

# TRACE32 and VxWorks 6.9

• Richard Copeman •  
2017 / June / 06

[www.lauterbach.com](http://www.lauterbach.com)



• 2 •

## 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**

## 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

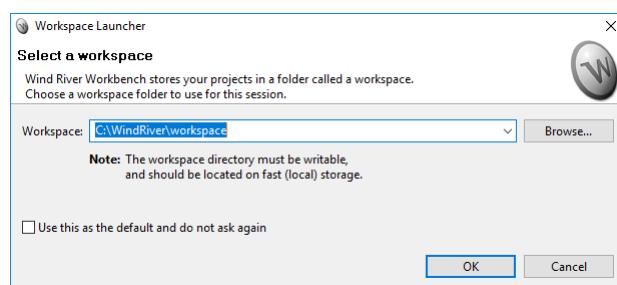
- Development Host & Target
- Build OS Image (no MMU, no RTP)
- Build OS Image (MMU, RTP)
- Configure TRACE32 (no MMU, non RTP)
- Configure TRACE32 (MMU, RTP)
- Example

## 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'



• 7 •

## Build OS Image (non RTP)

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

Basic Device Development - Wind River Workbench

File Edit Navigate Search Project Run Window

Project...

Build Target

Folder

File

File from Template

Example...

Other... Ctrl+N

Select a wizard

Creates a new VxWorks image project with all available kernel build specs

Wizards:

VxWorks 6.x

Example EMF Model Creation Wizards

JavaScript

Remote System Explorer

VxWorks fx

VxWorks Boot Loader / BSP Project

VxWorks Downloadable Kernel Module Project

**VxWorks Image Project**

VxWorks Real Time Process Project

VxWorks ROMFS File System Project

VxWorks Shared Library Project

VxWorks Source Build (Kernel Library) Project

Show All Wizards.

Next > Finish Cancel

TRACE32 and VxWorks • Richard Copeman • 2017 / 06 / 06 • www.lauterbach.com

LAUTERBACH DEVELOPMENT TOOLS

• 8 •

## Build OS Image (non RTP)

- 1) Enter Project name
- 2) Click 'Next>'

New VxWorks Image Project

Create a new VxWorks image project with all available kernel build specs.

Project

Project name: **SabreLite** 1

Location

Create project in workspace

Create project at external location

Directory: C:\WindRiver\workspace\SabreLite 2 Browse...

Next > Finish Cancel

TRACE32 and VxWorks • Richard Copeman • 2017 / 06 / 06 • www.lauterbach.com

LAUTERBACH DEVELOPMENT TOOLS

• 9 •

## 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>’

Project Setup  
Base the new project either on an existing project, or on a board support package and a tool chain.

Setup the project  
Based on: a board support package 1  
Project: fsl\_imx6\_sabrelite 2  
BSP: fsl\_imx6\_sabrelite 3  
Address mode: 32-bit kernel 4  
Tool chain: gnu

Target Agent  
 Enable WDB Target Agent

BSP validation test suite  
 Add support to project Options...

Setup information  
Base directory: C:/WindRiver/vxworks-6.9/target/config/fsl\_imx6\_sabrelite

?

5

Next > Finish Cancel

TRACE32 and VxWorks • Richard Copeman • 2017 / 06 / 06 • www.lauterbach.com

**LAUTERBACH**  
DEVELOPMENT TOOLS

• 10 •

## Build OS Image (non RTP)

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

Options  
Select the options to be used.

Options  
 Select All  Deselect All  
 SMP support in kernel 1  
 IPv6 enabled kernel libraries  
 Debug enabled and compiler optimizations disabled 2  
 Debug enabled and normal compiler optimizations enabled

?

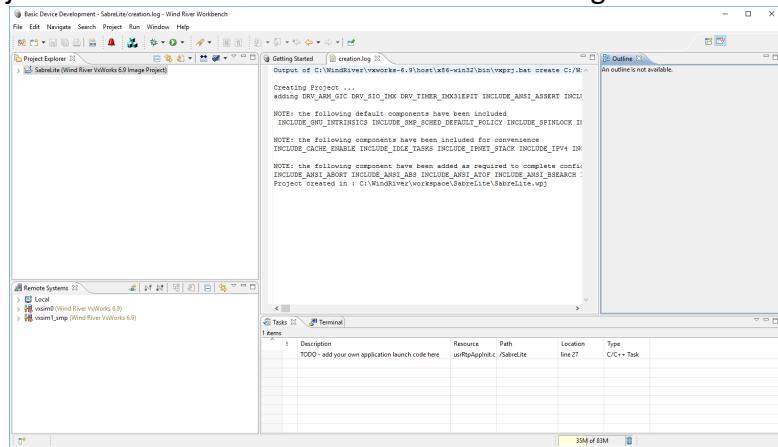
< Back 3 Finish Cancel

TRACE32 and VxWorks • Richard Copeman • 2017 / 06 / 06 • www.lauterbach.com

**LAUTERBACH**  
DEVELOPMENT TOOLS

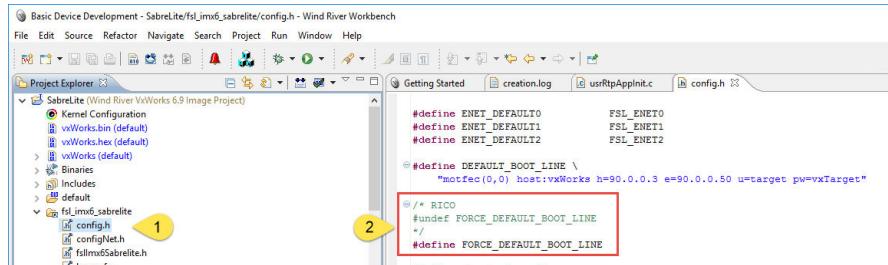
## Build OS Image (non RTP)

- Project is created and the WorkBench view changes



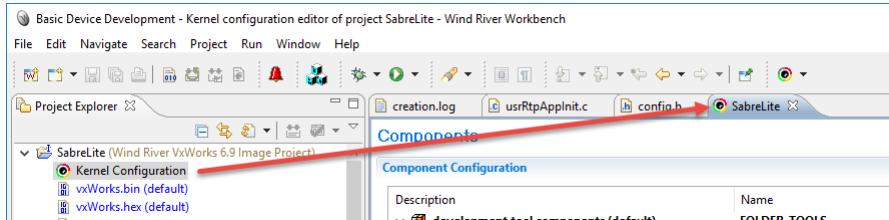
## Build OS Image (non RTP)

- Edit **config.h**
- Ensure that **FORCE\_DEFAULT\_BOOT\_LINE** is defined



## Build OS Image (non RTP)

- Edit Kernel Configuration



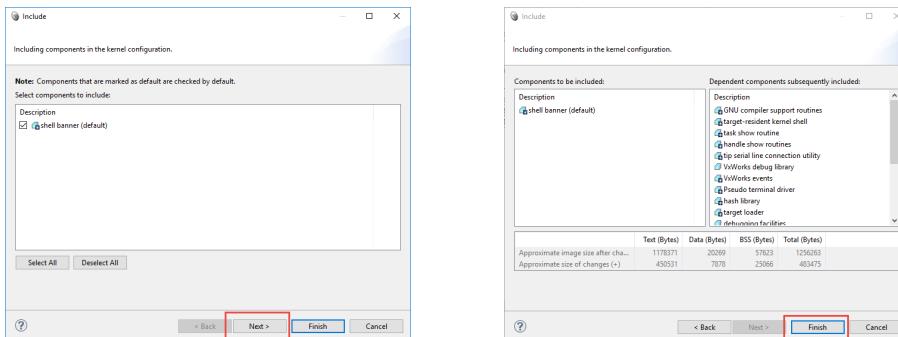
## Build OS Image (non RTP)

