - For every Node, the SMM is analyzed by the framework

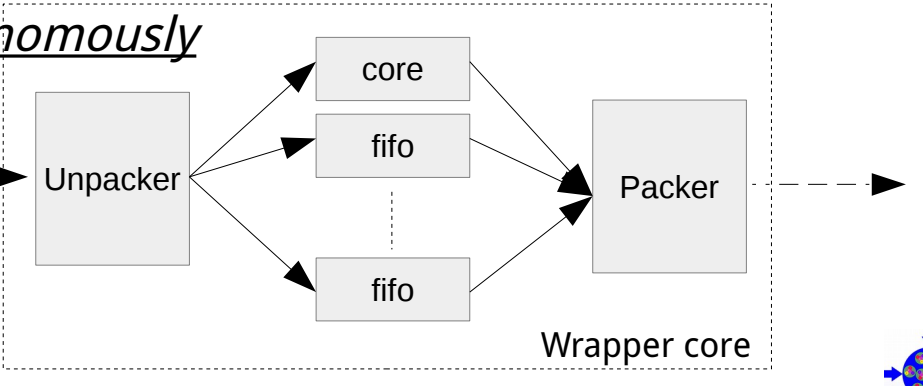
- **Stream word decoding**

  → Node operations can be done on any stream message word (metadata / payload)

  → The Framework creates a wrapper for a given core where the message decoding is implemented

  → This is generated by the framework *autonomously*



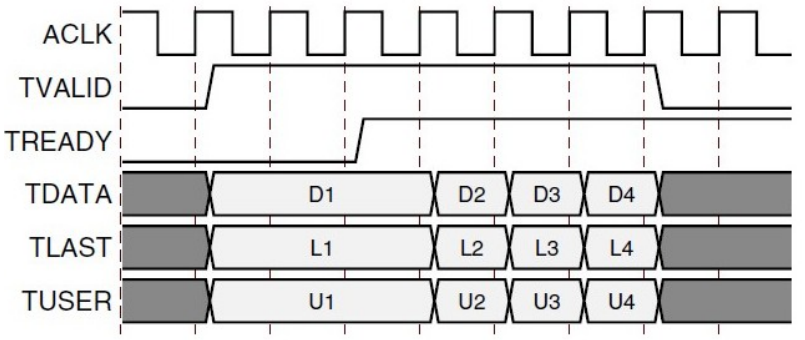


Stream Message Modeling - Transport

- **Stream message protocol: AXI4 Streamer**

  → TDATA channel: Stream message transmission
  → TUSER channel: Token transmission

  → Every clock cycle a stream-word is read

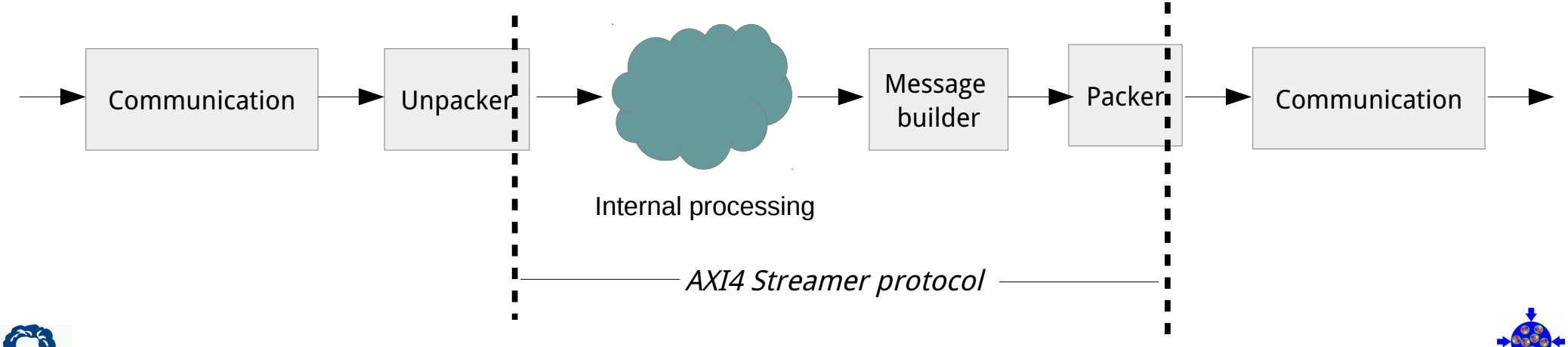


AXI4 Streamer transmission example

"*Outside world*" interface:
(Interfacing to other modules or off-chip)

