Open-Source SCA Implementation-Embedded and Software Communications Architecture

OSSIE and SCA Waveform Development

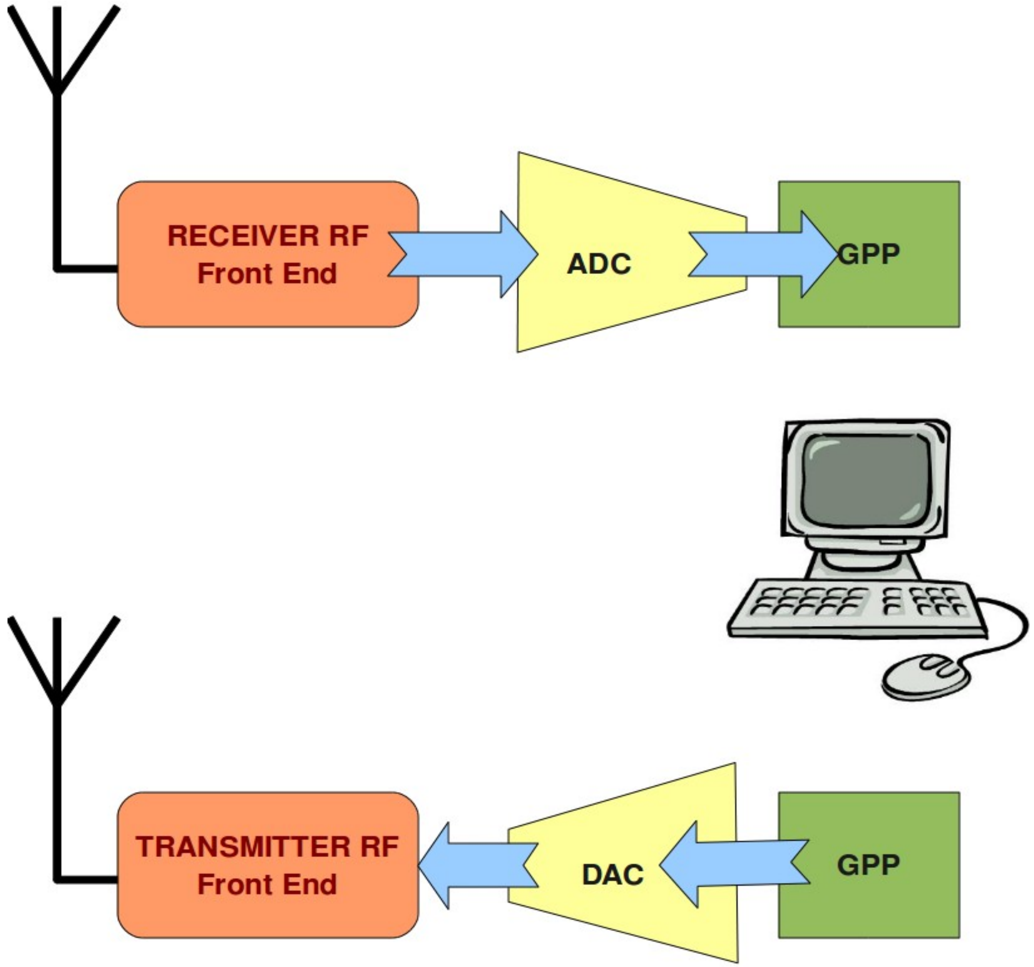
Edoardo Paone

The thesis project

Develop a OSSIE waveform based on IEEE 802.15.4:

- Understand SCA and the OSSIE waveform development
- Analyze the IEEE 802.15.4 standard protocol
- Import GNU Radio libraries into OSSIE components
- Implement a RX/TX waveform prototype using the USRP
- Evaluate the OSSIE waveform development tools and process

SDR – Software Defined Radio



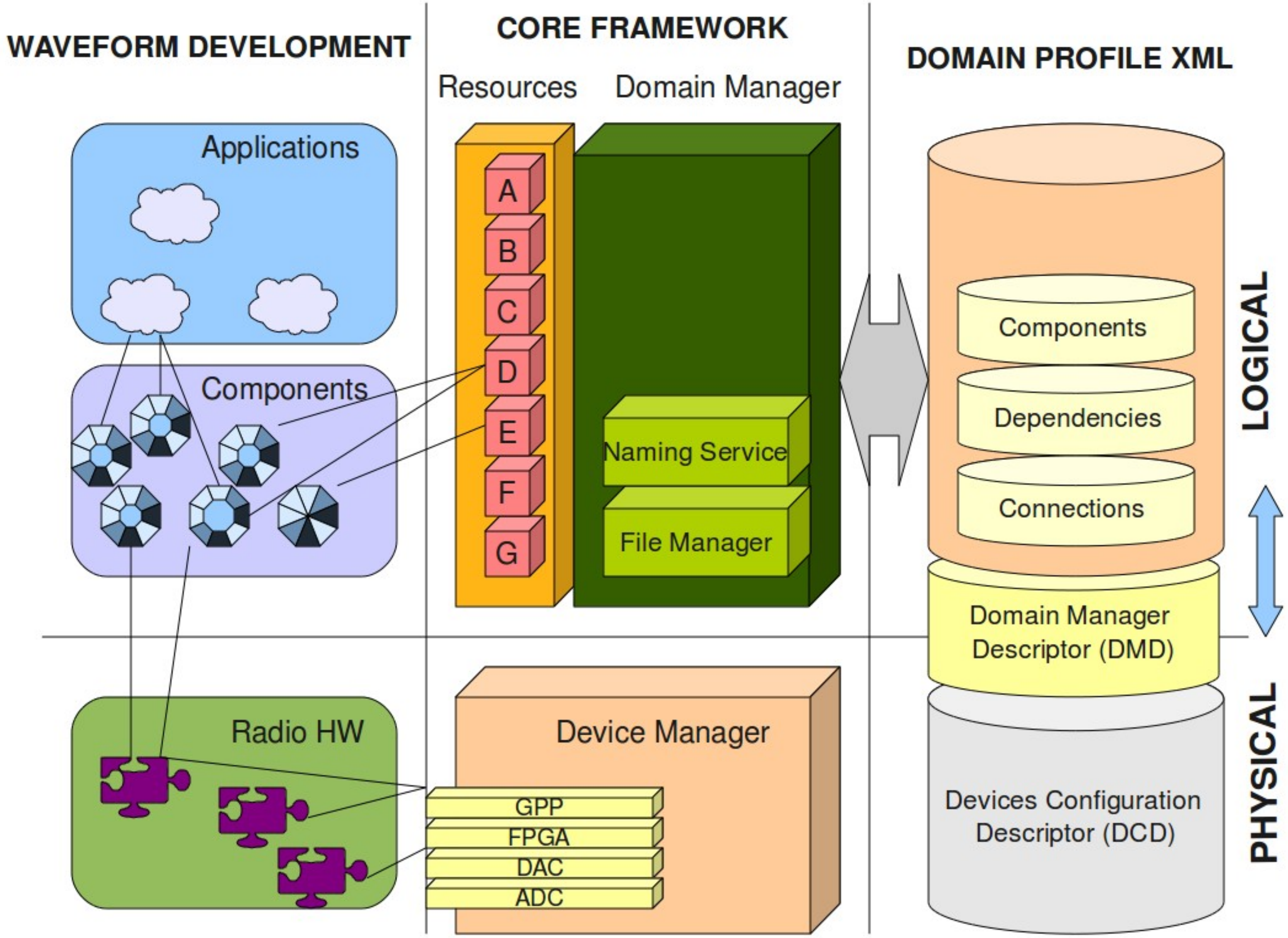
SoftModem

- modem with minimal hardware
- it uses host computer’s resources instead of dedicated hardware
- easier upgrades to new modem standards
- reduction in production costs, component size and weight

SCA Waveform Development

- Abstraction of underlying hardware and sofware
  - platform independence, portability of applications
  - reuse of waveform design modules
- CORBA for intercomponent communication
  - component implementation in different languages
  - distributed waveform (TCP transport rule)

SCA – Abstraction layers



Universal Software Radio Peripheral (USRP)

Materials for the thesis project:

- A personal computer (dualcore processor, 2 GB RAM)
- Ubuntu Linux 8.04
- GNU Radio 3.2.2
- OSSIE 0.7.4
- 2 USRPs
- Eclipse

OSSIE project

- **O**pen-**S**ource **S**CA-based **I**mplementation – **E**mbedded (OSSIE)
- Developed at Wireless@Virginia Tech (Blacksburg, VA)
- Professor-in-charge: Dr. Jeffrey H. Reed
- Source code, documentation, tutorials available at:

http://ossie.wireless.vt.edu

OSSIE distribution

- Latest version: OSSIE 0.8.0 (released in January 2010)
- It can be installed on PCs running Linux (or in a virtual machine on Windows)
- The OSSIE package includes:
  - The Core Framework (CF)
  - The OSSIE Eclipse Feature (OEF)
  - A set of tools for waveform development
  - A component library

