# TRACE32 and VxWorks 6.9

• Richard Copeman •
2017 / June / 06

www.lauterbach.com

LAUTERBACH DEVELOPMENT TOOLS



## Agenda

- Development Host & Target
- Build OS Image (no MMU, no RTP)
- Build OS Image (MMU, RTP)
- Configure TRACE32 (no MMU, non RTP)
- Configure TRACE32 (MMU, RTP)
- Example



LAUTERBACH DEVELOPMENT TOOLS

## Development Host & Target

- Windows 10 x64 PC
- 32bit Java runtime
- VxWorks 6.9
- WindRiver Workbench 3.3
- iMX6Quad SabreLite board
- Lauterbach TRACE32 for ARM
- PowerDebug Pro with ARM License Cable
- 20-pin JTAG to 10-pin MIPI convertor





## Agenda

- Development Host & Target
- Build OS Image (no MMU, no RTP)
- Build OS Image (MMU, RTP)
- Configure TRACE32 (no MMU, non RTP)
- Configure TRACE32 (MMU, RTP)
- Example

# Build OS Image (non RTP)

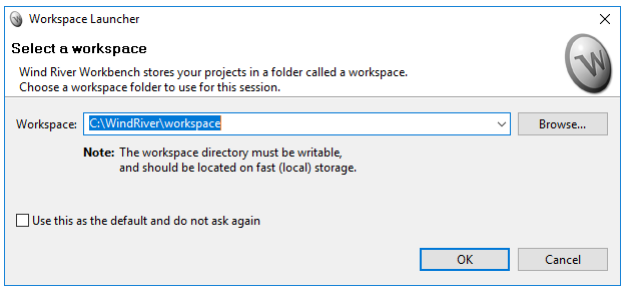
- This section will show how to build a simple VxWorks target image with a simple application for debugging
- For more details please consult the VxWorks documentation





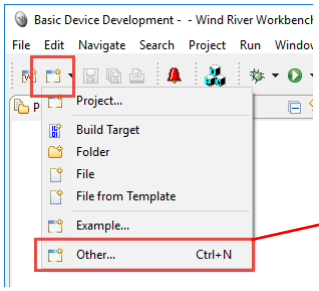
# Build OS Image (non RTP)

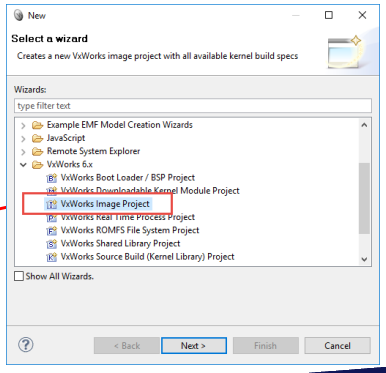
- Launch WindRiver Workbench
  - Select a directory to use as a workspace
  - Click ‘OK’



# Build OS Image (non RTP)

- Create a new Project
  - Select Other
  - Select VxWorks 6.x -> VxWorks Image Project
  - Click ‘Next>’



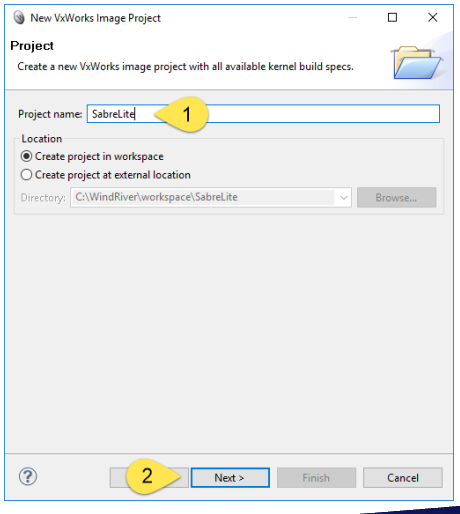







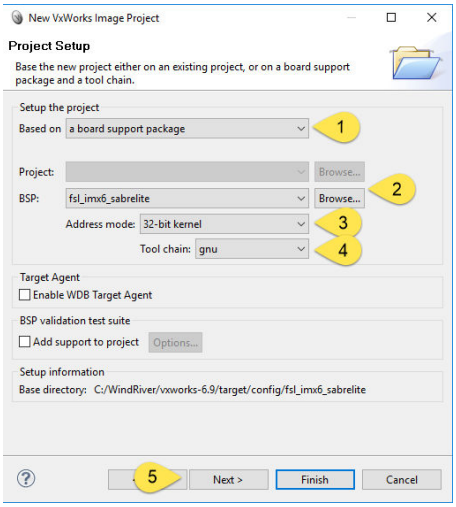

# Build OS Image (non RTP)

1) Enter Project name
2) Click ‘Next>’



# Build OS Image (non RTP)

1) Select “A board support package”
2) Select BSP from the list or Browse for an archive
3) Set the addressing mode
4) Select the toolchain to use
5) Click ‘Next>’

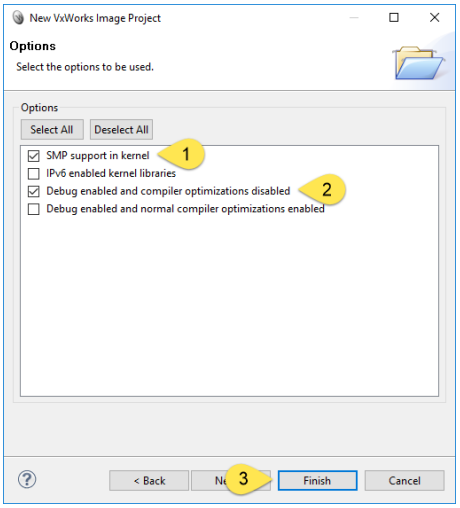






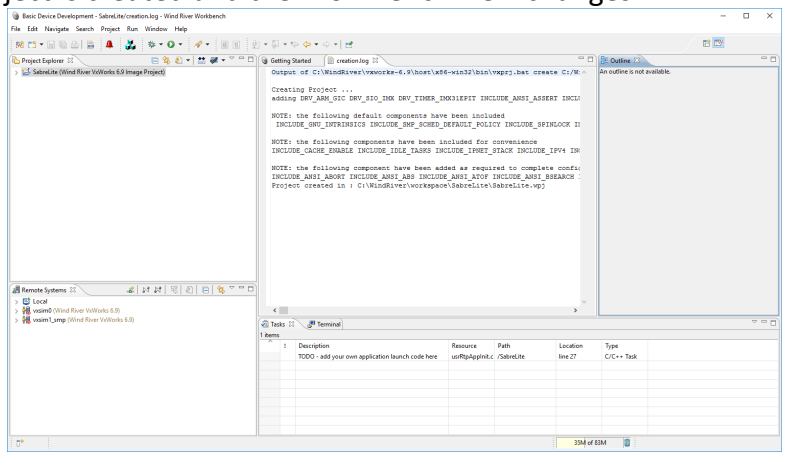

# Build OS Image (non RTP)

1) Set “SMP support in kernel”
2) Set “Debug enabled and compiler optimisations disabled”
3) Click ‘Finish’



# Build OS Image (non RTP)

- Project is created and the WorkBench view changes

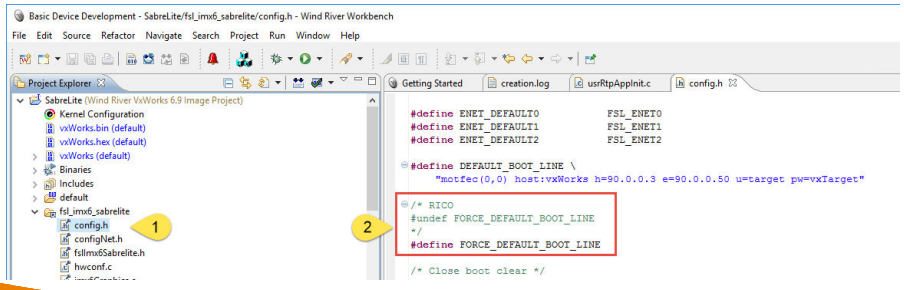






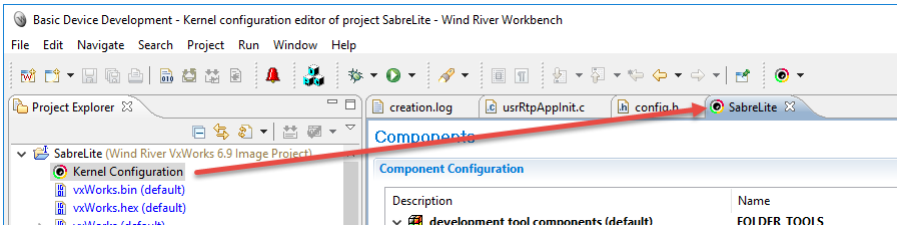
# Build OS Image (non RTP)

- Edit **`config.h`**
- Ensure that **`FORCE_DEFAULT_BOOT_LINE`** is defined



# Build OS Image (non RTP)

- Edit Kernel Configuration

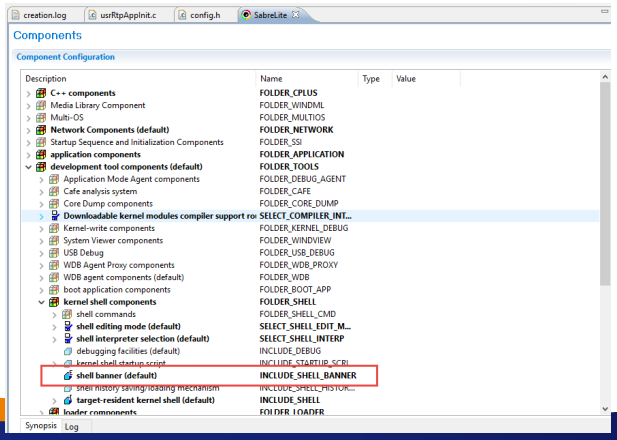






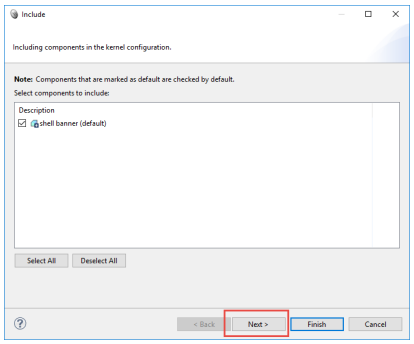

# Build OS Image (non RTP)

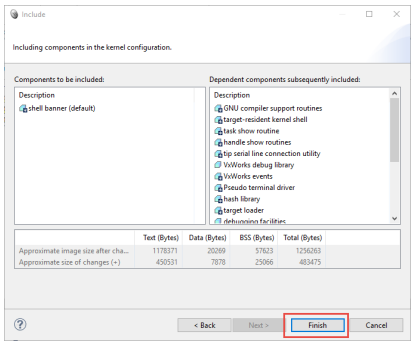
- Include Shell Banner(right-click->Include)
- This adds all of the required shell components too