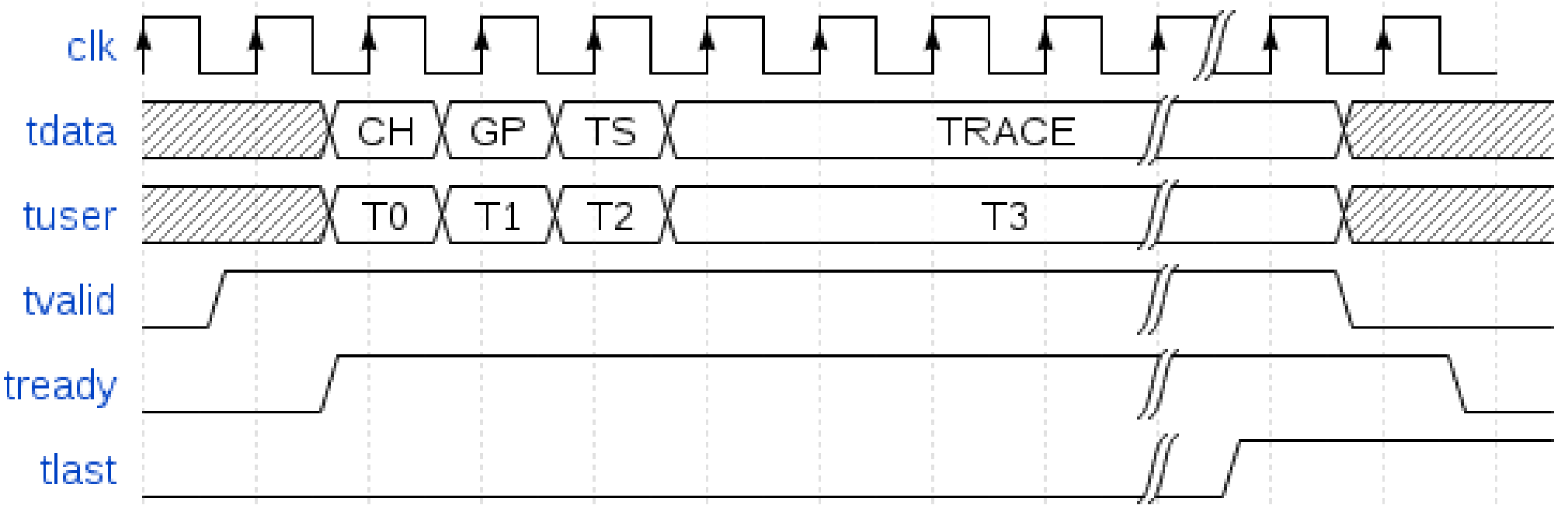
→ Unpackers for most used communication protocols are provided by the framework

→ For new protocols, an unpacker must be created by the designer



Stream Message Modeling – Transport

Transmission Example



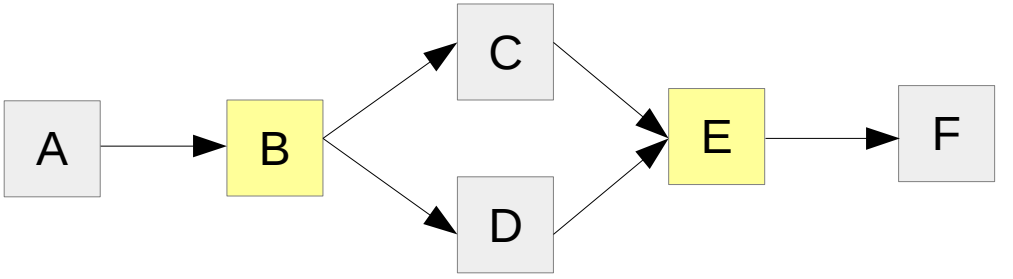
tdata : Stream Message
- CH : Channel
- GP : Group
- TS : Timestamp
- TRACE : Timebin

tuser : Word Token
- T0 : Token for channel
- T1 : Token for group
- T2 : Token for timestamp
- T3 : Token for trace

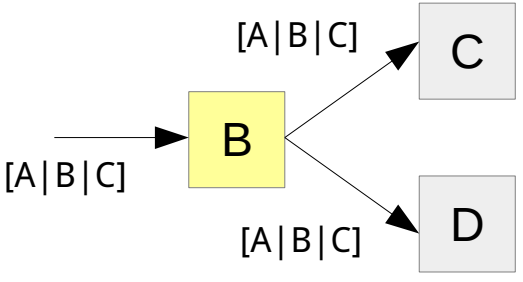
Stream Message Handling

Splitting and merging cores

→ Automatically generated by the framework



Stream Network
Yellow nodes Split and merge nodes



Stream Network
Yellow node: split node

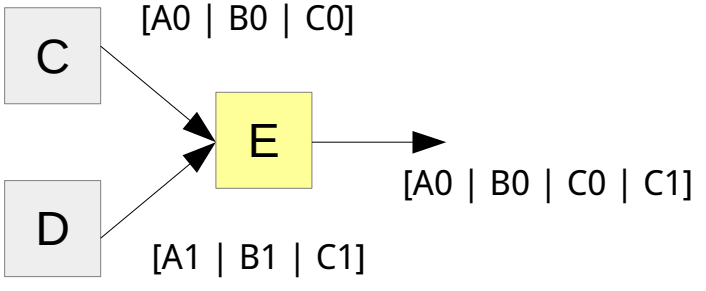
Stream splitter:

