LAB 3 IPC SCALABILITY

ex13_notifypeer

Version 1.01

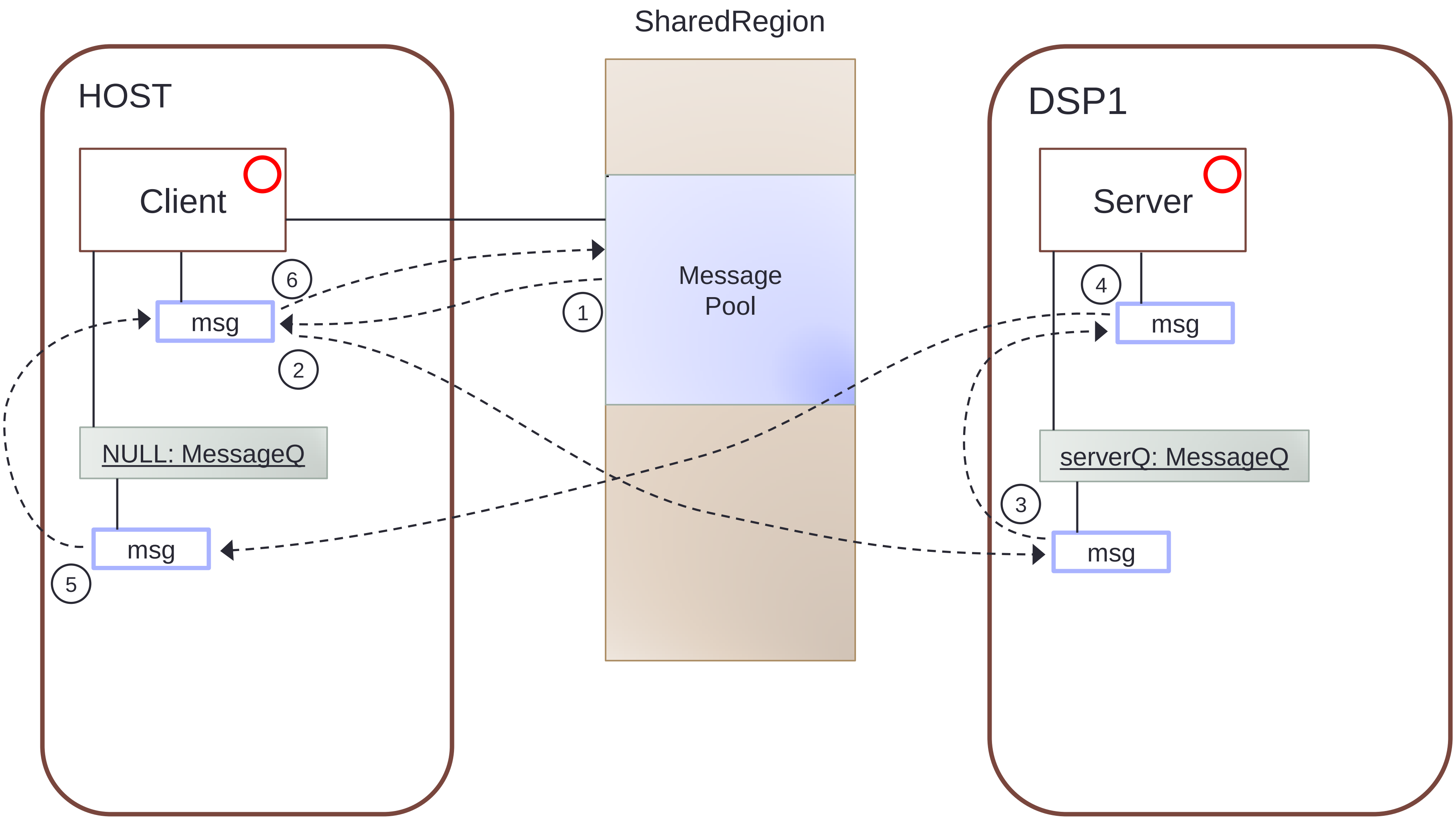
Overview

- This is an example of scaling IPC down to just Notify.
- Goals
  - Add an EVE processor to an existing two processor application.
  - IPC scalability, Notify only
  - Use a SYS/BIOS Event object to wait on two input sources