# Build OS Image (non RTP)

- Click Next>
- Click Finish



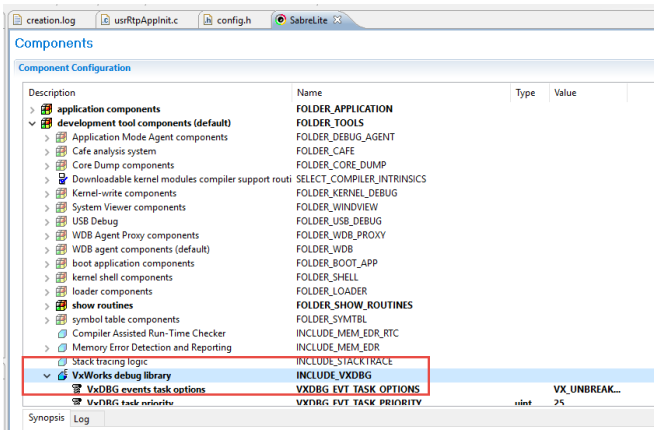







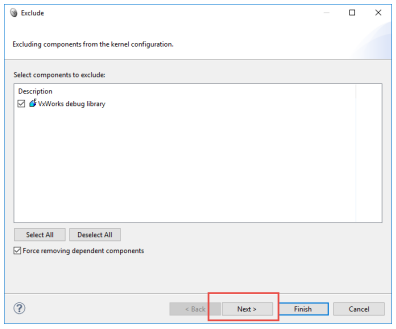

# Build OS Image (non RTP)

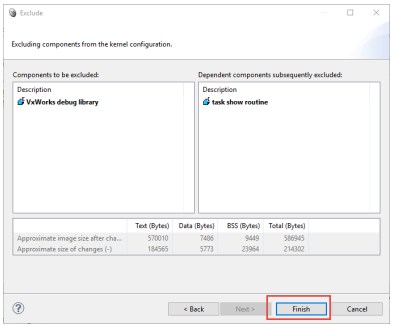
- Exclude VxWorks debug library (right-click->Exclude)



# Build OS Image (non RTP)

- Click Next>
- Click Finish



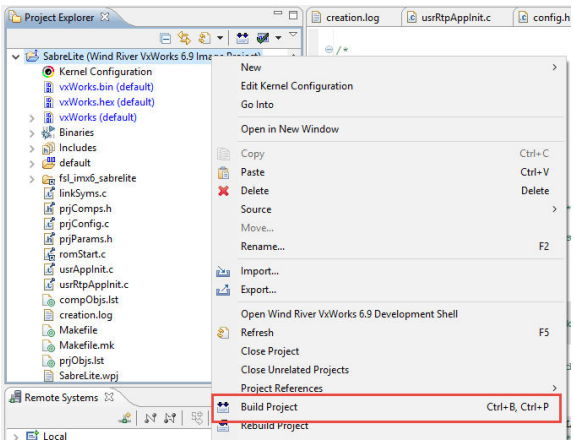








# Build OS Image (non RTP)

- Right-click the Project and select Build Project from the menu



## Build OS Image (non RTP)

- The output from the build is located in
  - C:\WindRiver\workspace\SabreLite\default\vxworks
- This is a non RTP build so does not use the MMU
  - All tasks are run in kernel space
  - Debugger does not require MMU configuration



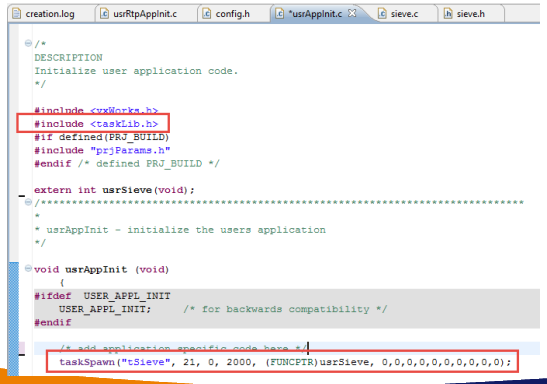


## Build OS Image (non RTP)

- Add two new files to the project
  - **`sieve.c`**
  - **`sieve.h`**
- These contain the code for my demo applications
- The entry point in **`sieve.c`** is function **`usrSieve()`**

# Build OS Image (non RTP)

- Edit the file **`usrAppInit.c`**
  - This is where VxWorks will start the tasks for the system.
  - Add your task creation code here








# Build OS Image (non RTP)

- Save and build the project
- The ELF file will be located at
  - **`C:\WindRiver\workspace\SabreLite\default\vxworks`**
- Now skip to slide 65 for NON-RTP configuration of TRACE32

# Agenda

- Development Host & Target
- Build OS Image (no MMU, no RTP)
- Build OS Image (MMU, RTP)
- Configure TRACE32 (non MMU, no RTP)
- Configure TRACE32 (MMU, RTP)
- Example



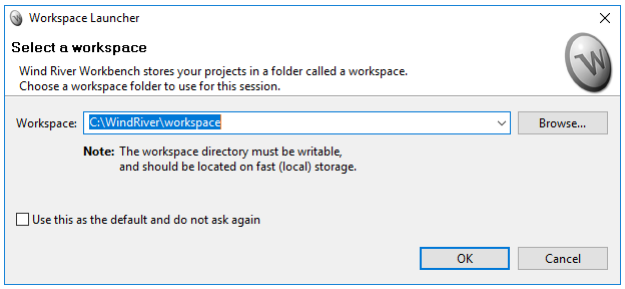


# Build OS Image (RTP)

- This section will show how to build a more complex VxWorks target image with an RTP application for debugging
- For more details please consult the VxWorks documentation

# Build OS Image (RTP)

- Launch WindRiver Workbench
  - Select a directory to use as a workspace
  - Click ‘OK’

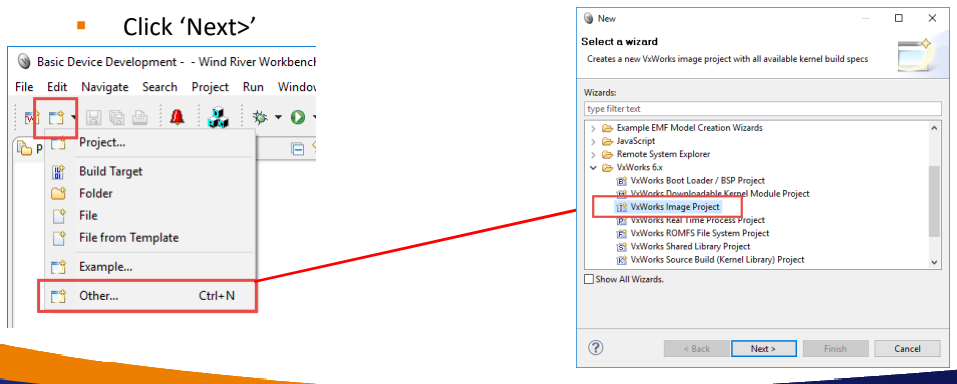






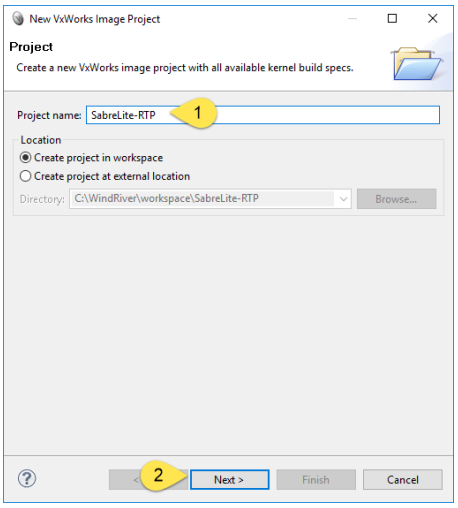

# Build OS Image (RTP)

- Create a new Project
  - Select Other
  - Select VxWorks 6.x -> VxWorks Image Project
  - Click ‘Next>’



# Build OS Image (RTP)

1) Enter Project name
2) Click ‘Next>’

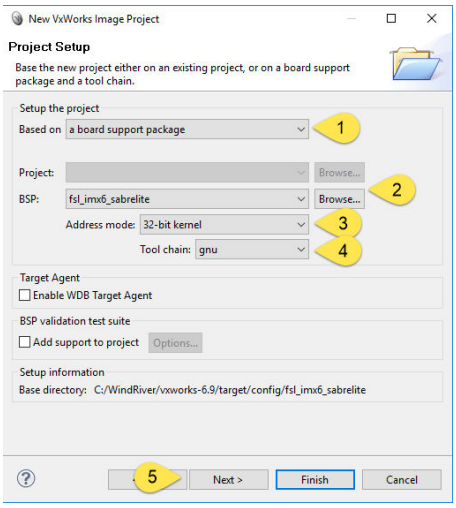






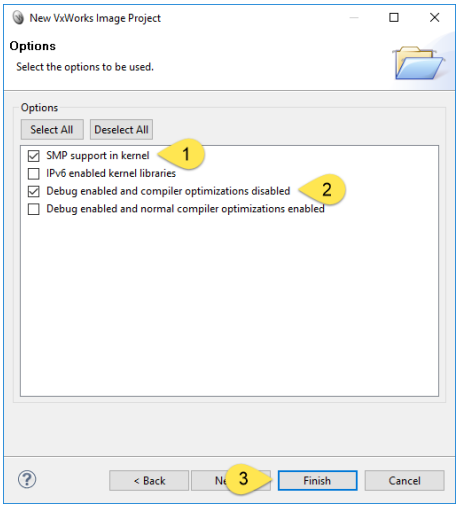

# Build OS Image (RTP)

1) Select “A board support package”
2) Select BSP from the list or Browse for an archive
3) Set the addressing mode
4) Select the toolchain to use
5) Click ‘Next>’