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

Description	Name	Type	Value
> C++ components	FOLDER_CPLUS	FOLDER	
> Media Library Component	FOLDER_VIMODM	FOLDER	
> Multi-OS	FOLDER_MULTIOS	FOLDER	
> Network Components (default)	FOLDER_NETWORK	FOLDER	
> Startup Sequence and Initialization Components	FOLDER_SSI	FOLDER	
> application components	FOLDER_APPLICATION	FOLDER	
> development tool components (default)	FOLDER_DEBUG_AGENT	FOLDER	
> File analysis system	FOLDER_CAFE	FOLDER	
> Core Dump components	FOLDER_CORE_DUMP	FOLDER	
> Downloadable kernel modules compiler support	SELECT_COMPILER_INT...	SELECT	
> Kernel-write components	FOLDER_KERNEL_DEBUG	FOLDER	
> System Viewer components	FOLDER_WINDVIEW	FOLDER	
> USB Debug	FOLDER_USB_DEBUG	FOLDER	
> WDB Agent Proxy components	FOLDER_WDB_PROXY	FOLDER	
> Web browser components (default)	FOLDER_BROWSER	FOLDER	
> boot application components	FOLDER_ROOT_APP	FOLDER	
> kernel shell components	FOLDER_SHELL	FOLDER	
> shell commands	FOLDER_SHELL_CMD	FOLDER	
> shell editing mode (default)	SELECT_SHELL_EDIT_M...	SELECT	
> shell interpreter selection (default)	SELECT_SHELL_INTERP	SELECT	
> debugging facilities (default)	INCLUDE_DEBUG	INCLUDE	
> kernel shell startup script	INCLUDE_STARTUP_SCI	INCLUDE	
> shell environment	INCLUDE_SHELL_ENV	INCLUDE	
> user memory sampling loading mechanism	INCLUDE_SPECIFICATOR...	INCLUDE	
> target-resident kernel shell (default)	INCLUDE_SHELL	INCLUDE	
> loader components	FOLDER_LOADER	FOLDER	

## Build OS Image (non RTP)

- Click Next>
- Click Finish



## Build OS Image (non RTP)

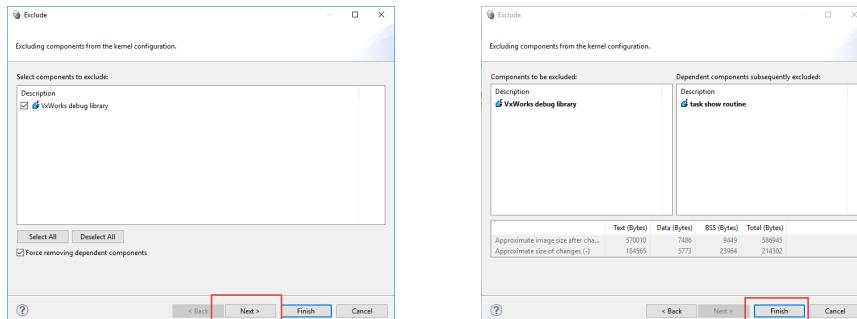
- Exclude VxWorks debug library (right-click->Exclude)

Screenshot of the 'Components' tab in a configuration tool. The 'VxWorks debug library' component is selected and highlighted with a red box. Underneath it, the 'VxDDBG events task options' and 'VxDDBG task priority' sub-options are also highlighted with a red box.

Description	Name	Type	Value
> application components	FOLDER_APPLICATION		
> development tool components (default)	FOLDER_TOOLS		
> Application Mode Agent components	FOLDER_DEBUG_AGENT		
> Cafe analysis system	FOLDER_CAFE		
> Core Dump components	FOLDER_CORE_DUMP		
> Downloadable kernel modules compiler support routine	SELECT_COMPILER_INTRINSICS		
> Kernel-write components	FOLDER_KERNEL_DEBUG		
> System Viewer components	FOLDER_WINDVIEW		
> USB Debug	FOLDER_USB_DEBUG		
> WDB Agent Proxy components	FOLDER_WDB_PROXY		
> WDB agent components (default)	FOLDER_WDB		
> boot application components	FOLDER_BOOT_APP		
> kernel shell components	FOLDER_SHELL		
> loader components	FOLDER_LOADER		
> show routines	FOLDER_SHOW_ROUTINES		
> symbol table components	FOLDER_SYMBOL		
> Compiler Assisted Run-Time Checker	INCLUDE_MEM_EDR_RTC		
> Memory Error Detection and Reporting	INCLUDE_MEM_EDR		
> Stack tracing logic	INCLUDE_STACKTRACE		
> VxWorks debug library	INCLUDE_VXDDBG		
VxDDBG events task options	VXDDBG_EVT_TASK_OPTIONS	VX_UNBREAK...	
VxDDBG task priority	VXDDBG_EVT_TASK_PRIORITY	uint	25

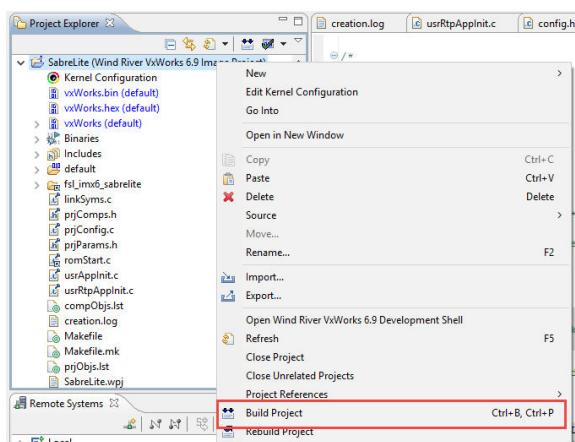
## Build OS Image (non RTP)

- 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

```

/*
 * DESCRIPTION
 * Initialize user application code.
 */
#include <vxWorks.h>
#include <tasKlib.h>
#ifndef PRJ_BUILD
#include "pjParams.h"
#endif /* defined PRJ_BUILD */

extern int usrSieve(void);
/* ****
 * usrAppInit - initialize the users application
 */

void usrAppInit (void)
{
    #ifndef USER_APPL_INIT
    USER_APPL_INIT; /* for backwards compatibility */
    #endif

    /* add application specific code here */
    taskSpawn("tSieve", 21, 0, 2000, (FUNCFT)usrSieve, 0,0,0,0,0,0,0,0,0,0,0,0);
}

```

## 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

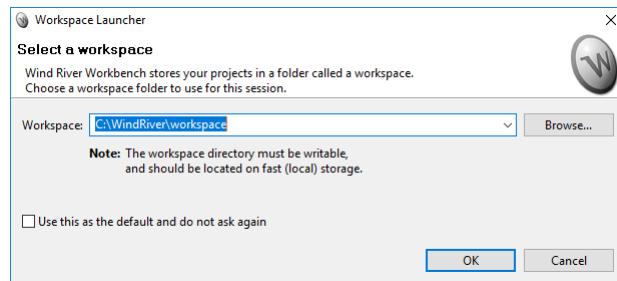
- Development Host & Target
- Build OS Image (no MMU, no RTP)
- **Build OS Image (MMU, RTP)**
- Configure TRACE32 (non MMU, no RTP)
- Configure TRACE32 (MMU, RTP)
- Example

## 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

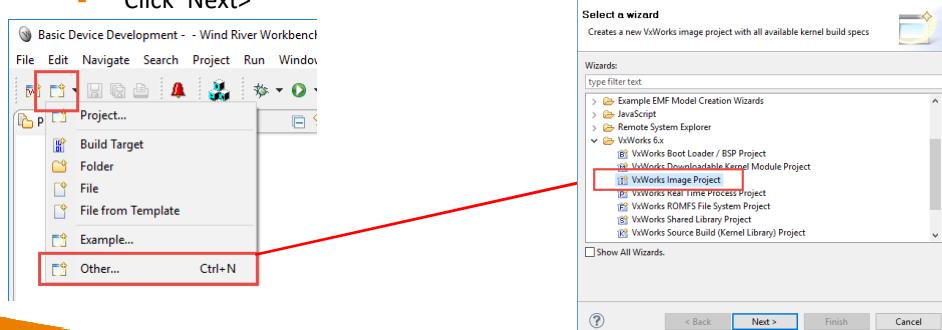
## Build OS Image (RTP)

