Circuit Data Flow (CIDAF)

A generic, flexible, discrete-time simulator

L.Calligaris* (RAL), luigi.calligaris@stfc.ac.uk

Participants: D.Cieri*, K.Harder, C.Shepherd, I.Tomalin (RAL), P.Hobson, A.Morton, R.Powell, I.Reid (Brunel University), P.Vichoudis (CERN), G.Hall, G.Iles, T.James, M.Pesaresi, A.Rose, A.Shtipliyski, A.Tapper, K.Uchida (Imperial College London), T.Schuh (KIT), F.Ball, J.Brooke, D.Newbold (University of Bristol), R.Vamosi, T.Matsushita (Austrian Academy of Science)

One of the proposed implementations of the **CMS L1 track trigger upgrade** for the HL-LHC uses a **Hough Transform** algorithm implemented in firmware, running on **FPGA** electronics. Many designs have been proposed for a firmware demonstrator[1], and their performance in terms of latency, localized data rates, pipe stalls, buffer overflows and dropped payloads needs to be studied. A common **data flow simulation** software, CIDAF, is being developed to ease sharing, modification and extension of the circuit simulations.

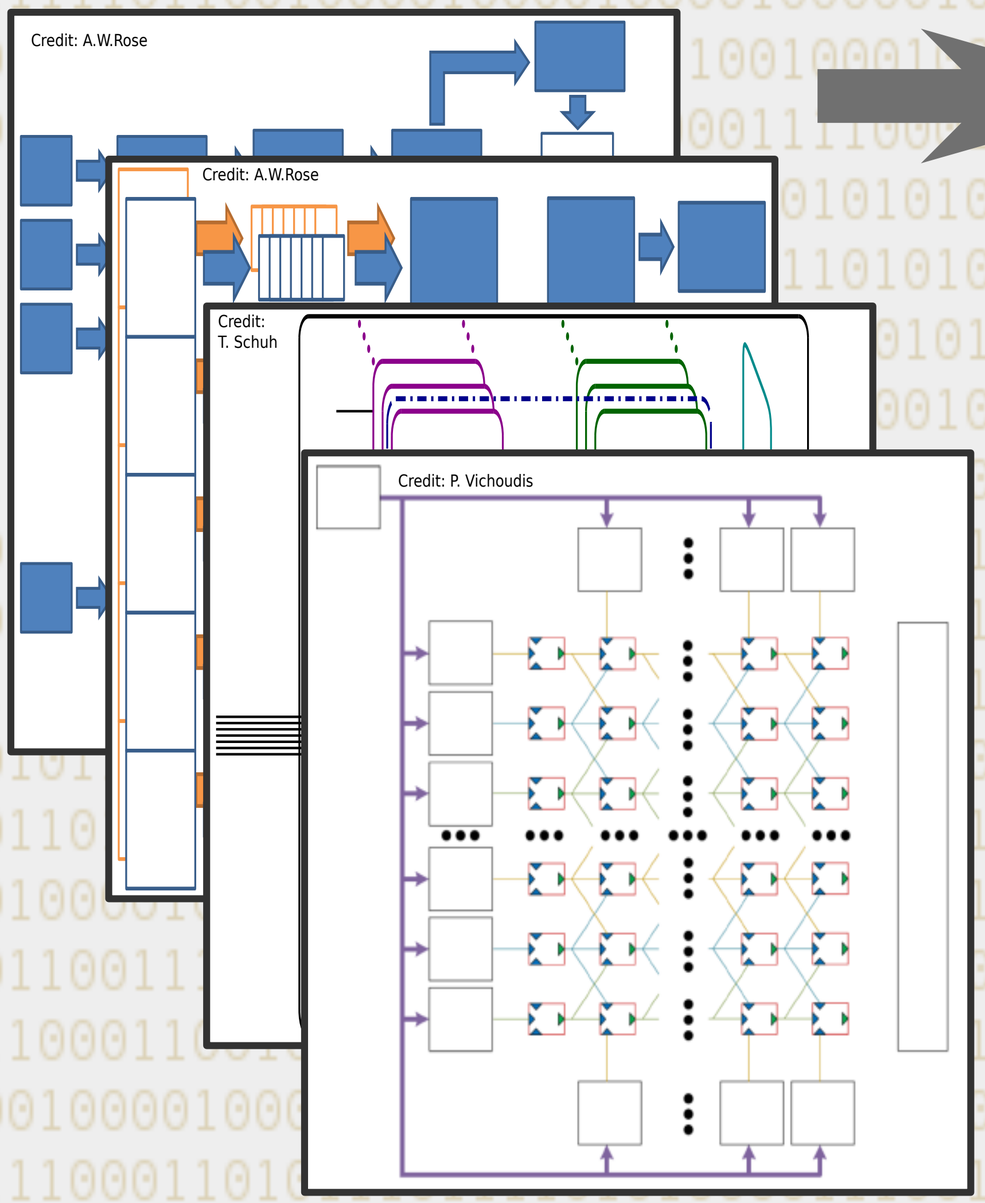
CIDAF performs a **discrete-time simulation** of a processing network previously loaded into memory. The network topology and the parameters configuring its constituents are loaded at **run-time**, allowing the connections between components to be defined on the fly.