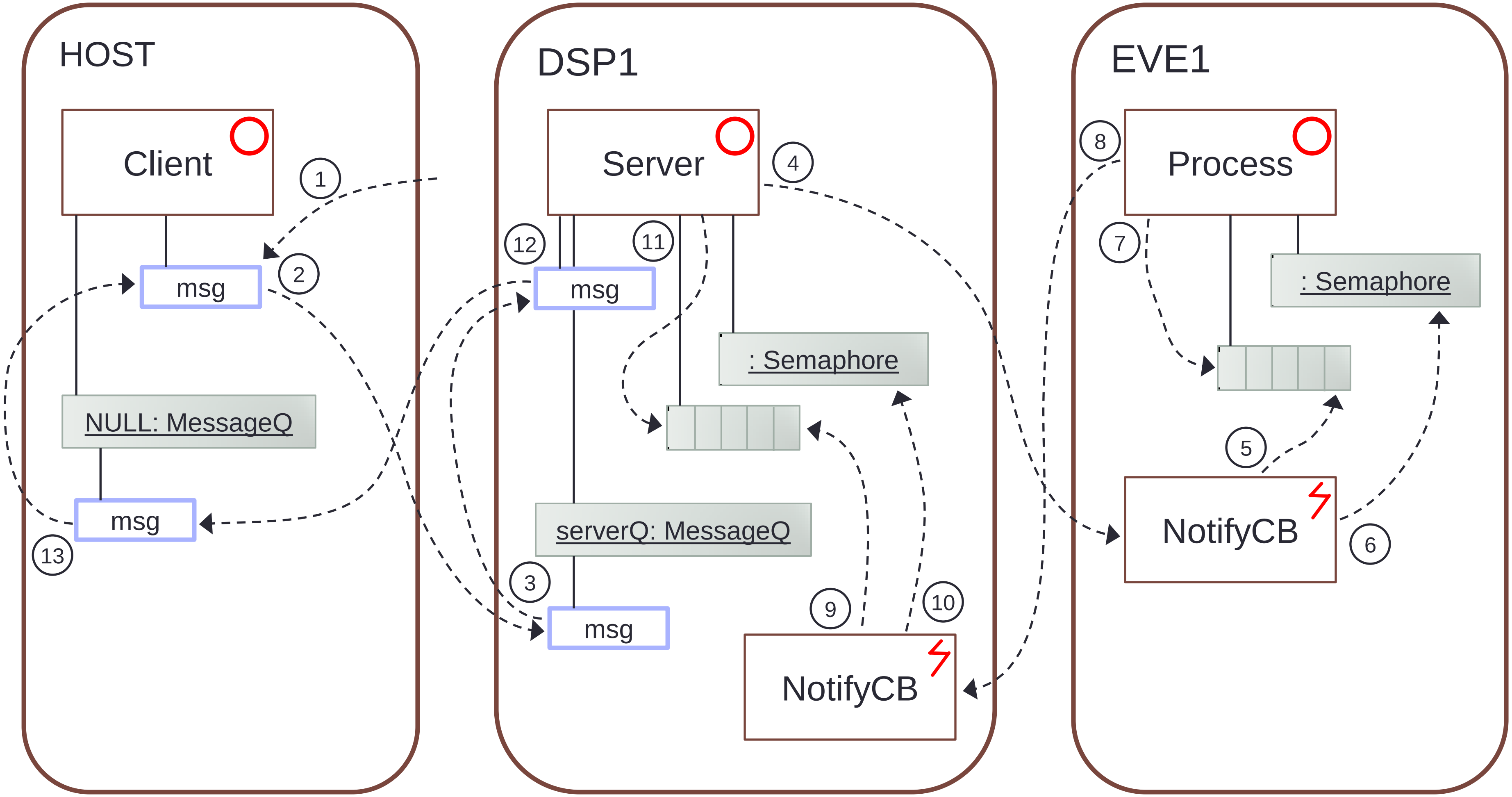
ex13_notifypeer

- Initial setup is for two processors; HOST and DSP1
- HOST uses message queue to send jobs to DSP1.
- You will modify DSP1 to forward the job to EVE1.
- You will add EVE1 to application using only IPC Notify.

Initial Data Flow



Final 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\ex13_notifypeer.zip`

Step 2 – Build Environment

- Set the product install paths as defined by your physical environment.
  - Edit `ex13_notifypeer/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 `ex13_notifypeer/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 ex13_notifypeer
make
```

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

