IPC LAB 1

ex01_helloworld

Version 1.02

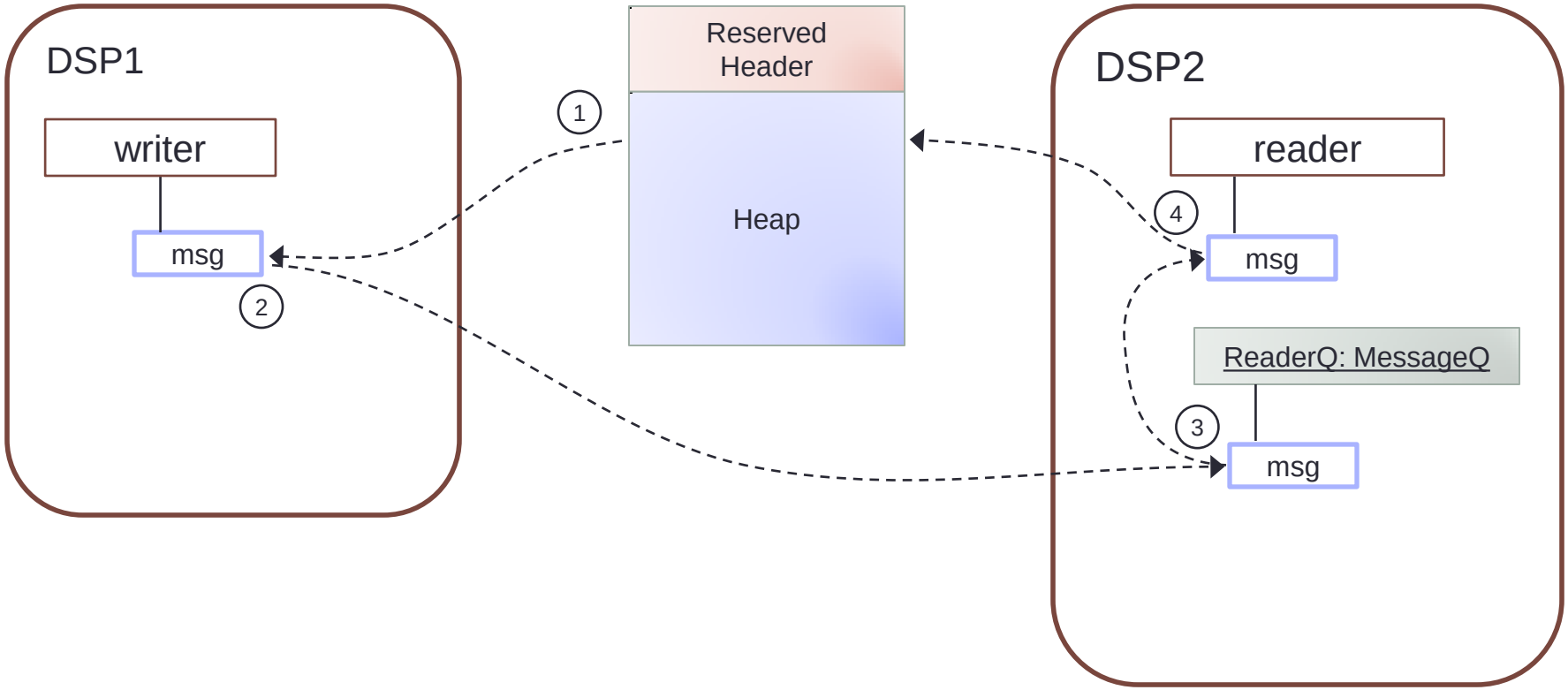
Overview

- This is the "Hello World" example for IPC.
- Goals
  - Learn how to build an IPC example
  - Setup a CCS Target Configuration for Vayu
  - Load and run the processors
  - Use the RTOS Object Viewer (ROV) to inspect IPC modules

ex01_helloworld

- This is a two processor example. You can build it for any two processors on your device, but only for two at a time.
- You will assign one processor the role of “reader" and the other the role of "writer".
- The reader will create a message queue and wait for messages.
- The writer will open the message queue and send messages.
- The heap in shared region #0 is used for the message pool.