This solution makes the circuit simulations easily extendable by adding components to an existing circuit, and allows to study in isolation the behavior of subsets of components in the circuit.

CIDAF allows the presence of an **arbitrary number of clocks**: the use of different clocks for piloting component state transitions and to initiate the transmission of data in the links between components solves the problem of determining the **correct order of execution** of component state transitions, which is problematic in the case of logic circuits implementing **feedback loops**.

[1] See poster by D.Cieri in the same session.

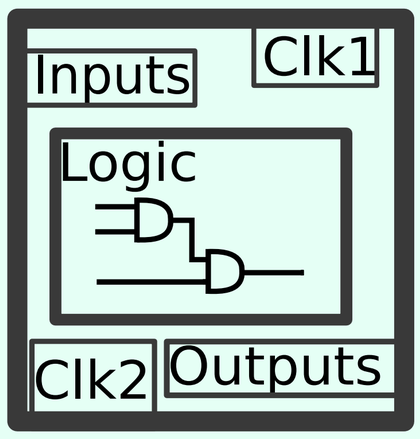
Processing network designs



Simulation Workflow

1) Define Internal Logic of the components

(at compile time)

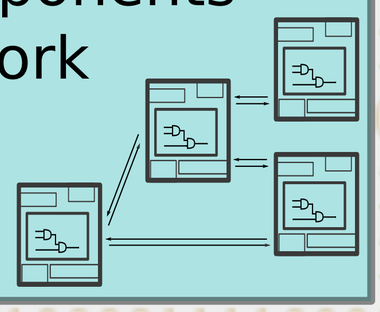


2a) Assign components to clocks

```
Clock internalclock001("internalclock001");
Clock linkclock001("linkclock001");
internalclock001.RegisterComponent(latchA11);
linkclock001.RegisterComponent(latchA11.Out());
```

2b) Connect the components to build the network

```
component01.RegisterOutput(&component02);
component01.RegisterOutput(&component04);
component02.RegisterOutput(&component03);
component02.RegisterOutput(&component04);
```

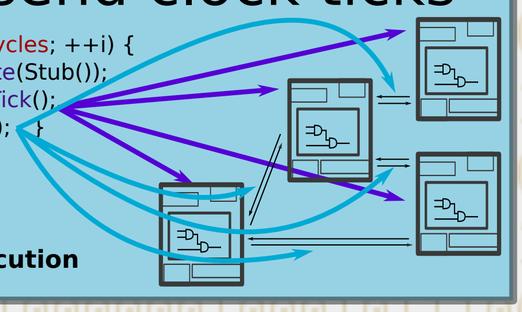


3) Simulation: send clock ticks

```
for (int i = 0; i < Ncycles; ++i) {
  component01.Write(Stub());
  internalclock001.Tick();
  linkclock001.Tick(); }
```

Separation of internal state and link clocks

Solves ambiguity in order of execution



*Supported by the EU FP7-PEOPLE-2012-ITN project nr 317446, INFIERI, "Intelligent Fast Interconnected and Efficient Devices for Frontier Exploitation in Research and Industry

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement n° [317446] INFIERI "INtelligent Fast Interconnected and Efficient Devices for Frontier Exploitation in Research and Industry''