# Build OS Image (RTP)

1) Set “SMP support in kernel”
2) Set “Debug enabled and compiler optimisations disabled”
3) Click ‘Finish’

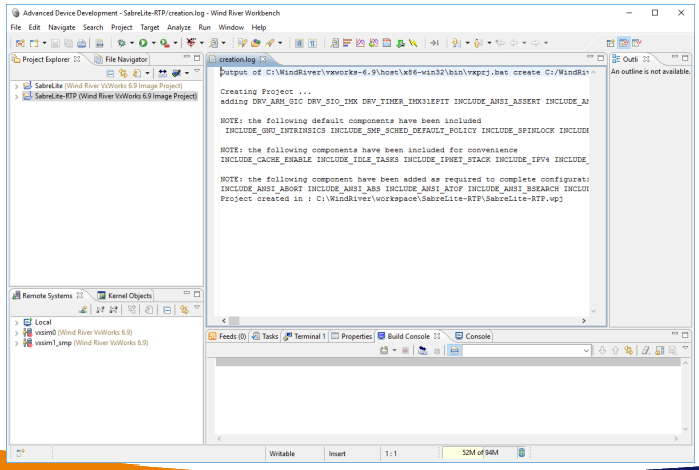






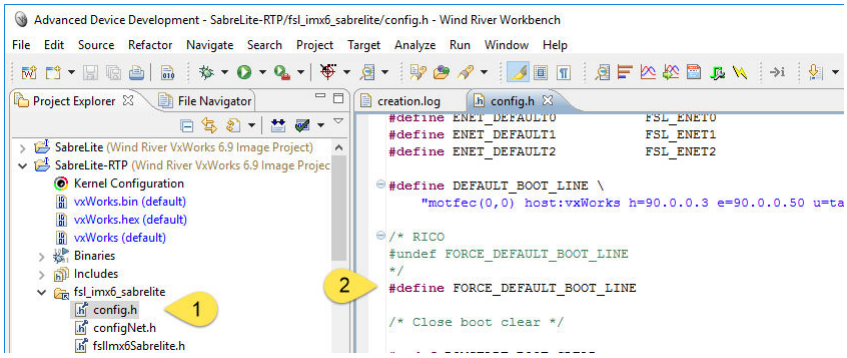

# Build OS Image (RTP)

- Project is created and the WorkBench view changes



# Build OS Image (RTP)

- Edit **`config.h`**
- Ensure that **`FORCE_DEFAULT_BOOT_LINE`** is defined

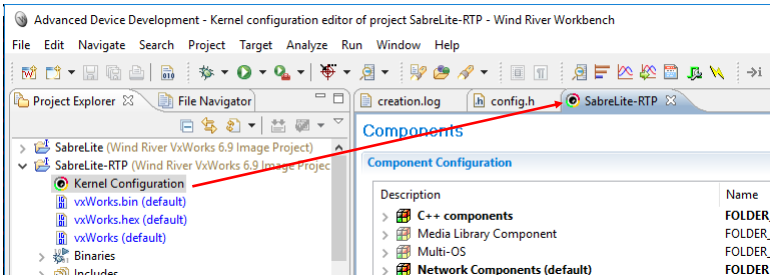






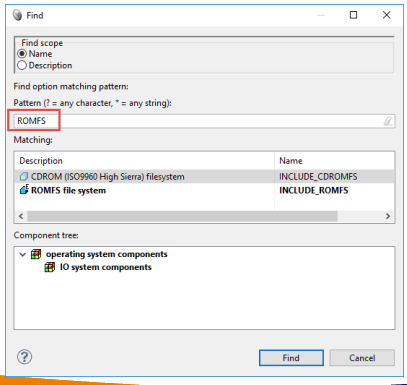

# Build OS Image (RTP)

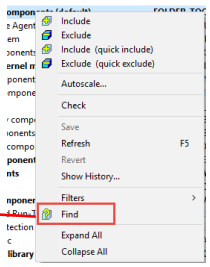
- Edit Kernel Configuration



# Build OS Image (RTP)

- Build configuration options can be found
- Right-click an element in kernel component configuration, select Find, enter the key to search for





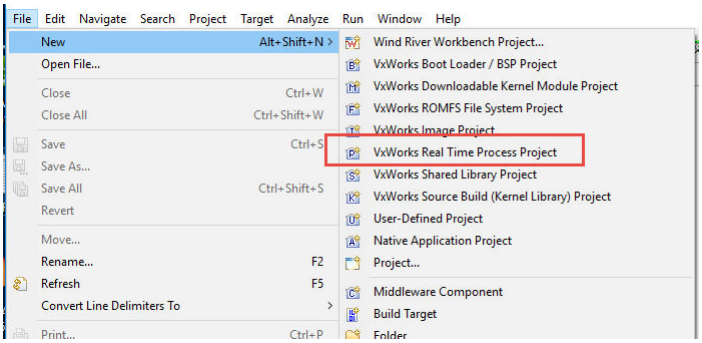



# Build OS Image (RTP)

- Add or remove the following components
- INCLUDE_ROMFS
- INCLUDE_RTP
- INCLUDE_SHELL_BANNER
- INCLUDE_DISK_UTIL_SHELL_CMD
- EXCLUDE_VXDBG

# Build OS Image (RTP)

- To build the Real Time Process:
  - Create a new VxWorks Real Time Process Project

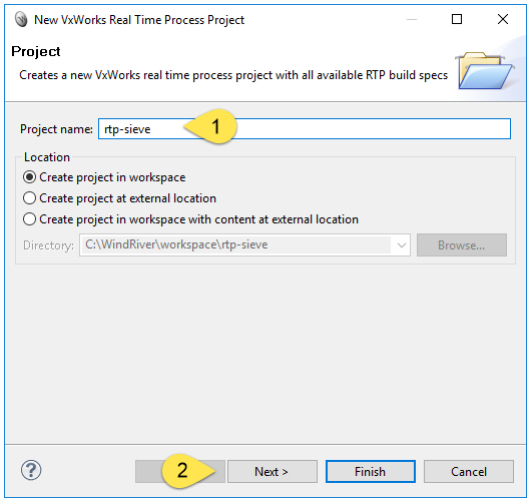






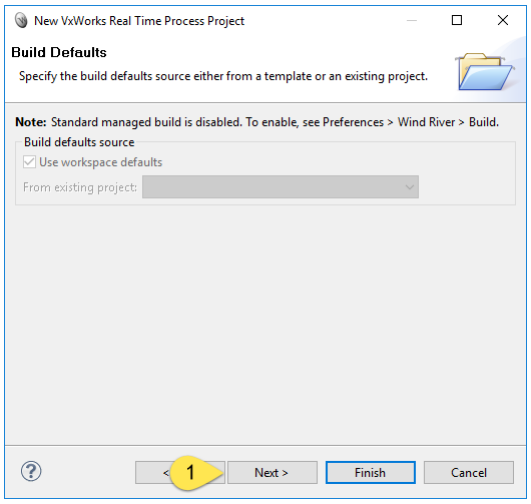

# Build OS Image (RTP)

- Name the project
- Click ‘Next>’



# Build OS Image (RTP)

- Click ‘Next>’

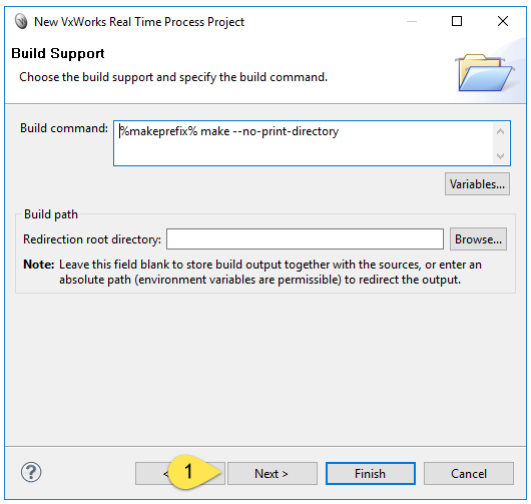






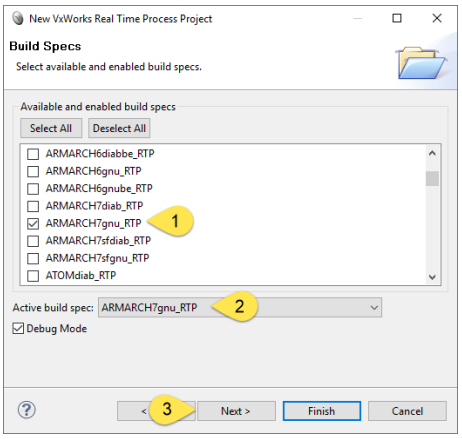

# Build OS Image (RTP)

- Click ‘Next>’



# Build OS Image (RTP)

- Select the correct architecture/compiler
- Build spec should automatically change
- Click ‘Next>’