```
ex13_notifypeer\host\bin\debug\app_host.xa15fg
ex13_notifypeer\dsp1\bin\debug\server_dsp1.xe66
```

CCS Auto Run Configuration

- Disable Run to Main in your target configuration.
  - 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.

Step 4 – Load Processors

- Load HOST with executable
  - Debug view > CortexA15_0 > Select
  - Run > Load > Load Program
  - Click Browse, select the HOST executable

    `ex13_notifypeer\host\bin\debug\app_host.xa15fg`

- Load DSP1 with executable
  - Debug view > C66xx_DSP1 > Select
  - Run > Load > Load Program
  - Click Browse, select the DSP1 executable

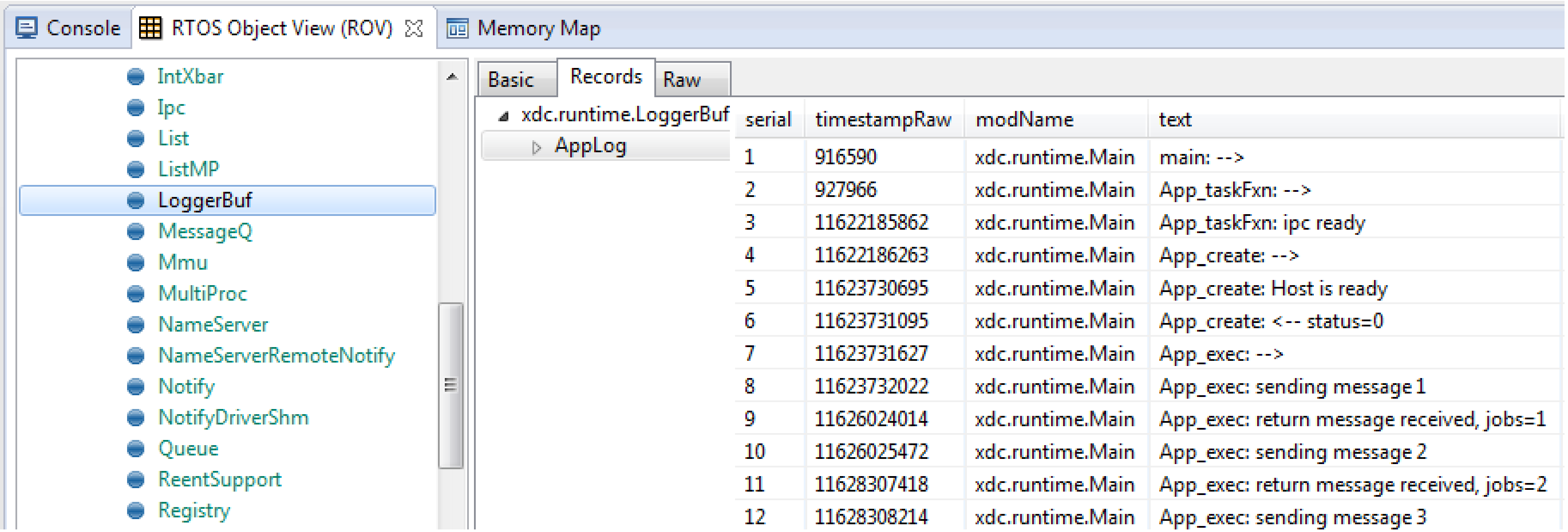
    `ex13_notifypeer\dsp1\bin\debug\server_dsp1.xe66`

Step 5 — Run the Example

- Run HOST processor
  - Debug view > CortexA15_0 > Select
  - Run > Resume
- Run DSP1 processor
  - Debug view > C66xx_DSP1 > Select
  - Run > Resume
- The example completes very quickly
- Halt DSP1 processor
  - Debug view > C66xx_DSP1 > Select
  - Run > Suspend
- Halt HOST processor
  - Debug view > CortexA15_0 > Select
  - Run > Suspend

ROV – LoggerBuf Module

- When the example completes, use ROV to inspect the LoggerBuf module to see the log events.
  - Debug view > CortexA15_0 > Select
  - RTOS Object View (ROV) > LoggerBuf > Select
  - Records (tab) > Select
  - AppLog > Select (don’t open it)
- You will see a list of log events.



serial	timestampRaw	modName	text
1	916590	xdc.runtime.Main	main: -->
2	927966	xdc.runtime.Main	App_taskFxn: -->
3	11622185862	xdc.runtime.Main	App_taskFxn: ipc ready
4	11622186263	xdc.runtime.Main	App_create: -->
5	11623730695	xdc.runtime.Main	App_create: Host is ready
6	11623731095	xdc.runtime.Main	App_create: <-- status=0
7	11623731627	xdc.runtime.Main	App_exec: -->
8	11623732022	xdc.runtime.Main	App_exec: sending message 1
9	11626024014	xdc.runtime.Main	App_exec: return message received, jobs=1
10	11626025472	xdc.runtime.Main	App_exec: sending message 2
11	11628307418	xdc.runtime.Main	App_exec: return message received, jobs=2
12	11628308214	xdc.runtime.Main	App_exec: sending message 3

Step 6 — Adding EVE1 Processor

- To build the EVE1 executable, you need to edit the top-level makefile. Add EVE1 to the PROCLIST macro.
  - Edit `ex13_notifypeer/makefile`

    ```
    PROCLIST = dsp1 eve1 host
    ```

- To enable DSP1 to EVE1 IPC communication, edit the server source file and uncomment the EVE macro.
  - Edit `ex13_notifypeer/dsp1/Server.c`

    ```
    /* define the EVE peer */
    #define EVE "EVE1"
    ```

- Build the example

  -
    ```
    cd ex13_notifypeer
    make
    ```

Step 7 – Connect to EVE1 Processor

- 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... `ex13_notifypeer/eve1/ex13_notifypeer_eve1.gel`
- Connect to EVE1
  - CortexA15_0 > Select
  - Scripts > DRA7xx MULTICORE Initialization > EVE1SSClkEnable_API
  - CS_DAP_DebugSS > Select
  - Scripts > EVE MMU Configuration > ex13_notifypeer_eve1_mmu_config
  - ARP32_EVE_1 > RMB > Connect Target
  - Run > Reset > CPU Reset