→ Broadcast an input stream message to multiple channels

Stream merger:

→ Merges input streams into a single message

Where A , B => Metadata & C => Payload



Stream Network
Yellow node: merge node

Framework's Web User Interface

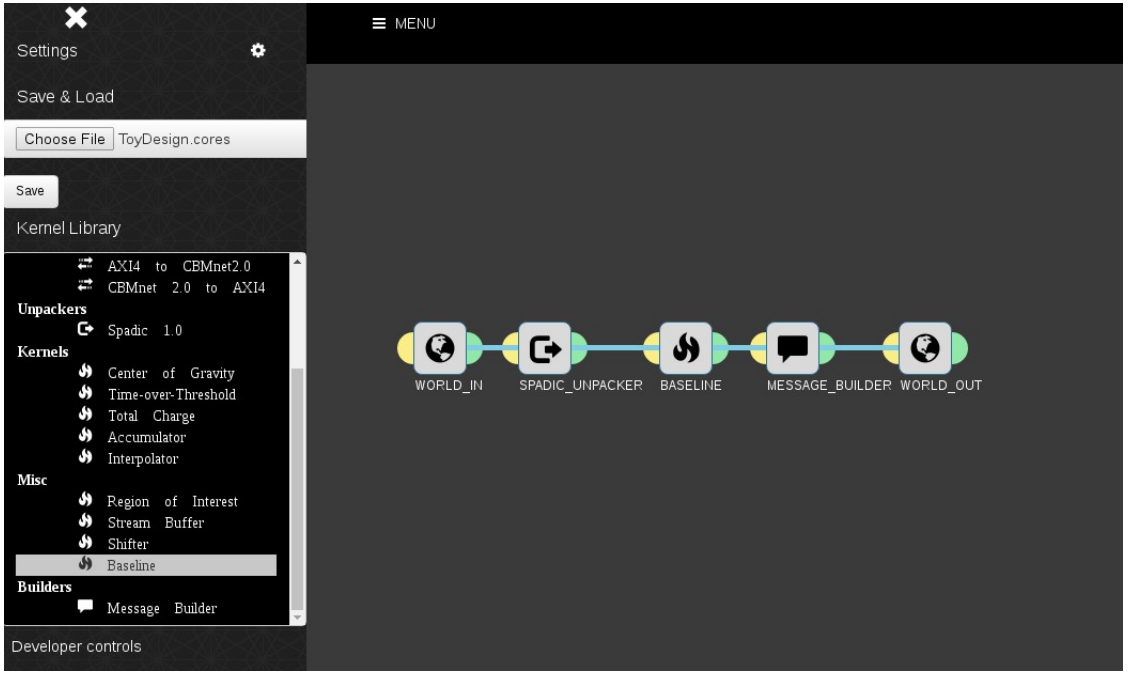
→ Built on top of the framework Application