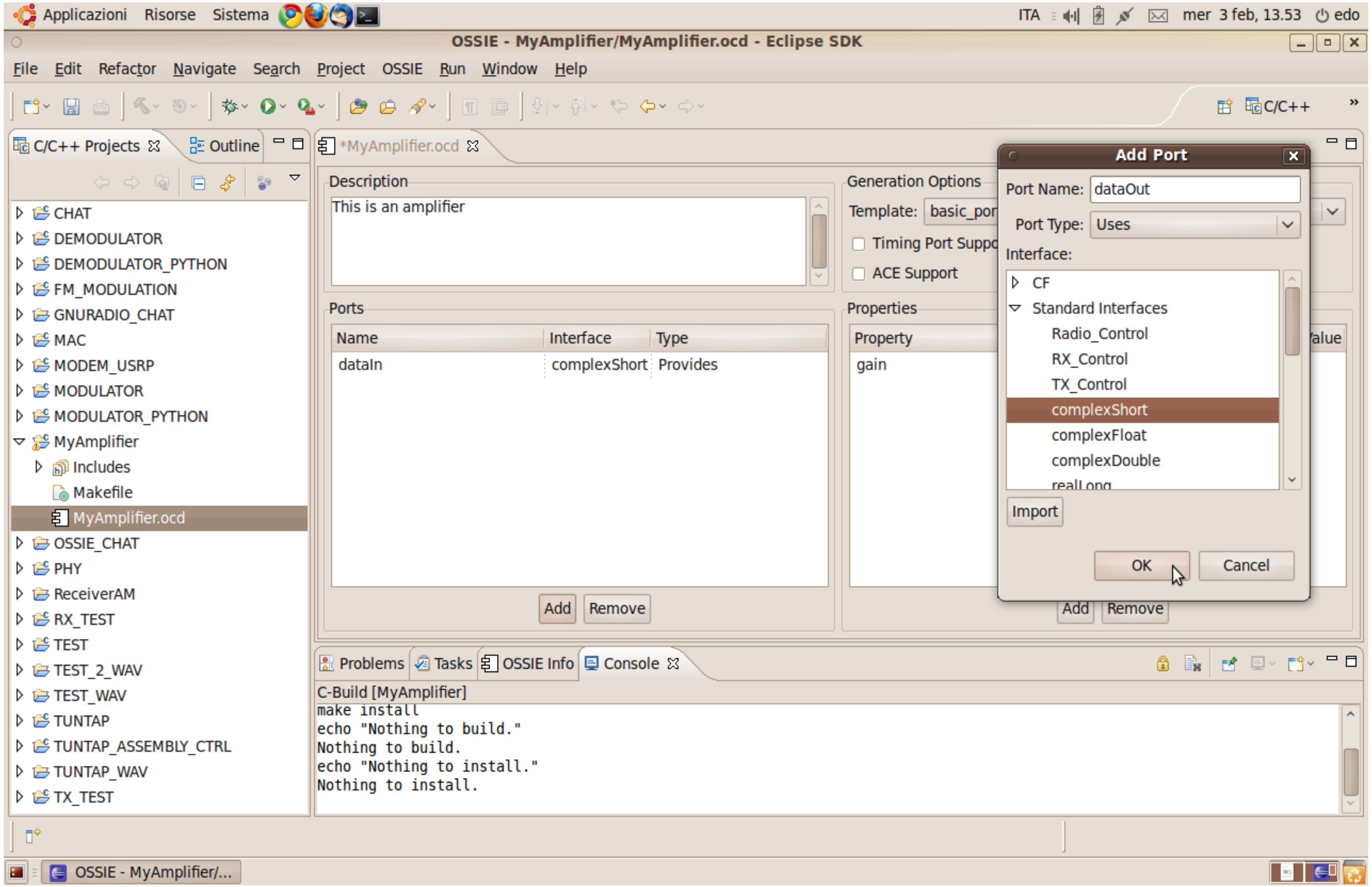
OSSIE Core Framework

- written in C++ using the omniORB CORBA ORB (open source)
- implements key elements of the SCA specification
- CORBA transport rules define inter-component communication:
  - TCP/IP, for distributed waveforms
  - Unix domain sockets

P. Balister, et al., “Impact of the use of CORBA for Inter-Component Communication in SCA Based Radio”, Nov 2006

T. Tsou, et al., "Latency Profiling for SCA Software Radio", Nov 2007

OSSIE Component



OSSIE Component

- **C++ or Python implementation**

- **Standard Interfaces:**

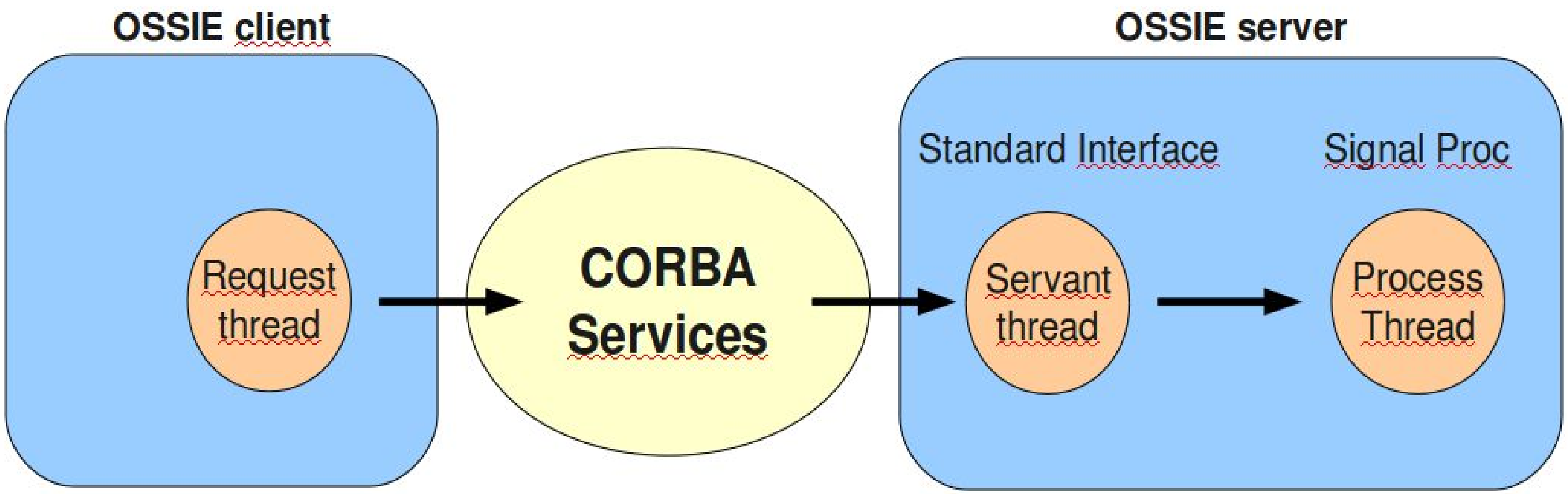
  basic real and complex data representations in 8, 16, and 32 bit sizes passed in the form of CORBA sequences

- **Custom Interfaces:**

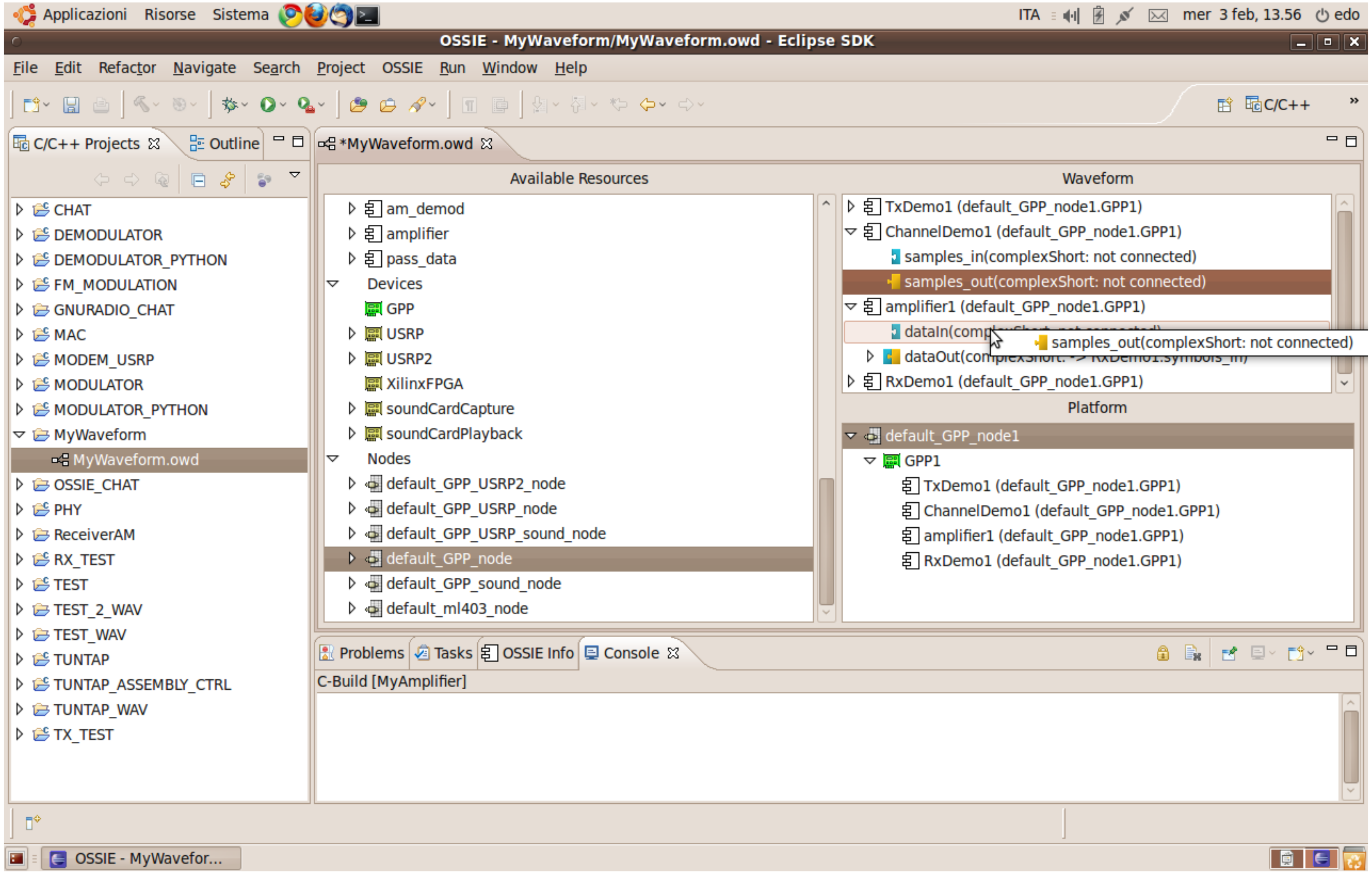
  can be defined using Interface Description Language (IDL) and compiled with the IDL compiler

OSSIE Component

- **Data buffering:**
  effectively decouples the CORBA call sequence from signal processing code



OSSIE Waveform



OSSIE Waveform

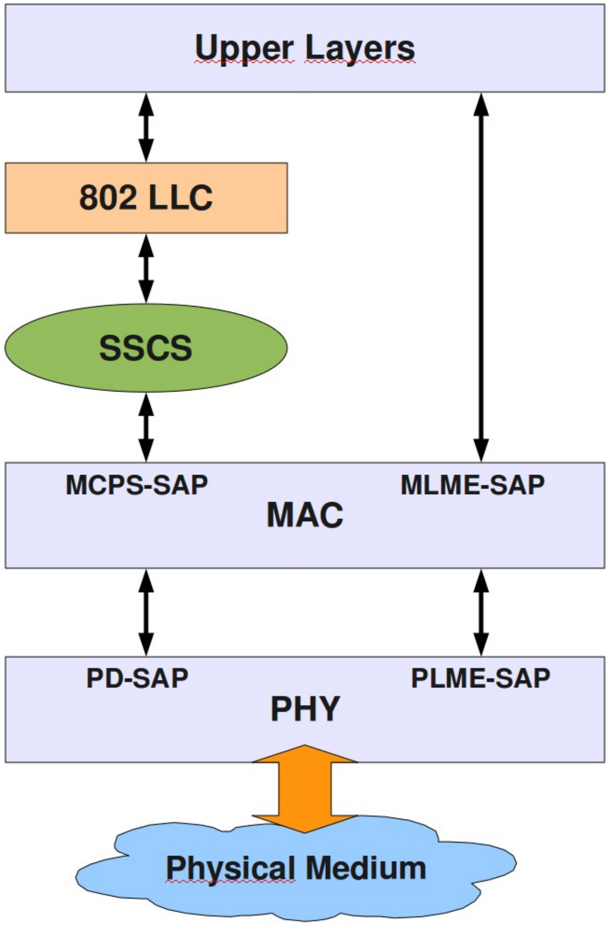
- Drag-and-drop of component library, graphical tools for interconnecting components