Step 8 — Load Processors

- Reload HOST with executable
  - Reset HOST
  - Run > Load > Reload Program
- Reload DSP1 with executable
  - Reset DSP1
  - Run > Load > Reload Program
- Load EVE1 with executable
  - Debug view > ARP32_EVE_1 > Select
  - Run > Load > Load Program
  - Click Browse, select the EVE1 executable

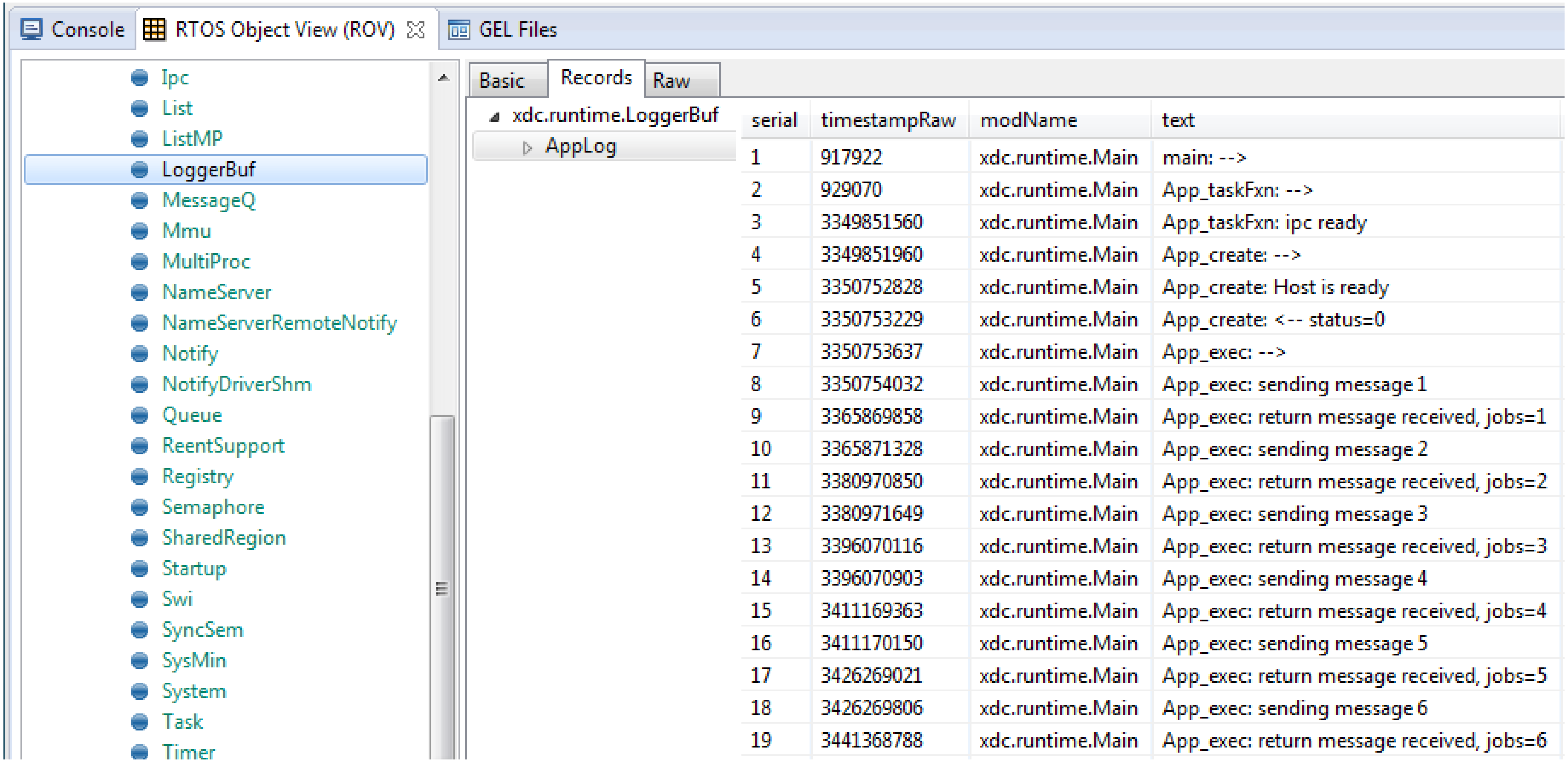
    `ex13_notifypeer\eve1\bin\debug\alg_eve1.xearp32F`

Step 9 — Run the Example

- Run HOST processor
  - Debug view > CortexA15_0 > Select
  - Run > Resume
- Run EVE1 processor
  - Debug view > ARP32_EVE_1 > Select
  - Run > Resume
- Run DSP1 processor
  - Debug view > C66xx_DSP1 > Select
  - Run > Resume
- The example completes quickly. Halt all three processors.

