Lab 5: Time Demux

Using Multiple OpCodes in RCC Workers

Objectives

- Learn how [RCC] Workers can:
    - Use different Protocols on different Ports
        - Nonstandard input:output port ratio (1:2 vs. 1:1)
    - Process incoming data based on message type (OpCode)
        - I/Q Data with Timestamps

- Reiterate:
    - C++ conventions
        - Accessing Port data and Properties
    - Framework interactions
        - `RCC_ADVANCE` vs. `RCC_OK`

Overview

- The “Time Demux” component receives I/Q sample data that has time stamps interleaved within it. This component recovers the original data stream and writes it out while also providing a second stream containing only the timestamps.

- Having data separated allows additional processing by tools expecting “just data,” *e.g.* plotting an FFT

Application Worker Development Flow

1. Protocol (OPS): Create new or select pre-existing
2. Component (OCS): Create new or select pre-existing
3. Create new App Worker (Modify OWD, Makefile, and source RCC/HDL code)
4. Build the App Worker for target device(s)
5. Create Unit Test (<component>-test.xml, generate, verify and view scripts)
6. Build Unit Test
7. Run Unit Test

Step 1: Protocol

- Review the component's datasheet and familiarize yourself with the two protocols the component will be using:
  - The incoming *combined* (in-band timestamp) data format is described in `iqstream_with_sync_protocol.xml`, found in `<Core Project>/specs`.
  - The outgoing ***data*** format is described in `iqstream_protocol.xml`, found in `<Core Project>/specs`.
  - The outgoing ***time*** format is also *iqstream_with_sync*.

Step 2: Component

- Create the Component Specfile (OCS) based on the datasheet’s Properties and Ports

- Notables:
  - *Volatile* flags on all status Properties
  - Most attributes don’t need to be set if “False”
  - Output ports are *Producers*

Step 3: Worker

- Create an RCC Worker (OWD) based on the OCS
  - This lab will use C++
  - Enable the “start” ControlOperations

- Test the build with the “empty” worker using IDE
  - You should be familiar at this point
  - What to expect:
    - `Linking final artifact file <X> and adding metadata to it…`

Step 3 (cont.)

- Copy `time_demux.cc` from `/home/training/provided/lab5` over your skeleton
- Fill in the logic, noting (read the next slide too):
  - `gen/time_demux-worker.hh` exposes a lot of information, including enums
  - C++ data access
    - Sequences: <PortName>.<OpName>().<ArgName>()`.data()` and `.size()`
    - Scalars: <PortName>.<OpName>().<ArgName>()
  - On output ports:
    - <PortName>.<OpName>().<ArgName>().`resize()` counts data *elements*
    - `setLength()`/`setDefaultLength()` tell show much to send in *bytes*
    - `setOpCode()`/`setDefaultOpCode()` set the opcode
  - Use `RCC_OK`, not `RCC_ADVANCE`
    - manually `advance()` ports if/when used

Step 3 (cont.) - Additional Hints

- A helper function like `increment_counters` is useful
  - Isolates *Messages_Read* and *Bytes_Read* Properties' logic
- Example `start()` method guarantees only *Time* opcode is ever sent
- Search for "???"
- **Cannot directly copy I/Q Data structures across different protocols!**
- *Most* C++11 constructs can be used by adding to Worker-level Makefile:
  - `RccExtraCompileOptionsCC_centos6=--std=c++0x`
  - `RccExtraCompileOptionsCC_centos7=--std=c++11`
  - `RccExtraCompileOptionsCC_xilinx13_3=--std=c++11`

Step 4: Build Worker

- Build the Worker for the host and Matchstiq (using IDE)

Step 5: Create Unit Test

- Copy “`time_demux.test`” into components library of your training project from the “`/home/training/provided/lab5/`” directory

Step 6: Build Unit Test

- Build the test
  - (Use standard test procedures as in other labs)

Step 7: Run Unit Test on the Host

- Run the tests

- Property values are not verified
  - Can be found in `run/centos7/case00.00.time_demux.rcc.propsfile_read`
  - Manually verify that UUT’s messages and bytes read match file_read’s
  - *Did you count timestamps as a single 64-bit payload?*
  - *Current_Second* should be close to

start time + bytes read ÷ (samples per second × 4)

Step 7b: Run Unit Test on Matchstiq-Z1

- (Use standard test procedures as in other labs)