- Waveform XML representation generated by the OEF:
  - List of components
  - Configuration of components
  - Component interconnections
  - Target deployment node

OSSIE Waveform

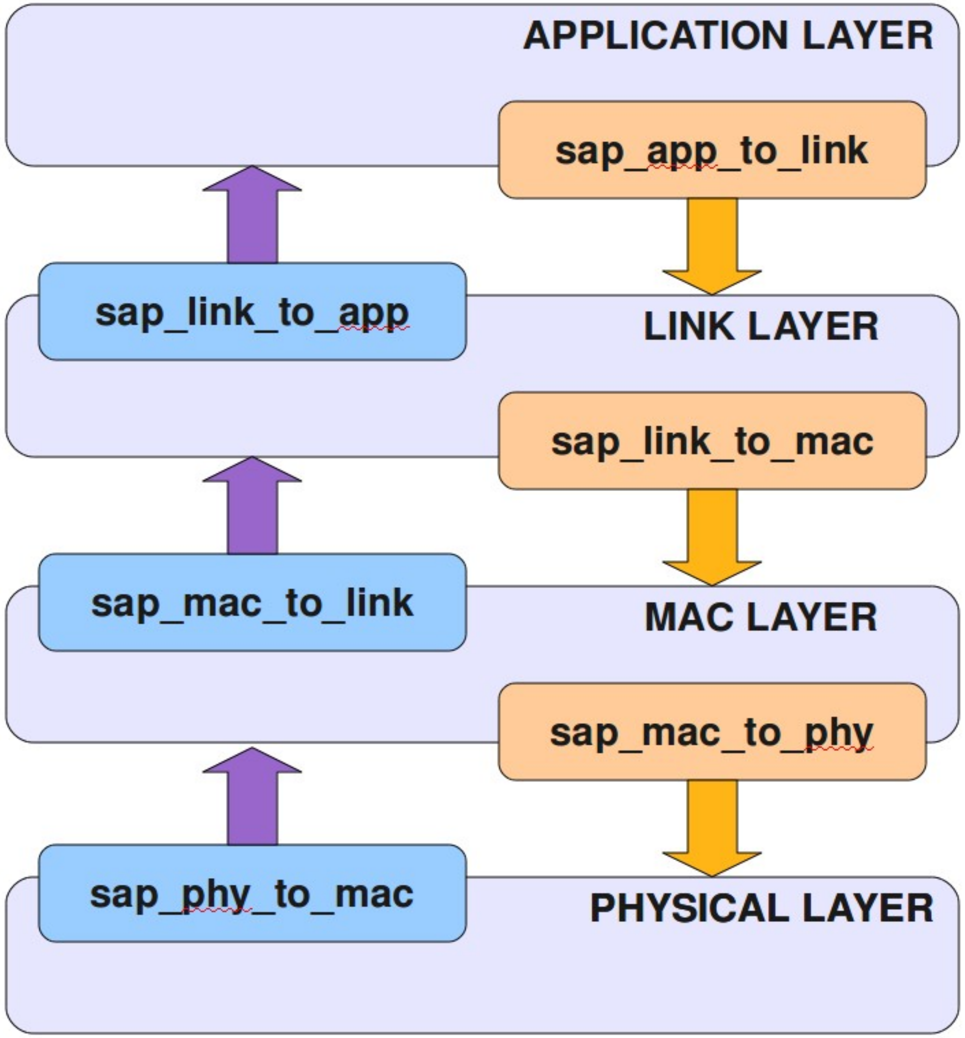
```
<connections>
    <connectinterface id="DCE:e1aa5d66-824a-11dc-adea-00123f63025f">
        <providesport>
            <providesidentifier>samples_in</providesidentifier>
            <findby>
                <namingservice name="ChannelDemo1"/>
            </findby>
        </providesport>
        <usesport>
            <usesidentifier>symbols_out</usesidentifier>
            <findby>
                <namingservice name="TxDemo1"/>
            </findby>
        </usesport>
    </connectinterface>
    <connectinterface id="DCE:e1ad7154-824a-11dc-85b0-00123f63025f">
        <providesport>
            <providesidentifier>symbols_in</providesidentifier>
            <findby>
                <namingservice name="RxDemo1"/>
            </findby>
        </providesport>
        <usesport>
            <usesidentifier>samples_out</usesidentifier>
            <findby>
                <namingservice name="ChannelDemo1"/>
            </findby>
        </usesport>
    </connectinterface>
</connections>
```

IEEE 802.15.4



- MCPS = MAC common part sublayer
- MCPS-SAP = MCPS service access point
- MLME = MAC layer management entity
- MLME-SAP = MLME service access point
- PD = PHY data
- PLME = PHY layer management entity
- SSCS = service-specific convergence sublayer
- LLC = logical link control