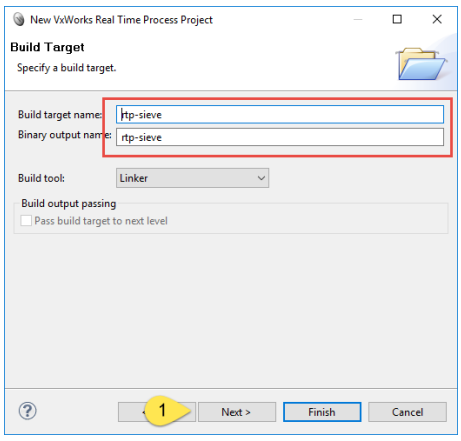






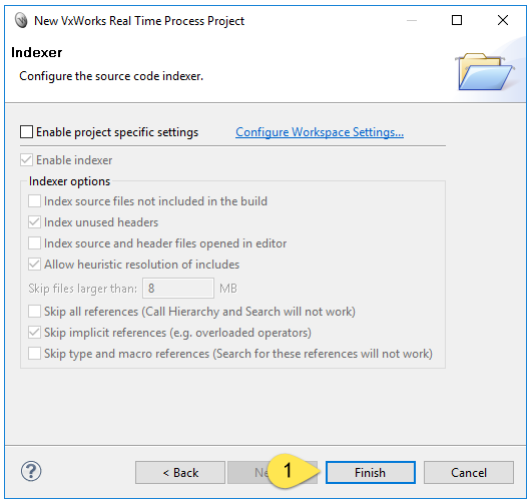

# Build OS Image (RTP)

- The project name and output file can be changed here, if required
- Click ‘Next>’



# Build OS Image (RTP)

- Click ‘Finish’

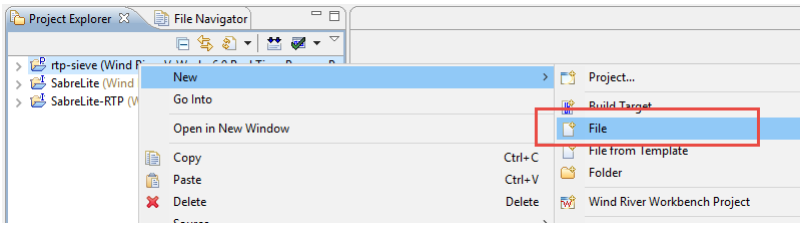






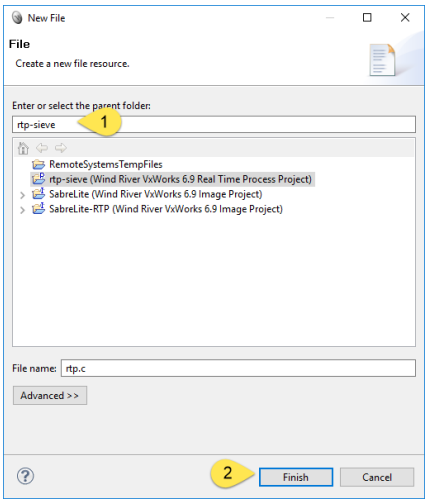

# Build OS Image (RTP)

- Add New Source File



# Build OS Image (RTP)

- Enter filename
- Click ‘Finish’

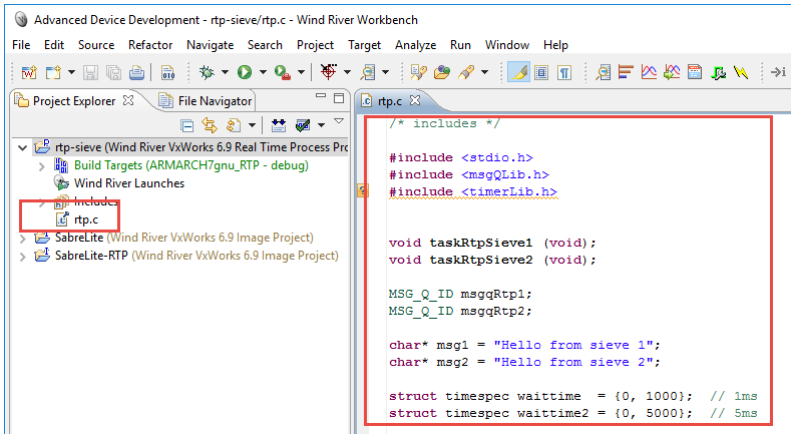






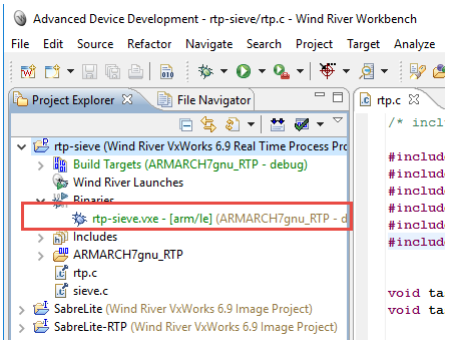

# Build OS Image (RTP)

- Enter your source code



# Build OS Image (RTP)

- Build the application
- Should produce **`rtp-sieve.vxe`**

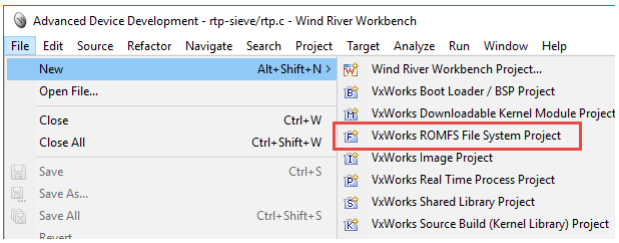






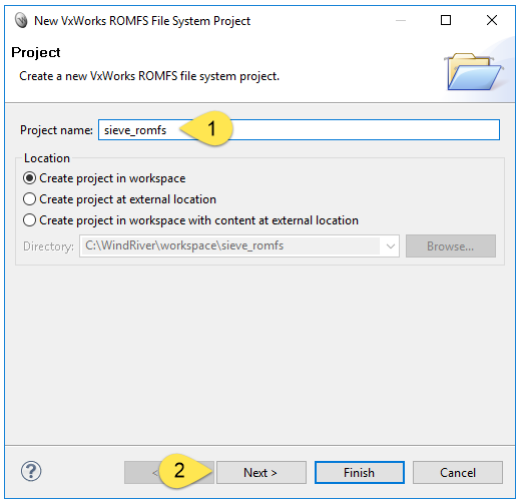

# Build OS Image (RTP)

- Create a new VxWorks ROMFS File System Project



# Build OS Image (RTP)

- Name the project
- Click ‘Next>’

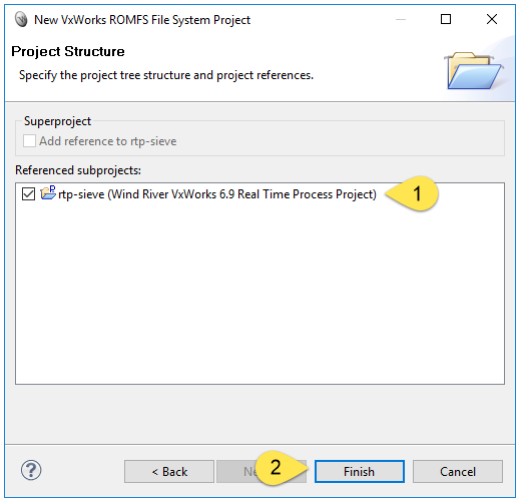






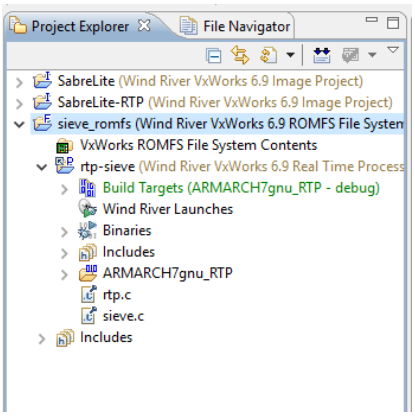

# Build OS Image (RTP)

- Select all required project references
- Click ‘Finish’



# Build OS Image (RTP)

- Project **`rtp-sieve`** is now a sub project of **`sieve_romfs`**

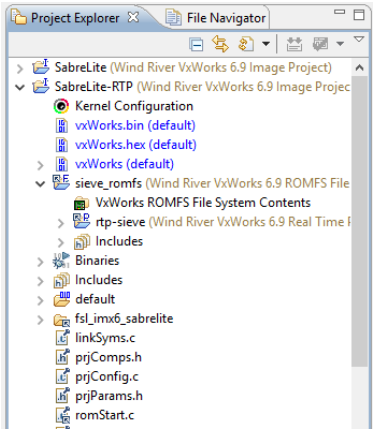






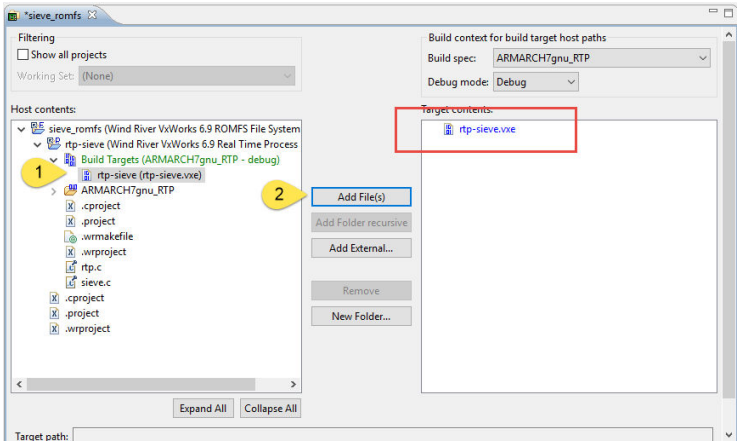

# Build OS Image (RTP)

- Drag and drop **`sieve_romfs`** so that it becomes a sub project of **`SabreLite-RTP`**