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



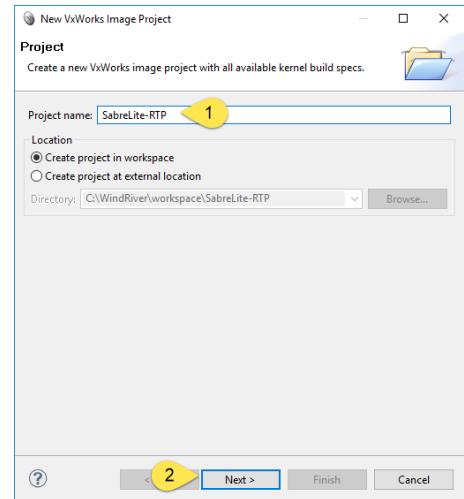
## Build OS Image (RTP)

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



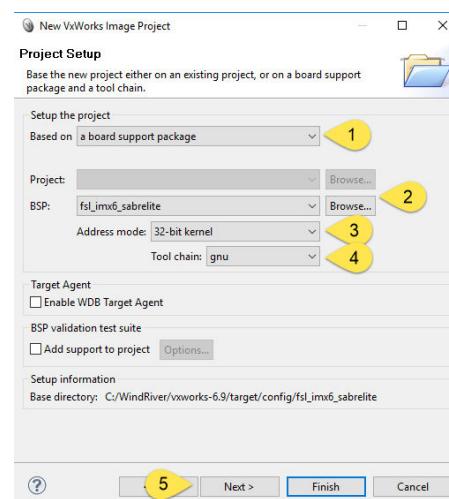
## Build OS Image (RTP)

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



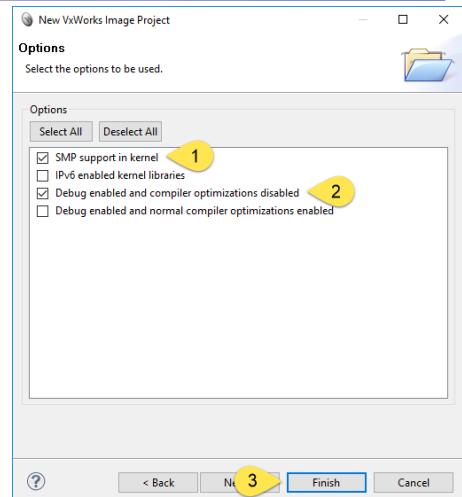
## Build OS Image (RTP)

- 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>’



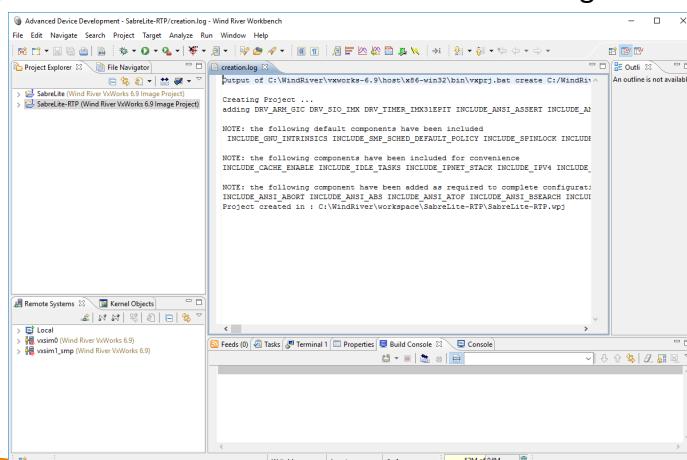
## Build OS Image (RTP)

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



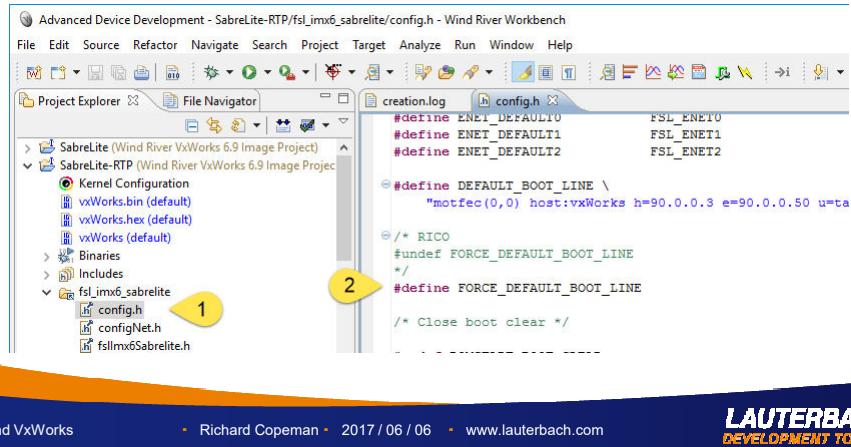
## Build OS Image (RTP)

- Project is created and the WorkBench view changes



## Build OS Image (RTP)

- Edit config.h
- Ensure that **FORCE\_DEFAULT\_BOOT\_LINE** is defined



```
#define ENET_DEFAULT0 FSL_ENET0
#define ENET_DEFAULT1 FSL_ENET1
#define ENET_DEFAULT2 FSL_ENET2

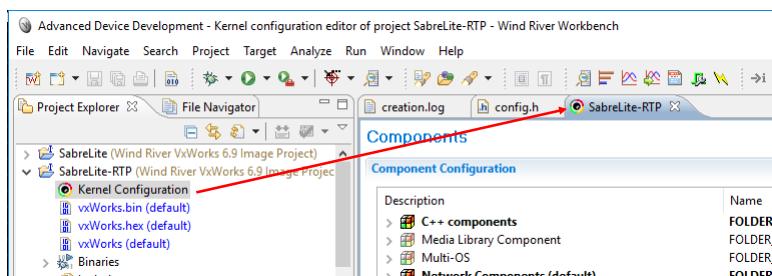
#define DEFAULT_BOOT_LINE \
    "motfec(0,0) host:vxWorks b=90.0.0.3 e=90.0.0.50 u=t"

/* RICO
#undef FORCE_DEFAULT_BOOT_LINE
*/
#define FORCE_DEFAULT_BOOT_LINE

/* Close boot clear */
```

## Build OS Image (RTP)

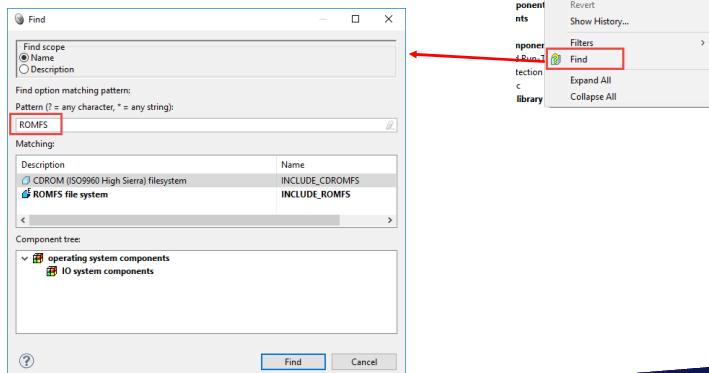
- Edit Kernel Configuration



Description	Name
> C++ components	FOLDER
> Media Library Component	FOLDER
> Multi-OS	FOLDER
> Network Components (default)	FOLDER

## 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

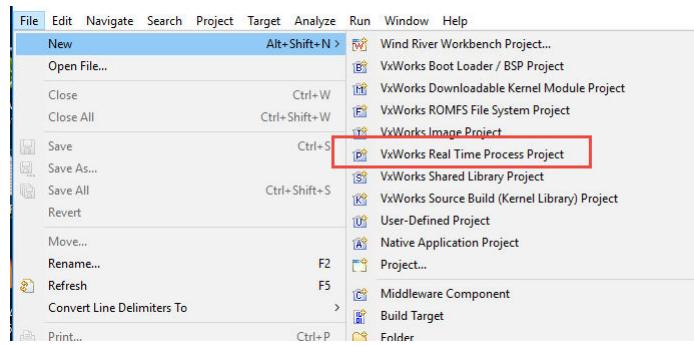


## Build OS Image (RTP)

- Add or remove the following components
- INCLUDE\_ROMFS
- INCLUDE\_RTP
- INCLUDE\_SHELL\_BANNER
- INCLUDE\_DISK\_UTIL\_SHELL\_CMD
- EXCLUDE\_VXDBG