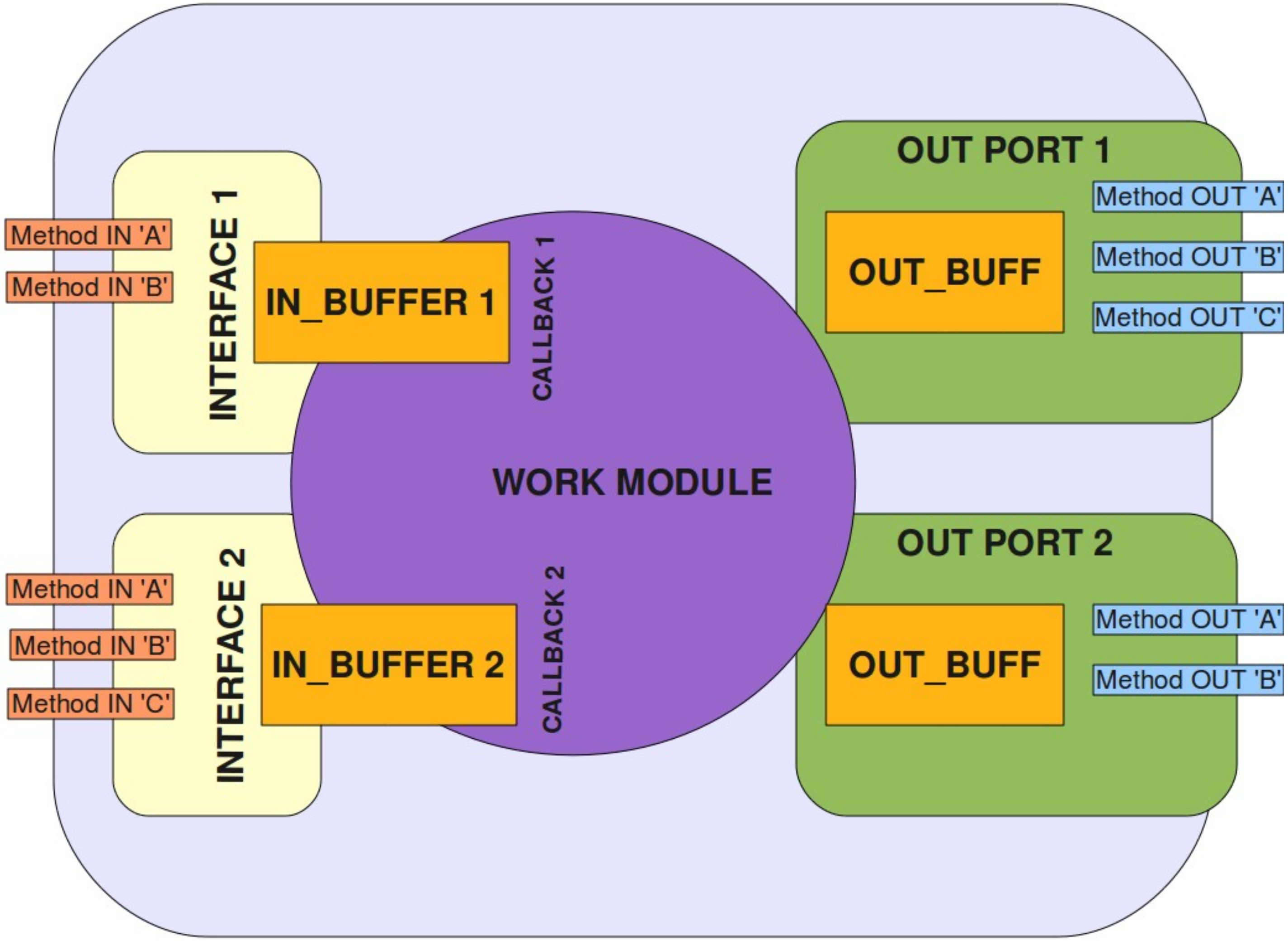
Custom Interfaces - IDL



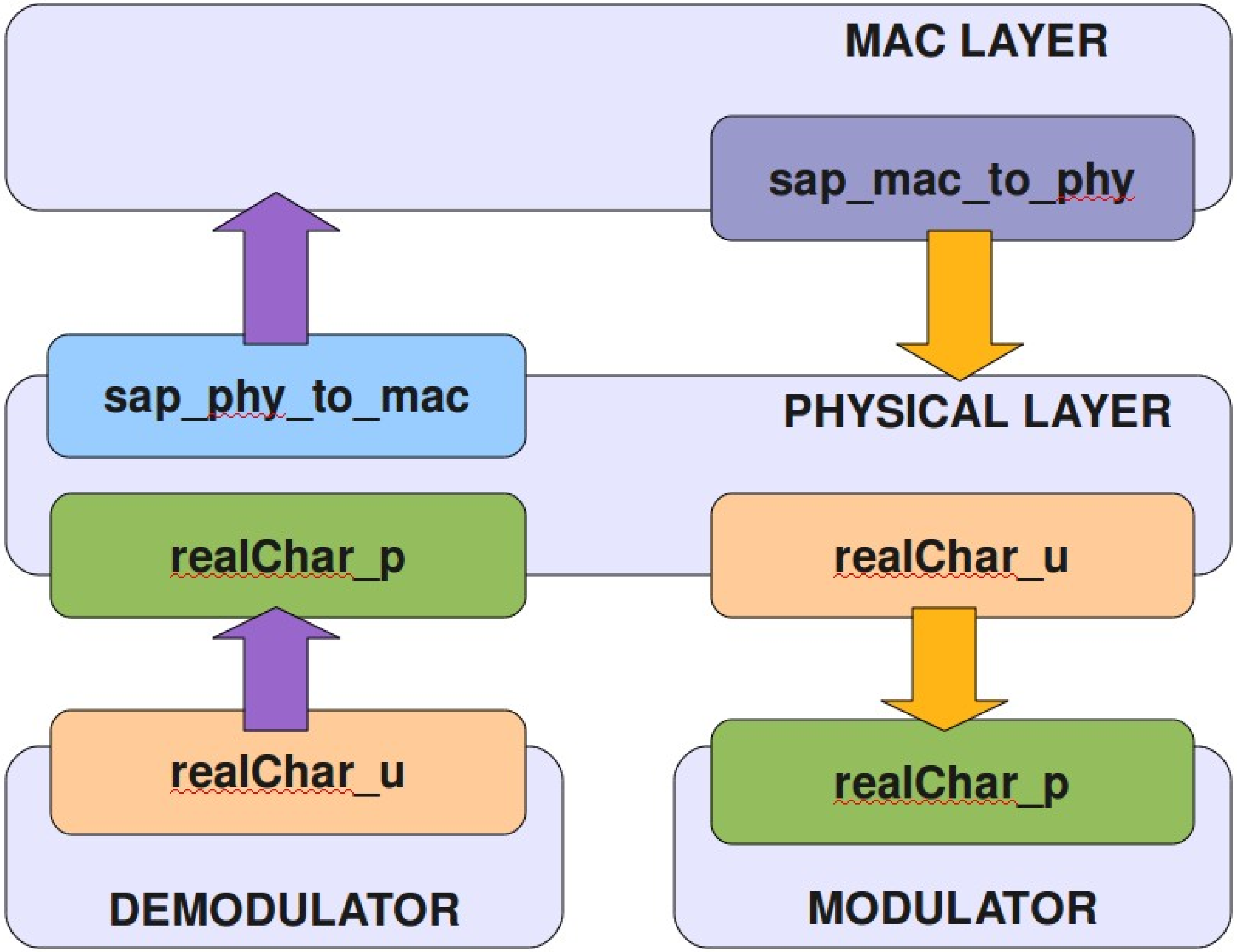
Custom Interfaces - IDL

```
#include "ossie/PortTypes.idl"

module customInterfaces{

/* link layer SAP for mac layer */
interface sap_mac_to_link{

    void macpsData_confirm(     in octet msduHandle,
                                in octet status,
                                in unsigned long Timestamp     );

    void mcpsData_indication(   in octet SrcAddrMode,
                                in unsigned short SrcPANId,
                                in unsigned long long SrcAddr,
                                in octet DstAddrMode,
                                in unsigned short DstPANId,
                                in unsigned long long DstAddr,
                                in octet msduLength,
                                in PortTypes::CharSequence msdu,
                                in octet msduLinkQuality,
                                in octet DSN,
                                in unsigned long Timestamp,
                                in octet SecurityLevel,
                                in octet KeyIdMode,
                                in unsigned long long KeySource,
                                in octet KeyIndex              );
};
};
```

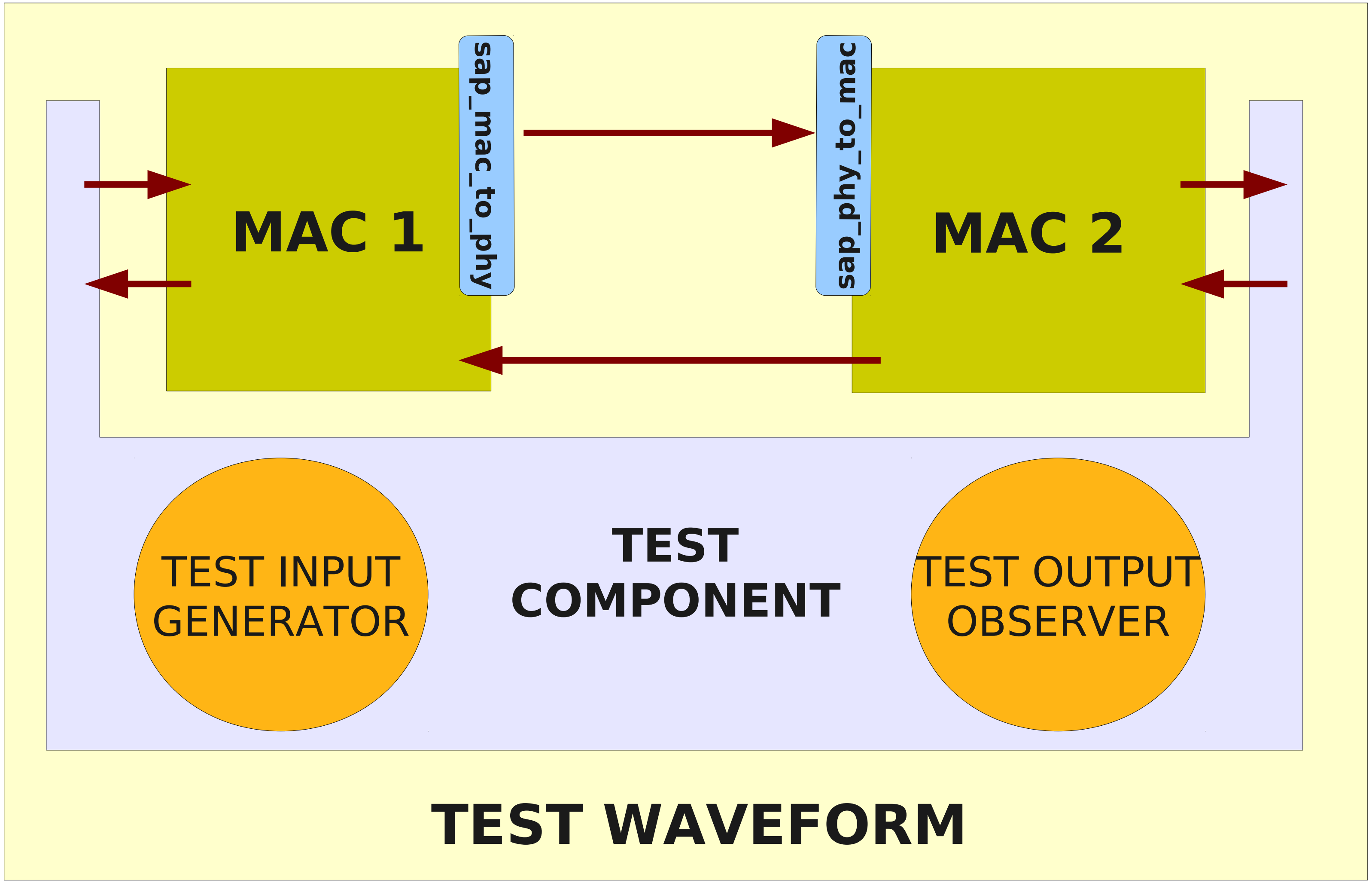
Component Structure



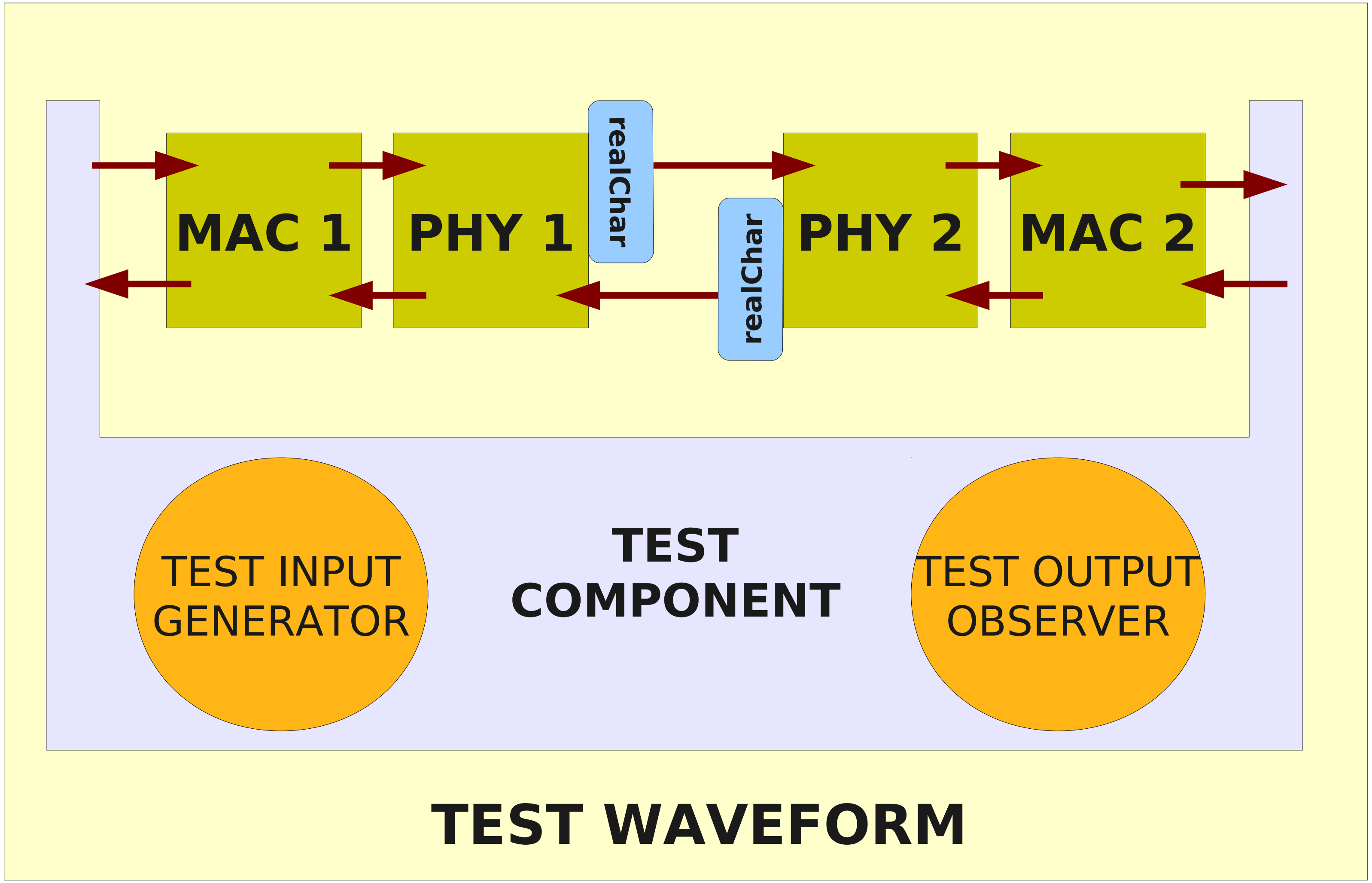
PHY layer


MAC LAYER
sap_mac_to_phy
sap_phy_to_mac
PHYSICAL LAYER
realChar_p
realChar_u
realChar_u
realChar_p
DEMODULATOR
MODULATOR