Data Flow



Step 1 – Work Area

- Create a work folder for this lab

  `C:\TI_Demo`

- Extract the example into the work folder

  `<ipc_3_30_pp_bb>\examples\DRA7XX_bios_elf\ex01_hello.zip`

Step 2 — Build Environment

- Set the product install paths as defined by your physical environment.
  - Edit `ex01_hello/products.mak`

```
DEPOT = C:/Products
IPC_INSTALL_DIR  = $(DEPOT)/ipc_m_mm_pp_bb
BIOS_INSTALL_DIR = $(DEPOT)/bios_m_mm_pp_bb
XDC_INSTALL_DIR  = $(DEPOT)/xdctools_m_mm_pp_bb
```

- Set the tool paths (only need the ones you actually plan to use).
  - Edit `ex01_hello/products.mak`

```
CCS = C:/CCS/CCS_6_0_0_00190/ccsv6/tools/compiler
gnu.targets.arm.A15F           = $(CCS)/gcc_arm_none_eabi_m_m_p
ti.targets.elf.C66             = $(CCS)/c6000_m_m_p
ti.targets.arm.elf.M4          = $(CCS)/arm_m_m_p
ti.targets.arp32.elf.ARP32_far = $(CCS)/arp32_m_m_p
```

- Each example has its own products.mak file; you may also create a products.mak file in the parent directory which will be used by all examples.

Step 3 — Build Executables

- Open a Windows Command Prompt

```
Start > Run
cmd
```

- TIP: Use the following command to create an alias for the make command

```
doskey make="C:\Products\xdctools_3_30_04_52\gmake.exe" $*
```

- TIP: Use dosrc.bat to setup your build environment
  - `<ipc_3_30_pp_bb>/examples/dosrc.bat` — copy to your work folder
  - Edit dosrc.bat, set product paths
  - Run script in your command prompt
- Build the example

```
cd ex01_hello
make
```

- The executables will be in their respective "bin" folders

```
ex01_hello\dsp1\bin\debug\hello_dsp1.xe66
ex01_hello\dsp2\bin\debug\hello_dsp2.xe66
```

Step 4 – CCS Target Configuration (1/2)

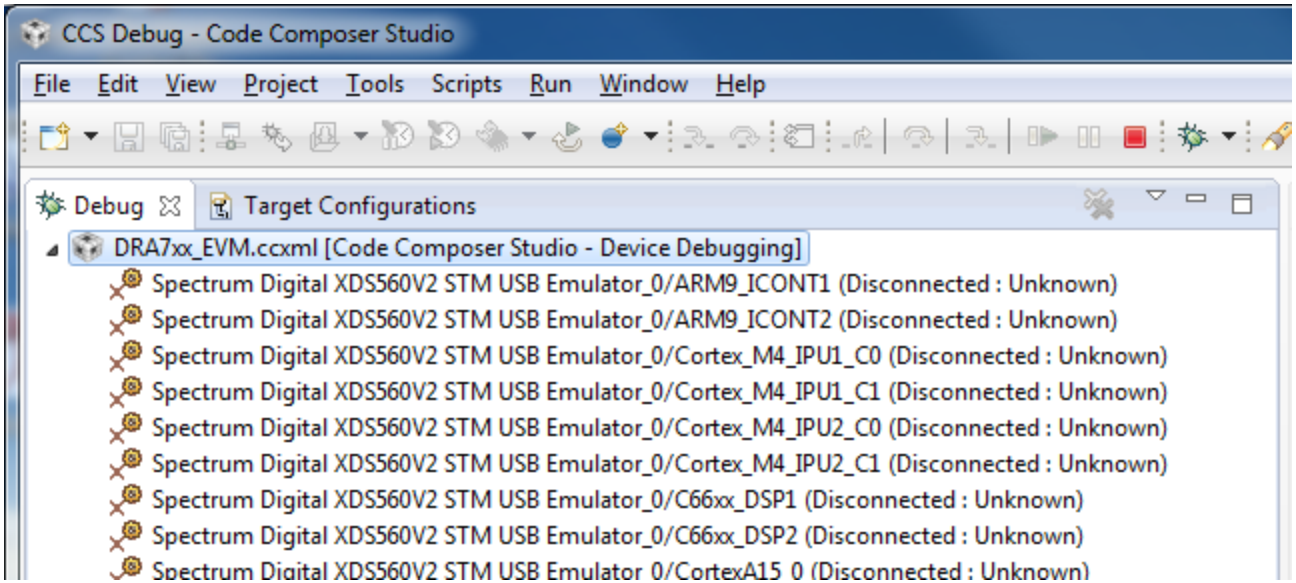
- Create an empty project. We will use this to store the target configuration.
  - File > New > Project...
  - General > Project
  - Next
  - Project name: TargetConfiguration
  - Finish
- Create a new target configuration file.
  - File > New > Target Configuration File
  - File name: DRA7xx_EVM
  - Use shared location > Unselect
  - Workspace...
  - Select the project you just created
  - OK
  - Finish