Inspect the Logs

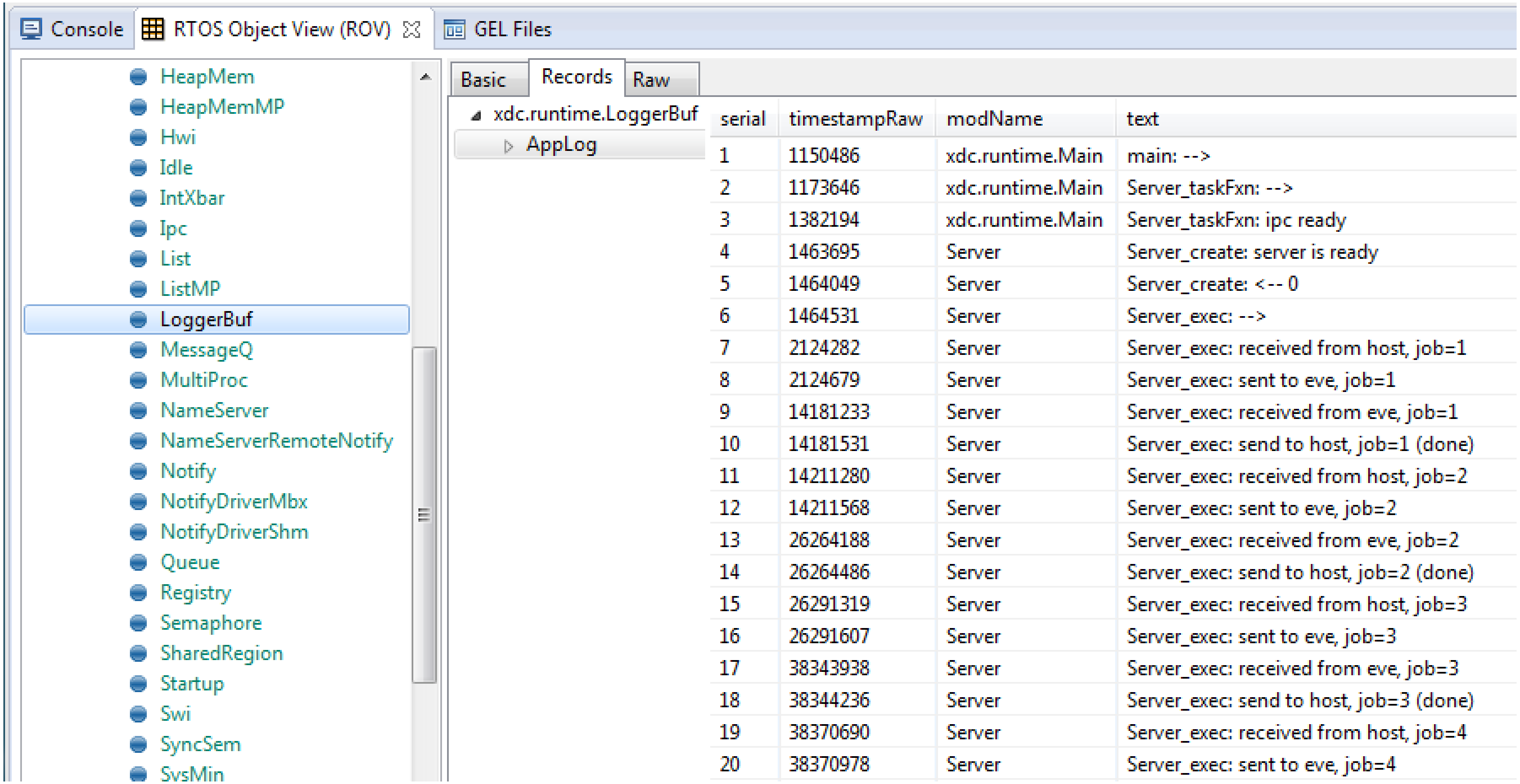
- Use ROV to inspect the logs from each processor.
- The HOST logs should look identical.



serial	timestampRaw	modName	text
1	917922	xdc.runtime.Main	main: -->
2	929070	xdc.runtime.Main	App_taskFxn: -->
3	3349851560	xdc.runtime.Main	App_taskFxn: ipc ready
4	3349851960	xdc.runtime.Main	App_create: -->
5	3350752828	xdc.runtime.Main	App_create: Host is ready
6	3350753229	xdc.runtime.Main	App_create: <-- status=0
7	3350753637	xdc.runtime.Main	App_exec: -->
8	3350754032	xdc.runtime.Main	App_exec: sending message 1
9	3365869858	xdc.runtime.Main	App_exec: return message received, jobs=1
10	3365871328	xdc.runtime.Main	App_exec: sending message 2
11	3380970850	xdc.runtime.Main	App_exec: return message received, jobs=2
12	3380971649	xdc.runtime.Main	App_exec: sending message 3
13	3396070116	xdc.runtime.Main	App_exec: return message received, jobs=3
14	3396070903	xdc.runtime.Main	App_exec: sending message 4
15	3411169363	xdc.runtime.Main	App_exec: return message received, jobs=4
16	3411170150	xdc.runtime.Main	App_exec: sending message 5
17	3426269021	xdc.runtime.Main	App_exec: return message received, jobs=5
18	3426269806	xdc.runtime.Main	App_exec: sending message 6
19	3441368788	xdc.runtime.Main	App_exec: return message received, jobs=6