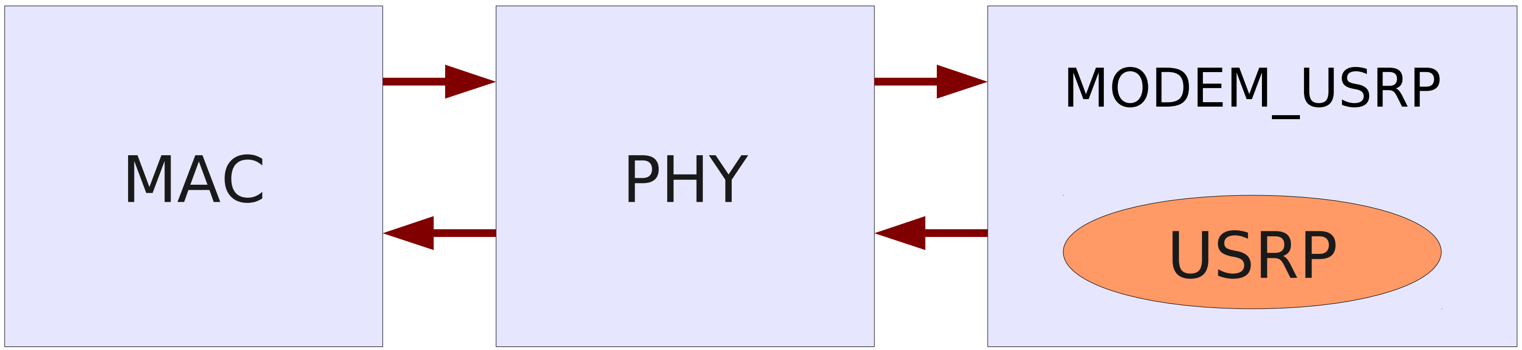
MAC - Simulation



PHY - Simulation

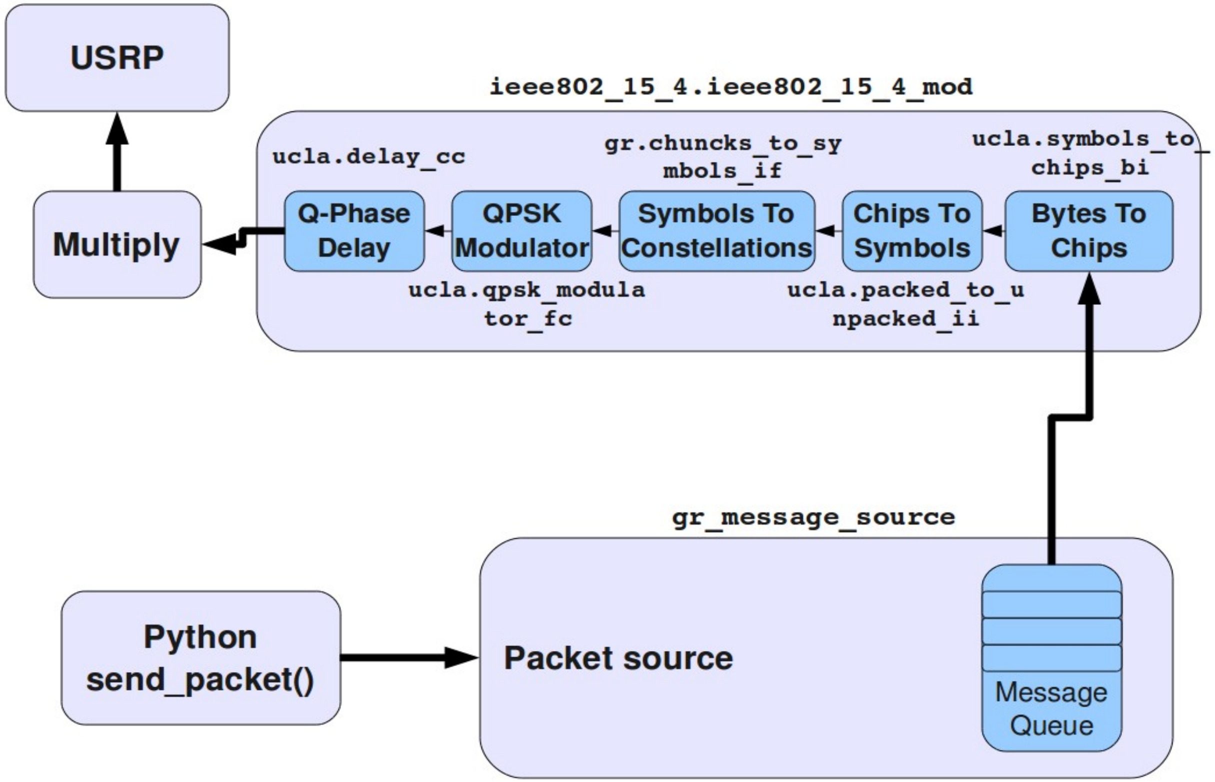


RX/TX path – GNU Radio solution

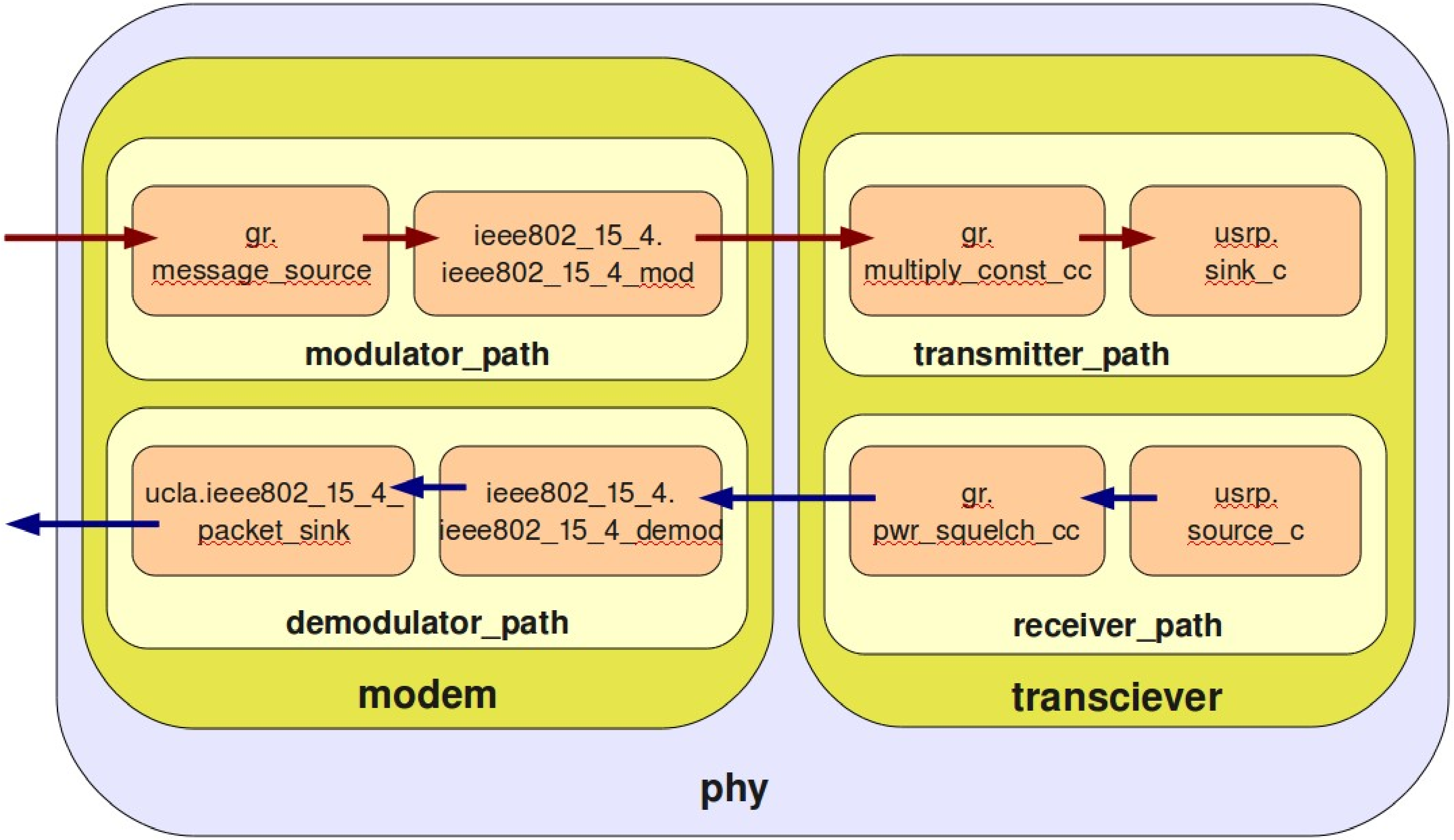


UCLA Zigbee PHY

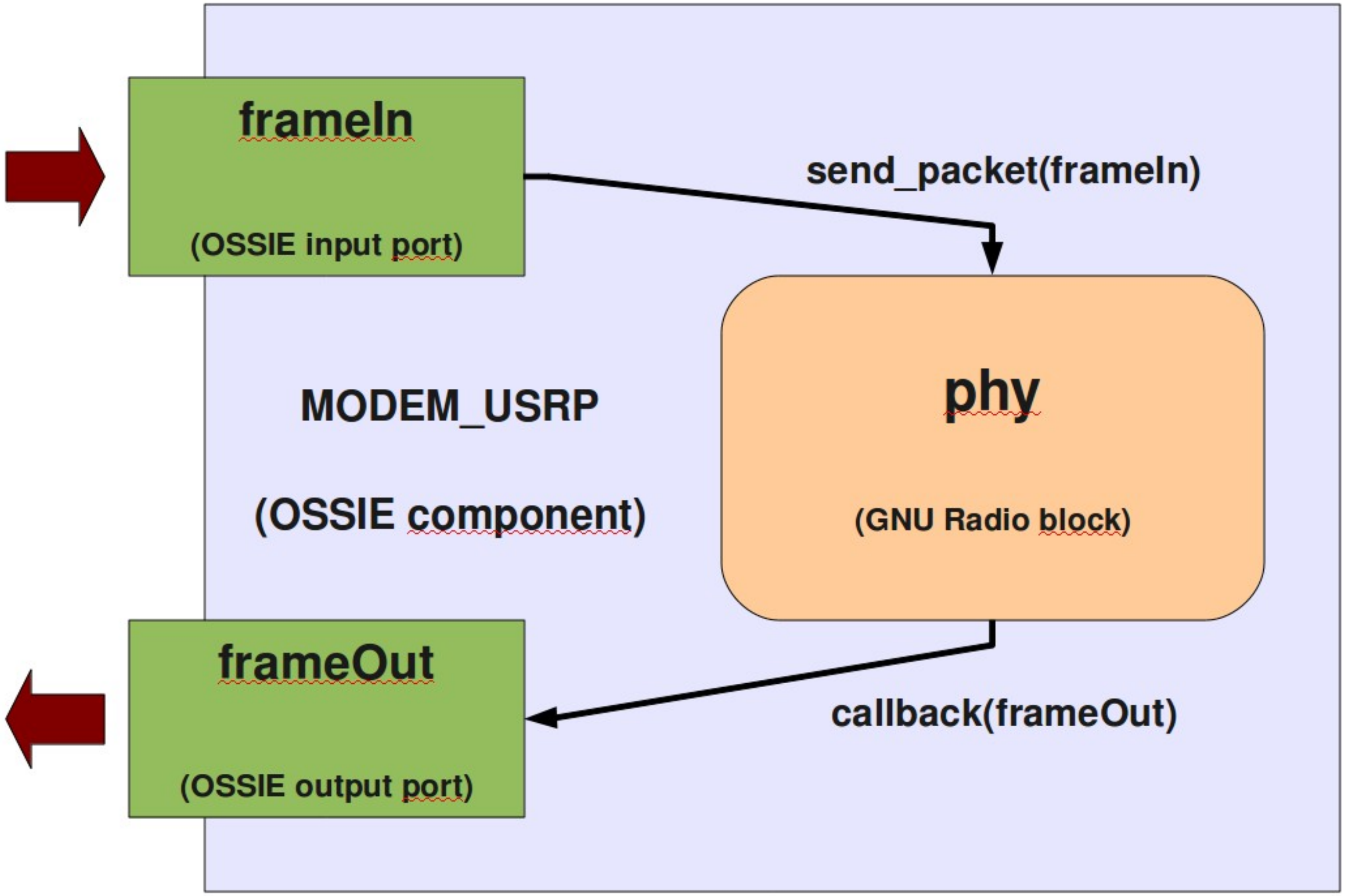
GNU Radio implementation of the IEEE 802.15.4 physical layer.