Step 4 – CCS Target Configuration (2/2)

- Setup the new target configuration
  - Connection: Spectrum Digital XDS560V2 STM USB Emulator
  - Board or device: DRA7xx
  - Save
- Lower JTAG clock speed (optional)
  - Advanced Setup > Target Configuration
  - Spectrum Digital XDS560V2 STM USB Emulator_0 > Select
  - JTAG TCLK Frequency > Automatic with legacy 10.368 MHz limit
  - Save

Step 5 – Launch Debug Session

- Set new target configuration as default
  - Open Target Configuration view (View > Target Configurations)
  - Projects > TargetConfiguration > DRA7xx_EVM.ccxml
  - RMB > Set as Default
  - RMB > Launch Selected Configuration
- Open Debug view. You should see a list of processors.

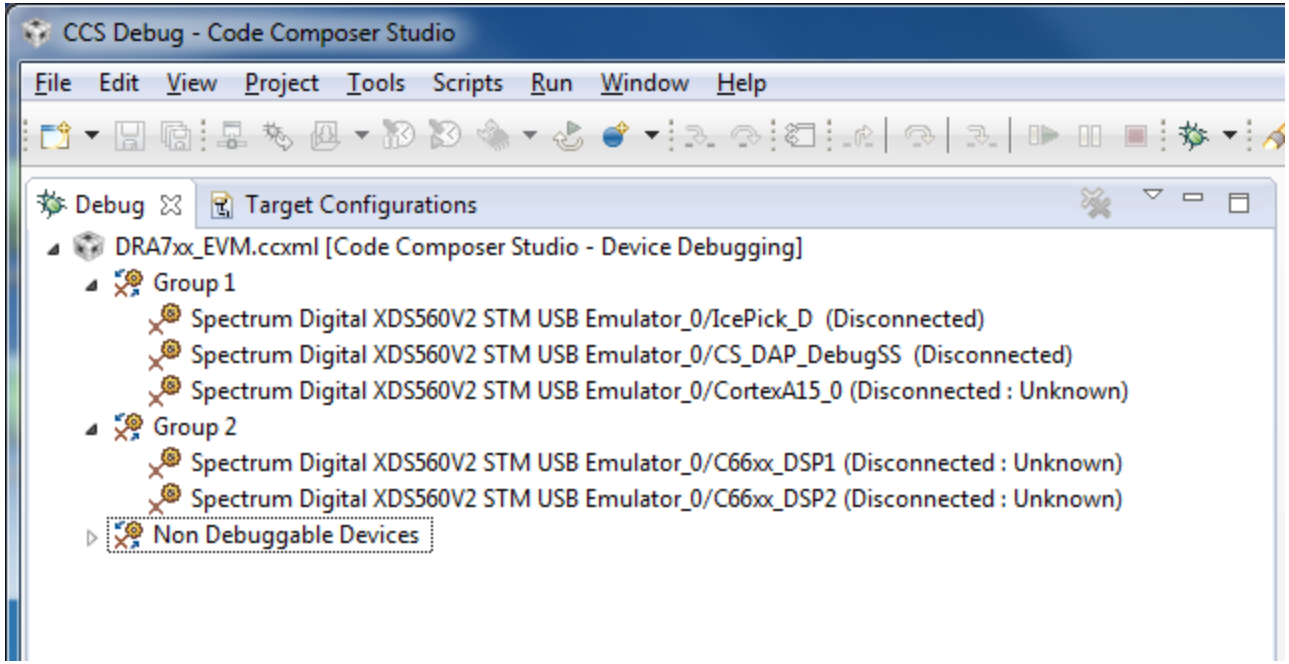


Step 6 – Group Processors (1/2)