# Build OS Image (RTP)

- Add **`rtp-sieve.vxe`** to the ROMFS target contents





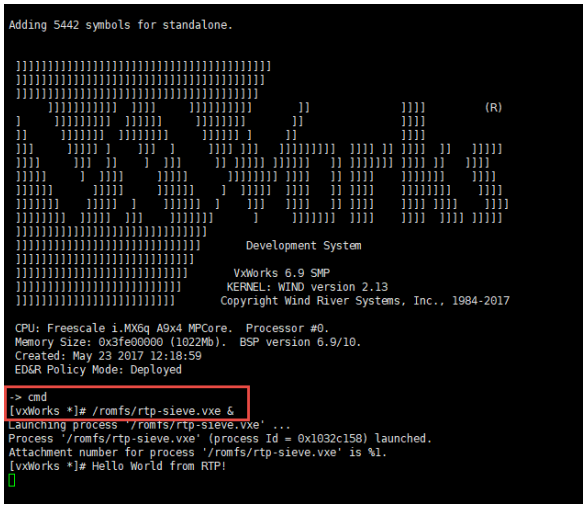



# Build OS Image (RTP)

- Save all
- Build SabreLite-RTP

# Build OS Image (RTP)

- Launch RTP tasks (1)
  - Via command shell








# Build OS Image (RTP)

- Launch RTP tasks (2)
  - Via **`rtpSpawn()`** in **`usrRtpAppInit()`**
  - Set **`INCLUDE_RTP_APPL_USER`** in Kernel Configuration

```
#include <rtpLib.h>

void usrRtpAppInit (void)
{
  /* TODO - add your own application launch code here */
  const char *vxeName = "/romfs/rtp-sieve.vxe";
  const char *argv[1];
  RTP_ID rtpID = NULL;

  argv[0] = NULL;

  if (( rtpID = rtpSpawn(vxeName, argv, NULL, 220, 0x10000,0, 0 ))==NULL)
  {
    printf("Unable to spawn rtp-sieve\n");
  }
}
```

# Build OS Image (RTP)

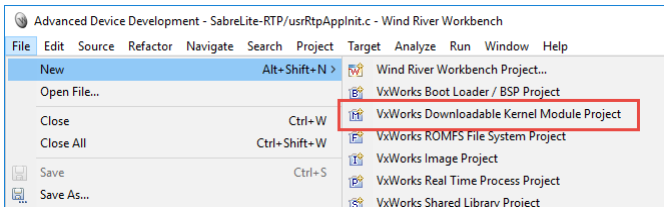
- Launch RTP tasks (3)
  - Other options include:
  - Boot command line
  - Custom start-up shell script





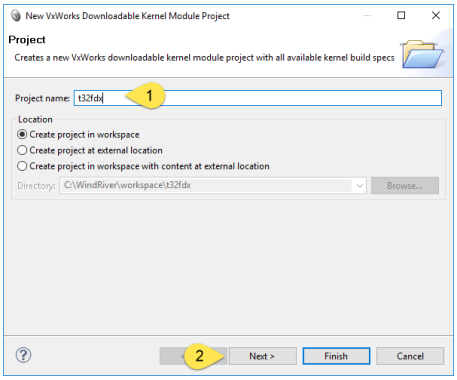
# Build OS Image (RTP)

- Add Kernel Module



# Build OS Image (RTP)

- Enter name
- Click ‘Next>’



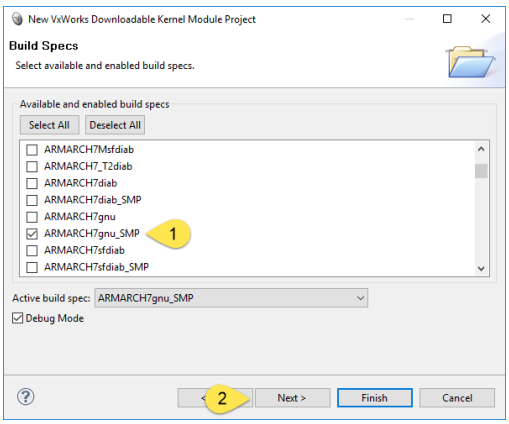

- **Click ‘Next>’ on the following two slides as well**





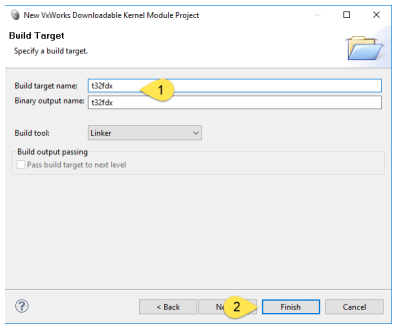
# Build OS Image (RTP)

- Select build spec
- Click ‘Next>’

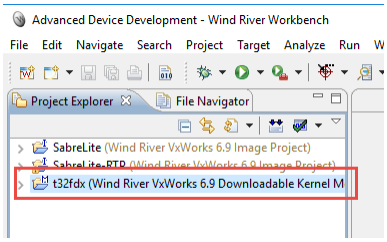


# Build OS Image (RTP)

- Enter target name and output file name
- Click ‘Finish’




- New project appears in Workspace







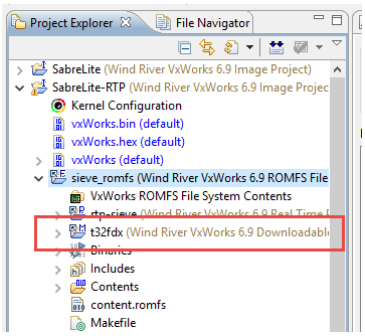

# Build OS Image (RTP)

- Add your source files to the DKM project

```
t32fdx.c   rtp.c   sieve_romfs
#include "vxWorks.h"
#include "stdio.h"
#include <msgQLib.h>
#include <time.h>

struct timespec waittime = { 0, 100000};     /* 100ms */

int t32fdxStart(void)
{
    int i;
    printf ("Hello World from DKM!\n");

    for(i=0; i< 10000; i++)
    {
        nanosleep(&waittime, NULL);
    }
    return 0;
}
```

# Build OS Image (RTP)

- Drag and drop the DKM project so that it becomes a sub-project of romfs (the filesystem)

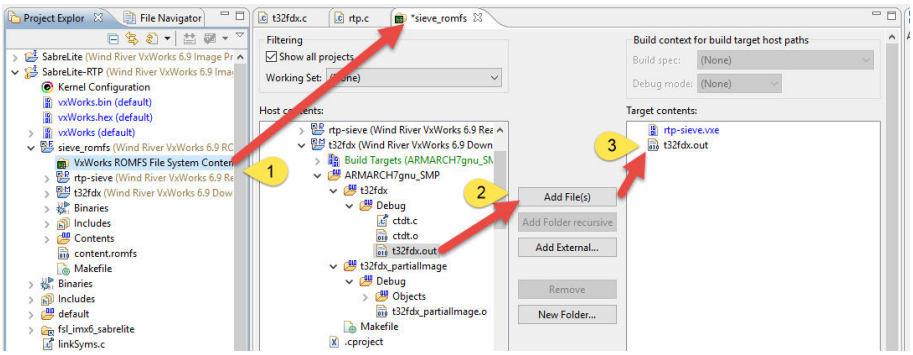






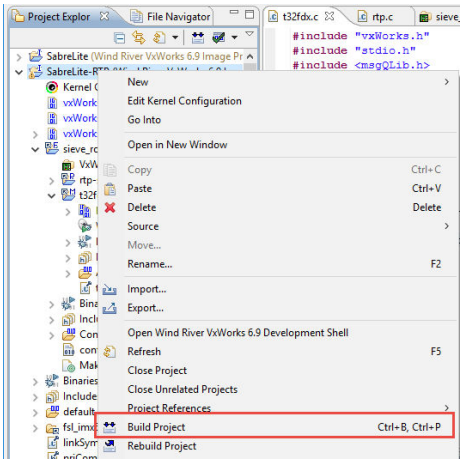

# Build OS Image (RTP)

- Edit the file system contents and add the .out file



# Build OS Image (RTP)

- Save all and build the root (RTP) project








# Build OS Image (RTP)

- DKMs can be launched from the VxWorks shell
  - Load the DKM from the filesystem
  - Switch back to the VxWorks shell form the Cshell
  - Execute the entry point

```
-> cmd
# ld /romfs/t32dkm.out
# C
-> t32dkmStart
```

# Agenda

- Development Host & Target
- Build OS Image (no MMU, no RTP)
- Build OS Image (MMU, RTP)
- Configure TRACE32 (non MMU, no RTP)
- Configure TRACE32 (MMU, RTP)
- Example





# Configure TRACE32

- These next few slides will demonstrate an example TRACE32 configuration for the Non-MMU (RTP) system we have just built
  - The script will be in a text box
  - The explanation will be in the bullet points

# Configure TRACE32

- Declare some macros at the top of the setup file

```
LOCAL &image

&image="C:\WindRiver\workspace\SabreLite\default\vxworks"
```





# Configure TRACE32

- Initial target setup options
  - Configure for only a single core as the others are released by VxWorks during initial boot
  - Kernel will use DABORT, PABORT and UNDEF for page miss handling
  - iMX6 does not like ETM enabled during the boot phase so either switch it off or set it to onchip (if required)

```
AREA.view
PRINT "Initialising..."
SYStem.CPU iMX6Quad
CORE.ASSIGN 1.
SYStem.Option DACR on
TrOnchip.Set DABORT OFF
TrOnchip.Set PABORT OFF
TrOnchip.Set UNDEF OFF
SYStem.Option ResBreak OFF
SYStem.Option WaitReset 1.0s
SYStem.JtagClock 20.MHz
trace.Method ONCHIP
SYStem.Up
```

# Configure TRACE32

- Target configuration and kernel loading
  - This setup lets uBoot configure the target. uBoot is resident in internal FLASH on the device.
  - Load VxWorks ELF file
  - This will automatically detect where to load to and set the PC to the entry point of the image.

```
;Let Uboot configure the board
;for us
Go.direct
PRINT "Uboot target setup"
WAIT 2.s
Break.direct
WAIT !RUN()

Register.RESet

Data.LOAD.Elf "&image"
```