Inspect the Logs

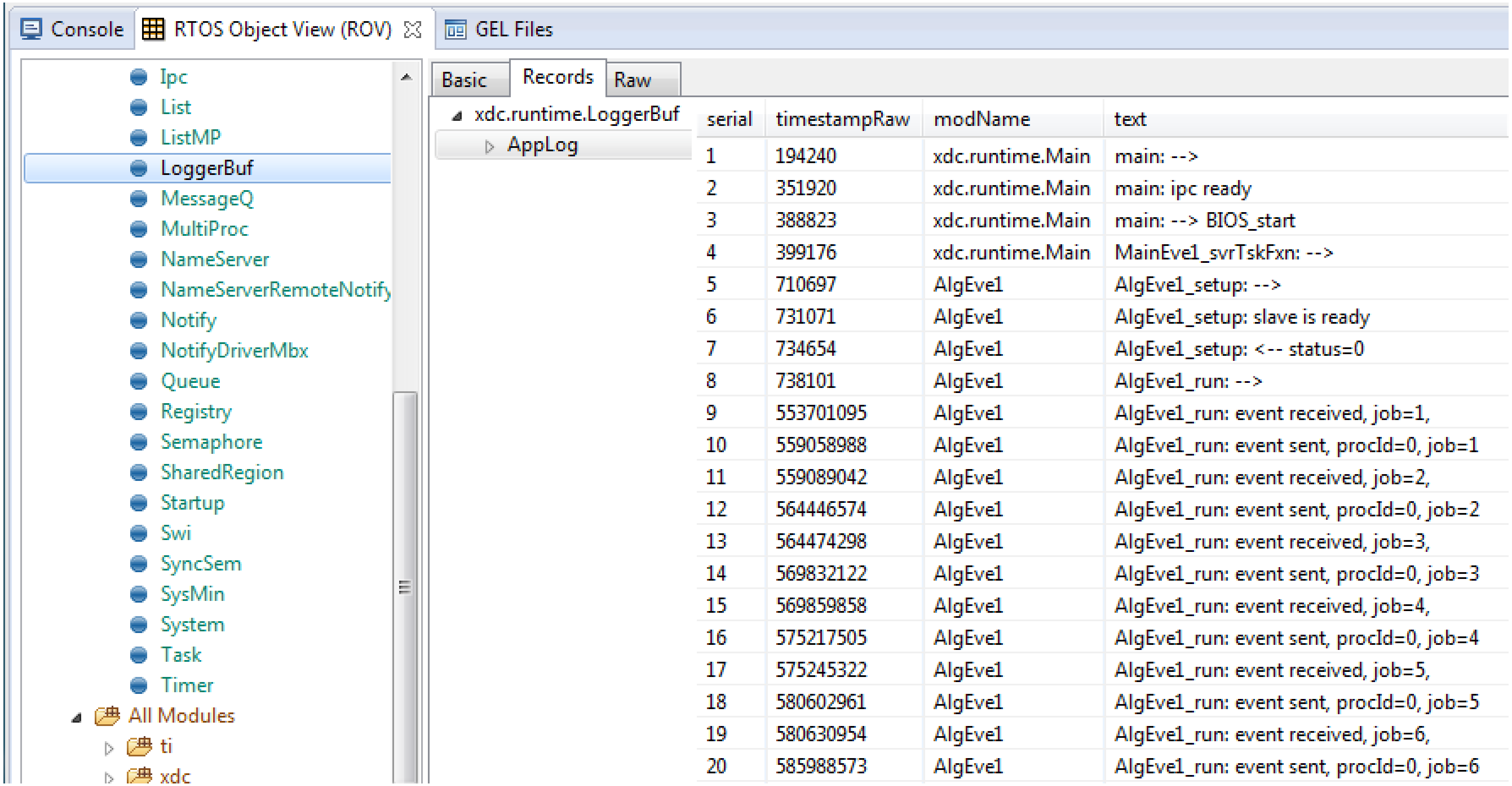
- The DSP1 logs will contain additional EVE notifications.