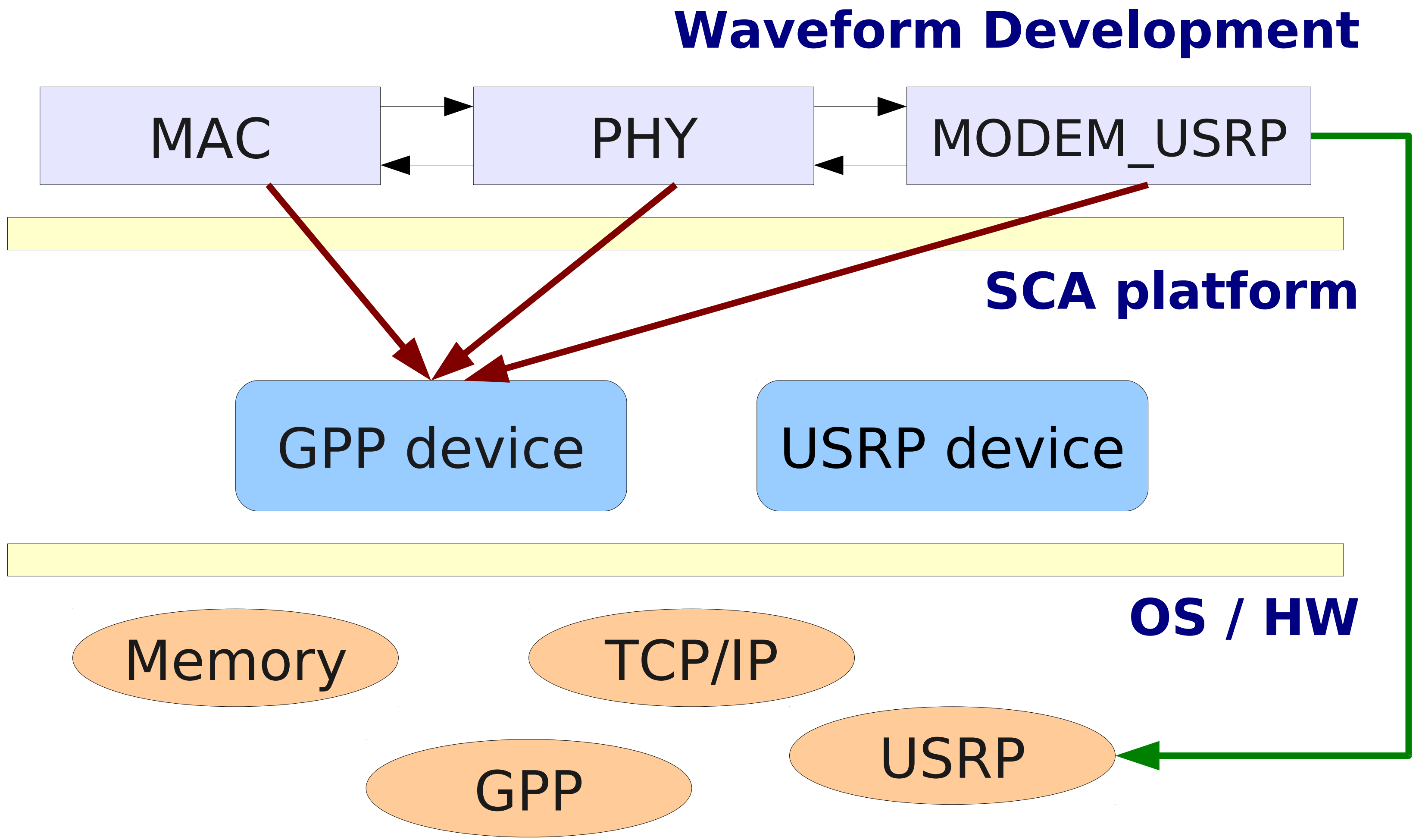
MODEM_USRP WorkModule



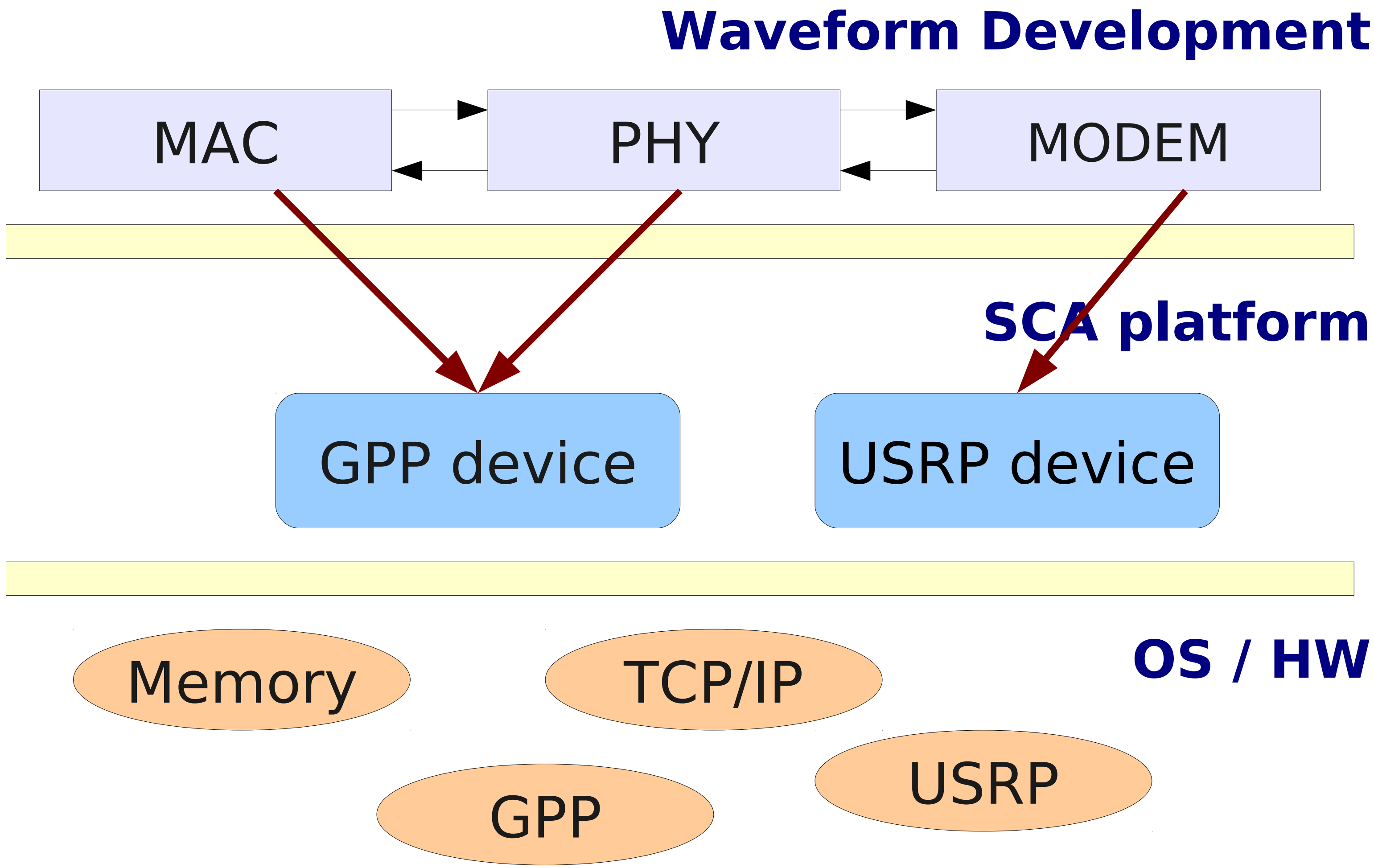
MODEM_USRP component



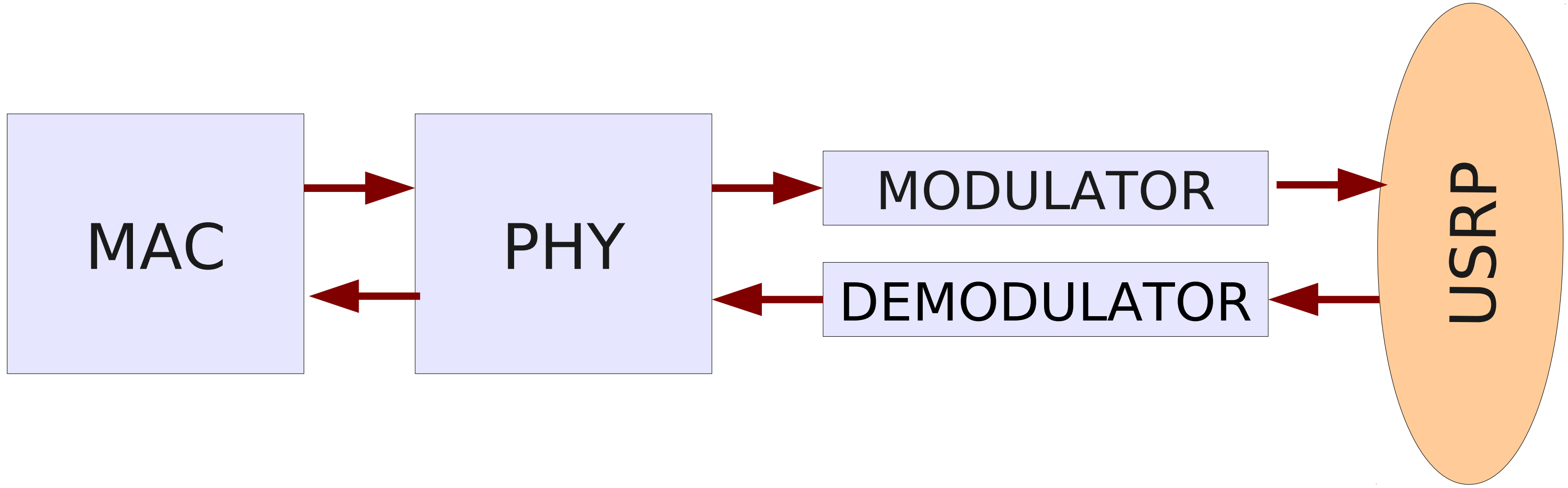
RX/TX path – GNU Radio solution



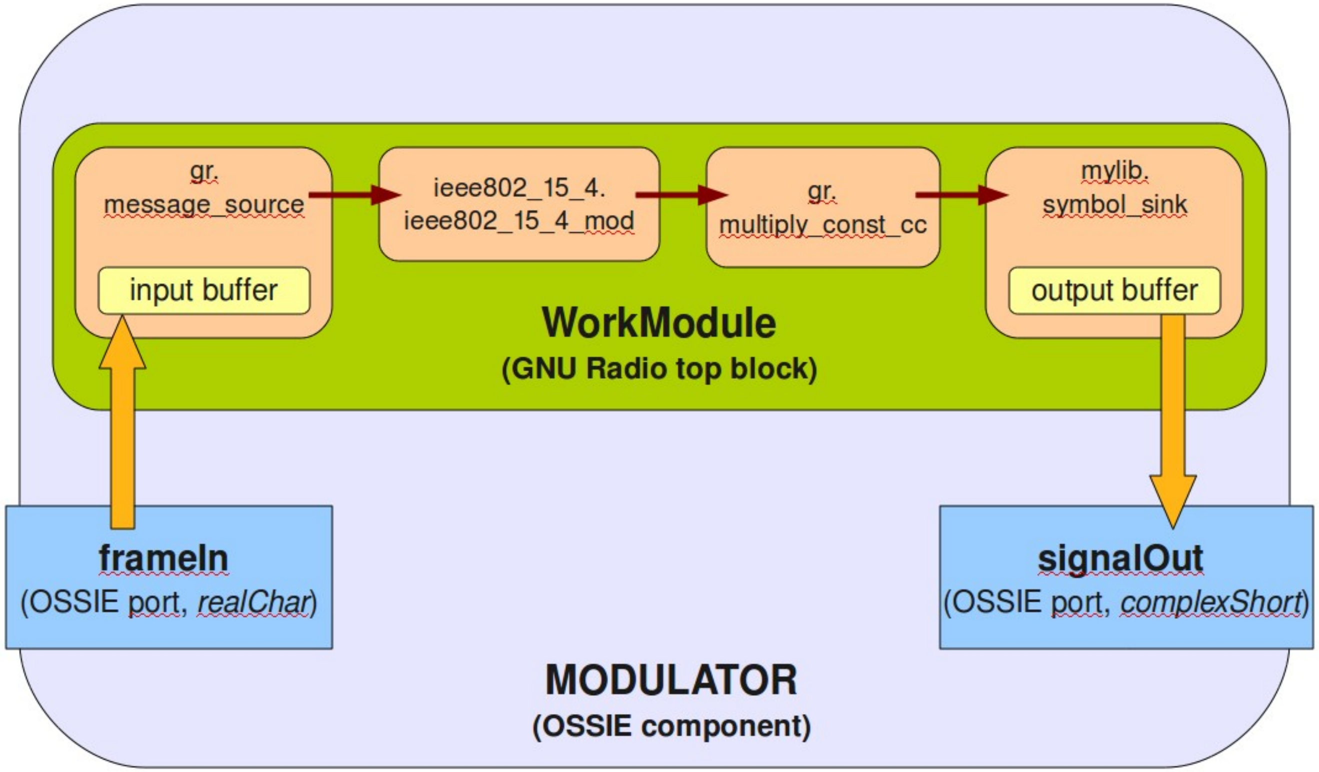