# Configure TRACE32

- Run to the entry point of VxWorks
  - Configure the TRACE32 kernel awareness and extension menu
  - ~~ resolves to the TRACE32 installation directory

```
Go usrRoot
WAIT !STATE.RUN()

;Initialize VxWorks Support
PRINT "initializing VxWorks support..."
TASK.CONFIG ~~/demo/arm/kernel/vxworks/vxworks.t32
MENU.ReProgram ~~/demo/arm/kernel/vxworks/vxworks.men
```

# Configure TRACE32

- Let VxWorks completely boot.
  - Once it has booted, all four cores in the SMP array are initialised
  - Detach the debugger, configure it for 4core SMP debugging and re-attach

```
PRINT "Starting VxWorks..."
Go usrAppInit
WAIT !STATE.RUN()

; Assign all cores to allow SMP debugging
SYStem.Down                     ; detach from CPU
CORE.ASSIGN 1. 2. 3. 4.         ; assign to all cores
SYStem.Attach                   ; reattach to CPU
```





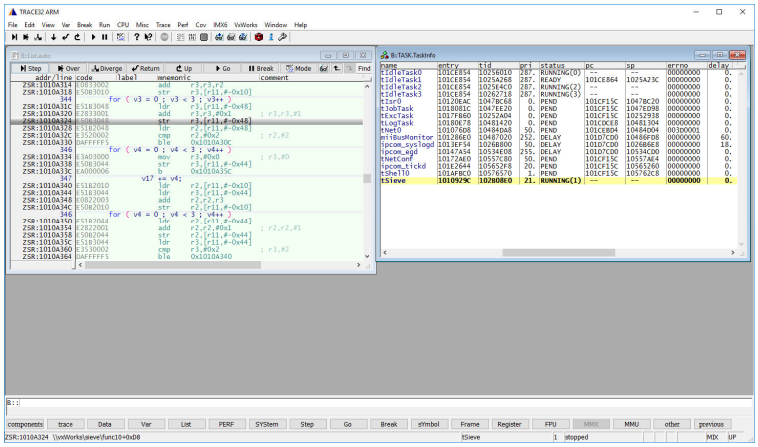
# Configure TRACE32

- Open some windows
- End the script

```
WinPOS 0.6 0.1875 98. 26. 22. 1. W002
WinTABS 10. 10. 25. 62.
List

WinPOS 104.9 0.25 80. 26. 0. 1. W001
WinTABS 10. 8. 8. 6. 5.
TASK.TaskList

ENDDO
```

# Configure TRACE32

- Launch TRACE32, run the script. The results will look like this







# Agenda

- Development Host & Target
- Build OS Image (no MMU, no RTP)
- Build OS Image (MMU, RTP)
- Configure TRACE32 (non MMU, no RTP)
- Configure TRACE32 (MMU, RTP)
- Example

# Configure TRACE32 (MMU, RTP)

- These next few slides will demonstrate an example TRACE32 configuration for the MMU (RTP) system we have just built
  - The script will be in a text box
  - The explanation will be in the bullet points





# Configure TRACE32 (MMU, RTP)

- Declare some macros at the top of the setup file

```
LOCAL &image

&image="C:\WindRiver\workspace\SabreLite-RTP\default\vxworks"
```

# Configure TRACE32 (MMU, RTP)

- Initial target setup options
  - Configure for only a single core as the others are released by VxWorks during initial boot
  - Kernel will use DABORT, PABORT and UNDEF for page miss handling
  - Enable MMUSPACES for Virtual to physical address mapping
  - iMX6 does not like ETM enabled during the boot phase so either switch it off or set it to onchip (if required)

```
AREA.view
PRINT "Initialising..."
SYStem.CPU iMX6Quad
CORE.ASSIGN 1.
SYStem.Option DACR on
TrOnchip.Set DABORT OFF
TrOnchip.Set PABORT OFF
TrOnchip.Set UNDEF OFF
SYStem.Option MMUSPACES ON
SYStem.Option ResBreak OFF
SYStem.Option WaitReset 1.0s
SYStem.JtagClock 20.MHz
trace.Method ONCHIP
SYStem.Up
```





# Configure TRACE32 (MMU, RTP)

- Target configuration and kernel loading
  - This setup lets uBoot configure the target. uBoot is resident in internal FLASH on the device.
  - Load VxWorks ELF file
  - This will automatically detect where to load to and set the PC to the entry point of the image.

```
;Let Uboot configure the board
;for us
Go.direct
PRINT "Uboot target setup"
WAIT 2.s
Break.direct
WAIT !RUN()

Register.RESet

Data.LOAD.Elf "&image"
```

# Configure TRACE32 (MMU, RTP)

- Run to the entry point of VxWorks
  - Configure the TRACE32 kernel awareness and extension menu
  - ~~ resolves to the TRACE32 installation directory

```
Go usrRoot
WAIT !STATE.RUN()

;Initialize VxWorks Support
PRINT "initializing VxWorks support..."
TASK.CONFIG ~~/demo/arm/kernel/vxworks/vxworks.t32
MENU.ReProgram ~~/demo/arm/kernel/vxworks/vxworks.men
```





# Configure TRACE32 (MMU, RTP)

- Let VxWorks completely boot.

```
PRINT "Starting VxWorks..."
Go usrAppInit
WAIT !STATE.RUN()
```

# Configure TRACE32 (MMU, RTP)

- Configure the MMU
  - Table format is always STD
  - Table base address is TTB of kernel
  - Kernel range is fixed and can be found in adrSpaceArchLibP.h
  - Base address of physical RAM

```
PRINT "initializing debugger translation..."
MMU.FORMAT STD task.rtp.ttb(0) 0x10000000--0x4fffffff 0x10000000

; kernel area is common for all RTPs
TRANSlation.COMMON 0x10000000--0x4fffffff

TRANSlation.TableWalk ON
TRANSlation.ON
```





# Configure TRACE32 (MMU, RTP)

- Let VxWorks completely boot.
  - Once it has booted, all four cores in the SMP array are initialised
  - Detach the debugger, configure it for 4core SMP debugging and re-attach

```
; Assign all cores to allow SMP debugging
SYStem.Down                     ; detach from CPU
CORE.ASSIGN 1. 2. 3. 4.         ; assign to all cores
SYStem.Attach                   ; reattach to CPU
```

# Configure TRACE32 (MMU, RTP)

- Open some windows
- End the script

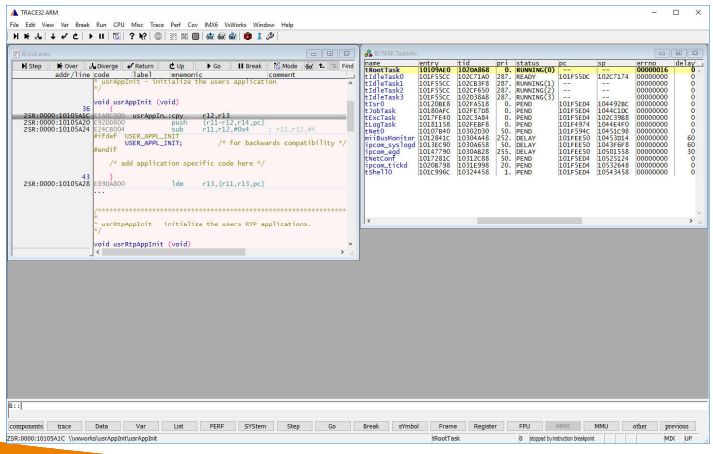
```
WinPOS 0.6 0.1875 98. 26. 22. 1. W002
WinTABS 10. 10. 25. 62.
List

WinPOS 104.9 0.25 80. 26. 0. 1. W001
WinTABS 10. 8. 8. 6. 5.
TASK.TaskList

ENDDO
```





# Configure TRACE32 (MMU, RTP)

- Launch TRACE32, run the script. The results will look like this



# Agenda

- Development Host & Target
- Build OS Image (no MMU, no RTP)
- Build OS Image (MMU, RTP)
- Configure TRACE32 (non MMU, no RTP)
- Configure TRACE32 (MMU, RTP)
- Example



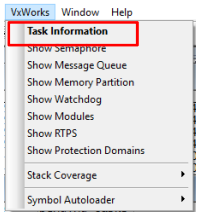


# Example

- This section provides examples of using the TRACE32 VxWorks kernel awareness features
  - For more details please refer to **~~/pdf/rtos_vxworks.pdf**
- It also provides an example of loading symbols for an RTP task and source level debugging for that task
  - Symbols for non RTP tasks will be already included in the VxWorks kernel symbol file.