- You will need to use the non-debuggable processors. These are not visible by default.
  - In Debug View > RMB > Show all cores
  - Open the Non Debuggable Devices group
- Group the following devices. These are used for system control.
  - Select IcePick_D, CS_DAP_DebugSS, CortexA15_0
  - RMB > Group cores
- Group the processors used by your example.
  - Select C66xx_DSP1, C66xx_DSP2
  - RMB > Group cores
- Hide the remaining processors. This removes clutter form the debug view.

Step 6 – Group Processors (2/2)

- Your Debug view should look like this.



Step 7 – Connect to Processors

- I recommend you connect to the IpcPick first and issue a system reset.
  - Debug view > IcePick_D > RMB > Connect Target
  - Scripts > ICEPick_D_Utility > SystemReset
- You must connect to the CortexA15_0 first. This will automatically run GEL scripts to enable the device.
  - Debug view > CortexA15_0 > RMB > Connect Target
- Connect to DSP1
  - CortexA15_0 > Select
  - Scripts > DRA7xx MULTICORE Initialization > DSP1SSClkEnable_API
  - C66xx_DSP1 > RMB > Connect Target
  - Run > Reset > CPU Reset (or use toolbar icon, )
- Repeat previous step for DSP2. Remember to select the CortexA15_0 before running the GEL script.

Step 8 – Load Processors

- Run the host processor. This is required to enable the DSPs to reach main when loaded (timers are halted when host is halted).
  - Debug view > CortexA15_0 > Select
  - Run > Run (or use toolbar icon, )
- Load DSP1 with the executable you just built.
  - Select C66xx_DSP1
  - Run > Load > Load Program (or use toolbar icon, )
  - Click Browse, select the DSP1 executable

    `C:\TI_Demo\ex01_hello\dsp1\bin\debug\hello_dsp1.xe66`
- You should see DSP1 run to main and then stop.
- Load DSP2 using the same procedure. Be mindful to load the proper executable.

  `C:\TI_Demo\ex01_hello\dsp2\bin\debug\hello_dsp2.xe66`

CCS Auto Run Configuration (optional)

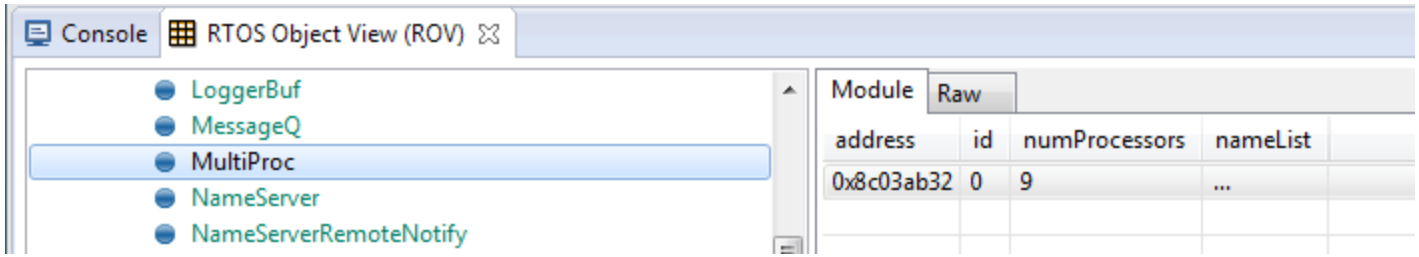
- By default, when you load a program, CCS will run to the function main and then stop. You can disable this with the Auto Run option. This comes in handy when there is a bug in the boot code.
  - Target Configurations
  - Projects > TargetConfiguration > DRA7xx_EVM.ccxml
  - RMB > Properties
  - Device (menu) > C66xx_DSP1
  - Auto Run and Launch Options > Select
  - Auto Run Options (group) > On a program load or restart > Unselect
  - Use the Device pull-down menu to select the next processor. Repeat for each processor.
- The changes above will not effect the current debug session (only subsequent ones). Use these steps to modify the current session.
  - Debug View
  - Select C66x_0
  - Tools > Debugger Options > Auto Run and Launch Options
  - Auto Run Options (group) > On a program load or restart > Unselect
  - Click Remember My Settings to make this change permanent

Step 9 — Run to Main

- If you disabled the Auto Run option, you should see the processor at the `_c_int00` entry point. Run the processor to main.
  - Select the processor in the Debug view
  - Run > Go Main
- Run both processors to main. You should see the source code in the editor view.