serial	timestampRaw	modName	text
1	1150486	xdc.runtime.Main	main: -->
2	1173646	xdc.runtime.Main	Server_taskFxn: -->
3	1382194	xdc.runtime.Main	Server_taskFxn: ipc ready
4	1463695	Server	Server_create: server is ready
5	1464049	Server	Server_create: <-- 0
6	1464531	Server	Server_exec: -->
7	2124282	Server	Server_exec: received from host, job=1
8	2124679	Server	Server_exec: sent to eve, job=1
9	14181233	Server	Server_exec: received from eve, job=1
10	14181531	Server	Server_exec: send to host, job=1 (done)
11	14211280	Server	Server_exec: received from host, job=2
12	14211568	Server	Server_exec: sent to eve, job=2
13	26264188	Server	Server_exec: received from eve, job=2
14	26264486	Server	Server_exec: send to host, job=2 (done)
15	26291319	Server	Server_exec: received from host, job=3
16	26291607	Server	Server_exec: sent to eve, job=3
17	38343938	Server	Server_exec: received from eve, job=3
18	38344236	Server	Server_exec: send to host, job=3 (done)
19	38370690	Server	Server_exec: received from host, job=4
20	38370978	Server	Server_exec: sent to eve, job=4

Inspect the Logs

- The EVE logs contain the jobs messages.



serial	timestampRaw	modName	text
1	194240	xdc.runtime.Main	main: -->
2	351920	xdc.runtime.Main	main: ipc ready
3	388823	xdc.runtime.Main	main: --> BIOS_start
4	399176	xdc.runtime.Main	MainEve1_svrTskFxn: -->
5	710697	AlgEve1	AlgEve1_setup: -->
6	731071	AlgEve1	AlgEve1_setup: slave is ready
7	734654	AlgEve1	AlgEve1_setup: <-- status=0
8	738101	AlgEve1	AlgEve1_run: -->
9	553701095	AlgEve1	AlgEve1_run: event received, job=1,
10	559058988	AlgEve1	AlgEve1_run: event sent, procId=0, job=1
11	559089042	AlgEve1	AlgEve1_run: event received, job=2,
12	564446574	AlgEve1	AlgEve1_run: event sent, procId=0, job=2
13	564474298	AlgEve1	AlgEve1_run: event received, job=3,
14	569832122	AlgEve1	AlgEve1_run: event sent, procId=0, job=3
15	569859858	AlgEve1	AlgEve1_run: event received, job=4,
16	575217505	AlgEve1	AlgEve1_run: event sent, procId=0, job=4
17	575245322	AlgEve1	AlgEve1_run: event received, job=5,
18	580602961	AlgEve1	AlgEve1_run: event sent, procId=0, job=5
19	580630954	AlgEve1	AlgEve1_run: event received, job=6,
20	585988573	AlgEve1	AlgEve1_run: event sent, procId=0, job=6

IPC Notify Scalability

- Need only two modules

```
xdc.useModule('ti.sdo.ipc.Notify');
xdc.useModule('ti.sdo.utils.MultiProc');
```

- Configure notify to use mailbox driver

```
/* configure the notify driver */
var NotifySetup = xdc.useModule('ti.sdo.ipc.family.vayu.NotifySetup');

NotifySetup.connections.$add(
    new NotifySetup.Connection({
        driver: NotifySetup.Driver_MAILBOX,
        procName: "EVE1"
    })
);
```

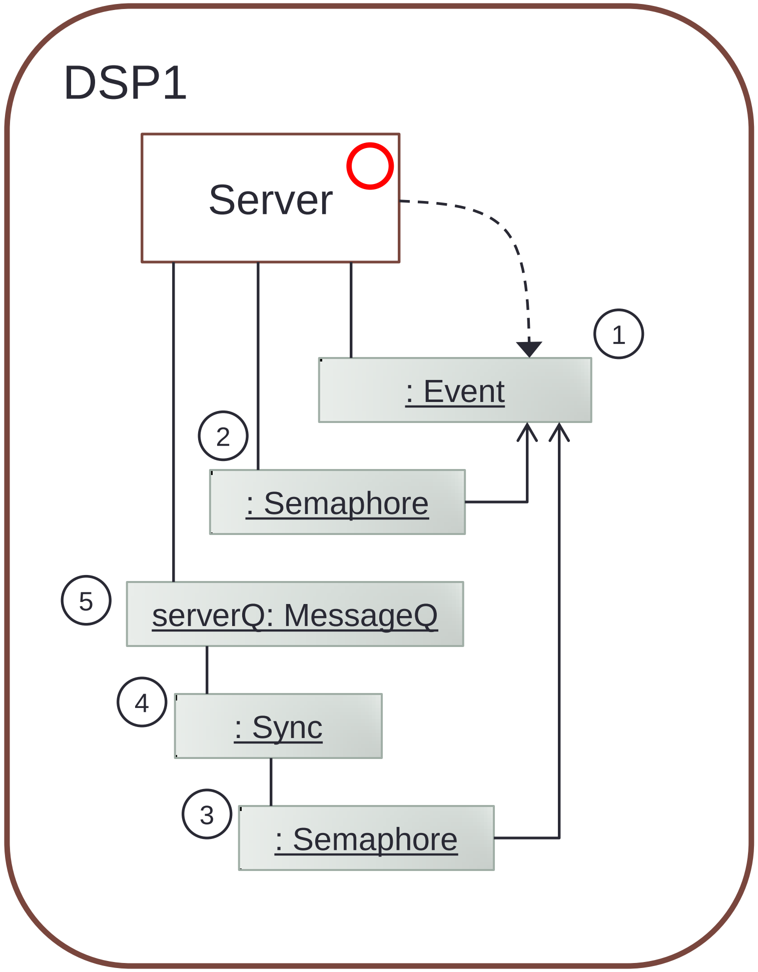
- Attach has no handshake

```
/* setup IPC-notify with eve processor */
Notify_attach(Module.eveProcId, 0);
```