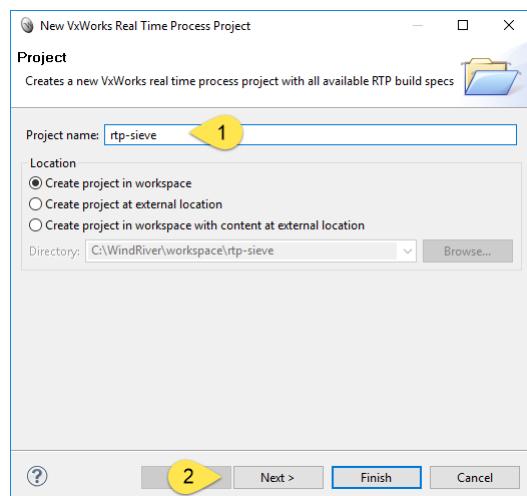
## Build OS Image (RTP)

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



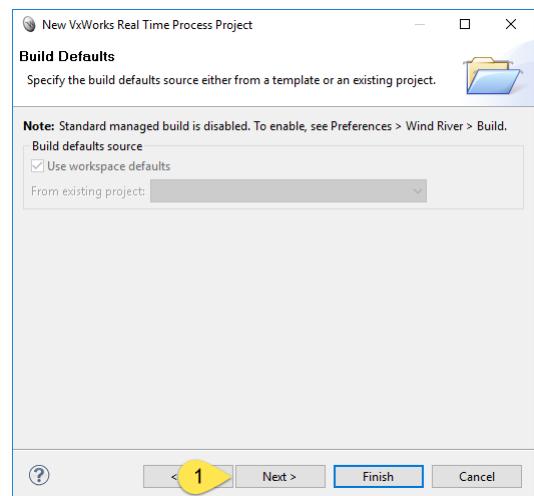
## Build OS Image (RTP)

- Name the project
- Click 'Next>'



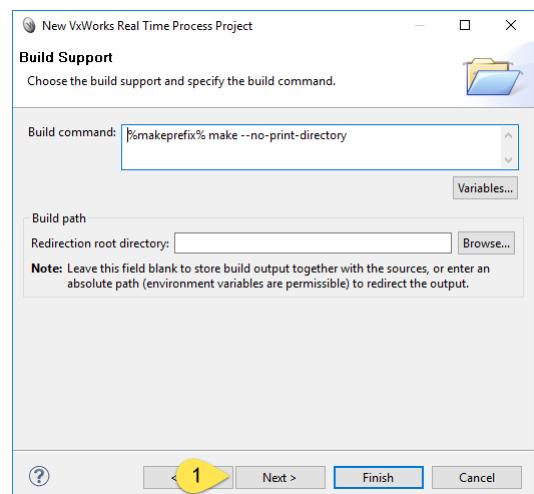
## Build OS Image (RTP)

- Click 'Next>'



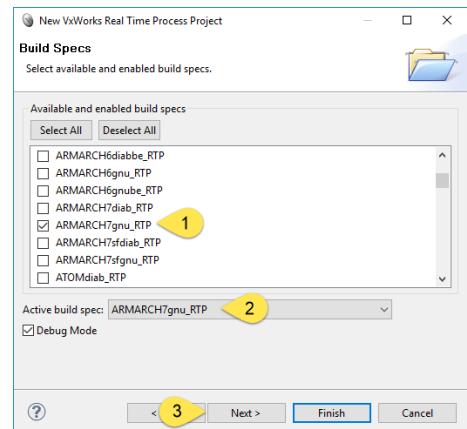
## Build OS Image (RTP)

- Click 'Next>'



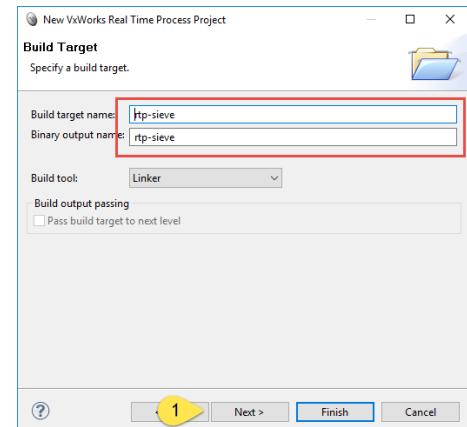
## Build OS Image (RTP)

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



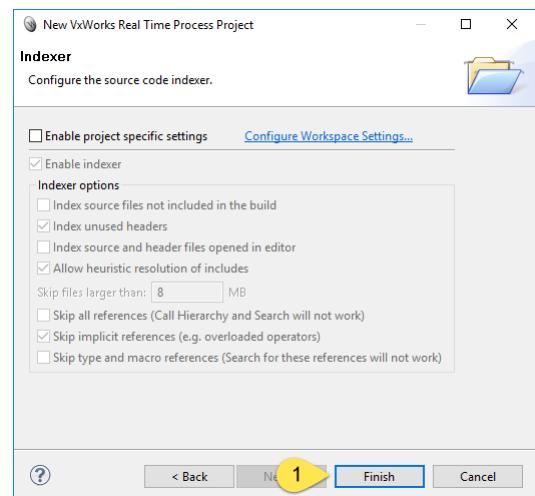
## Build OS Image (RTP)

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



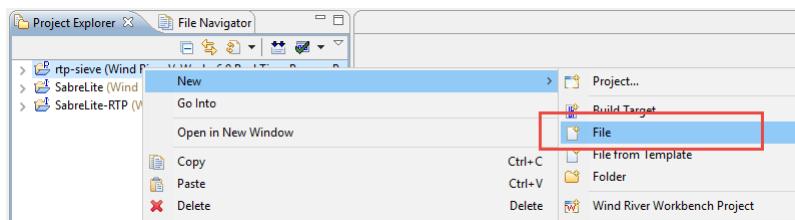
## Build OS Image (RTP)

- Click 'Finish'



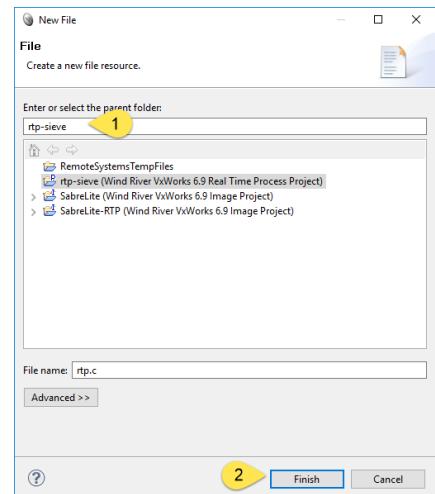
## Build OS Image (RTP)

- Add New Source File



## Build OS Image (RTP)

- Enter filename
- Click ‘Finish’



## Build OS Image (RTP)

- Enter your source code

```

/* includes */

#include <stdio.h>
#include <msgQLib.h>
#include <timerLib.h>

void taskRtpSieve1 (void);
void taskRtpSieve2 (void);

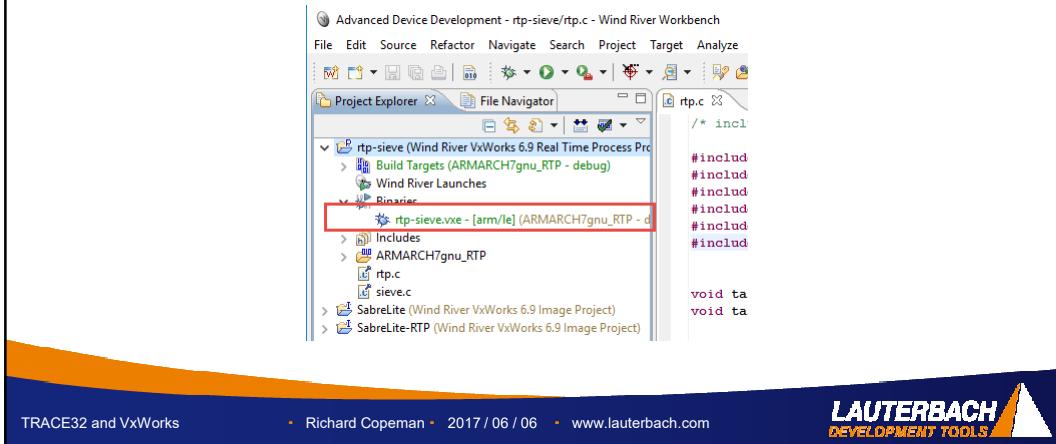
MSG_Q_ID msgqRtp1;
MSG_Q_ID msgqRtp2;

char* msg1 = "Hello from sieve 1";
char* msg2 = "Hello from sieve 2";

struct timespec waittime = {0, 1000}; // 1ms
struct timespec waittime2 = {0, 5000}; // 5ms
  
