Adopting WINNF Transceiver Facility for Spectrum Sensors

Tomaž Šolc, Jožef Stefan Institute
Unified transceiver API

- CREW uses a large variety of transceiver hardware
  - True SDR nodes (Ettus Research USRP, WARP)
  - Spectrum sensors (SNE-ESHTER, Imec Sensing Engine)
  - [Low-power, narrow band radios on embedded devices]

- Each transceiver has its own native interface
Unified transceiver API (cont.)

- To perform an experiment, testbed users:
  - develop an application running on nodes,
  - remotely upload and start the application and
  - perform measurements using test instrumentation.
- One API for all radio hardware would simplify
  - application development for testbed users and
  - portability of an experiment between testbeds
- WINNF Transceiver Facility
  - selected as an API for SDR nodes early in the project,
  - could be also used for access to spectrum sensors?
WInnF Transceiver Facility

Waveform application

Transceiver API

Transceiver

MAC

Modem

Digital frontend

Analog frontend

R/W properties access

Transmit Channel

Receive Channel

Base-band Signal

RF Signal

Real-time control

**SDR** vs. **Spect. sensor**

- RX and TX
- Continuous reception/unlimited burst length
- Frequency agile, low-latency
- Fast turn on/turn off
- Optimized for signal processing in software
- RX only
- Limited buffer for a sample-process cycle
- Uses a predetermined sequence
- Continuous scanning of a frequency band
- Optimized for on-board signal processing
Compact, low-cost spectrum sensor for VHF/UHF frequencies

- Selectable sensing bandwidths from 8 MHz to 500 kHz
- Baseband signal capture (up to 2 Msample/s, 25000 samples)
- Statistical processing on sensor node CPU (covariance, Eigenv. det.)
- Low-latency programmable hardware trigger for energy detection
- Compressive and multi-antenna, multi-channel sensing

SNE-ESHTER
(embedded sensing hardware for TVWS experimental radio)
RF analog front-end

42 – 870 MHz single-conversion, low-IF receiver

analog energy detector
**Typical setup in a testbed**

SNE-ESHTER (RF analog front-end)

SNE-ESHTER (slave)

SNE-ESHTER (master)

SNC

VESNA Sensor Node Core
(56 MHz ARM CPU, ADC)

RS-232 serial
(or Ethernet with TCP/IP)

Host PC running
GNU/Linux

PC
Throughput: 

- **SNE-ESHTER**: Analog front-end
- **SNC (STM32F103)**: Cortex-M3 CPU, f = 56 MHz, 64 kB SRAM
- **SNR-ETH**: DigiConnect ME
- **Host PC**

**Connections and Throughput Rates**:
- **A/D** to **BUS**: 32 Mbit/s
- **BUS** to **USART**: 576 kbit/s
- **USART** to **DigiConnect ME**: 230 kbit/s (Ethernet connection)
- **DigiConnect ME** to **Host PC**: (direct serial connection)

Throughput (for calculating 25 element sample covariance vector): ~2 Mbit/s
Implementation overview

- C++ library, linked with user application
  - Transceiver Facility specification is language agnostic.
  - When changing hardware, just link with a different library.
- Running on the Host PC
  - No need to reprogram sensor's firmware,
  - but limited by the serial interface, host PC clock.
Latency benchmarks

- From `receiveStartTime` event to `pushBBSamples` call

```
1441102358141563854 ns
createReceiveCycleProfile() enter
1441102358141944585 ns
createReceiveCycleProfile() exit
1441102358625030773 ns
pushBBSamplesRX() enter
...
```
Latency measurement with scope

- createReceiveCycleProfile() receiveStartTime event
- pushBBSamplesRX()
- sensor starts sending signal samples
- computer sends configuration, »sample on« command

[Diagram showing oscilloscope traces with annotations]
Histogram of measurements

- Measured using the host PC clock
  - inaccurate due to task switching, etc.
• **Latency approximately 570 ms** (for 2048 samples)
  - Frontend slow to resume from power saving mode.
  - Sending ASCII formatted samples over 576 kbps RS-232 connection – latency depends on packet size

• Blind time of sensor in this mode > 99.5%
Improving latency

- Keep front-end constantly powered-up
  - No calibration for each RX cycle, but increased power usage.
- Send samples in binary format instead of ASCII
  - Higher bitrate due to lower CPU usage, no ASCII overhead,
  - but complicates sensor interface, backwards incompatible.

- **frontend power-up, calibration**: 290 ms
- **RS-232 send samples**: 270 ms
  - **Total**: 270 ms

- **Internal latency**: 62 ms
  - **Total**: 53 ms
Exploiting on-board processing

...  
Transceiver::ULong packet_size = 2048;
Transceiver::UShort tuning_preset = 2;
Transceiver::Function func = Transceiver::covarianceFunction;

device->receiveChannel.createReceiveCycleProfile(
    start, stop, packet_size, tuning_preset, freq, func);

nullFunction, covarianceFunction, fftFunction,
...

Application

processed baseband signal

f(x)
Timing information

- Add a timestamp/stream offset to `BBPacket` class
  - defines timing in case of sample loss.

```cpp
class BBPacket
{
  public:
    ULong SampleNumber;
    BBSample* packet;
    ULong time;
    BBPacket(ULong inSamplesNumber, BBSample* inPacket);
};
```
Conclusions

- Transceiver Facility has been adopted for spectrum sensors in CREW project testbeds.
- Latency with SNE-ESHTER hardware is ~570 ms.
- Support for spectrum sensors could be improved with minor changes of Transceiver Facility.
  - Support signal processing in hardware,
  - add timing information to packets of signal samples.
- Time will tell whether this abstraction is useful for use in testbeds.
Questions?

Tomaž Šolc
tomaz.solc@ijs.si

http://www.crew-project.eu