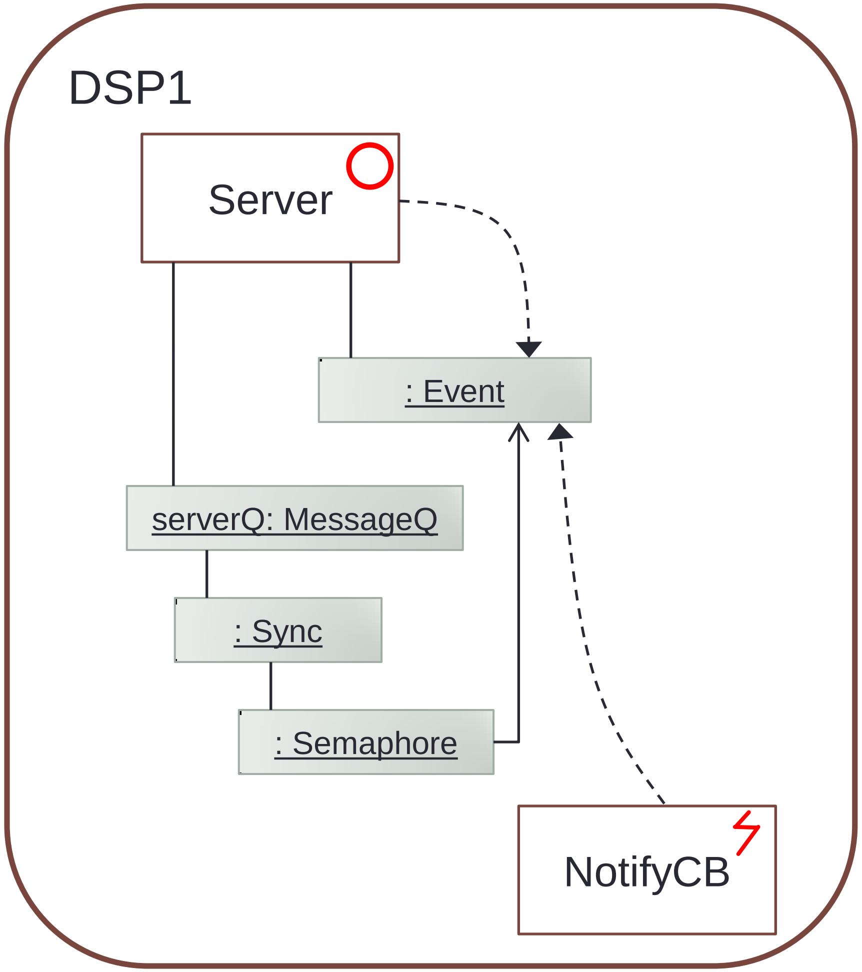
Waiting on Two Input Sources

- The DSP is blocked, waiting on input from two sources.
  - Source 1: Waiting on the message queue for a new message.
  - Source 2: Waiting on the semaphore for a post event.
- How is this possible?
- We use a SYS/BIOS Event instance. The event object has a binding to both the message queue and the semaphore.

SYS/BIOS Event Object



SYS/BIOS Event Object



Create Phase

- Create event object

```
Event_Params_init(&eventP);
event = Event_create(&eventP, NULL);
```

- Create the message queue with sync and semaphore objects

```
Semaphore_Params_init(&semP);
semP.event = event;
semP.eventId = Event_Id_01; /* message queue */
semP.mode = Semaphore_Mode_BINARY;
sem = Semaphore_create(0, &semP, NULL);

SyncSem_Params_init(&syncSemP);
syncSemP.sem = sem;
sync = SyncSem_create(&syncSemP, NULL);

MessageQ_Params_init(&msgQueP);
msgQueP.synchronizer = (Void *)sync;
messageQ = MessageQ_create("ServerQue", &msgQueP);
```

Execute Phase

- Notify callback will post the event object directly.

```
Event_post(event, Event_Id_00);
```

- Server task will pend on event object.

```
mask = Event_Id_01 | Event_Id_00;
evts = Event_pend(event, Event_Id_NONE, mask, BIOS_WAIT_FOREVER);

if (evts & Event_Id_00) {
    /* get payload from the notify queue */
    job = Server_dequeueEvent(&Module.notifyQ);
}

if (evts & Event_Id_01) {
    /* get message from message queue */
    MessageQ_get(Module.messageQ, (MessageQ_Msg *)&msg, BIOS_NO_WAIT);
}
```

Congratulations!
End of Lab 3