```

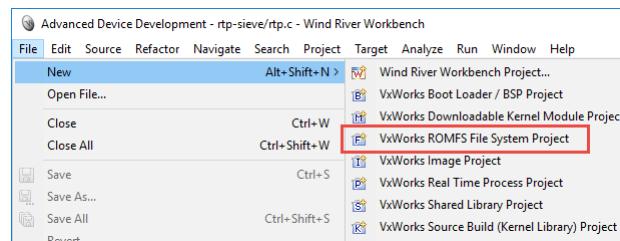
## Build OS Image (RTP)

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



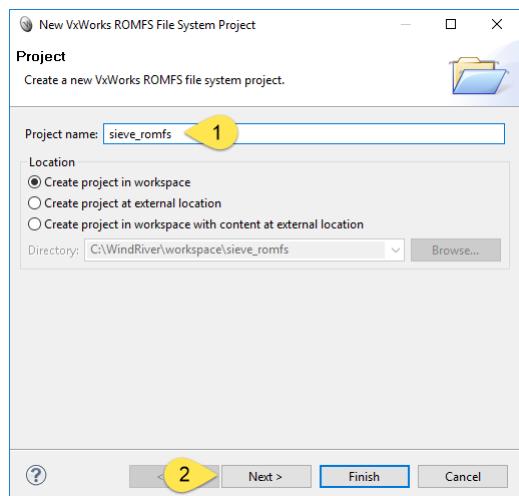
## Build OS Image (RTP)

- Create a new VxWorks ROMFS File System Project



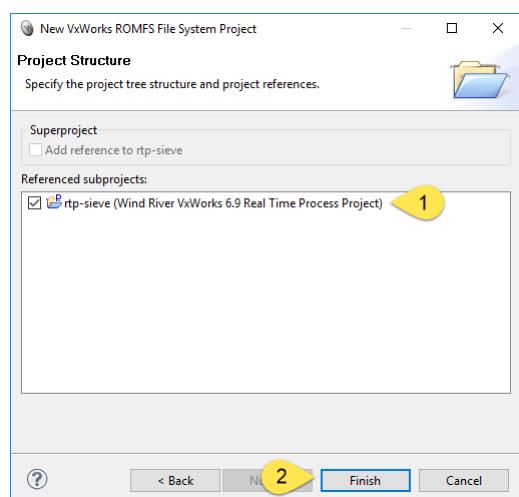
## Build OS Image (RTP)

- Name the project
- Click 'Next>'



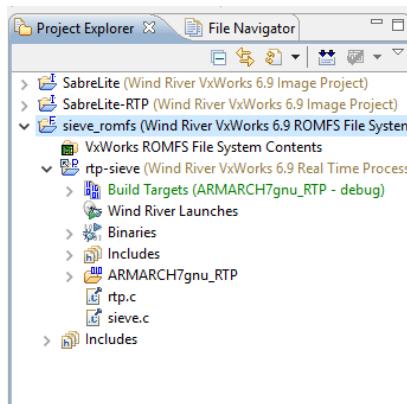
## Build OS Image (RTP)

- Select all required project references
- Click 'Finish'



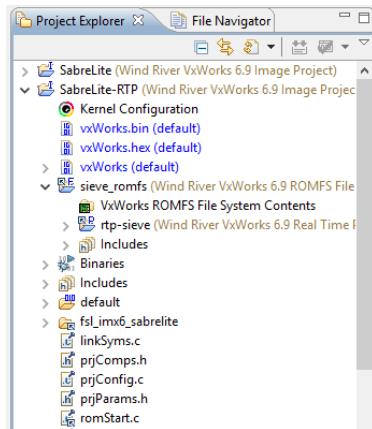
## Build OS Image (RTP)

- Project **rtp-sieve** is now a sub project of **sieve\_romfs**



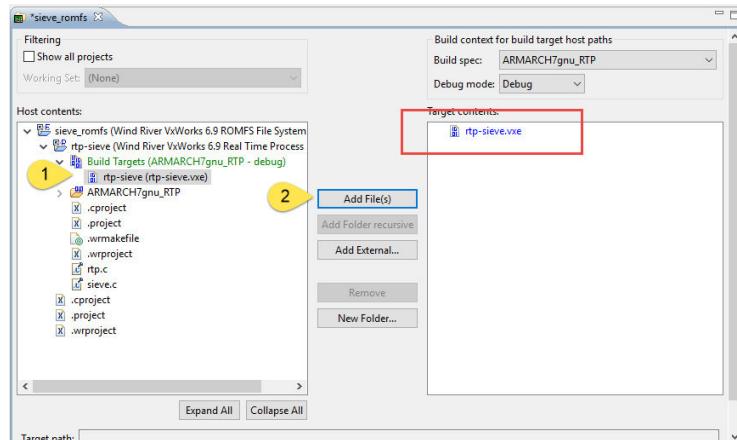
## Build OS Image (RTP)

- 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;

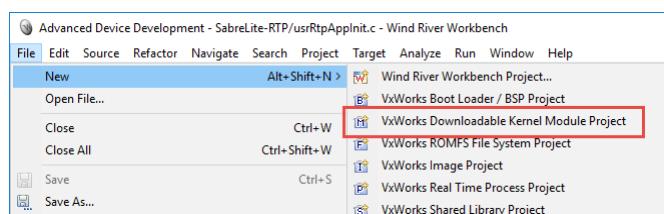
    if (( rtpID = rtpSpawn(vxeName, argv, NULL, 220, 0x10000,0, 0 )==NULL)
    {
        printf("Unable to spawn rtp-sieve\n");
    }
}
```

## Build OS Image (RTP)

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

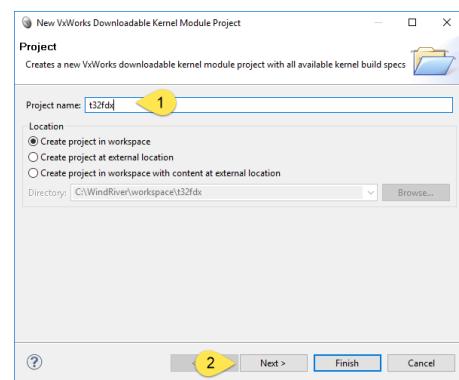
## Build OS Image (RTP)

- Add Kernel Module



## Build OS Image (RTP)

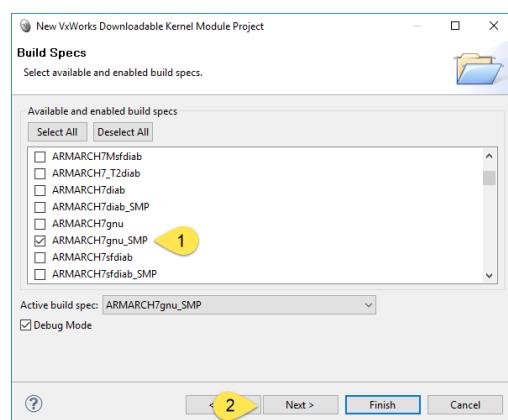
- Enter name
- Click 'Next>'



- 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

The screenshot shows the Wind River Workbench interface. On the left, the Project Explorer window displays three projects: 'SabreLite (Wind River VxWorks 6.9 Image Project)', 'Sabrelite\_RTD (Wind River VxWorks 6.9 Image Project)', and 't32fdx (Wind River VxWorks 6.9 Downloadable Kernel M...)' which is highlighted with a red box. On the right, a dialog box titled 'New VxWorks Downloadable Kernel Module Project' is open, showing the 'Build Target' section where 't32fdx' is selected in the 'Build target name' field. A yellow arrow labeled '1' points to this field. Below it, the 'Binary output name' field also contains 't32fdx'. A yellow arrow labeled '2' points to the 'Finish' button at the bottom right of the dialog. The status bar at the bottom of the screen shows 'TRACE32 and VxWorks', 'Richard Copeman • 2017 / 06 / 06', and 'www.lauterbach.com'. The Lauterbach logo is visible in the bottom right corner.

## Build OS Image (RTP)

- Add your source files to the DKM project

The screenshot shows the Wind River Workbench code editor with an open file named 't32fdx.c'. The code contains C code for a function 't32fdxstart' that prints a message and enters a loop with a 100ms delay using 'nanosleep'. A yellow box highlights the line 'nanosleep(&waittime, NULL);'. The status bar at the bottom of the screen shows 'TRACE32 and VxWorks', 'Richard Copeman • 2017 / 06 / 06', and 'www.lauterbach.com'. The Lauterbach logo is visible in the bottom right corner.