RX/TX path – OSSIE solution



RX/TX path – OSSIE solution

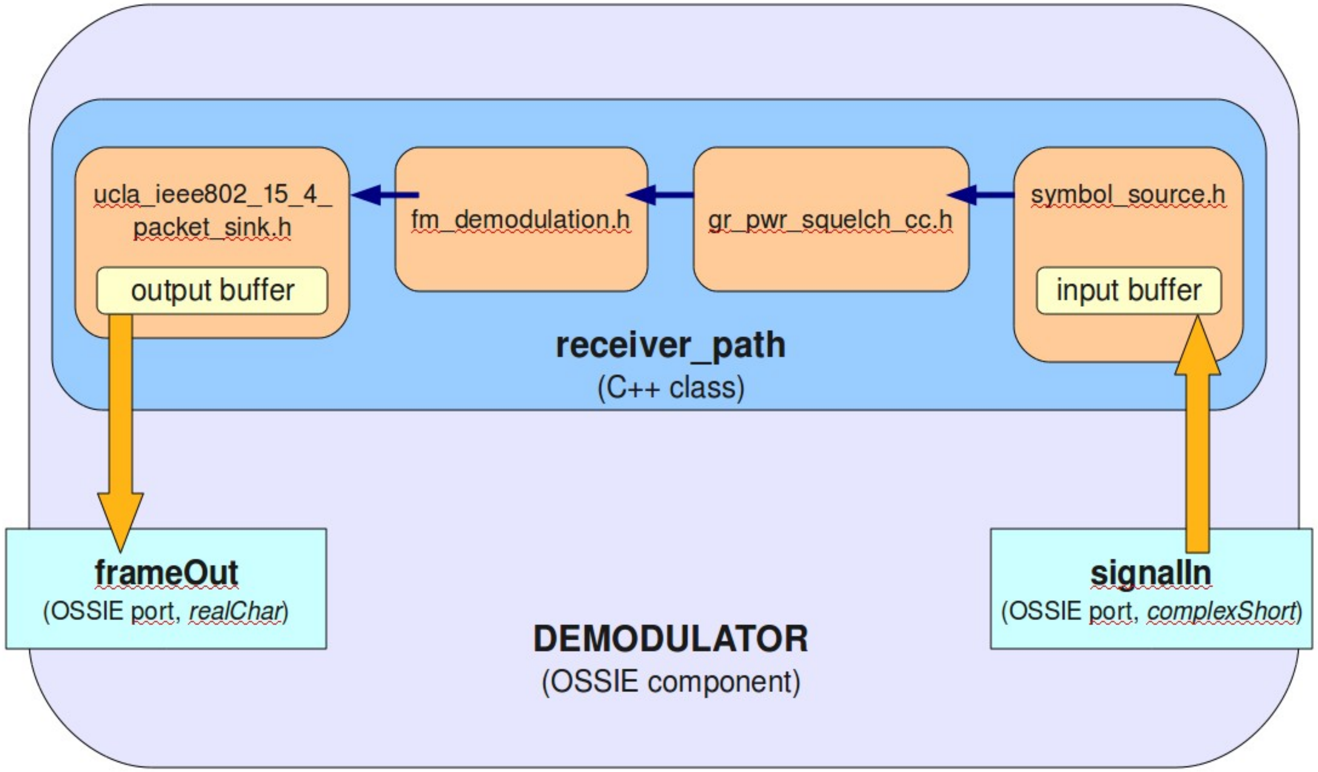


MODULATOR component



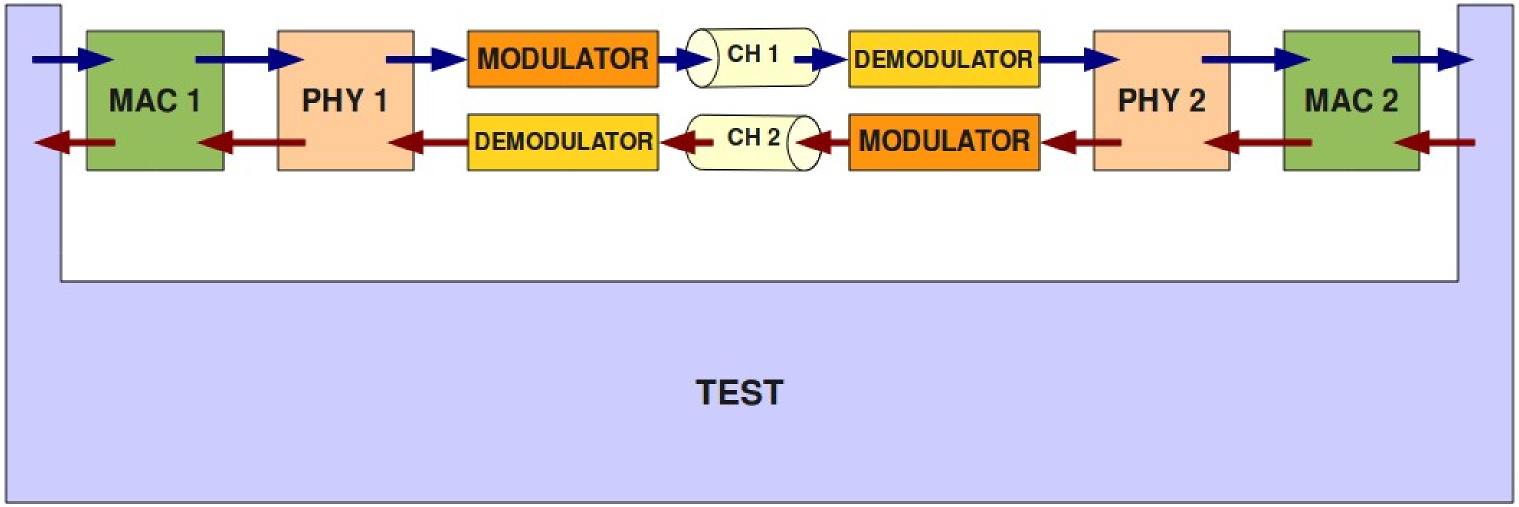
- Python implementation
- `mylib.symbol_sink`
- `ieee802_15_4_mod`
- input/output buffers