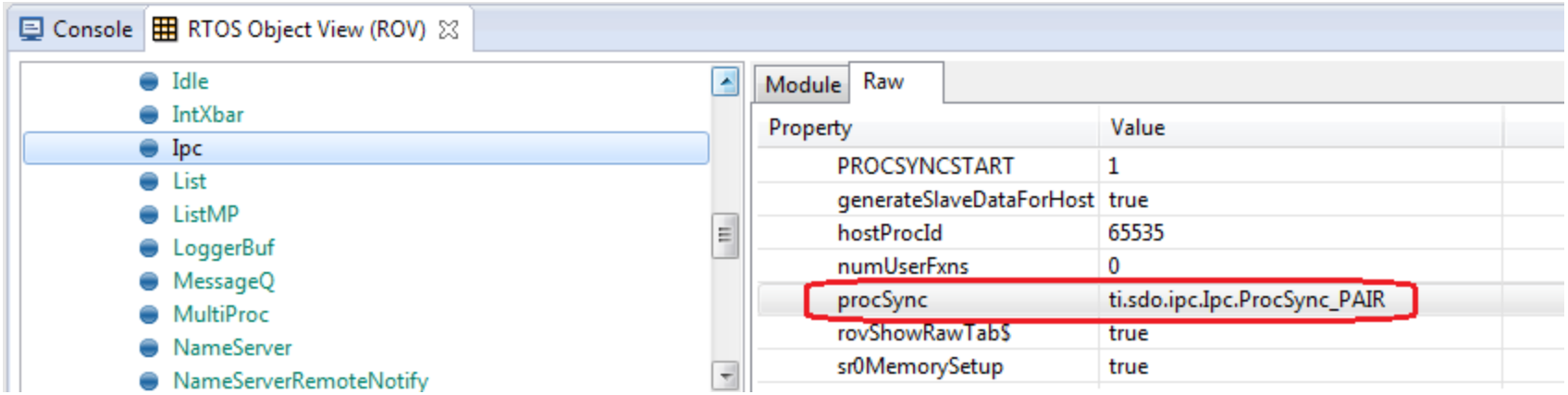
RTSC Object View (ROV)

- Once you reach main, you can use ROV to inspect the target data.
  - Debug view > C66xx_DSP1 > Select
  - Tools > RTOS Object View (ROV)
- TIP: Dock the ROV view along the bottom edge. Then maximize it.
- Inspect the MultiProc module
  - Select MultiProc in the object browser (on the left)
  - Select the Module tab (on the right)



ROV – Ipc Module

- Select the Ipc module in ROV. On the Module tab you will see a list of all the other processors in the system. Notice that the attached column shows 'false'. That is because we have not yet made it through `Ipc_attach`.
- What is our Ipc attach mode? Click on the Raw tab. Expand the Configuration object. Use the Auto Fit tool bar button ( ) to adjust the column sizes. The information displayed here reflects the configuration for the module.
- Scroll down and look for the procSync config param. You should see it was set to `ti.sdo.ipc.Ipc.ProcSync_PAIR`. This is a handy way to inspect your configuration values.