# Example

- Display a list of active tasks




B::TASK.TaskInfo

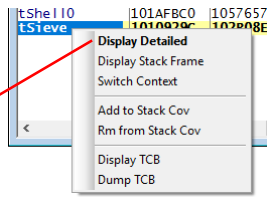
| name | entry | tid | pri | status | pc | sp | errno | delay |
|---|---|---|---|---|---|---|---|---|
| tIdleTask0 | 101CE854 | 10256010 | 287. | RUNNING(0) | -- | -- | 00000000 | 0. |
| tIdleTask1 | 101CE854 | 1025A268 | 287. | READY | 101CE864 | 1025A23C | 00000000 | 0. |
| tIdleTask2 | 101CE854 | 1025E4C0 | 287. | RUNNING(2) | -- | -- | 00000000 | 0. |
| tIdleTask3 | 101CE854 | 10262718 | 287. | RUNNING(3) | -- | -- | 00000000 | 0. |
| tIsr0 | 10120EAC | 1047BC68 | 0. | PEND | 101CF15C | 1047BC20 | 00000000 | 0. |
| tJobTask | 1018081C | 1047EE20 | 0. | PEND | 101CF15C | 1047ED98 | 00000000 | 0. |
| tExcTask | 1017FB60 | 10252A04 | 0. | PEND | 101CF15C | 10252938 | 00000000 | 0. |
| tLogTask | 10180E78 | 10481420 | 0. | PEND | 101CDCE8 | 10481304 | 00000000 | 0. |
| tNet0 | 1010760D8 | 10484DA8 | 50. | PEND | 101CEBD4 | 10484D04 | 003D0001 | 0. |
| miiBusMonitor | 101286E0 | 10487020 | 252. | DELAY | 101D7CD0 | 10486FD8 | 00000000 | 60. |
| ipcom_syslogd | 1013EF54 | 1026B800 | 50. | DELAY | 101D7CD0 | 1026B6E8 | 00000000 | 18. |
| ipcom_egd | 10147A54 | 10534E08 | 255. | DELAY | 101D7CD0 | 10534CD0 | 00000000 | 0. |
| tNetConf | 10172AE0 | 10557C80 | 50. | PEND | 101CF15C | 10557AE4 | 00000000 | 0. |
| ipcom_tickd | 101E2644 | 105652F8 | 20. | PEND | 101CF15C | 1056526C | 00000000 | 0. |
| tShell0 | 101AFBC0 | 10576570 | 1. | PEND | 101CF15C | 105762C8 | 00000000 | 0. |
| tSieve | 1010929C | 102B08E0 | 21. | RUNNING(1) | -- | -- | 00000000 | 0. |





# Example

- Each entry has a right-click menu



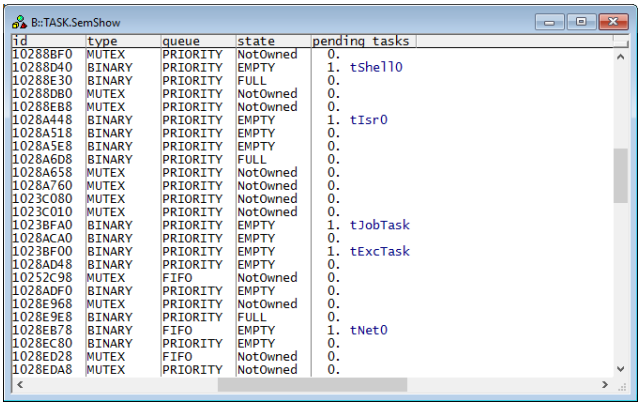

B::TASK.TaskInfo 0x102B08E0

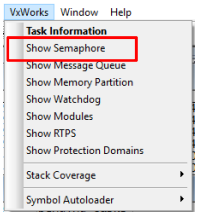
| name | entry | tid | pri | status | pc | sp | errno | delay |
|---|---|---|---|---|---|---|---|---|
| tSieve | 1010929C | 102B08E0 | 21. | RUNNING(1) | -- | -- | 00000000 | 0. |

```
task stack: base 102B08E0  end 102B0110
            size 2000.  high 272.  margin 1728.
exc. stack: base 1052F3AC  end 1052E3B0  start 1052F3B0
            size 4092.  high 0.  margin 4092.

proc id: 1023C460

options: 00009001
VX_SUPERVISOR_MODE VX_DEALLOC_EXC_STACK VX_DEALLOC_TCB

events:  pended on 00000000  received 00000000  options 00

⊞ registers:
```

# Example

- Display a list of Semaphores



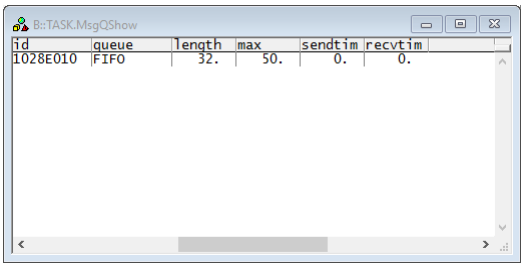



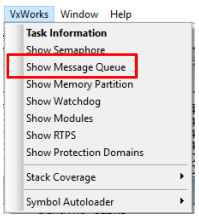





# Example

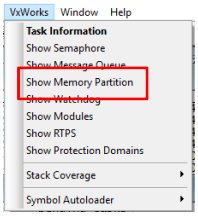
- Display a list of Message Queues



# Example

- Display a list of Memory Partitions




B::TASK.MemPShow

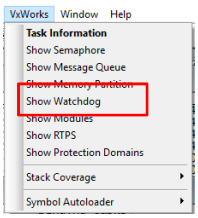
| id | total bytes | free bytes | alloc bytes | free blocks | alloc blocks |
|---|---|---|---|---|---|
| 10253338 | 1069135196. | 1066882144. | 2252628. | 242. | 777. |
| 102A2BB8 | 12040. | 9904. | 2136. | 1. | 34. |





# Example

- Display a list of Watchdog tasks

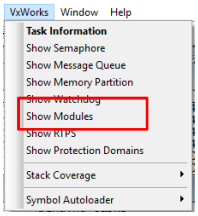



B::TASK.WDShow

| id | state | ticks | routine | parameter | |
|---|---|---|---|---|---|
| 1029E4A8 | IN_Q | -142. | 101E5AE4 | 10298F20 | ipcom_job_queue_schedule_single.. |

# Example

- Display a list of Modules




B::TASK.ModShow

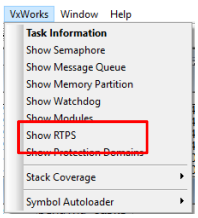
| id | name | group | text start/length | data start/length | bss start/length |
|---|---|---|---|---|---|
| 103346B0 | t32fdx.out | 1. | 1033841C 00000080 | 103384AC 00000008 | 103384B4 0000000C |





# Example

- Display a list of Real Time Processes (Requires RTP)

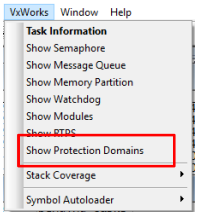



B::TASK.RTPShow

| id | name | space | state | entry | options | task cnt |
|---|---|---|---|---|---|---|
| 1032C1D8 | /romfs/rtp-sieve.vxe | 0001 | normal | 00800274 | 00000041 | 3. |

# Example

- Display a list of Protection Domains (Requires VxWorks653)

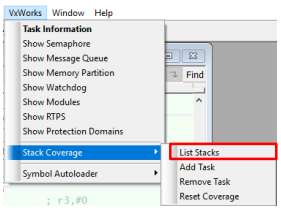


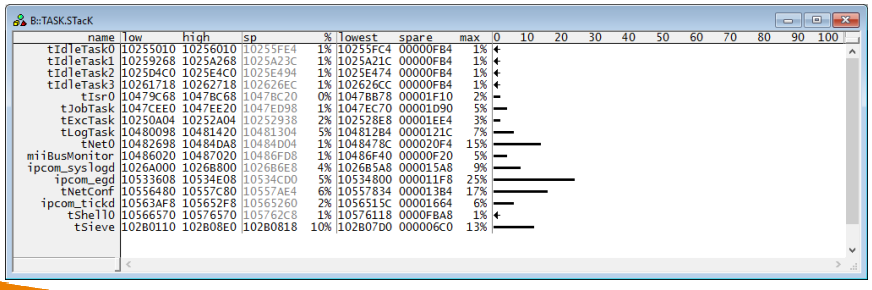





# Example

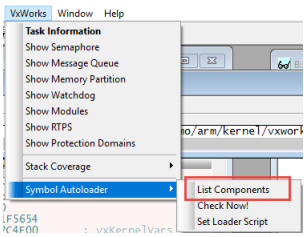
- Display a list of task stacks

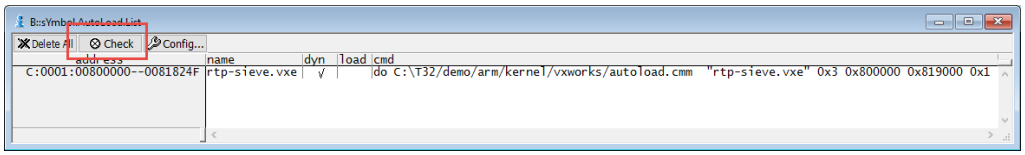


# Example

- To debug an RTP task (1)
  - Open the Symbol Loader
  - If the window is empty, click “Check” to populate the list



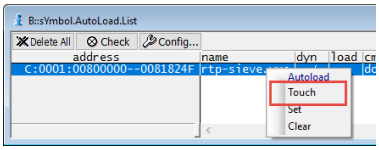



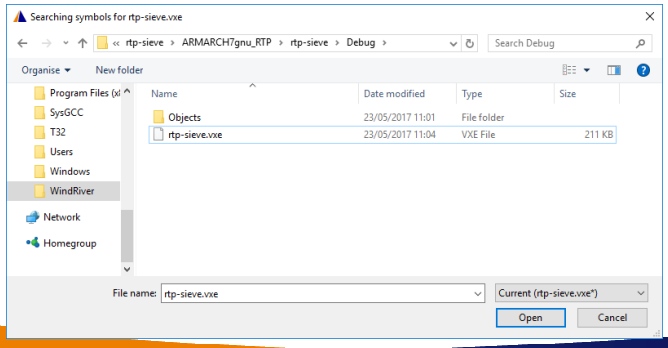





# Example

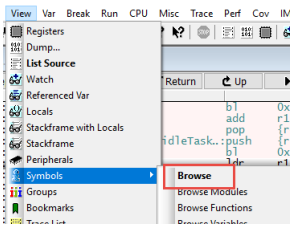
- To debug an RTP task (2)
  - Right-click the RTP and select “Touch” from the menu.
  - Browse for the symbol file