→ Web Service

→ Visual application

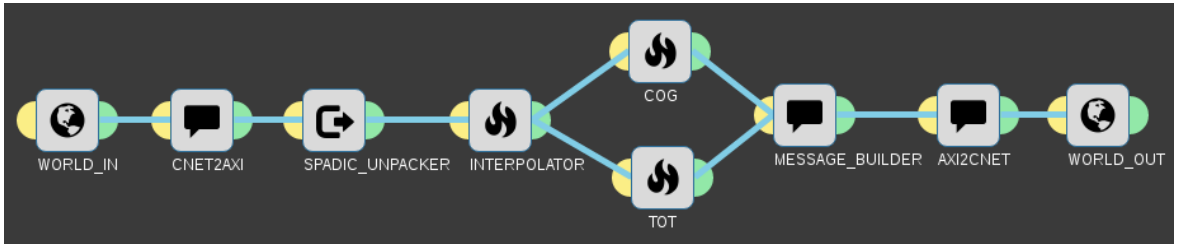
- *No need to write code (DSL)*

→ Back-end generates the DSL file



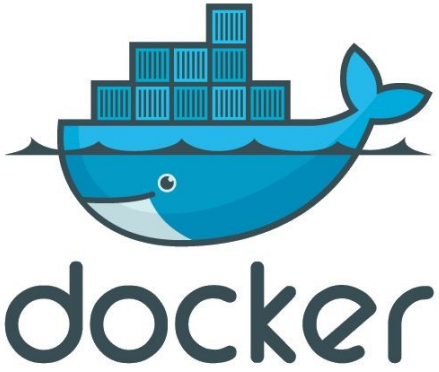
WebApp Snapshot

→ Stream Network is made by interconnection of nodes



Stream Network Example

Framework's Web Application - Virtualization



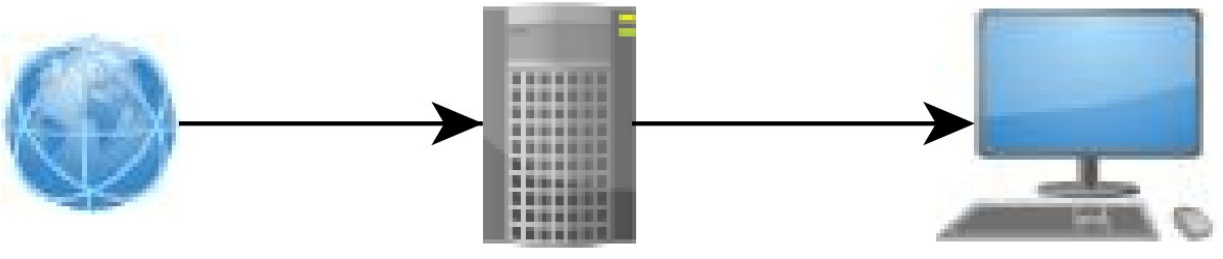
→ **Data-base for users and projects**

- Only registered users can access the service
- Projects can be saved/loaded on the server infrastructure