Step 10 — Break after Ipc_attach

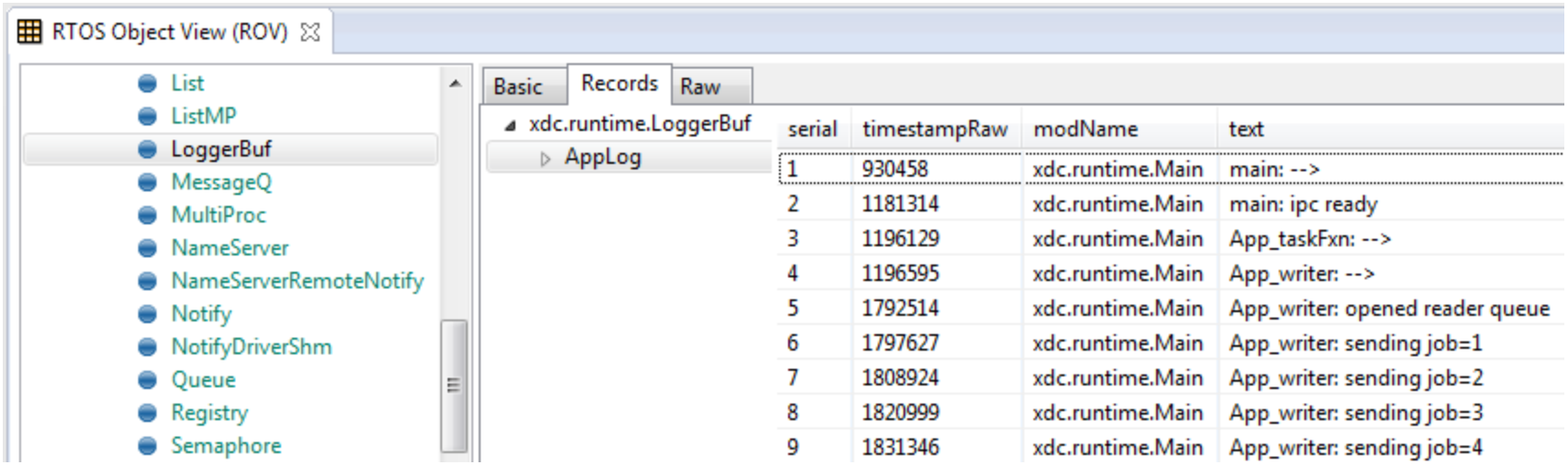
- Let's use breakpoints to run each processor through `Ipc_attach`. In the Debug view, select `main` under C66xx_DSP1. Scroll down to the function App_taskFxn. Set a breakpoint in the source window just after the Ipc_attach loop. Observe the new breakpoint in the Breakpoints view. Use the pop-up context menu.
- Do the same for DSP2.
- TIP: It helps to name your breakpoints to keep track of them.
  - In Breakpoint view, RMB on breakpoint
  - Breakpoint Properties...
  - Edit the name field
- Run DSP1. Notice it does not hit the breakpoint. It is spinning in the attach loop.
- Run DSP2. After a short run, both processors will hit their respective breakpoints.
- Inspect the Ipc module in ROV. In the attached column, you should see 'true' for the respective peer processor.

Step 11 – Run to Completion

- Set a breakpoint on the last line of App_taskFxn. Do this for both processors.
- Run both processors
  - Select the processor group in the Debug view
  - Run > Run (or use toolbar icon, )
- After a short run, both processors should hit their breakpoints.

ROV – LoggerBuf Module

- When the example completes, use ROV to inspect the LoggerBuf module to see the log events.
  - Debug view > C66xx_DSP1 > Select
  - RTOS Object View (ROV) > LoggerBuf > Select
  - Records (tab) > Select
  - AppLog > Select (don't open it)
- You will see a list of log events. Use the Auto Fit Columns ( ) button if necessary.



Example Reloaded

- Use the following sequence to reload the example.
  - C66xx_DSP1 > Select
  - CPU Reset
  - Run > Load > Reload Program (or use toolbar menu, )
- Repeat the previous step for DSP2
- You are now ready to run the example again.

Rebuild with Different Processors

- To rebuild the example with a different processor pair, you need to edit the top-level makefile. Modify the PROCLIST macro to specify which processors to build.
  - Edit `ex01_hello/makefile`

    ```
    PROCLIST = dsp1 eve1
    ```

- Next, you need to edit the source file for each processor and specify its role and name its peer.
  - Edit `ex01_hello/dsp1/HelloDsp1.c`

    ```
    Int role = Role_WRITER;
    String peer = "EVE1"
    ```

  - Edit `ex01_hello/eve1/HelloEve1.c`

    ```
    Int role = Role_READER;
    String peer = "DSP1";
    ```

- Remember to delete the error directive.