DEMODULATOR component



- C++ implementation
- `fm_demodulation.h`
- `symbol_source.h`
- input/output buffers

Waveform Simulation

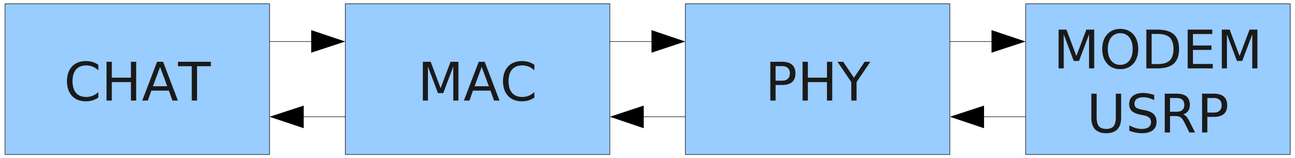


- all components deployed on the GPP (no USRP)
- radio channel simulated by the *channel* component
- test case implemented in the TEST component

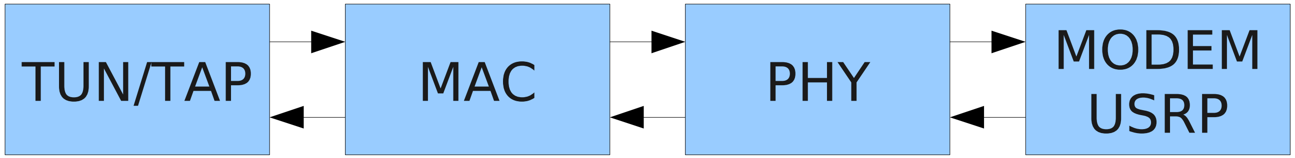
The target waveform

Two configurations:

- a chat application



- a TUN/TAP interface



GNU Radio vs. OSSIE

GNU Radio	OSSIE
Waveform structure: • the `top_module` class represents a container for the flowgraph • the flowgraph is then implemented as a single thread	• defined in the XML descriptor files and instanciated at runtime • one or more threads for each component
Component interconnections: • defined in the source code with the `connect()` method • functional de-composition and hierarchical structure	• defined in the XML descriptor files and instanciated at runtime • functional composition but flat structure
Data transfer: • if data is available in the input buffer, the `top_module` calls the `work` method • synchronization and scheduling tecniques	• performed by CORBA calls, data is temporarely stored in input/output buffers • no synchronization between components, independent threads

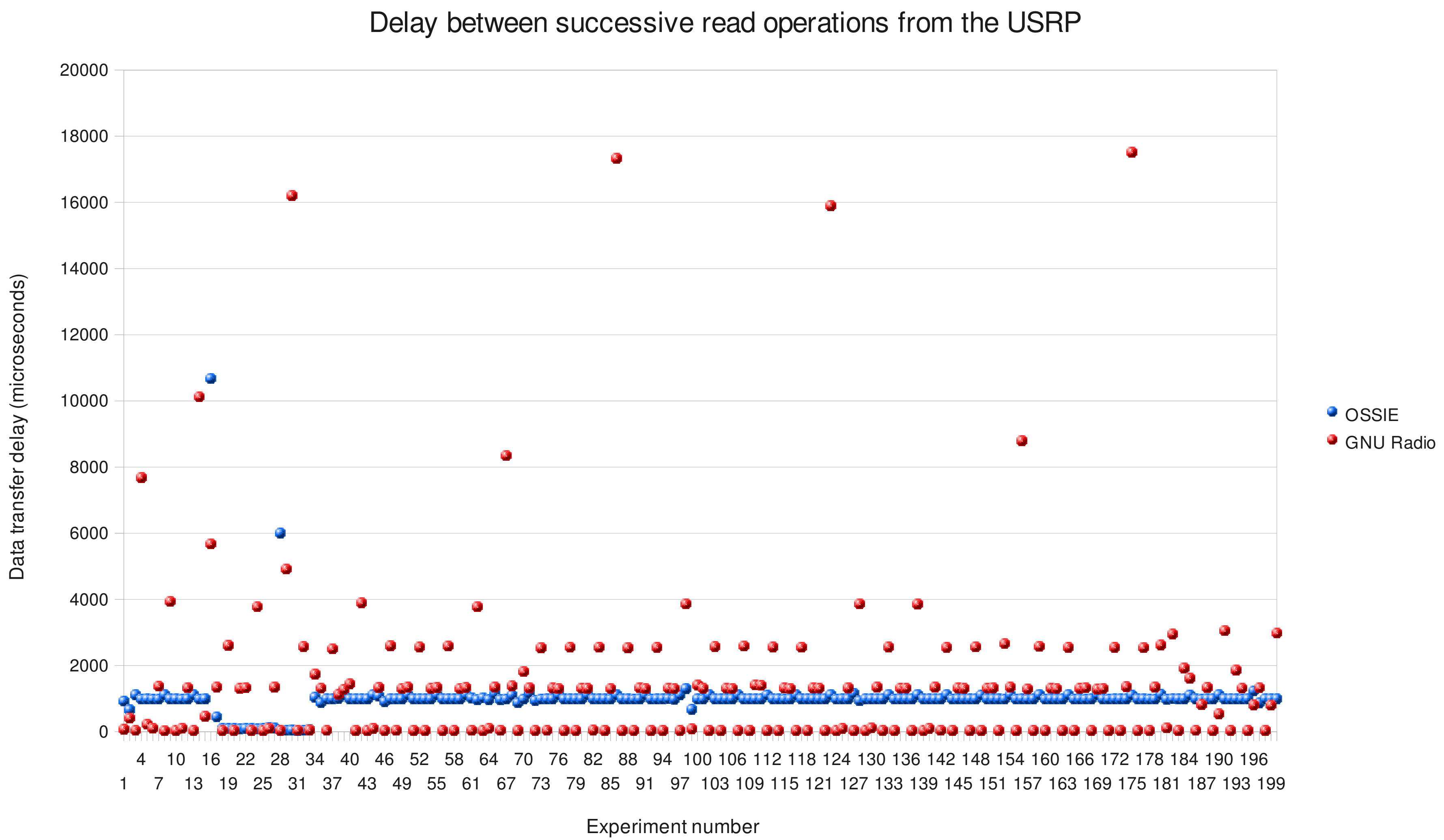
Measurements

	GNU Radio	OSSIE
Average data transfer size	12.5 KB	16 KB
Average data transfer delay	1.5 ms	1 ms
Average input data rate	8 MB/s	15.4 MB/s

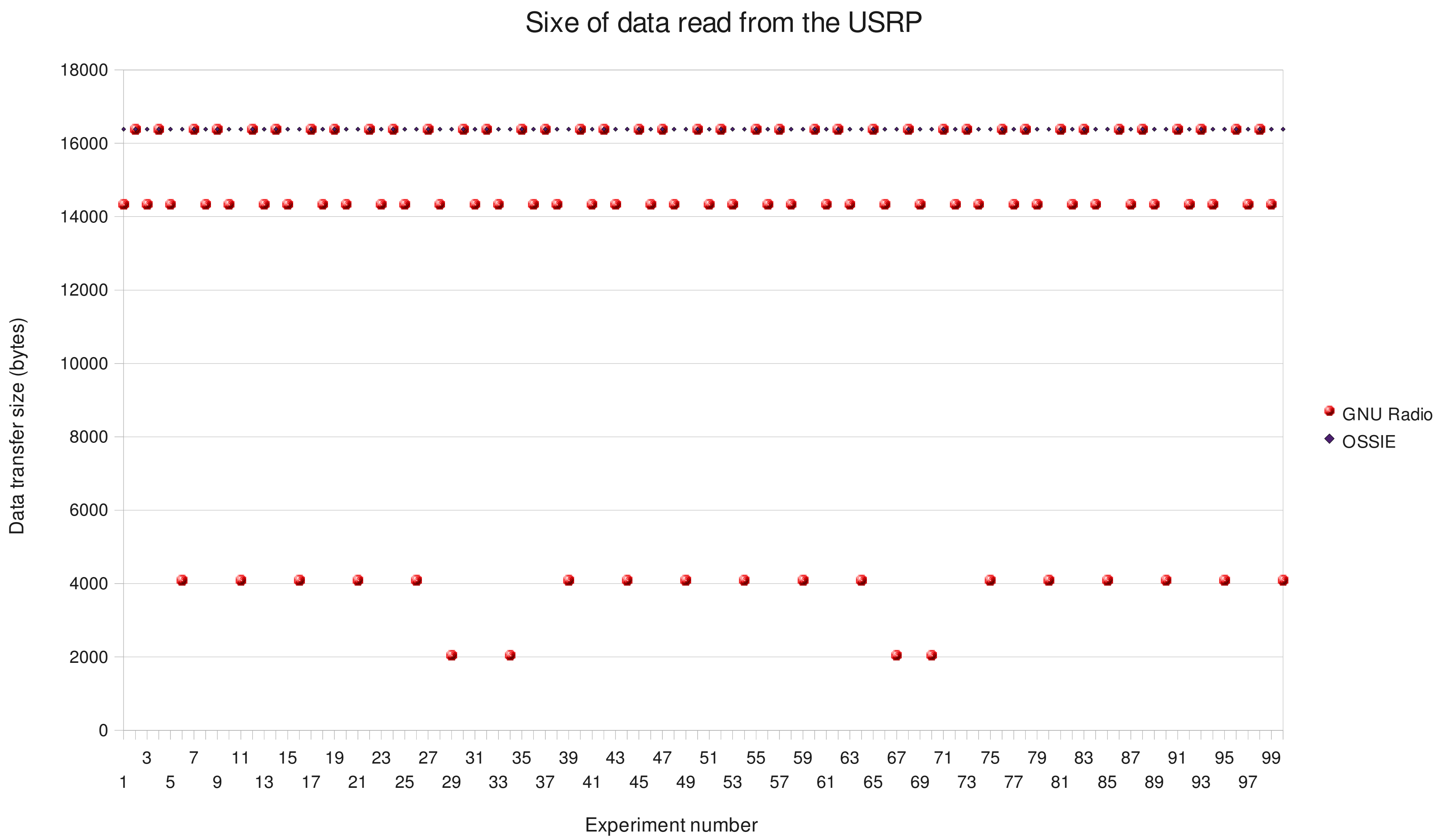
USRP:

- ADC sample rate: 64 Msps
- Decimation factor: 16
- Output sample rate: 4 Msps
- Theoretical limit USB 2.0: 8 Msps

USRP – Data transfer delay



USRP – Data transfer size



Results

- GNU Radio:
  - Adaptive scheduling
  - Variable data rate
  - Data transfer synchronization

- OSSIE:
  - Independent scheduling for each component
  - Constant data rate
  - No synchronization between components

Thank you for your attention

Acknowledgements

SAAB TECHNOLOGIES SAAB