→ **Web service runs powered by Go and Docker**

- Increased security => Isolation
- Dedicated synthesis run per user

Network Layout

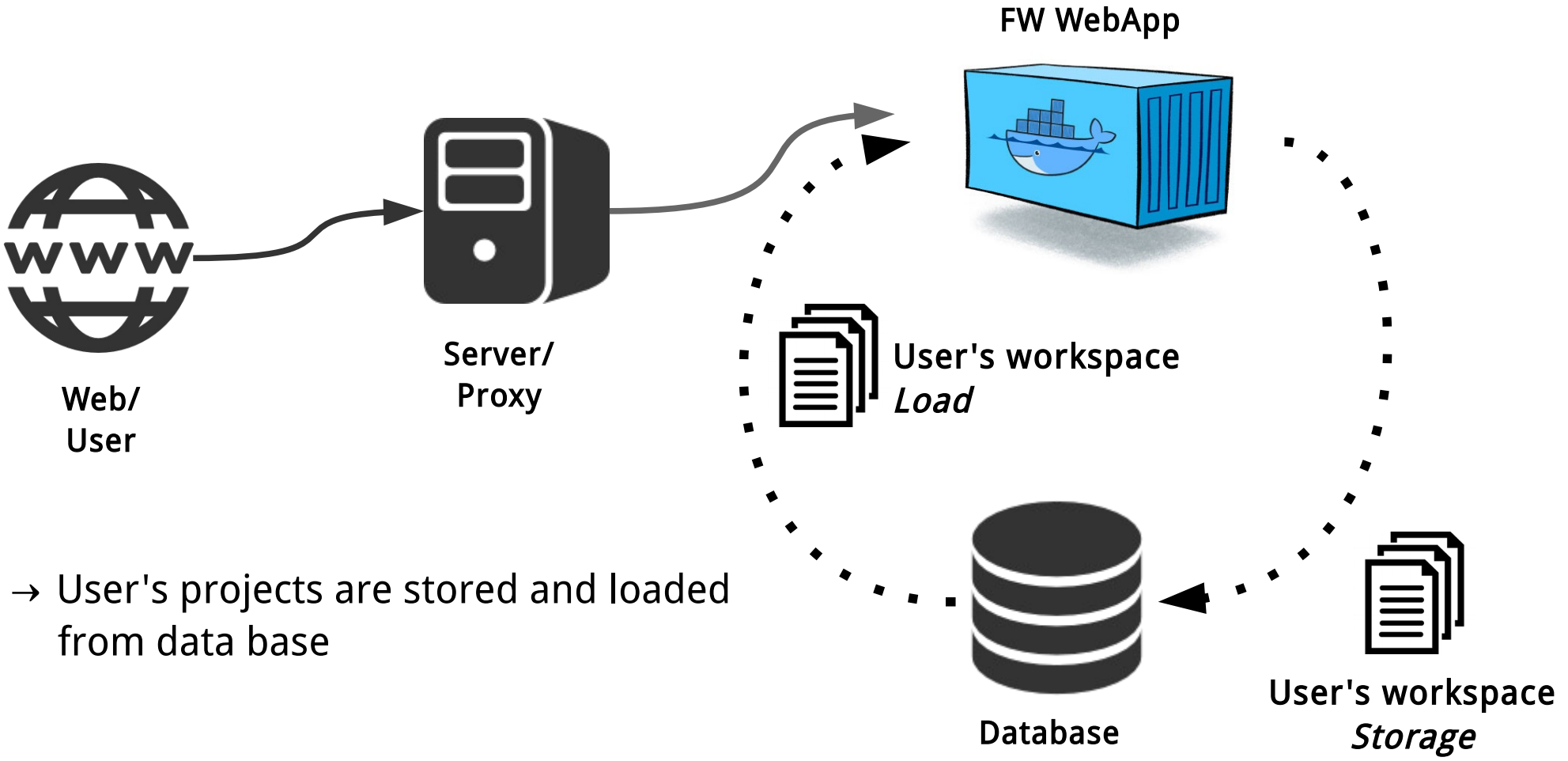


User/
Request

Server/
Proxy

WebApp
Docker container

Framework's Web Application - Projects

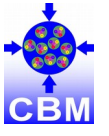


→ User's projects are stored and loaded from data base

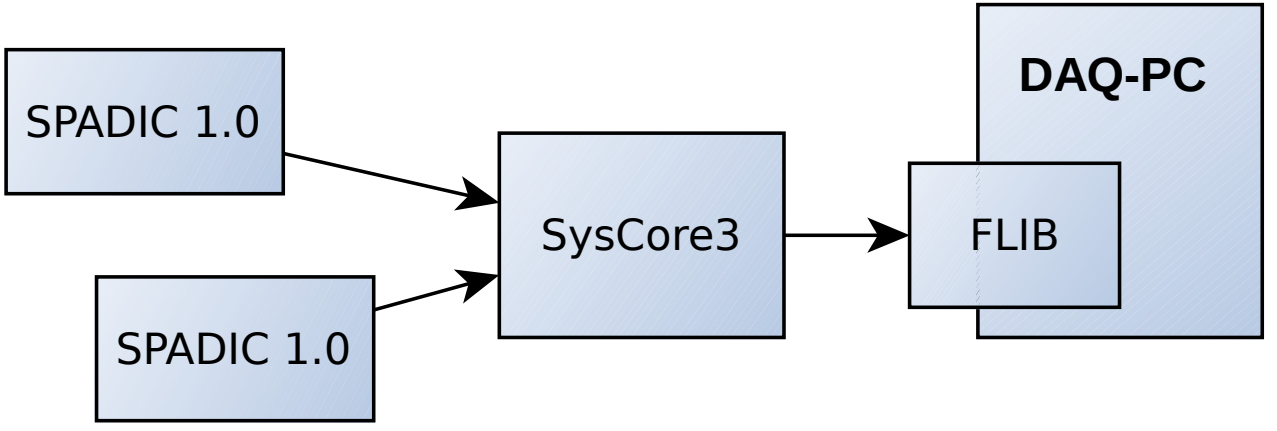
→ What is kept in the data base?

- Stream network description with configurations
- HDL sources
- Synthesis project sources (for 3rd party tools)