```

#include "vxWorks.h"
#include "stdio.h"
#include <msgQLib.h>
#include <time.h>

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

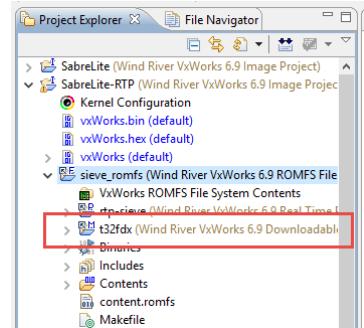
int t32fdxstart(void)
{
    int i;
    printf ("Hello World from DKM!\n");

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

```

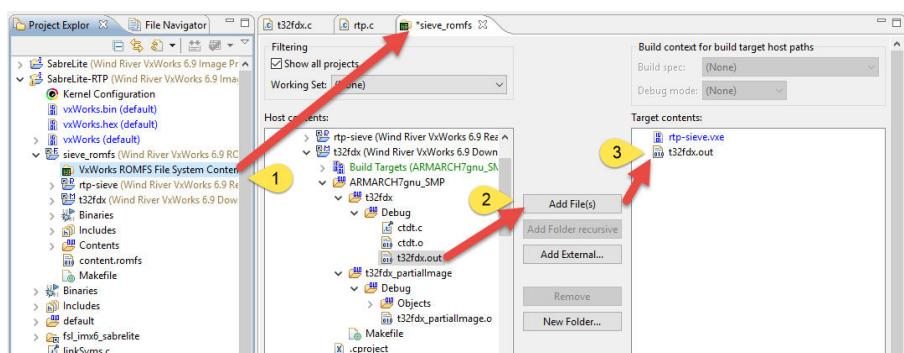
## Build OS Image (RTP)

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



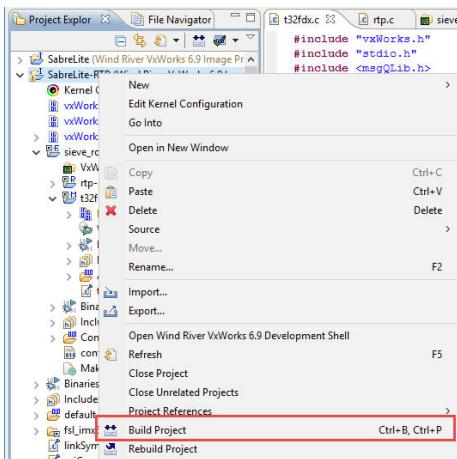
## Build OS Image (RTP)

- 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
```

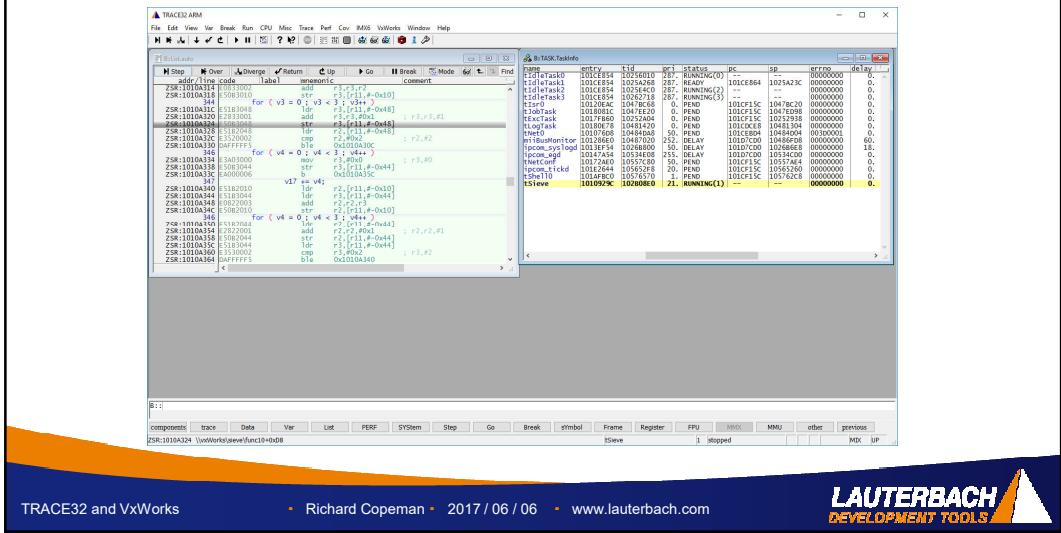
## Configure TRACE32

- 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

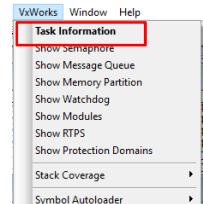
## Example

- 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

name	entry	tid	pri	status	pc	sp	errno	delay
tidlerTask0	101CE854	10256010	287.	RUNNING(0)	--	101CE864	1025A23C	00000000
tidlerTask1	101CE854	1025A268	287.	READY	101CE864	1025A23C	00000000	0.
tidlerTask2	101CE854	1025E4C0	287.	RUNNING(2)	--	--	00000000	0.
tidlerTask3	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	101076D8	10484DAB	50.	PEND	101CEBD0	10484D04	003D0001	0.
mt18usMonitor	101076D0	10484D00	250.	DELAY	101CEBD0	10484D08	00000000	60.
ipcom_syslogd	1013EF54	1026B800	50.	DELAY	101D7C00	1026B6E8	00000000	18.
ipcom_eggd	10147A54	10534E08	255.	DELAY	101D7C00	10534C00	00000000	0.
tNetConf	10172AE0	10557C80	50.	PEND	101CF15C	10557AE4	00000000	0.
ipcom_ticd	101E2644	105652F8	20.	PEND	101CF15C	10565260	00000000	0.
tShell10	101AFBC0	10576570	1.	PEND	101CF15C	105762C8	00000000	0.
<b>tSieve</b>	<b>1010929C</b>	<b>102B08E0</b>	<b>21.</b>	<b>RUNNING(1)</b>	<b>--</b>	<b>--</b>	<b>00000000</b>	<b>0.</b>

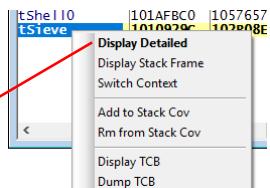


## Example

- Each entry has a right-click menu

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

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: 102C460  
options: 00009001  
VX\_SUPERVISOR\_MODE VX DEALLOC\_EXC\_STACK VX DEALLOC\_TCB  
events: pending on 00000000 received 00000000 options 00  
@ registers:



## Example

- Display a list of Semaphores

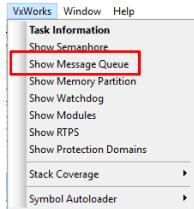


The screenshot shows a Windows-style application window titled "B:TASK.SemShow". Inside, there is a table with columns: id, type, queue, state, and pending tasks. The table lists various semaphores, such as tshell0, tIsr0, tJobTask, tExcTask, and tNet0, along with their properties like type (MUTEX or BINARY), priority, and current state (NotOwned or FULL). A context menu is open over the table, with the "Show Semaphore" option highlighted.