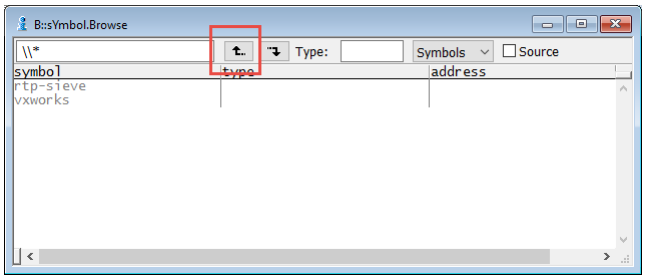


# Example

- To debug an RTP task (3)
  - Open Symbol Browser and navigate to the top



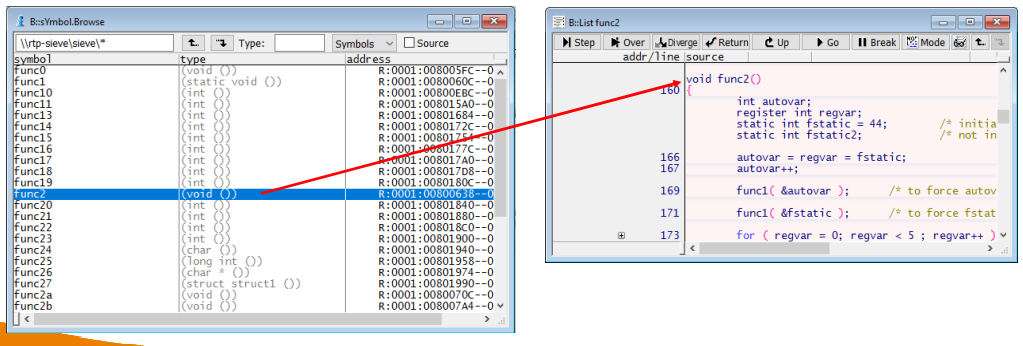







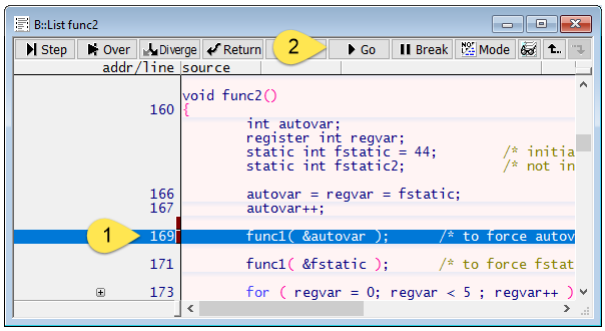

# Example

- To debug an RTP task (4)
  - Navigate Symbol browser
  - Double-click function name to open source listing window



# Example

- To debug an RTP task (5)
  - Double-click line to set breakpoint
  - Run target








# Example

- To debug an RTP task from it's entry point
  - This requires writing a small script to catch the system as it loads the RTP and configures the MMU
  - Once the MMU settings are known the symbols can be loaded and matched to the RTP
  - The next few slides show an example of such a script
  - The script must be executed BEFORE the RTP is started

# Example

- Declare some macros for use in the script

```
LOCAL &rtpname &filename &basename &rtpid &break &entry
```





# Example

- Present the user with a dialog to gather the RTP name and entry point details

```
DIALOG
(
    ICON "[:ecomp]"
    HEADER "Catch RTP at main()"
    POS 0. 0. 32. 3.
    BOX "Enter RTP name (eg: /romfs/myRtp.vxe)"
    POS 1. 1. 30. 1.
R:  EDIT "" ""
    POS 0. 3. 32. 3.
    BOX "Enter entry point (default main)"
    POS 1. 4. 30. 1.
E:  EDIT "" ""
    POS 25. 6. 7. 1.
B:  BUTTON "Start" "CONTinue"
)
STOP
```

# Example

- Populate the macros with the user provided data
- Close the dialog

```
&rtpname=DIALOG.STRing(R)
&entry=DIALOG.STRing(E)
IF "&entry"==""
    &entry="main"
&filename=OS.FILE.NAME("&rtpname")
&basename=STRing.CUT("&filename", \
       -STRing.LENgth(OS.FILE.EXTENSION("&filename")))

DIALOG.END
```





# Example

- If the symbols have already been loaded, clear them
- Halt the target, if required, in order to set the breakpoint

```
// Delete symbols if they already exist
sYmbol.AutoLoad.CLEAR "&filename"
IF sYmbol.EXIST("\\&basename")
    sYmbol.Delete \\&basename

// Halt the target if necessary
IF STATE.RUN()
    Break

// Wait for RTP to be started
Break.Set rtpUserModeSwitch /CONDition TASK.RTP.ID("&filename")!=0xFFFFFFFF
```

# Example

- Pause the script until the breakpoint has been reached
  - Only occurs when a new RTP is started
- Delete the breakpoint an clear the event handler

```
ON PBREAKAT ADDRESS.OFFSET(rtpUserModeSwitch) GOTO continue1

GO
STOP       // halt script until breakpoint reached

// Breakpoint hit, load symbols
continue1:

Break.Delete rtpUserModeSwitch
ON PBREAKAT ADDRESS.OFFSET(rtpUserModeSwitch)    // delete script action
```





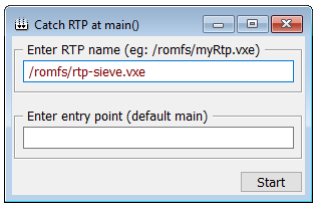
# Example

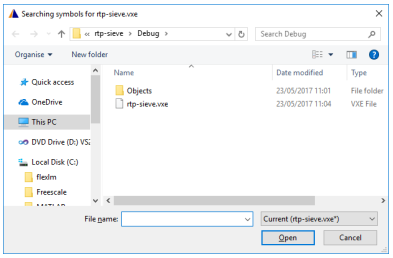
- Force the autoloader to check for new RTPs and load the symbols
- Set a breakpoint at the new RTP's entry point
- Run the target until this is reached and then delete the breakpoint

```
sYmbol.AutoLOAD.CHECK
sYmbol.AutoLOAD.TOUCH "&filename"

// Set Breakpoint on entry
&break="`"+"\\&basename\\&entry"+"`"
Break.Set &break
GO
WAIT !STATE.RUN()
Break.Delete &break

ENDDO
```

# Example

- Enter RTP pathname
- Provide entry point (if not main() )
- Click “Start”
- Launch RTP
- Browse for symbols, when prompted



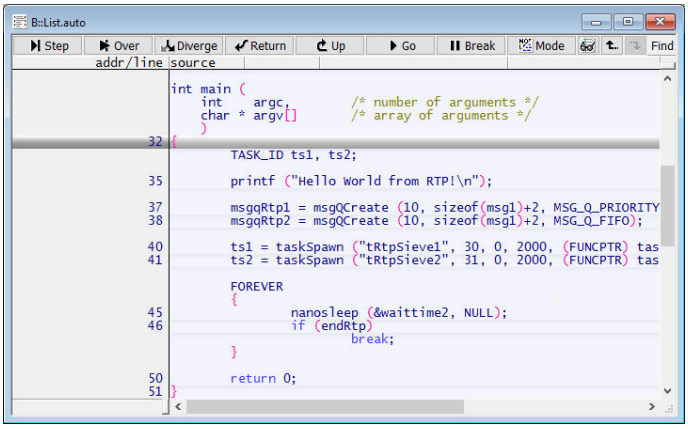








# Example

- RTP halted at entry point



# Example

- To debug a Kernel Module (DKM) from the entry point
  - Use a script.
  - An example will be shown here.
  - Assumes TRACE32 has a terminal window to the target.
  - This script will load and start the DKM.





# Example

- To debug a Kernel Module (DKM) from the entry point
  - Check for arguments

```
LOCAL &file &entry

; get and check parameters
ENTRY &file &entry
IF ("&file"=="")
(
    PRINT "usage: do startdkm <file> [<entry>]"
    ENDDO
)
```

# Example

- To debug a Kernel Module (DKM) from the entry point
  - Populate the macros with data from the arguments
  - Halt the target if necessary

```
; get filename and basename
&filename=OS.FILE.NAME(&file)
&basename=STRing.CUT("&filename", \
      -STRING.LENgth(OS.FILE.EXTENSION(&filename)))

; run target if halted
IF !STATE.RUN()
    Go
```





# Example

- To debug a Kernel Module (DKM) from the entry point
  - Clear any existing symbols for the DKM
  - Switch the target shell to the Cshell

```
; delete possibly existing symbols
sYmbol.AutoLOAD.CLEAR "&filename"
IF sYmbol.EXIST("\\&basename")
    sYmbol.Delete \\&basename

; change to command mode
TERM.Out "cmd" 0x0a
WAIT 0.2s
```

# Example

- To debug a Kernel Module (DKM) from the entry point
  - If the DKM has already been loaded, remove it

```
; check if DKM already loaded and remove it if necessary
Break
IF task.modid("&filename")!=0xffffffff
(
  Go
  TERM.Out "module unload &filename" 0x0a
  WAIT 0.2s
)
ELSE
  Go
```





# Example

- To debug a Kernel Module (DKM) from the entry point
  - Load the module via the shell and list all loaded modules
  - Force the autoloader to check and load the symbols

```
; load DKM
TERM.Out "ld &file" 0x0a
WAIT 0.2s
TERM.Out "module list" 0x0a
WAIT 0.2s

; load symbols of DKM
Break
sYmbol.AutoLOAD.CHECK
sYmbol.AutoLOAD.TOUCH "&filename"
```

# Example

- To debug a Kernel Module (DKM) from the entry point
  - Set a breakpoint on entry and start the module
  - Target will halt at the entry and then remove the breakpoint

```
if "&entry"!=""
(
     ; set breakpoint on DKM entry point,
     Break.Set &entry
     Go
     TERM.Out "C" 0x0a
     WAIT 0.2s
     TERM.Out "&entry" 0x0a
     WAIT !RUN()
     Break.Delete &entry
)
ENDDO
```





# Example

- For more information how to drive the debugger please refer to the Lauterbach training manuals
  - `~~/pdf/training_debugger.pdf`
  - `~~/pdf/training_hll.pdf`
- More information on the TRACE32 VxWorks awareness can be found in
  - `~~/pdf/rtos_vxworks.pdf`

# THANK YOU!

▪ Richard Copeman ▪
Richard.copeman@lauterbach.com

# Questions?



LAUTERBACH
DEVELOPMENT TOOLS