Overview CERN-SPS TRD Beam-test



TRD Experimental Data Acquisition Diagram



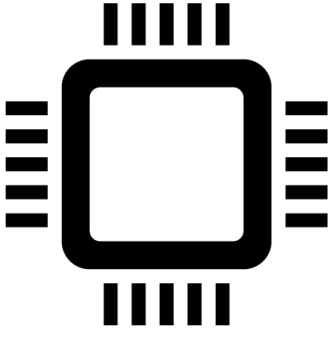
DAQ Diagram

Front-End Electronics

→ SPADIC 1.0

→ Each connected to a TRD detector

SPADIC
Self triggered Pulse Amplification and Digitization asIC



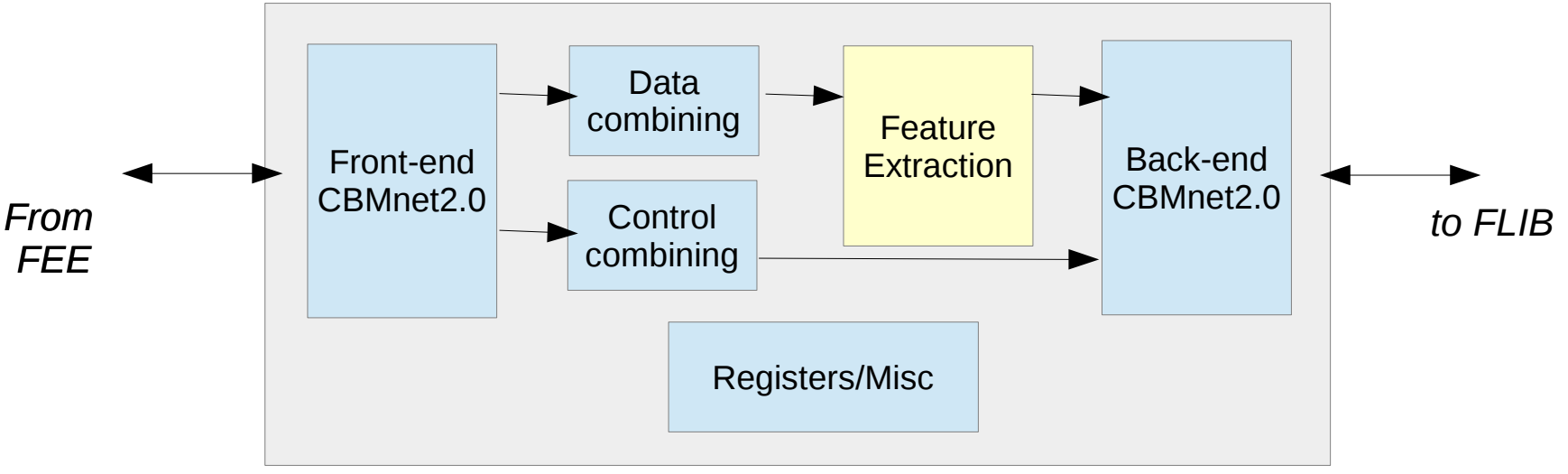
Readout

→ SysCore3 – Spartan-6 FPGA

DAQ-PC

→ FLIB (First-Level Interface Board)
Kintex-7 based FPGA board

SysCore3 Validation Design - Architecture



FPGA Internal Structure Diagram

→ SysCore3 FPGA development platform

→ ROC design already available

→ Feature Extraction included as a module inside existing design

→ Original ROC design functionality was kept

Original Design / Extended Design

Original ROC design functionality:

→ Read-out up to 3 front-end electronics

→ Communication by CBMnet 2.0 front/back end protocol

→ Configuration registers
- Front-ends
- Internal ROC

Extended ROC design

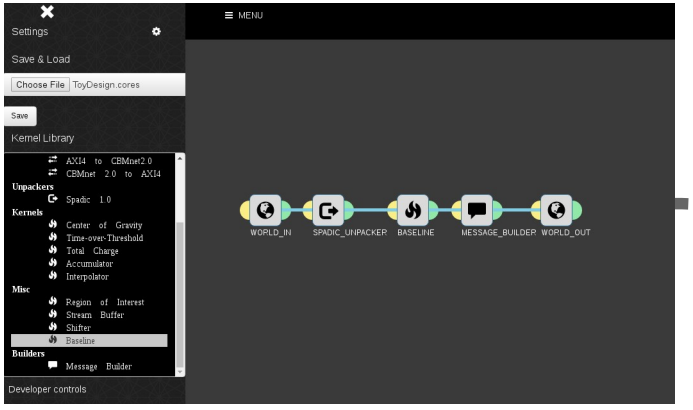
→ Feature Extraction as “black-box”

→ Feature extraction intercepts stream before going to the back-end module

→ No hand-written code
- All HDL source generated by the framework

CBM

Firmware Design Flow



Design entry by Web-Application

DSL file

Framework Application

→ HDL Sources
→ Synthesis Project

XILINX ISE DESIGN SUITE

3rd Party Synthesis Tool

Bitfile

SysCore3 ROC FPGA

Temporal Analysis (time-based)

Data mining in every message:

→ Center of Gravity
→ Time over threshold
→ Peak finder
→ Signal Integrator

Spatial Analysis (cluster-based)

Basic cluster finder in *sorted* stream messages

→ Cluster signal integrator
→ Cluster center of gravity

Message Transport

Evaluation of different packer modules
(AXI4-Streamer => CBMnet 2.0)

CBM

Closing Notes

→ A Feature Extraction Framework for FPGA firmware generation is available

→ Firmware can be generated without need of writing HDL code

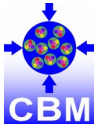
→ Framework's library contains common used processing algorithms

→ Reutilization of already existing code is possible
IpCores / VHDL / Verilog / Netlists

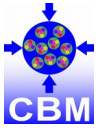
→ Performance increase against HLS tools (Vivado)
- Compared against similar algorithms implemented in Vivado HLS

CBM

Thank you for your attention.