<b>id</b>	<b>type</b>	<b>queue</b>	<b>state</b>	<b>pending tasks</b>
10288BF0	MUTEX	PRIORITY	NotOwned	0.
10288D40	BINARY	PRIORITY	EMPTY	1. tshell0
10288E30	BINARY	PRIORITY	FULL	0.
10288DB0	MUTEX	PRIORITY	NotOwned	0.
10288EB0	MUTEX	PRIORITY	NotOwned	0.
1028A448	BINARY	PRIORITY	EMPTY	1. tIsr0
1028A518	BINARY	PRIORITY	EMPTY	0.
1028A5E8	BINARY	PRIORITY	EMPTY	0.
1028A6D8	BINARY	PRIORITY	FULL	0.
1028A658	MUTEX	PRIORITY	NotOwned	0.
1028A760	MUTEX	PRIORITY	NotOwned	0.
1028A800	MUTEX	PRIORITY	NotOwned	0.
1023C010	MUTEX	PRIORITY	NotOwned	0.
1023BF40	BINARY	PRIORITY	EMPTY	1. tJobTask
1028AC40	BINARY	PRIORITY	EMPTY	0.
1023BF00	BINARY	PRIORITY	EMPTY	1. tExcTask
1028AD48	BINARY	PRIORITY	EMPTY	0.
10252C98	MUTEX	FIFO	NotOwned	0.
1028ADF0	BINARY	PRIORITY	EMPTY	0.
1028E968	MUTEX	PRIORITY	NotOwned	0.
1028E9E8	BINARY	PRIORITY	FULL	0.
1028EB78	BINARY	FIFO	EMPTY	1. tNet0
1028EC80	BINARY	PRIORITY	EMPTY	0.
1028ED28	MUTEX	FIFO	NotOwned	0.
1028EDA8	MUTEX	PRIORITY	NotOwned	0.

## Example

- Display a list of Message Queues

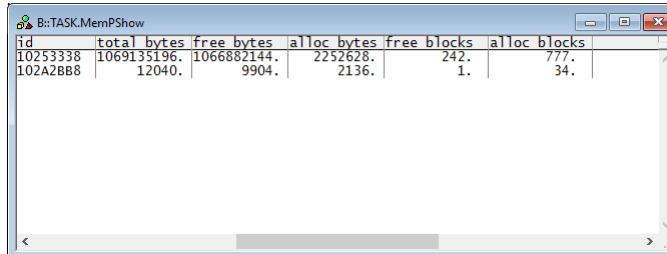


The screenshot shows a Windows-style application window titled "B:TASK.MsgQShow". Inside, there is a table with columns: id, queue, length, max, sendtim, and recvtim. The table lists a single message queue named tNet0, which is FIFO, has a length of 32, a maximum length of 50, and no scheduled send or receive times. A context menu is open over the table, with the "Show Message Queue" option highlighted.

<b>id</b>	<b>queue</b>	<b>length</b>	<b>max</b>	<b>sendtim</b>	<b>recvtim</b>
1028E010	FIFO	32.	50.	0.	0.

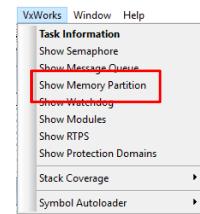
## Example

- Display a list of Memory Partitions



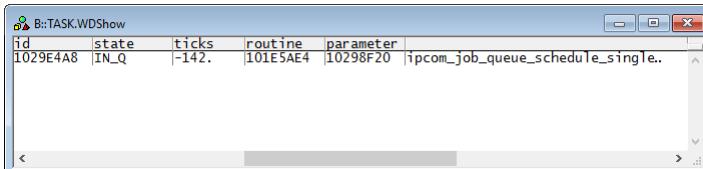
B:TASK.MemPShow

<b>id</b>	<b>total bytes</b>	<b>free bytes</b>	<b>alloc bytes</b>	<b>free blocks</b>	<b>alloc blocks</b>
10253338	1069135196.	1066882144.	2252628.	242.	777.
102A2BB8	12040.	9904.	2136.	1.	34.



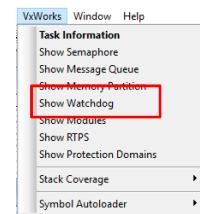
## Example

- Display a list of Watchdog tasks



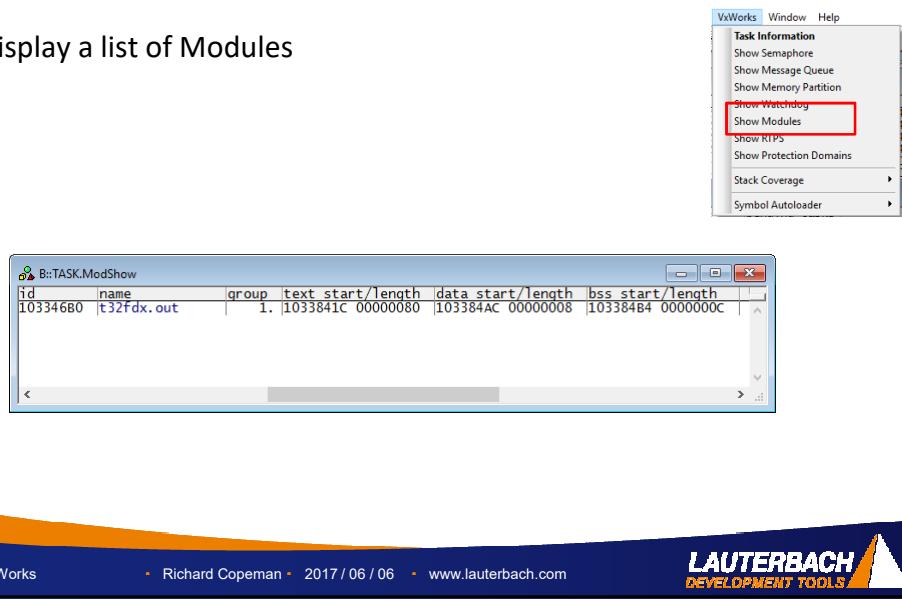
B:TASK.WDShow

<b>id</b>	<b>state</b>	<b>ticks</b>	<b>routine</b>	<b>parameter</b>
1029E4AB	IN_Q	-142.	101E5AE4	10298F20 ipcom_job_queue_schedule_single..



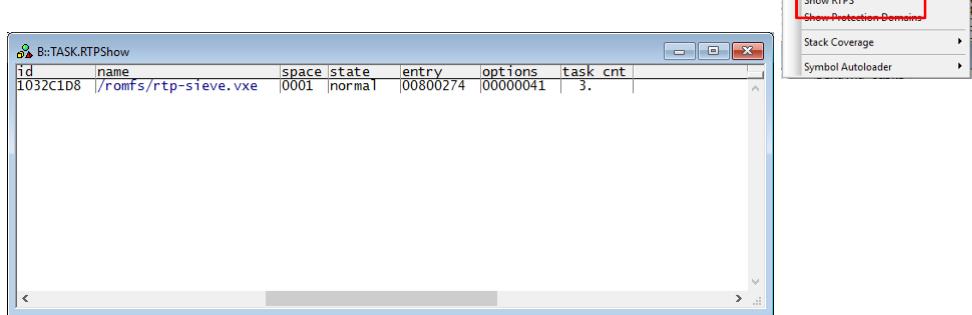
## Example

- Display a list of Modules



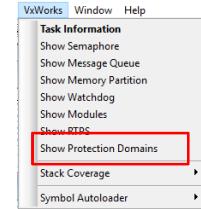
## Example

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



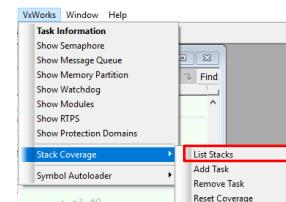
## Example

- Display a list of Protection Domains (Requires VxWorks653)