  ~~`#error Must define role and peer`~~

Running on IPU Processor (1/3)

- Load GEL file
  - Cortex_M4_IPU1_C0 > Select
  - Tools > GEL Files
  - GEL Files (view) > GEL Files Panel (right side) > RMB > Load GEL...
    `ex01_hello/ipu1/ex01_hello_ipu1.gel`
- Connect to IPU1
  - CortexA15_0 > Select
  - Scripts > DRA7xx MULTICORE Initialization > IPU1SSClkEnable_API
  - Cortex_M4_IPU1_C0 > RMB > Connect Target
  - Run > Reset > CPU Reset
  - Cortex_M4_IPU1_C1 > RMB > Connect Target
  - Run > Reset > CPU Reset
- Program AMMU
  - Cortex_M4_IPU1_C0 > Select
  - Scripts > ex01_hello > ex01_hello_ipu1_ammu_config

Running on IPU Processor (2/3)

- Run host processor
  - CortexA15_0 > Select
  - Run > Run
- Load IPU1_C0 with program
  - Cortex_M4_IPU1_C0 > Select
  - Run > Load > Load Program...
  - Click Browse, select the IPU1 executable

    `ex01_hello\ipu1\bin\debug\hello_ipu1.xem4`
- Load symbols on IPU1_C1. With SYS/BIOS SMP, you only need symbols on the second core.
  - Cortex_M4_IPU1_C1 > Select
  - Run > Load > Load Symbols...
  - Use same file as above. Usually, its already selected in dialog box.
- Restart IPU1_C1
  - Run > Restart

Running on IPU Processor (3/3)

- Run IPU1_C1. It will just spin until Core 0 calls BIOS_start.
  - Cortex_M4_IPU1_C1 > Select
  - Run > Run
- Run IPU1_C0 to main
  - Cortex_M4_IPU1_C0 > Select
  - Run > Go Main
- You are now ready to proceed with the example.

Running on EVE Processor (1/2)

- Load GEL file. Needed for programming the MMU.
  - CS_DAP_DebugSS > Select (must show all cores to see the DebugSS)
  - Tools > GEL Files
  - GEL Files (view) > GEL Files Panel (right side) > RMB > Load GEL...
    `ex01_hello/eve1/ex01_hello_eve1.gel`
- Connect to EVE1
  - CortexA15_0 > Select
  - Scripts > DRA7xx MULTICORE Initialization > EVE1SSClkEnable_API
  - CS_DAP_DebugSS > Select
  - Scripts > EVE MMU Configuration > ex01_hello_eve1_mmu_config
  - ARP32_EVE_1 > RMB > Connect Target
  - Run > Reset > CPU Reset

Running on EVE Processor (2/2)

- Run host processor
  - CortexA15_0 > Select
  - Run > Run
- Load program
  - ARP32_EVE_1 > Select
  - Run > Load > Load Program...
  - Click Browse, select the EVE1 executable

    `ex01_hello\eve1\bin\debug\hello_eve1.xearp32F`
- Run to main
  - ARP32_EVE_1 > Select
  - Run > Go Main
- You are now ready to proceed with the example.
  - Note: Run EVE processor first, before running the peer processor.

Running on HOST Processor

- Connect to HOST
  - CortexA15_0 > RMB > Connect Target
- Connect, load, and run to main the peer processor before loading program on HOST.
- Load program
  - CortexA15_0 > Select
  - Run > Reset > CPU Reset
  - Run > Load > Load Program...
  - Click Browse, select the HOST executable

    `ex01_hello\host\bin\debug\hello_host.xa15fg`
- Run to main
  - CortexA15_0 > Select
  - Run > Go Main
- Run HOST before running peer
  - Run > Run
- You are now ready to proceed with the example.

Extra Credit

- Here are some suggestions on extra credit tasks
- Inspect the SharedRegion module in ROV. Note the cache setting for SR #0.
- Inspect the MessageQ module in ROV. Use breakpoints before and after the reader creates the message queue. Look for the message queue in ROV.
- Set a breakpoint after the reader has received the first message. Let the writer continue. Use ROV to observe the messages in the queue.

Congratulations!
End of Lab 1