Backup



Stream Message Modeling - Format

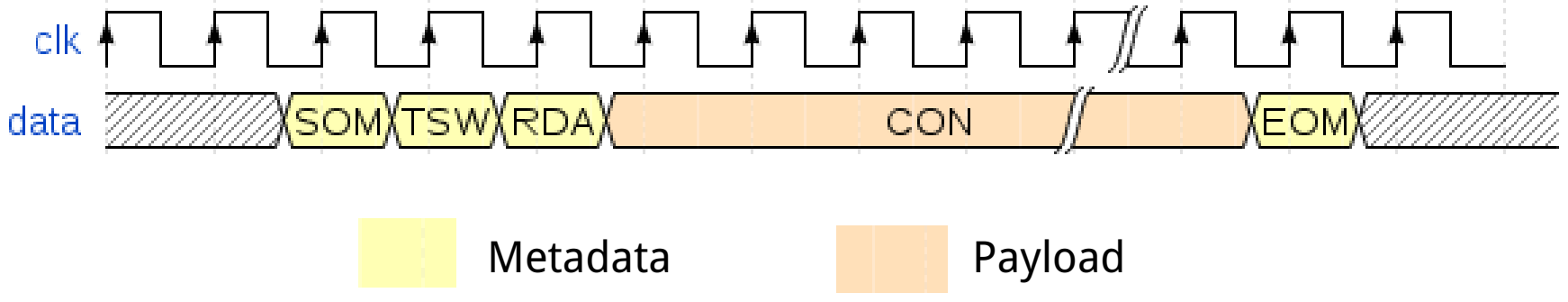
- **Every message should be 'self-contained'**

Example: Spadic 1.0 Hit message (stream message structure)

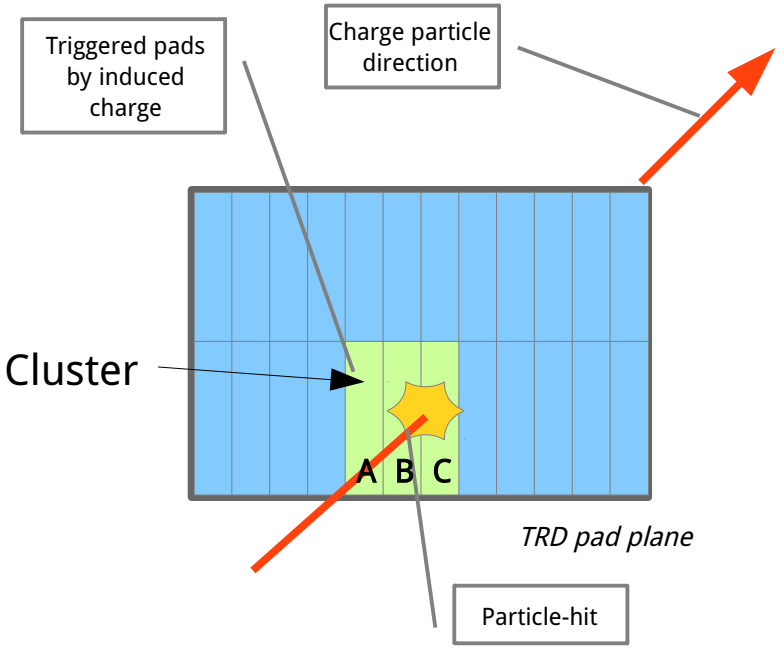
Word type	Format	Content
SOM	1000 gggg gggg cccc	g: group ID, c: channel ID
TSW	1001 tttt tttt tttt	t: timestamp
RDA	1010 dddd dddd dddd	d: raw data
CON	0ddd dddd dddd dddd	d: raw data
⋮	⋮	⋮
CON	0ddd dddd dddd dddd	d: raw data
EOM	1011 nnnn nnhh -sss	n: number of samples contained in raw data block, h: hit type, s: stop type

Source sus.ziti.uni-heidelberg.de

- Spadic 1.0 stream message modeled by the framework
  No control signals shown



CBM-TRD hit-event data generation

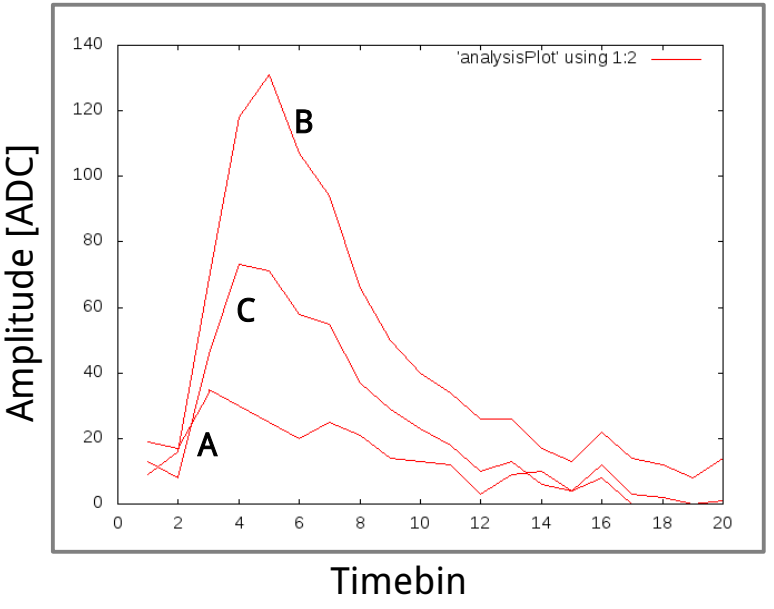


Physical event

A charged particle crosses the TRD detector

Measurement

The particle's charge is spread-out across neighboring pads



We could calculate specific parameters e.g. :

- Signal charge
- Hit cluster (based on neighbor pads)

Self-triggered Pulse Amplification and Digitization ASIC - SPADIC 1.0

What is the SPADIC 1.0 FEE?