## Example

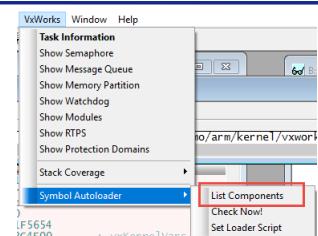
- Display a list of task stacks



	name	low	high	sp	% lowest	spare	max	0	10	20	30	40	50	60	70	80	90	100
	tidleTask0	10255010	10256010	10255FE4	1%	10255FFC4	00000FB4	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%
	tidleTask1	10259268	1025A268	1025A23C	1%	1025A21C	00000FB4	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%
	tidleTask2	1025D4C0	1025E4C0	1025E494	1%	1025E474	00000FB4	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%
	tidleTask3	10261710	10262718	102626EC	1%	102626CC	00000FB4	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%
	tsr0	10479C60	1047BC68	1047BC20	0%	1047BBC8	00001F10	2%	2%	2%	2%	2%	2%	2%	2%	2%	2%	2%
	tjobTask	1047E000	1047EE20	1047ED98	1%	1047EC70	00001D90	5%	5%	5%	5%	5%	5%	5%	5%	5%	5%	5%
	texecTask	1047E5000	1047E508	1047E508	2%	1047E508	00001D90	3%	3%	3%	3%	3%	3%	3%	3%	3%	3%	3%
	tloopTask	10480098	10481420	10481304	2%	104812B4	0000121C	3%	3%	3%	3%	3%	3%	3%	3%	3%	3%	3%
	tNet0	10482698	10484048	1048AD04	1%	1048478C	000020F4	15%	15%	15%	15%	15%	15%	15%	15%	15%	15%	15%
	miibusMonitor	10486020	10487020	10486FD8	1%	10486F40	00000E20	5%	5%	5%	5%	5%	5%	5%	5%	5%	5%	5%
	ipcom_syslogd	1026A000	1026B800	1026B6E8	4%	1026B5A8	000015A8	9%	9%	9%	9%	9%	9%	9%	9%	9%	9%	9%
	ipcom_egd	10533608	10534E08	10534CD0	5%	10534B00	000011F8	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%
	tNetConfig	10557552	10557CB8	10557A4E	6%	10557870	000011F8	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%	17%
	ipcom_tickd	10563A98	10564000	10563F5C	6%	10563F5C	00001664	6%	6%	6%	6%	6%	6%	6%	6%	6%	6%	6%
	tShe10	10566570	10576570	105762C8	1%	10576118	0000FBAB	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%
	tSieve	102B0110	102B08E0	102B0818	10%	102B07D0	000006C0	13%	13%	13%	13%	13%	13%	13%	13%	13%	13%	13%

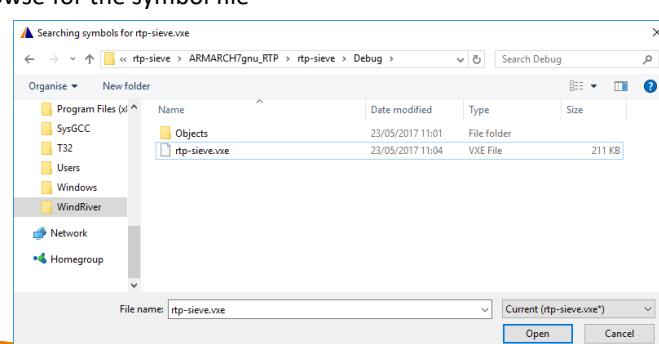
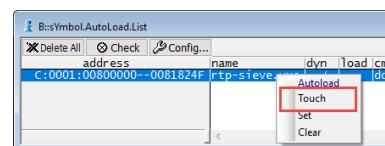
## Example

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



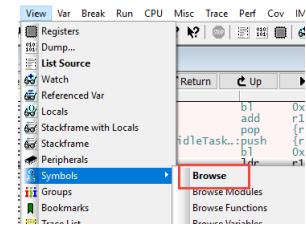
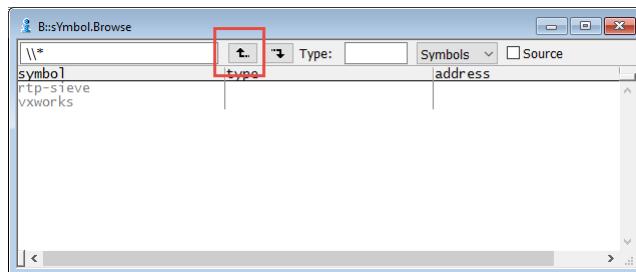
## Example

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



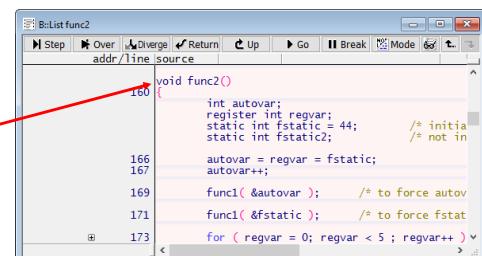
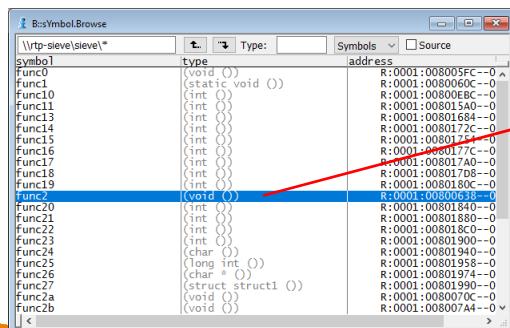
## Example

- 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

```

B:List func2
Step | Over | Diverge | Return | Go | Break | Mode | Up | ...
addr/line source

160 void func2()
161     int autovar;
162     register int regvar;
163     static int fstatic = 44; /* initial value */
164     static int fstatic2; /* not initialized */
165
166     autovar = regvar = fstatic;
167     autovar++;
168
169     func1( &autovar ); /* to force autovar to be initialized */
170     func1( &fstatic ); /* to force fstatic to be initialized */
171
172     for ( regvar = 0; regvar < 5 ; regvar++ ) {
173         ...
174     }
  
```

## 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 continuel

GO
STOP      // halt script until breakpoint reached

// Breakpoint hit, load symbols
continuel:

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

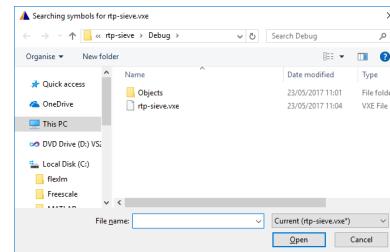
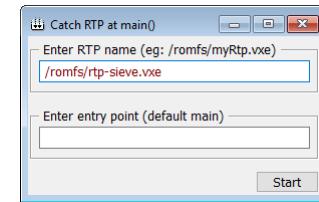
```
sYmbol.AutoLOAD.CHECK
sYmbol.AutoLOAD.TOUCH "&filename"

// 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

```

B:List.auto
Step Over Diverge Return Up Go Break Mode Find
addr/line source
int main (
    int argc,
    char * argv[] /* number of arguments */
)
{
    TASK_ID ts1, ts2;
    printf ("Hello World from RTP!\n");
    msgqRtp1 = msgQCreate (10, sizeof(msg1)+2, MSG_Q_PRIORITY);
    msgqRtp2 = msgQCreate (10, sizeof(msg1)+2, MSG_Q_FIFO);
    ts1 = taskSpawn ("tRtpSieve1", 30, 0, 2000, (FUNCPTR) tas
    ts2 = taskSpawn ("tRtpSieve2", 31, 0, 2000, (FUNCPTR) tas
    FOREVER
    {
        nanosleep (&waittime2, NULL);
        if (endRtp)
            break;
    }
    return 0;
}

```

## 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?

[www.lauterbach.com](http://www.lauterbach.com)