- Default Front-end for CBM-TRD
- An 32 channel oscilloscope-like charge pulse readout ASIC
- Complete multi-channel mixed-mode readout system
- Flexible, adjustable and programmable

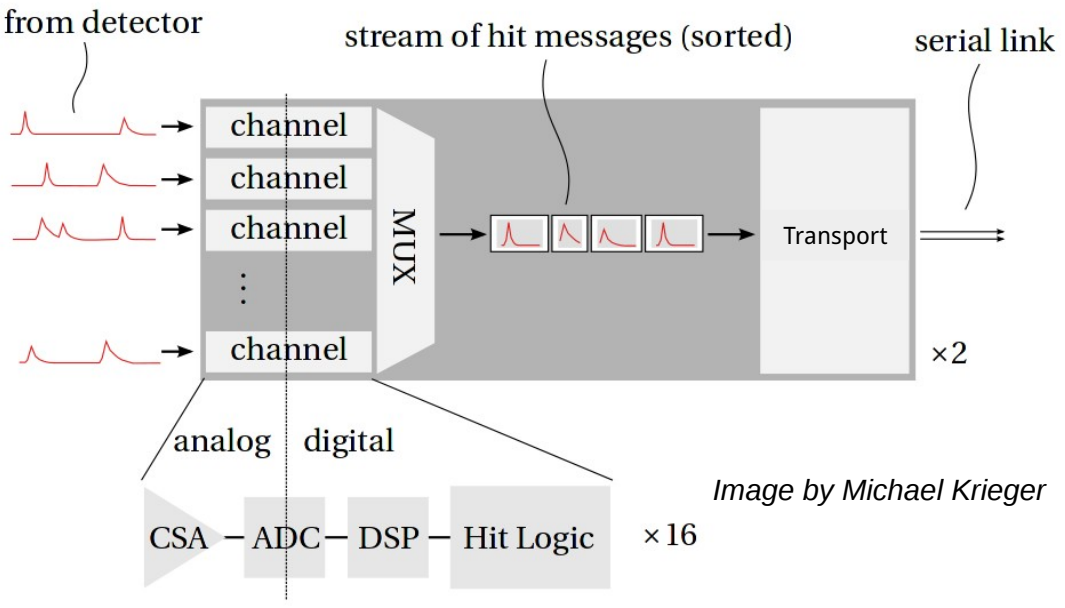
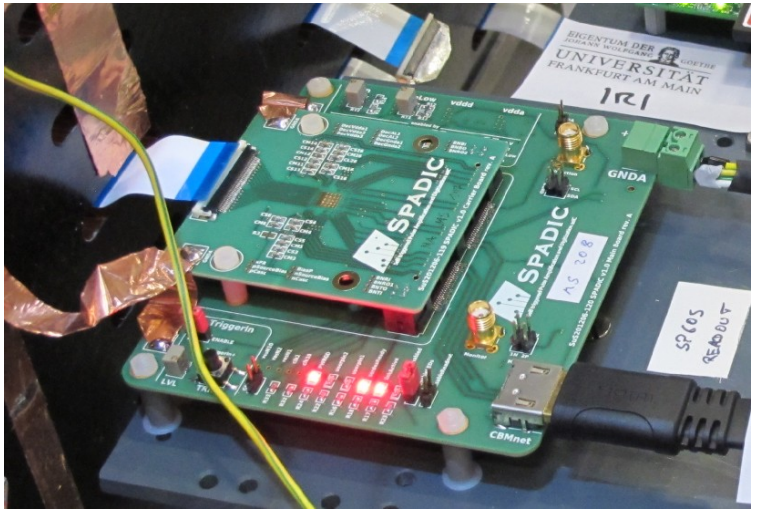


Image by Michael Krieger

Features:

- Payload is wrapped together with meta-data
  - Timestamp, Channel ID, Group ID, Hit type Stop Type, Epoch between others.
- Multi-hit logic detection
- Backend transport
  - 2 x 500 Mbps downlinks
- More info: http://spadic.uni-hd.de/

SPADIC
Self triggered Pulse Amplification and Digitization asIC



SPADIC 1.0 @ 2012 CERN PS9 Beamtest
Connected to a Münster TRD